COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

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

Autor: Mario Kroll Merino

Director: Santiago Canales Cano

Madrid

Junio 2025

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

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

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

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

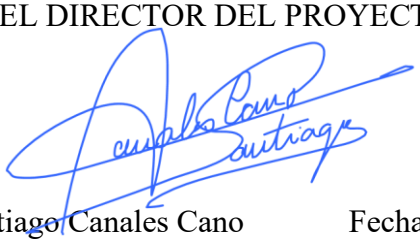
El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Mario Kroll Merino Fecha: 17/06/2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Santiago Canales Cano Fecha: 16/06/2025

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

Autor: Kroll Merino, Mario.
Director: Canales Cano, Santiago.
ICAI – Universidad Pontificia Comillas

Resumen

La visibilidad con alcance limitado es un problema relevante en geometría computacional con múltiples aplicaciones en robótica, planificación urbana y gráficos por ordenador. Este trabajo se centra en una de sus variantes: determinar el punto dentro de un polígono simple cuya región de visibilidad, limitada por un alcance fijo, sea máxima. Dado que este problema es de naturaleza **NP-hard**, se propone una aproximación basada en técnicas metaheurísticas, como son Algoritmo Genético, Simulated Annealing y Búsqueda Tabú. Los dos últimos se combinan con Random Search como técnica de inicialización. Para el funcionamiento de estos algoritmos, se han desarrollado dos nuevos algoritmos para calcular la región de visibilidad.

Palabras clave: Visibilidad, alcance limitado, polígonos, Algoritmo Genético, Simulated Annealing, Búsqueda Tabú, Random Search, **NP-hard**.

1. Introducción

La memoria parte del campo de la Geometría Computacional, en concreto de los problemas de visibilidad surgidos a raíz del célebre "problema de la galería de arte" de Klee-Chvátal [1]. El trabajo aborda la versión con alcance limitado propuesta por Ntafos [2], donde dos puntos solo se ven si el segmento que los une no se cruza con obstáculos *y* su distancia es ≤ *r*. El objetivo general es *maximizar la región de visibilidad de un punto interior a un polígono simple dado un alcance* d; se trata de un problema **NP-hard**, por lo que se recurre a metaheurísticas para hallar soluciones casi óptimas en tiempos razonables**.**

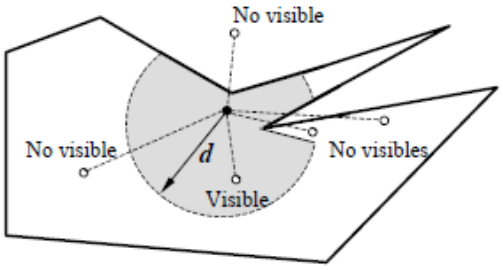


Ilustración 1: Visibilidad de alcance limitado d. En la imagen se pueden apreciar todos los casos de puntos visibles y no visibles dentro del contexto del problema. (Adaptado de Canales Cano S., (2004, p.56)).

2. Definición del proyecto

Para calcular la región de visibilidad de alcance limitado, se introducen dos nuevos algoritmos en la literatura: el primero de complejidad cuadrática y el segundo de complejidad lineal. Estos algoritmos serán usados por las técnicas metaheurísticas Algoritmo Genético, Simulated Annealing y Búsqueda Tabú, para tratar de buscar el punto que maximice el área visible. Para ver y evaluar su funcionamiento, se ha preparado una aplicación interactiva y

un estudio comparativo. Se ha hecho uso de un generador aleatorio de polígonos para generar 100 polígonos de 25 vértices sobre los que evaluar dichos algoritmos. En dicha evaluación, se consideran dos alcances distintos: 80 y 150 píxeles.

3. Descripción del sistema

El software empleado para el desarrollo de este proyecto se puede dividir en 3 grandes grupos: algoritmos, interfaz, y datos y resultados. En el apartado de algoritmos se incluyen el generador de polígonos aleatorio, los dos algoritmos para calcular el área de visibilidad y las técnicas metaheurísticas para determinar la solución al problema. Para desarrollar la interfaz, se ha hecho uso de la librería pygame. La aplicación incluye una serie de escenas que permiten ver de primera mano el funcionamiento de los algoritmos, permitiendo ejecutar 1 iteración, 10 iteraciones o resolverlo directamente. Por último, se incluye la base de datos de polígonos sobre la que se han hecho los experimentos, así como los resultados obtenidos.

4. Metodología

Los algoritmos propuestos parten ambos del polígono de visibilidad de p con respecto a P. A partir de este se determinan los puntos de intersección de la circunferencia con dicho polígono y, para cada par de puntos consecutivos, ambos algoritmos evalúan si debemos considerar para esa sección el área del sector circular o del polígono compuesto por el punto candidato, ambos puntos de intersección y a veces por los lados del polígono que unan los puntos de intersección.

Para los experimentos, se ha hecho uso de las tres técnicas mencionadas anteriormente, con una selección de parámetros determinada mediante observaciones. Además, se ha probado también a proporcionar como la solución inicial de Simulated Annealing y Búsqueda Tabú la solución de aplicar Random Search con un máximo de 100 iteraciones.

5. Resultados

Tras correr los algoritmos en la base de datos de polígonos previamente mencionada, se han recogido tres métricas diferentes: tiempo tardado por polígono, área total cubierta y porcentaje del área del polígono cubierta.

El Algoritmo Genético es el que mejor ha funcionado de todos en términos de resultados, obteniendo en las métricas de área total cubierta y porcentaje de área iluminado (ambos en promedio) puntuaciones más altas de las otras técnicas. Sin embargo, es el que más tiempo tarda, con mucha diferencia respecto a las otras dos técnicas.

Sin embargo, se ha observado que aplicando Random Search al inicio los algoritmos mejoran sus resultados, llegando a obtener resultados mucho más parecidos a los del Algoritmo Genético sin un incremento del tiempo de ejecución sustancial.

Si aumentamos el alcance, el tiempo de computación aumenta para Simulated Annealing y para Búsqueda Tabú, ya que cada vez es más complicado que el área de visibilidad del candidato sea la circunferencia de radio el alcance (solución óptima), con lo cual tiene que iterar más.

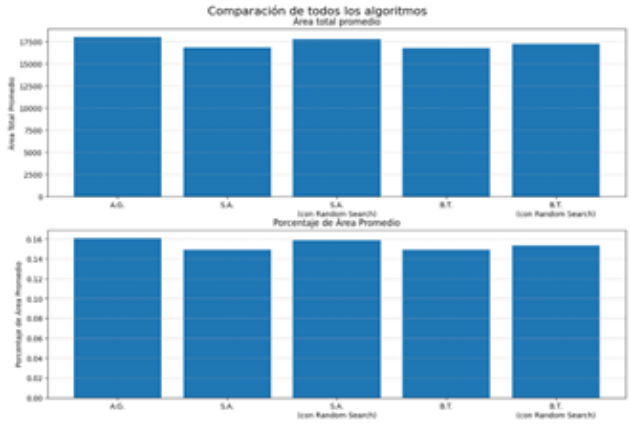


Ilustración 2: Resultados del área total y del porcentaje de área para los 5 algoritmos. Las columnas están ordenadas de la siguiente manera, de izquierda a derecha: Algoritmo Genético, Simulated Annealing, Simulated Annealing con Random Search, Búsqueda Tabú y Búsqueda Tabú con Random Search.

6. Conclusiones

Este trabajo ha abordado el problema de maximizar la región de visibilidad con alcance limitado de un punto dentro de un polígono simple, un problema conocido por ser **NP-hard**. Para aproximar una solución, se han implementado tres técnicas metaheurísticas: Algoritmo Genético, Simulated Annealing y Búsqueda Tabú. Además, se han desarrollado dos nuevos algoritmos para calcular la región de visibilidad de forma eficiente.

Los experimentos mostraron que el Algoritmo Genético obtiene los mejores resultados en cuanto a área total y porcentaje de área cubierta. Sin embargo, su alto coste computacional lo hace menos adecuado para aplicaciones con restricciones de tiempo. Por otro lado, Simulated Annealing y Búsqueda Tabú, aunque suelen producir soluciones ligeramente peores, son significativamente más rápidos y pueden beneficiarse de una solución inicial proporcionada por Random Search.

También se analizó la influencia del alcance de visibilidad en el rendimiento. Se observó que aumentar el alcance tiende a incrementar el tiempo de ejecución en la mayoría de los algoritmos, salvo en el caso del Algoritmo Genético. Esto se debe a la mayor dificultad de encontrar regiones óptimas a medida que el área de visibilidad aumenta.

En conclusión, combinar Random Search con técnicas de búsqueda local como Simulated Annealing o Búsqueda Tabú ofrece un buen equilibrio entre calidad de la solución y tiempo de ejecución, proporcionando una alternativa eficaz a métodos más costosos como el Algoritmo Genético.

7. Referencias

[1] Chvátal, V. (1975). *A combinatorial theorem in plane geometry*. *Journal of Combinatorial Theory, Series B, 18*(1), 39–41. https://doi.org/10.1016/0095-8956(75)90061-1

[2] Ntafos, S. (1992). *Watchman routes under limited visibility*. *Computational Geometry: Theory and Applications, 1*(3), 149–170. https://doi.org/10.1016/0925-7721(92)90014-J

[3] Auer†, T., & Held†, M. (1996). Heuristics for the generation of random polygons. *Canadian Conference on Computational Geometry*, 38–43. https://doi.org/10.1515/9780773591134-009

Maximization with Metaheuristic Techniques of the Limited Visibility Region from a Point in a Polygon

Author: Kroll Merino, Mario.
Supervisor: Canales Cano, Santiago.
ICAI – Universidad Pontificia Comillas

Abstract

Limited range visibility is a relevant problem in computational geometry with multiple applications in robotics, urban planning, and computer graphics. This work focuses on one of its variants: determining the point inside a simple polygon whose visibility region, limited by a fixed range, is the largest. Since this problem is NP-hard in nature, a metaheuristic-based approach is proposed, using Genetic Algorithm, Simulated Annealing, and Tabu Search. The last two are combined with Random Search as an initialization technique. To ensure the correct performance of these algorithms, two new algorithms have been developed to compute the visibility region.

Keywords: Visibility, limited range, polygons, Genetic Algorithm, Simulated Annealing, Tabu Search, Random Search, **NP-hard**.

1. Introduction

This thesis is based in the field of Computational Geometry, specifically on visibility problems that emerged from the famous "Art Gallery Problem" by Klee-Chvátal [1]. The work focuses on the limited range version proposed by Ntafos [2], where two points can see each other only if the segment connecting them does not intersect any obstacles and their distance is $\leq r$. The general goal is to maximize the visibility region of a point inside a simple polygon given a range d; this is an **NP-hard** problem, so metaheuristics are used to find near-optimal solutions in reasonable time.

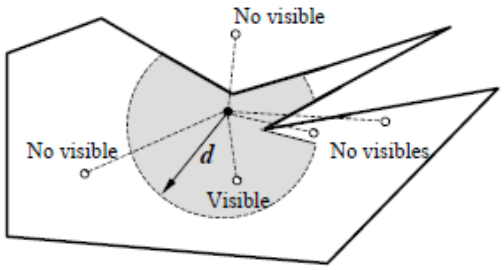


Illustration 3: Limited range visibility d. The image shows all possible cases of visible and non-visible points within the context of the problem. (Adapted from Canales Cano S., (2004, p.56)).

2. Project definition

To compute the visible region, two new algorithms are introduced in the literature: the first with quadratic complexity and the second with linear complexity. These algorithms are used by the metaheuristic techniques—Genetic Algorithm, Simulated Annealing, and Tabu Search—to try to find the point that maximizes the visible area. To visualize and evaluate their performance, an interactive application and a comparative study were prepared. A random polygon generator was used to create 100 polygons with 25 vertices, which were

used to test the algorithms. In this evaluation, two different ranges were considered: 80 and 150 pixels.

3. System description

The software used for the development of this project can be divided into three main groups: algorithms, interface, and data and results. The algorithms section includes the random polygon generator, the two algorithms for computing the visibility area, and the metaheuristic techniques used to solve the problem. To develop the interface, the Pygame library was used. The application includes a series of scenes that allow users to see how the algorithms work in real time, with the option to run 1 iteration, 10 iterations, or solve the problem directly. Finally, the database of polygons used for the experiments is included, along with the results obtained.

4. Methodology

Both proposed algorithms start from the visibility polygon of point p with respect to P. From this polygon, the intersection points between the circle and the polygon are determined. For each pair of consecutive points, both algorithms evaluate whether the area for that section should be calculated using the circular sector or the polygon formed by the candidate point, the two intersection points, and sometimes the polygon edges that connect them.

For the experiments, the three techniques mentioned earlier were used, with parameter settings selected based on observation. Additionally, it was tested whether using the result of applying Random Search (with a maximum of 100 iterations) as the initial solution for Simulated Annealing and Tabu Search could improve their performance.

5. Results

After running the algorithms on the previously mentioned polygon database, three different metrics were collected: time taken per polygon, total area covered, and percentage of the polygon area covered.

The Genetic Algorithm performed the best of all in terms of results, achieving the highest average scores in both total area covered, and percentage of area illuminated. However, it is also the slowest by far compared to the other techniques.

However, it was observed that using Random Search at the beginning improves the results of the algorithms, making them much closer to those of the Genetic Algorithm, without a significant increase in execution time.

Another study was conducted to determine how the visibility range affects execution time. It was found that, except for the Genetic Algorithm, increasing the range negatively impacts the time needed to solve the problem. This is because, as the range increases, it becomes more difficult for the candidate's visibility area to be a full circle with radius equal to the range (the optimal solution), so more iterations are required.

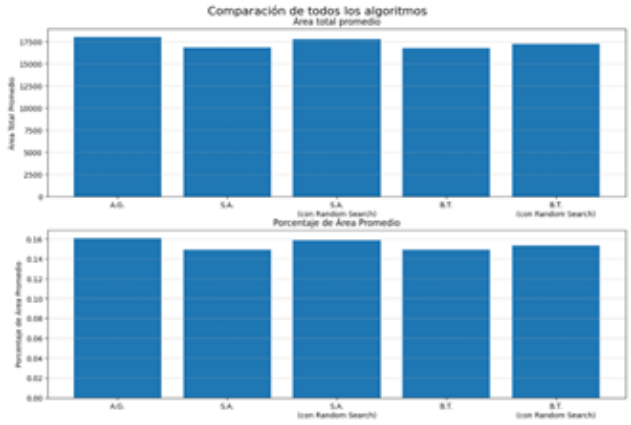


Illustration 4: Results for total area and percentage of area for the 5 algorithms. The columns are ordered as follows, from left to right: Genetic Algorithm, Simulated Annealing, Simulated Annealing with Random Search, Tabu Search, and Tabu Search with Random Search.

6. Conclusions

This work has addressed the problem of maximizing the limited visibility region of a point inside a simple polygon, a problem known to be NP-hard. To approximate a solution, three metaheuristic techniques were implemented: Genetic Algorithm, Simulated Annealing, and Tabu Search. Additionally, two new algorithms were developed to compute the visibility region efficiently, which are essential to evaluate the quality of each candidate point.

The experiments showed that the Genetic Algorithm achieves the best results in terms of both total area and percentage of area covered. However, its high computational cost makes it less suitable for time-constrained applications. On the other hand, Simulated Annealing and Tabu Search, while generally producing slightly worse solutions, are significantly faster and can benefit from an initial solution provided by Random Search.

The influence of the visibility range on performance was also analyzed. It was found that increasing the range tends to increase execution time for most algorithms, except the Genetic Algorithm. This is due to the added difficulty in finding optimal regions as the area of visibility expands.

In conclusion, combining Random Search with local search techniques such as Simulated Annealing or Tabu Search provides a good balance between solution quality and execution time, offering an effective alternative to more expensive methods like the Genetic Algorithm.

7. References

[4] Chvátal, V. (1975). *A combinatorial theorem in plane geometry*. *Journal of Combinatorial Theory, Series B, 18*(1), 39–41. https://doi.org/10.1016/0095-8956(75)90061-1

[5] Ntafos, S. (1992). *Watchman routes under limited visibility*. *Computational Geometry: Theory and Applications, 1*(3), 149–170. https://doi.org/10.1016/0925-7721(92)90014-J

[6] Auer†, T., & Held†, M. (1996). Heuristics for the generation of random polygons. *Canadian Conference on Computational Geometry*, 38–43. https://doi.org/10.1515/9780773591134-009

Índice de la memoria

Índice de figuras

1. INTRODUCCIÓN

1.1. INTRODUCCIÓN A LA GEOMETRÍA COMPUTACIONAL

La Geometría Computacional es la rama de las matemáticas y de la computación que estudia el diseño e implementación de algoritmos que resuelven problemas geométricos. Se desarrolla a partir de la geometría analítica y pretende implementar matemáticas avanzadas en computación mediante algoritmos. En el diseño de estos algoritmos, se busca reducir la cantidad de recursos necesarios para su implementación al mínimo, tratando de reducir tanto el espacio utilizado como el tiempo empleado. Esto introduce una serie de desafíos no triviales de resolver, especialmente cuando se manejan grandes volúmenes de datos. Por lo tanto, para resolver un problema de geometría computacional es necesario comprender la geometría del problema e implementar conceptos algorítmicos y estructuras de datos, utilizando de la manera más eficiente posible los recursos disponibles.

La Geometría Computacional aborda una gran cantidad de problemas clásicos y contemporáneos. Entre ellos destacan el cálculo del cierre convexo de una nube de puntos, problema ampliamente estudiado y sirve de punto de partida para algoritmos más complejos (ver sección 3.1); detección de colisiones e intersección de segmentos, que es un problema con aplicaciones en el mundo físico, como la planificación de trayectorias o la simulación física; la triangulación de polígonos y superficies para dividir un polígono en triángulos adyacentes, ampliamente usado en técnicas de renderizado de imágenes y siluetas.

Otra rama importante es la que trata con los diagramas de Voronoi y la triangulación de Delaunay. El diagrama de Voronoi divide el espacio en regiones que contienen los puntos más cercanos a un punto perteneciente a la nube de puntos que a cualquier otro punto de la nube. La triangulación de Delaunay representa el dual del diagrama de Voronoi y conecta los puntos de la región de Voronoi cuyas regiones compartían un lado. Se aplica ampliamente en redes de telecomunicaciones y en la geolocalización. Por ejemplo, dónde tengo que construir los hospitales de una ciudad para que cada ciudadano tenga un hospital a mínimo 15 minutos andando desde su casa.

Como se puede apreciar, la Geometría Computacional es una rama de las matemáticas y de la informática que está ampliamente presente en nuestro día a día, lo que despierta el interés para tratar problemas relacionados con este tema.

1.2. PROBLEMAS DE VISIBILIDAD

Una de las ramas de la geometría computacional que más interés atrae por sus aplicaciones al mundo físico es el de la visibilidad. Esta rama nace en 1973, cuando V. Klee introdujo el *problema de la galería de arte*. En él, se plantea la siguiente cuestión: ¿cuántas luces son a veces necesarias y siempre suficientes para iluminar un polígono P con n vértices?

Este problema fue resuelto por Chvátal [1] en el Teorema de la Galería de Arte, que logró demostrar que $\left\lfloor \frac{n}{3} \right\rfloor$ luces son siempre suficientes y a veces necesarias para iluminar todo el polígono. La idea de esta solución proviene de la afirmación de que todo polígono admite al menos una triangulación, es decir, se puede dividir en triángulos. Si utilizamos una luz por triángulo, usamos una luz por cada 3 vértices, quedando demostrado así que $\left\lfloor \frac{n}{3} \right\rfloor$ luces son siempre suficientes. Esto se puede visualizar en la Figura 1. En ella, se puede apreciar un polígono y su correspondiente triangulación. Si en cada vértice de cada triángulo se pinta un punto de un color diferente, con la única condición de que dos vértices del mismo color no pueden estar consecutivos, basta con tomar los vértices del color menos frecuente para tener el polígono iluminado, utilizando como máximo $\left\lfloor \frac{n}{3} \right\rfloor$ luces.

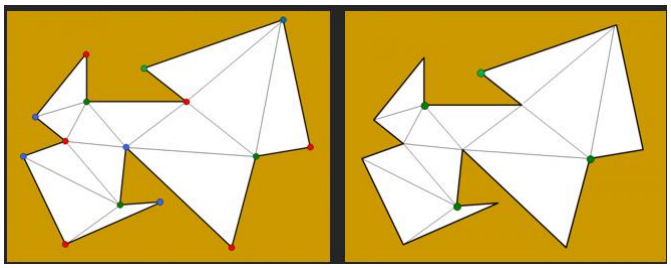


Figura 1: Demostración visual de la solución propuesta por Chvátal. De izquierda a derecha: triangulación del polígono, asignando a cada vértice del triángulo un color distinto al de sus vecinos; elección del color menos frecuente, obteniendo una solución que no supera a la cota teórica, como es esperable. *Nota:* Imagen utilizada con permiso del autor. Adaptado de Pérez Bienzobas, D. (2023, p.7) [12].

Como la triangulación se puede realizar de varias maneras, no siempre obtendremos el número mínimo de luces. En muchos casos, son necesarias menos, llegando en algunos polígonos a necesitar únicamente una luz, como ocurre en los polígonos convexos (Figura 2).

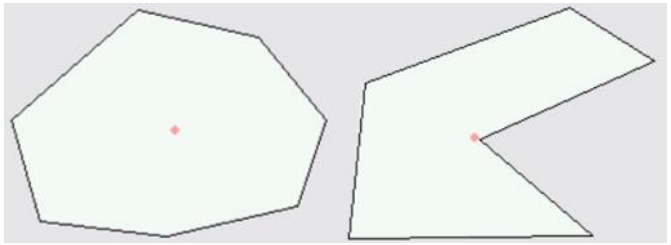


Figura 2: Ejemplo de un polígono convexo y uno cóncavo que solo necesitan una luz para iluminarlo entero. Los puntos que aparecen son las únicas luces necesarias.

La rama de visibilidad presenta una nueva área de investigación, que resulta de gran utilidad en campos como Robótica, Arquitectura, Computación Gráfica o Urbanismo, entre otros.

1.3. PROBLEMAS DE VISIBILIDAD CON ALCANCE LIMITADO

La introducción de estos problemas abrió una nueva área de estudio que hoy resulta fundamental en diversos campos. Sin embargo, pronto surgió un problema en cuanto a las aplicaciones de las soluciones de los problemas al mundo físico: la visibilidad no tiene alcance ilimitado. Por lo tanto, a raíz de este nuevo problema, Ntafos estableció el concepto de *visibilidad de alcance limitado* [2], en el que se define que dos puntos p y q son visibles entre sí siempre y cuando:

- $dist(p,q) \leq d$
- No exista ningún obstáculo que obstruya el segmento $\overline{pq}$.

, siendo d un dato del problema y $dist(p,q)$ la distancia euclídea en el plano entre dos puntos. En términos poligonales, dados un punto p en el interior de un polígono P y un alcance d, la región de visibilidad con alcance limitado d del punto p en el polígono P está formada por todos los puntos $q \in Int(P)$, tales que:

- $dist(p,q) \leq d$
- $\overline{pq} \cap P = \emptyset$

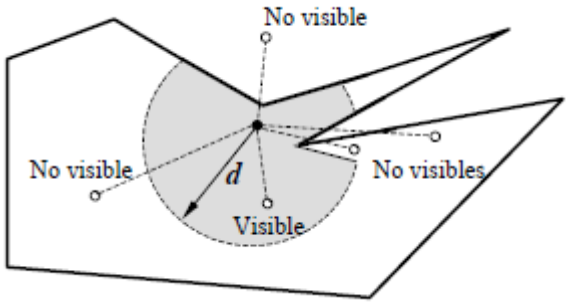


Figura 3: Visibilidad de alcance limitado d. En la imagen se pueden apreciar todos los casos de puntos visibles y no visibles dentro del contexto del problema. (Adaptado de Canales Cano S., (2004, p.56)).

A raíz de esta propuesta, surge un nuevo problema a resolver*: dónde tengo que colocar un punto p dado un polígono P, para que la región de visibilidad de p en P con alcance limitado d sea máxima*. Este es el problema que se tratará en este presente trabajo.

1.4. COMPLEJIDAD DEL PROBLEMA

La complejidad computacional de un problema es una función que describe, en términos del tamaño de la entrada n, la cantidad mínima de recursos computacionales necesarios para resolver cualquier instancia del problema. Los recursos considerados más comúnmente son el tiempo (número de

operaciones elementales requeridas) y el espacio (cantidad de memoria utilizada). En función de los recursos, existen diferentes clases de problemas. Las dos principales son los problemas de tipo **P**, que son los problemas que se pueden resolver de manera determinista en tiempo polinómico; y los problemas **NP**, que tratan de aquellos problemas para los cuales no existe ningún algoritmo que los resuelva de manera determinista de manera óptima. Para tratar estos problemas, se hace uso de heurísticas o técnicas metaheurísticas. Se hablará más adelante acerca de estas últimas.

Por otra parte, existe otra categoría de problemas que son los denominados **NP-hard**. Estos problemas no se encuentran dentro de la rama de los problemas **NP**. Si asumimos que **P≠NP**, los problemas **NP-hard** comparten con los problemas **NP** únicamente aquellos que son **NP-completos**. Los problemas **NP-completos** son problemas de decisión que pertenecen a la clase **NP** y son tan difíciles de resolver como cualquier otro problema **NP**. En caso de que **P=NP**, entonces **P=NP=NP-completos**, y los problemas **NP-hard** pueden estar en **P** o no estarlo. La estructura de los problemas según su complejidad se puede observar en la Figura 4.

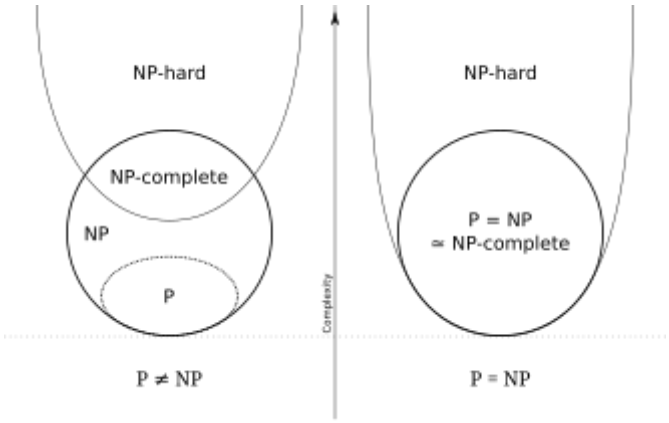


Figura 4: Estructura de los problemas según su complejidad computacional. A la izquierda, se asume que **NP≠P**. Por el contrario, a la derecha se muestra la estructura en el caso de que sea cierto que **NP=P**. (Fuente: https://en.wikipedia.org/wiki/NP-hardness)

Por lo tanto, los problemas **NP-hard** pueden ser o no ser **NP-completos**, lo que quiere decir que, dada una solución del problema, puede que no sea posible verificarla en un tiempo polinomial.

En relación con el problema a tratar en esta memoria, el problema de minimizar el número de luces punto que iluminan un polígono fue estudiado por Lee y Lin [12] y utilizando una reducción al problema 3-Sat, lograron demostrar que son problemas pertenecientes a la categoría de **NP-hard**.

Debido a que este problema es de categoría **NP-hard**, no existen algoritmos que lo resuelvan en tiempo polinómico y comprobar una solución de este problema puede realizarse también en tiempo no polinómico. Por ello, es necesario recurrir a técnicas metaheurísticas que proporcionan soluciones aproximadas y cuasi-óptimas. Las técnicas que se utilizarán para resolver este problema serán Algoritmo Genético, Simulated Annealing y Búsqueda Tabú. Cabe destacar también el uso de Random Search para determinar la solución inicial de la que partirán los algoritmos de Simulated Annealing y Búsqueda Tabú. En la sección 3 se puede encontrar una descripción más detallada sobre cada algoritmo.

2. ESTADO DEL ARTE

El concepto de visibilidad de alcance limitado fue establecido primeramente por Ntafos en 1992 [2]. Lo que puede parecer una pequeña ampliación en la teoría de visibilidad que se tenía hasta entonces, realmente abría una nueva rama de investigación para la que los teoremas y teorías previamente establecidos no funcionaban. A raíz de esta nueva rama, surgieron nuevos problemas de combinatoria, como, por ejemplo: Dado un alcance d, ¿cuál es el número mínimo de luces que necesita un polígono P para que quede completamente iluminado?; ¿Para qué valores de d de puede iluminar todo el polígono con un solo punto?

En este documento se tratará el siguiente problema: *determinar el punto p dentro de un polígono P, tal que dado un alcance d la región de visibilidad sea máxima.*

Tratar con problemas de categoría **NP-hard** no es una tarea sencilla y menos un proceso trivial. Requiere de un entendimiento profundo del problema a resolver y un conocimiento amplio de la rama a la que pertenece el problema. Por ello, se han desarrollado librerías como la de TriVis [5] que incluyen una gran variedad de algoritmos para calcular regiones de visibilidad de polígonos con y sin obstáculos de manera óptima. Entre otros, se incluye el algoritmo de TEA [6], que permite calcular la región de visibilidad mediante la triangulación de un polígono en orden $O(nh)$, siendo n el número de lados y h el número de agujeros del polígono (aunque no se tratará con este tipo de polígonos), pero que en la práctica lo hace en $O(n)$. Específicamente para este algoritmo, paradójicamente, se observa que añadiendo un alcance d, calcular el área de visibilidad es computacionalmente menos costoso que calcular la región de visibilidad sin alcance. Si bien es cierto que el algoritmo TEA devuelve un polígono, basta con intersecarlo con la circunferencia con radio d para encontrar el área exacta. Sin embargo, al estar la librería únicamente disponible en C++, se ha optado por implementar los algoritmos, en vez de hacer uso de la librería. El algoritmo que se ha implementado para calcular la región de visibilidad es el de *ray casting*.

En cuanto al estudio teórico sobre la relación entre el alcance y el número mínimo de luces necesarias para iluminar todo el polígono, en la tesis de Canales, S. [7] para polígonos escaleras y polígonos pirámide se realiza un estudio exhaustivo para responder a algunas de las preguntas que se han planteado al inicio de esta sección. Los polígonos escalera se caracterizan por "tener un lado horizontal h (respectivamente vertical v) cuya longitud es igual a la suma de los lados restantes horizontales (respectivamente verticales)" (Canales Cano, S., 2004, p.56).

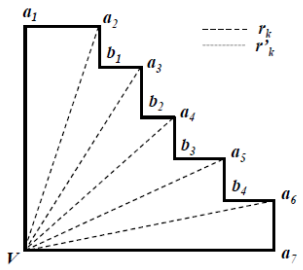


Figura 5: Polígono escalera. (Adaptado de Canales Cano, S. (2004, p.57))

Por otra parte, los polígonos pirámide son aquellos "polígonos ortogonales tal que existe un lado horizontal cuya longitud es igual a la suma de las longitudes de los restantes lados horizontales" (Canales Cano, S., 2004, p.64).

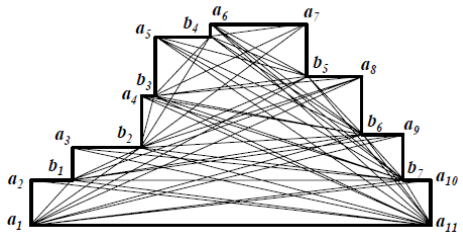


Figura 6: Polígono pirámide y su grafo de visibilidad (Adaptado de Canales Cano, S. (2004, p.64)).

Además de los polígonos escalera y pirámide, existen otros estudios que analizan las preguntas planteadas para polígonos convexos [8] y también para visibilidad desde un lado concreto de un polígono [9].

En esta memoria no se estudiarán tipos concretos de polígonos, si no que se tratará de ver cómo solucionar este problema, presentando dos nuevos algoritmos para calcular la región de visibilidad de un punto en un polígono con visibilidad limitada y evaluando cómo se desenvuelven las distintas metaheurísticas para el mismo problema.

3. METODOLOGÍA

En esta sección se explicará en detalle cada uno de los algoritmos que se han implementado, resumiendo su funcionamiento y analizando su complejidad.

Los algoritmos implementados para este proyecto se reparten en distintas categorías: construcción aleatoria de polígonos, cálculo de la región de visibilidad y determinación de una solución del problema.

3.1. GENERADOR ALEATORIO DE POLÍGONOS

Para realizar estudios comparativos se ha generado una base de datos de polígonos aleatorios, que nos permite elaborar dicho estudio. Ha sido por tanto necesario implementar un generador aleatorio de polígonos que fuera capaz de construir polígonos de n vértices, siendo n un número fácilmente modificable. Por ello, se ha escogido el generador propuesto por Auer y Held [3], que es capaz de generar polígonos con un tiempo de computación $O(n^2)$.

En este generador se parte de inicialmente de conjunto $C = \{v_1, v_2, \dots, v_n\}$ de n puntos en el plano, distribuidos de manera uniforme. A partir de este conjunto, se quiere construir el polígono simple P, cuyos vértices son los puntos del conjunto C.

El algoritmo parte de un conjunto inicial de tres puntos que forman un triángulo. Estos tres vértices, pertenecientes al conjunto original C, forman un triángulo dentro del cual no se encuentra ningún otro punto de C. A partir de esta terna, comienza la construcción del polígono aleatorio. Para determinar la terna inicial, se ha decidido tomar dos vértices aleatorios del conjunto C y buscar el tercer punto tal que ningún otro punto de C se encuentre dentro del triángulo (Figura 7).

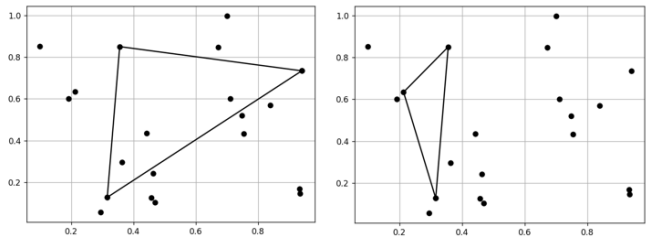


Figura 7: De izquierda a derecha: Terna de puntos no válida; Terna de puntos válida

Una vez se dispone del triángulo inicial, partimos de los siguientes conjuntos: $P = \{p_1, p_2, p_3\}$, cuyos puntos se corresponden a los tres vértices del triángulo, y por tanto pertenecen al conjunto C; y $S = C \setminus P$. En cada iteración $0 \leq i \leq n-3$, tenemos el conjunto P_i, que consistirá en el polígono que se tenga actualmente; y S_i, formado por los puntos que todavía no formen parte de P_i. De este modo, se tiene que $C = P_i \cup S_i, \forall\, 0 \leq i \leq n-3$.

Debido a que es un algoritmo iterativo, en cada iteración i se busca un vértice $v_k \in S_i$ de tal manera que el cierre convexo de $P_i \cup v_k$ (a partir de ahora se referirá al cierre convexo de un conjunto K como $CH(K)$) no contenga ningún otro vértice de $S_i \setminus v_k$.

Una vez determinado el punto v_k que cumpla las condiciones anteriores, tenemos que buscar una arista $\overline{p_j p_{j+1}}$, con $\overline{p_j, p_{j+1}} \in P_i$ que sea completamente visible desde v_k. Esto siempre se cumple, porque v_k se encuentra fuera de $CH(P_i)$. Después de determinar la arista, basta con introducir el punto v_k como un nuevo punto del polígono. Para ello, se elimina la arista $\overline{p_j p_{j+1}}$ y se construyen dos nuevas aristas: $\overline{p_j v_k}$ y $\overline{v_k p_{j+1}}$. De este modo, se consigue construir un polígono simple a partir de una nube de puntos distribuida de manera uniforme.

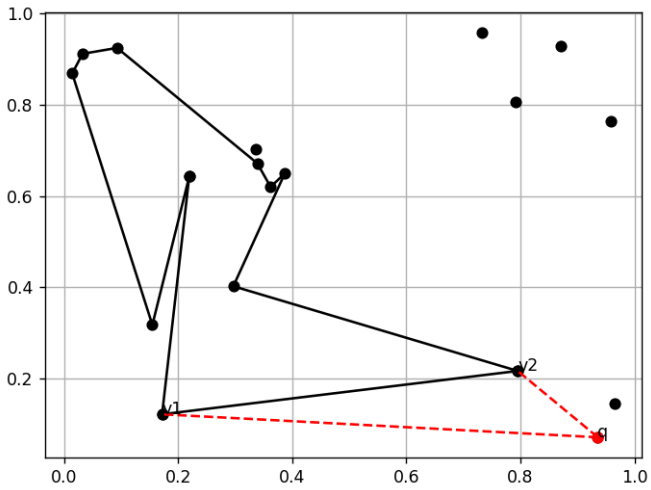


Figura 8: Inserción de un punto en el polígono actual. Nótese que el cierre convexo del polígono, más el punto q no contiene ningún otro punto que no sea del polígono P_i. La arista $\overline{v_1 v_2}$ es, este caso, la única arista visible desde q.

3.2. CÁLCULO DE LA REGIÓN DE VISIBILIDAD

En este apartado, se explicará cómo se ha determinado la región de visibilidad, exponiendo dos métodos nuevos para hacerlo: el primero que lo realiza en complejidad cuadrática y el segundo que lo realiza en complejidad lineal.

Como ya se ha explicado antes, la región de visibilidad de un punto p en un polígono P con un alcance limitado d está compuesta por todos los puntos $q \in Int(P)$, tal que se cumplan las dos siguientes condiciones:

- $dist(p,q) \leq d$
- $\overline{pq} \cap P = \emptyset$

, donde $dist(p,q)$ determina la distancia euclídea en el plano entre dos puntos.

Ambos algoritmos requieren de una fase de inicialización que es común para los dos. Además, se introducirá nuevos términos para que la explicación sea lo más clara posible.

En primer lugar, debemos calcular el polígono de visibilidad de p con respecto al polígono P. Esto nos devuelve un polígono estrellado cuyo núcleo contiene a p. De esta manera, tenemos que todos los puntos del polígono estrellado son visibles desde p. Se denota a este polígono como $VP(p,P)$. En la aplicación se ha optado por implementar el algoritmo de *ray casting*, una técnica ampliamente conocida en geometría computacional, que consiste en lanzar semirrectas desde el punto p y computar sus intersecciones con los bordes del polígono para determinar la región de visibilidad. Este algoritmo tiene una complejidad computacional de $O(n^2)$. Sin embargo, existen algoritmos que son capaces de realizarlo en $O(n)$, como el de Joe & Simpson [10].

Una vez disponemos del polígono de visibilidad de p con respecto a P, debemos encontrar los puntos de intersección de la circunferencia que tenga centro en p y radio d, siendo d el alcance determinado al inicio del problema, con el polígono $VP(p,P)$. El conjunto de puntos de intersección de $VP(p,P)$ se denotará como $IP(p,P,d)$. Se asume que los puntos del conjunto $IP(p,P,d)$ están ordenados angularmente. Para que la programación de este algoritmo sea más sencilla, se ha optado por insertar los puntos de intersección como puntos del polígono $VP(p,P)$. Este nuevo polígono se llamará $VPI(p,P,d)$.

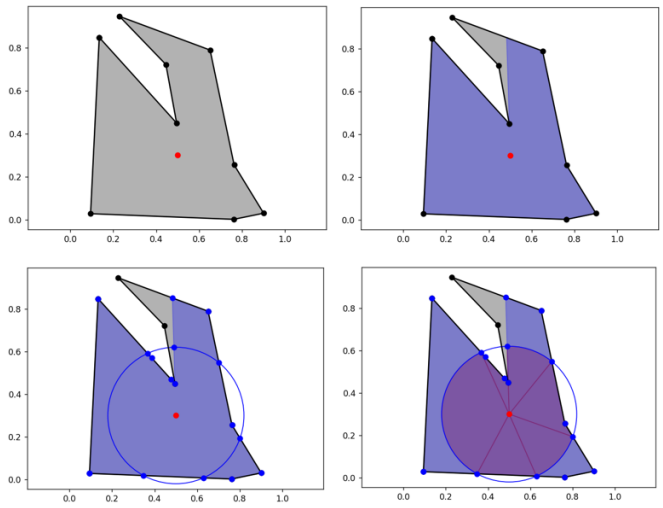


Figura 9: Representación de los pasos para determinar la región de visibilidad con alcance limitado. De izquierda a derecha, y de arriba a abajo: Polígono P y punto p; polígono de visibilidad sin restricciones de alcance $VP(p,P)$; polígono de visibilidad con las intersecciones del círculo con la figura $VPI(p,P,d)$ (los puntos que aparecen de más son el resultado de la aplicación del algoritmo de *ray casting*); región de visibilidad final con alcance limitado d.

Para calcular el área, debemos iterar para cada par de puntos consecutivos $q_i, q_{i+1} \in IP(p,P,d)$. A la hora de considerar el área que habrá que tener en cuenta para el cálculo del área visible, surgen dos posibles casos: considerar el sector circular entre q_i y q_{i+1}, o considerar el área del polígono formado por $\{q_i, p, q_{i+1}\}$ y todos aquellos puntos de $VPI(p,P,d)$ que se encuentren entre q_i y q_{i+1}. Nótese que en este último polígono no se incluye ningún punto de intersección además de q_i y q_{i+1}, pues se está tratando con puntos de intersección que son consecutivos. A continuación, se procederá a explicar cómo se puede determinar de dos maneras diferentes si hay que considerar el sector circular o el polígono.

3.2.1. ALGORITMO I

Sea k el número de puntos del conjunto $IP(p,P,d)$. En cada iteración $0 \leq i < k-1$, debemos calcular el punto medio entre q_i y q_{i+1}, con $q_i, q_{i+1} \in IP(p,P,d)$, al que llamaremos $MP(q_i,q_{i+1})$. Este punto medio será válido siempre que se cumpla que $\text{angle}(\overrightarrow{pq_i}, \overrightarrow{pq_{i+1}}) \leq \pi$. Si esto no ocurre, debemos tomar como punto medio el punto simétrico de $MP(q_i,q_{i+1})$ con respecto a p. Este punto medio servirá para calcular la semirrecta que tiene origen en p y pasa por $MP(q_i,q_{i+1})$, y se calcula la intersección de dicha semirrecta con $VPI(p,P,d)$, que se denotará t. Por la naturaleza tanto de la semirrecta como del polígono, existirá siempre una única intersección. Se distinguen dos casos en función de t.

- Si $dist(t,p) < d$, entonces debemos considerar el polígono previamente descrito que incluye los puntos $\{q_i, q_{i+1}, p\}$ y todos aquellos puntos pertenecientes a $VPI(p,P,d)$ que se encuentren entre q_i y q_{i+1}. El área que se sumará será el área de este polígono, ya que el área del sector circular sería mayor a la que realmente se debe considerar.
- Si $dist(t,p) > d$, entonces debemos considerar el sector circular con ángulo $\text{angle}(\overrightarrow{pq_i}, \overrightarrow{pq_{i+1}})$. En este caso, considerar el área del polígono sería erróneo, pues no estaríamos teniendo en cuenta el alcance d.

Como se puede deducir, este algoritmo es computacionalmente costoso, pues para cada par de puntos de intersección, necesitamos calcular otra intersección de una semirrecta con $VPI(p,P,d)$. La siguiente subsección introduce otro algoritmo que es computacionalmente menos demandante que este.

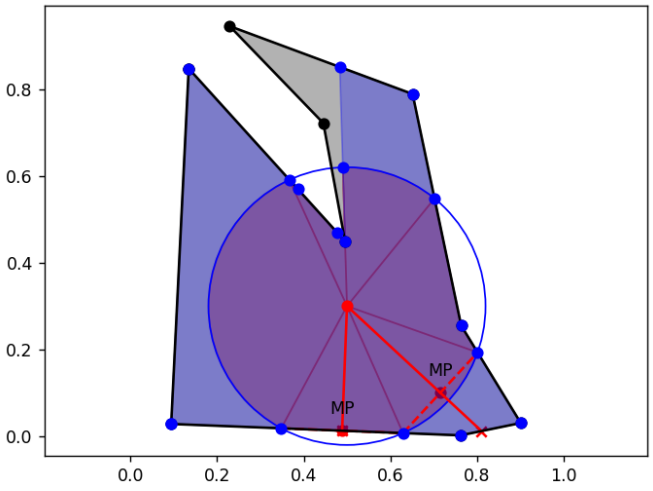


Figura 10: Representación del funcionamiento del Algoritmo 1. Se distinguen los dos casos: en uno de ellos la distancia desde el punto a la intersección es menor que el radio, por lo que nos quedamos con el polígono (triángulo en este caso); en el otro caso, el punto de intersección está más alejado que el radio, por lo que se usa el sector circular para añadir al área.

3.2.2. ALGORITMO II

Sea k el número de puntos del conjunto $IP(p,P,d)$. En cada iteración $0 \leq i < k-1$, debemos encontrar el punto $v \in VPI(p,P,d)$ que se encuentre a continuación de $q_i \in IP(p,P,d)$. Se distinguen tres casos:

- Si $dist(v,p) = d$, entonces $v = q_{i+1}$ y debemos considerar como área el triángulo formado por $\{q_i, p, q_{i+1}\}$.
- Si $dist(v,p) < d$, entonces debemos considerar el polígono previamente descrito.
- Si $dist(v,p) > d$, entonces debemos considerar el sector circular con ángulo $\text{angle}(\overrightarrow{pq_i}, \overrightarrow{pq_{i+1}})$.

Con este algoritmo, no hace falta encontrar intersecciones y basta con recorrer una única vez $VPI(p,P,d)$ para determinar el área de la región de visibilidad con alcance limitado d.

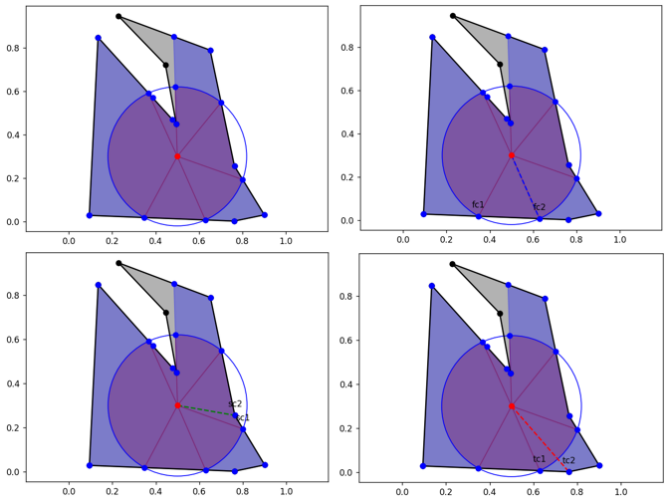


Figura 11: Distinción de casos para el algoritmo II. De izquierda a derecha, y de arriba a abajo: $VPI(p, P, d)$; primer caso, donde la distancia de $fc2$ al punto es igual al radio, así que consideramos el triángulo formado por $\{fc1, fc2, p\}$; segundo caso, donde la distancia de sc2 al punto p es menor que el radio; y tercer caso, donde la distancia de tc2 al punto es mayor que el radio.

3.3. DETERMINACIÓN DE LA SOLUCIÓN DEL PROBLEMA

En este apartado se describen los algoritmos que se han utilizado para resolver el problema planteado en este trabajo. Estos algoritmos se implementan mediante el uso de técnicas metaheurísticas, que se caracterizan por proporcionar soluciones que se acercan mucho a la solución óptimo en un tiempo razonable. Son las técnicas ideales para resolver problemas de naturaleza **NP-hard**, como el problema en cuestión y de los que se ha hablado antes.

Las tres técnicas que se han desarrollado en este apartado son Algoritmo de Recocido (Simulated Annealing), Algoritmo Genético y Búsqueda Tabú. En los siguientes subapartados se proporcionará una descripción sobre su funcionamiento. En la sección 4 se realiza un estudio comparativo detallado acerca del rendimiento de los distintos algoritmos.

3.3.1. SIMULATED ANNEALING

El algoritmo de Simulated Annealing fue introducido por primera vez en 1983, por Kirkpatrick, Gelatt y Vecchi [14]. Se trata de un algoritmo que se utiliza ampliamente para resolver algoritmos de la categoría **NP-hard**.

Este método está inspirado en un proceso físico, conocido como recocido. Es una técnica que se utiliza para ablandar metales y darle posteriormente su forma deseada. Para ello, se calienta el metal hasta la temperatura deseada y se deja enfriar poco a poco, hasta alcanzar la temperatura ambiente.

En el caso del algoritmo, realiza una búsqueda en el espacio de soluciones hasta que poco a poco converge a una solución. En un principio, se permiten soluciones de peor calidad que en pasos previos con una probabilidad más alta, para abandonar posibles mínimos locales y encontrar el mínimo global.

Según avanza el algoritmo, es lógico pensar que se está acercando a la solución global, por lo tanto, para evitar la exploración excesiva, reducimos la probabilidad en cada iteración para limitar la búsqueda aleatoria, priorizando movimientos a soluciones vecinas y limitando la exploración.

La probabilidad de aceptar una solución peor viene dada por la siguiente fórmula:

$$P(\Delta E) = e^{\frac{\Delta E}{T}}$$

, donde ΔE representa el cambio en el valor de la función objetivo y T la temperatura actual del sistema.

En nuestro caso, se propone una solución inicial y se realiza una pequeña perturbación sobre ella con una probabilidad $P(\Delta E)$. Esta perturbación. Se ha decidido realizar a través de una distribución normal con una desviación típica elegida manualmente.

Las perturbaciones se van aceptando o rechazando en función de la probabilidad. En cada iteración, se reduce la temperatura multiplicándolo por un factor

$0 < \alpha < 1$. Este factor es clave, porque indica cómo de rápido queremos dejar de explorar el espacio de soluciones.

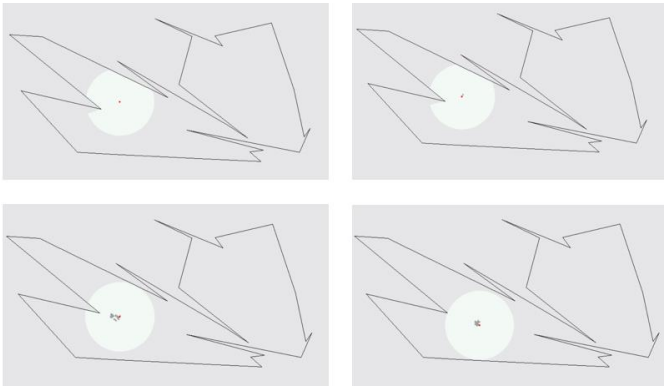


Figura 12: Funcionamiento de Simulated Annealing. De izquierda a derecha, y de arriba a abajo: inicialización, 5 iteraciones, 20 iteraciones, solución (en este caso es óptima).

3.3.2. ALGORITMO GENÉTICO

La idea de un algoritmo genético (AG) fue inicialmente propuesto por Holland en 1975 [15] y está basado en la evolución y la selección natural. En él, un conjunto de individuos, al que a partir de ahora llamaremos población, se relacionan y mutan para dar lugar a una nueva población que en teoría es una versión mejorada de la anterior población.

En este problema, cada individuo presenta una potencial solución al problema y se evalúa mediante una función de *fitness* (en este caso será el área de la región de visibilidad con alcance limitado). Existen tres operadores fundamentales en este algoritmo: selección, cruce y mutación.

La selección consiste en la elección de individuos de la población actual que se van a utilizar para generar una nueva población. Para ello, se eligen individuos de una distribución aleatoria generada a partir del *fitness* de cada individuo, de manera que los individuos con un mayor *fitness* tendrán mayor probabilidad de ser elegidos. Existen diferentes técnicas de muestreo para elegir de manera aleatoria los individuos. En nuestro caso, se utilizará el más sencillo de todos, que se trata del método de la ruleta.

La probabilidad de cada individuo de ser elegida viene dada por:

$$p_{it} = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

, donde i representa el individuo de la población, t la iteración y f_i representa el *fitness* del individuo i. Una vez conocemos todas las probabilidades, se determina S que es la suma acumulada de las probabilidades. Esto quiere decir que para cada $p'_i \in S$, $p'_i = \sum_{j=0}^{i} p_j$. Después basta con determinar n números aleatorios $m \in [0, p'_n)$, para cada cual debemos encontrar el p'_i tal que $p'_i \leq m < p'_{i+1}$. Entonces elegiremos los distintos individuos según los índices i obtenidos en el muestreo.

En el operador de cruce, se eligen dos padres de la población mediante el método de la ruleta y se intercambian información para dar lugar a un nuevo hijo. La utilidad de este operador reside en la idea de que un hijo que viene de dos padres que tengan una buena adaptación debería poseer también esa ventaja de adaptación. Este operador trabaja también con una probabilidad p_c, que determina si hay que cruzar a los dos padres o no. En caso de que no haya que cruzarlos, el hijo es idéntico a uno de los dos padres. Dados dos padres q_i, q_j obtenidos mediante el operador de selección. Recordemos que los padres son puntos en el plano con coordenadas (x, y). Determinamos el nuevo hijo q de la siguiente manera:

1. Escogemos un número aleatorio $\alpha \in [0,1)$
2. $q = q_i * \alpha + q_j * (1 - \alpha)$

Una vez tenemos el hijo, faltaría aplicarle el operador de mutación. Este operador introduce variaciones pequeñas en algunos de los individuos. Para ello, necesitamos una probabilidad de mutación p_m, que es global e igual para cada individuo. La variación introducida se realiza aplicándole una pequeña perturbación obtenida de

una distribución normal de media 0 y desviación típica predeterminada (en este caso, será de 1 píxel).

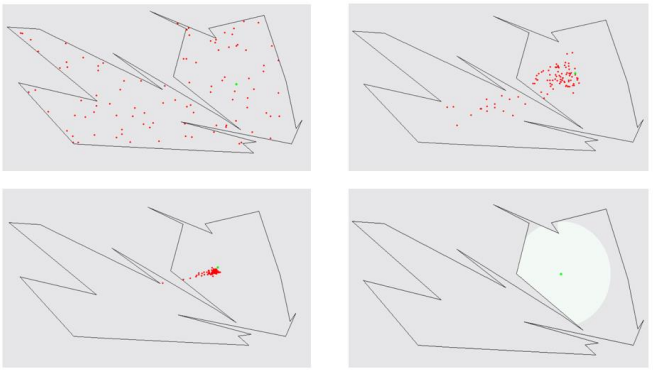


Figura 13: Funcionamiento del algoritmo genético tras 0, 5 y 20 iteraciones, y tras alcanzar el máximo de soluciones, ordenadas respectivamente de izquierda a derecha, y de arriba a abajo. *Nota:* El alcance usado es distinto al de la Figura 12.

3.3.3. BÚSQUEDA TABÚ

El algoritmo de Búsqueda Tabú (Tabu Search) fue introducido en 1986 por Fred W. Glover [11] y utiliza la búsqueda local para encontrar una solución al problema. Este algoritmo explora un conjunto de vecinos de una potencial solución y los evalúa, utilizando a su vez una memoria de las soluciones previamente consideradas. De esta manera, es un algoritmo que realiza una intensa búsqueda del óptimo global.

El concepto de memoria en este algoritmo se refiere a un conjunto de posiciones que están prohibidas durante una serie de pasos máximo k. De este modo, se evita que el algoritmo quede completamente estancado en un mínimo global y le impulsa a buscar el óptimo global. Aquellas posiciones prohibidas en el paso i vendrán dadas por todas las previas potenciales soluciones de los pasos $i-1$ hasta $i-k$. Lógicamente, no siempre se tendrán como mínimo n pasos dados, por lo que si $i < k$, los movimientos tabúes serán todas las posiciones anteriores.

El algoritmo parte de una potencial solución inicial. A partir de esta solución, se generan n vecinos. Para este problema, se ha escogido $n = 8$, ya que de este modo cubrimos las principales direcciones cardinales (norte, noreste, este, sureste, sur, suroeste, oeste, y noroeste). Sin embargo, para evitar explorar únicamente esas direcciones, ya que acabaríamos considerando el espacio como discreto, se ha decidido añadir a cada vector un pequeño vector perturbación generado a partir de una distribución normal $\mathcal{N}(\mu = 0, \sigma = 0.2)$. Se podría aplicar también una perturbación únicamente a la magnitud del vector y de esta manera aseguraríamos la consideración de muchos más candidatos. La Figura 14 muestra las posibles generaciones de vecinos.

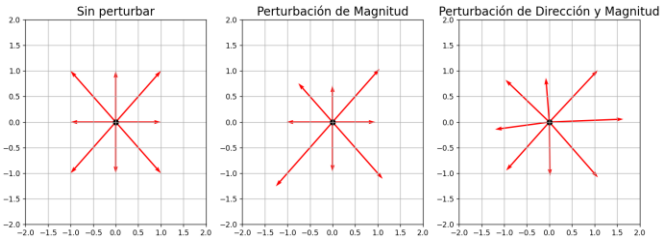


Figura 14: De izquierda a derecha: imagen que muestra la generación de vecinos en un espacio discreto; imagen que muestra la generación de vecinos con una perturbación en la magnitud del vector; imagen que muestra la generación de vecinos añadiendo al vector de la imagen de la izquierda un pequeño vector perturbación.

Una vez tenemos calculados los potenciales vecinos se comprueba si alguno de ellos está lo suficientemente cerca de un movimiento tabú. Si lo está, descartamos ese vecino. Cuando tenemos los vecinos filtrados, simplemente comprobamos cuál de ellos tiene mejor *fitness* y tomamos ese vecino como la potencial solución del paso $i+1$, y se añade a la lista de movimientos tabú la potencial solución del paso i.

Este bucle se repite hasta llegar a un número máximo de iteraciones o bien una potencial solución no tenga posibles vecinos, porque todos ellos son movimientos tabúes.

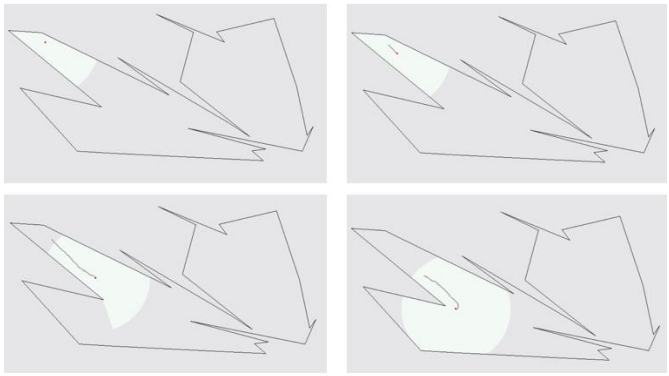


Figura 15: Funcionamiento del algoritmo de Búsqueda Tabú. De izquierda a derecha, y de arriba a abajo: inicialización, 20 iteraciones, 60 iteraciones, solución (número máximo de iteraciones alcanzado). *Nota:* el alcance es diferente al usado en la Figura 12 y en la Figura 13.

3.3.4. RANDOM SEARCH

Si bien es cierto que el Algoritmo Genético no depende tanto de la inicialización, pues considera a la vez muchas hipótesis distribuidas uniformemente por todo el polígono, los algoritmos de Simulated Annealing y Búsqueda Tabú siguen siendo muy sensibles a la inicialización. Esto se debe principalmente a que, si toman saltos de exploración demasiado grandes, estaría perdiendo mucha información entre medias, al estar tratando con un espacio continuo y no discreto. Además, para abandonar un mínimo local tiene que tomar muchos pasos que son soluciones peores que las que se tenían previamente, siendo muy difícil en el caso de ambos algoritmos por su estructura. Por esto, la inicialización de estos dos algoritmos no puede ser puramente aleatoria, si no que tienen que estar condicionados.

Existen muchas maneras de condicionar una solución inicial para este tipo de algoritmos, pero se ha optado por utilizar una muy sencilla de implementar, pero a la vez muy efectiva: Random Search. Este algoritmo se puede usar directamente para resolver el problema en cuestión, pero se usará como un paso previo a la inicialización de los algoritmos Simulated Annealing y Búsqueda Tabú.

El funcionamiento de Random Search es bastante intuitivo. Para cada paso $0 \leq i \leq n$, siendo n un parámetro, genero un punto aleatorio dentro del polígono. Si el punto generado en el paso i tiene mejor *fitness* que la mejor solución encontrada hasta el paso $i-1$, entonces mi nueva mejor solución es la que he generado en el paso i. Este proceso se repite hasta que $i = n$.

Se deduce que, si se utiliza únicamente este algoritmo para resolver el problema, necesitaríamos de un n lo suficientemente grande para encontrar una solución cercana a la óptima. Sin embargo, si usamos un n considerablemente menor para encontrar una solución razonablemente buena, este punto puede ser la solución de partida de algoritmos más precisos como pueden ser Simulated Annealing o Búsqueda Tabú. De este modo, aunque se aumente el tiempo de computación para solucionar el problema con los dos algoritmos mencionados, se estaría partiendo de una posición mucho más ventajista, que permitiría encontrar el óptimo global, en vez de estancarse en un mínimo local.

4. EXPERIMENTOS

Para comparar el desarrollo de los algoritmos a la hora de resolver el problema, se han llevado a cabo una serie de experimentos en los que se han tomado medidas de una serie de métricas que se usarán para evaluar el funcionamiento de los algoritmos.

Se ha diseñado una base de datos de 100 polígonos de 25 vértices mediante el Generador Aleatorio de Polígonos descrito previamente en la sección 3.1. Los puntos distribuidos uniformemente se han generado en el rango [20,780) para el eje x, y [170, 550) para el eje y. Se han elegido a su vez dos alcances diferentes para comprobar el funcionamiento de estos algoritmos, que son 80 y 150.

Una vez definidos los parámetros de entrada del problema (polígono P y alcance d), se ha procedido a seleccionar los parámetros de cada algoritmo. Si bien esto se ha realizado por pura observación y experimentación, se puede hacer uso de técnicas

estadísticas más complejas para determinar un valor óptimo de parámetros para cada algoritmo para la resolución de este problema. Sin embargo, esto puede introducir un sesgo y que sobreajuste a los datos de entrada (la base de datos de polígonos). Además, como esto requeriría de más recursos computacionales, se ha optado por descartar esta opción. Se dejará como un posible trabajo futuro.

Los parámetros utilizados para cada algoritmo son los siguientes:

1. Algoritmo Genético
    a. Tamaño de la población: 100. Esto se mantiene constante para todas las iteraciones.
    b. Probabilidad de cruce: 0,8.
    c. Probabilidad de mutación 0,05.
    d. Número de iteraciones máximo: 100.
    e. Número de iteraciones permitidas sin mejora: 20.
2. Búsqueda Tabú
    a. Tamaño de la lista tabú: 15
    b. Número de iteraciones máximo: 200
3. Simulated Annealing:
    a. Temperatura inicial: 1000
    b. Alpha: 0,99
    c. Temperatura mínima: 10^{-3}
    d. Número de iteraciones máximo: 1000
4. Random Search (este algoritmo se aplica solo como paso previo de inicialización a Simulated Annealing y Búsqueda Tabú):
    a. Número de iteraciones máximo: 100

Con estos parámetros se ha procedido a correr los siguientes 5 algoritmos en la base de datos de polígonos: Algoritmo Genético, Búsqueda Tabú, Búsqueda Tabú aplicando previamente Random Search, Simulated Annealing y Simulated Annealing aplicando previamente Random Search.

Para las comparaciones, se evaluará la media del porcentaje de área del polígono cubierta, el área total media cubierta y el tiempo medio en determinar la solución.

4.1. COMPARATIVA ALGORITMOS SIN RANDOM SEARCH

En este apartado se explorará el rendimiento de los algoritmos Algoritmo Genético, Simulated Annealing y Búsqueda Tabú, estos dos últimos sin aplicar Random Search previamente.

En cuanto al tiempo de ejecución, la diferencia es muy notable entre los tres algoritmos. Se puede ver en la Figura 16 que el algoritmo genético es con mucha diferencia el que más tarda en ejecutarse, incluso siendo el que menos iteraciones tiene. Sin embargo, si pensamos que para cada iteración tiene que evaluar 100 individuos no es de extrañar el resultado de esta gráfica.

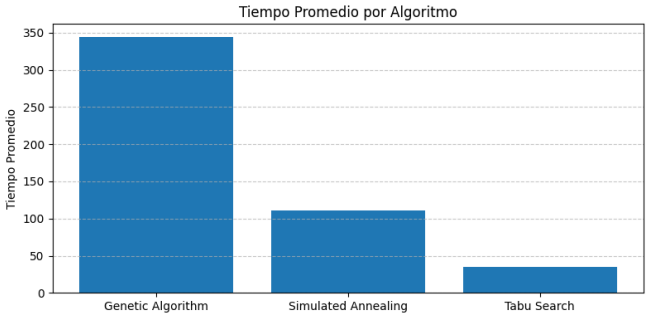


Figura 16: Comparación de tiempos medios de ejecución de las técnicas metaheurísticas Algoritmo Genético, Simulated Annealing y Búsqueda Tabú.

La diferencia entre Simulated Annealing y Búsqueda Tabú reside en el número máximo de iteraciones que se le ha permitido a cada algoritmo. Las 1000 iteraciones de Simulated Annealing vienen de que, en determinados algoritmos, la convergencia es muy lenta porque la perturbación no deja de ser aleatoria en cualquier dirección. Era necesario asegurarse que un algoritmo que solo evalúa una perturbación fuera capaz de competir con otros algoritmos que evalúan varios candidatos e hipótesis.

En la Figura 17, se presentan los resultados del porcentaje de área iluminada con respecto al área total del polígono. Esta métrica depende en gran medida del polígono de origen y del alcance dado,

pues en algunos es posible encontrar como solución toda el área del alcance (área de la circunferencia) y en otros no.

Aún así, se puede ver que hay cierta diferencia entre los algoritmos. El algoritmo genético lidera esta gráfica, lo que indica que, de media, el algoritmo genético ha encontrado las mejores soluciones al problema. Se da un salto más grande entre Búsqueda Tabú y Simulated Annealing. Esto se debe, una vez más, al número de iteraciones permitidas a Simulated Annealing.

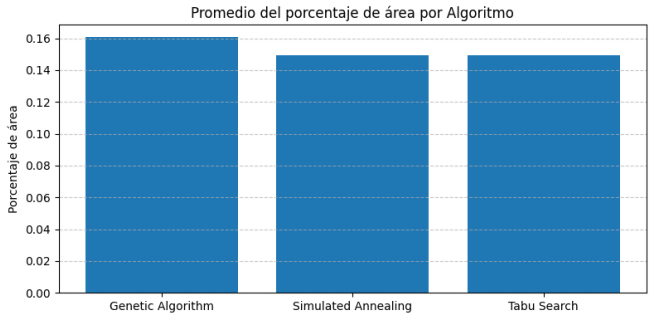


Figura 17: Comparación del porcentaje de área medio encontrada por las técnicas metaheurísticas, sin Random Search

Por último, en la Figura 18 se presentan los resultados acerca de la media del área total cubierta por los algoritmos. Los resultados son muy similares a los que se ven en la Figura 17.

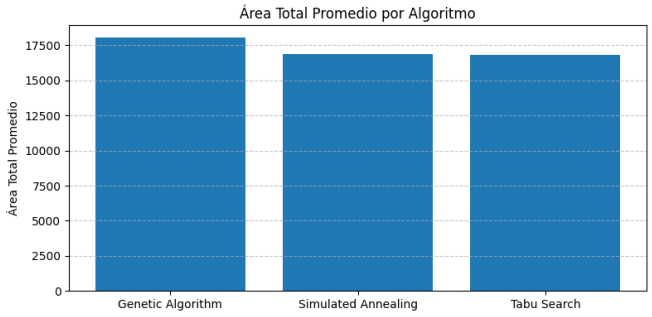


Figura 18: Comparación del área total iluminada media de las distinas técnicas metaheurísticas, sin Random Search.

Sin aplicar Random Search, se observa que, por una parte, el algoritmo que mejor ha rendido ha sido el Algoritmo Genético. Sin embargo, el tiempo que ha tardado en ejecutarse es significativamente mayor al de los otros algoritmos. Es difícil explicar qué algoritmo es mejor, pues depende de lo que se esté buscando. Si se busca maximizar el resultado, es conveniente utilizar el Algoritmo Genético. Si se busca minimizar el tiempo de ejecución, entonces sería mejor optar por la Búsqueda Tabú. Simulated Annealing, empleando más tiempo que Búsqueda Tabú, ha obtenido resultados muy similares, lo cual lo convierte en el peor de los tres algoritmos.

4.2. COMPARATIVA ALGORITMOS CON Y SIN RANDOM SEARCH

En este apartado se evaluará la influencia de inicializar con Random Search las técnicas de Simulated Annealing y Búsqueda Tabú.

En la Figura 19, se aprecian los tiempos de ejecución. Resulta interesante como, sin modificar nada los parámetros de las técnicas metaheurísticas, aplicar Random Search en Simulated Annealing disminuye el tiempo de ejecución y en Búsqueda Tabú aumenta.

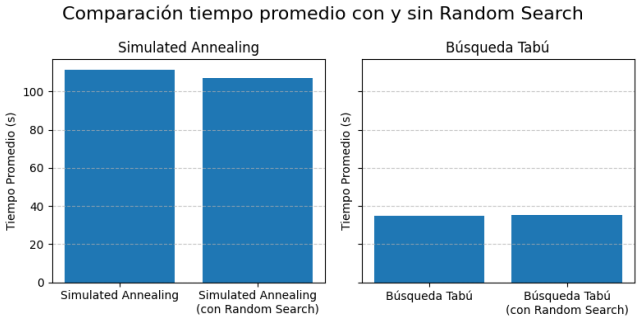


Figura 19: Comparación del tiempo de ejecución de Simulated Annealing y Búsqueda Tabú con y sin aplicar previamente Random Search. La columna de la izquierda de cada gráfica representa el algoritmo sin aplicar Random Search; la de la derecha, aplicándolo. Están escalados a sus respectivos tiempos, para poder ver la diferencia.

La explicación se debe a la propia naturaleza de cada algoritmo. Si bien es cierto que, para un mismo polígono, Random Search debería proporcionar a cada algoritmo soluciones iniciales similares, Simulated Annealing explora en todas las direcciones posibles. Al estar ya cerca de la solución, no es necesario hacer primero un desplazamiento largo de la solución, con explorar

alrededor es suficiente. Por esto, Búsqueda Tabú sufre, incluso con la generación de vecinos explicada en el apartado 3.3.3, ya que únicamente puede explorar en 8 direcciones diferentes, mientras que Simulated Annealing lo puede hacer en cualquier dirección. Además, con las probabilidades iniciales altas de exploración, acaba encontrando antes la solución que sin aplicar Random Search.

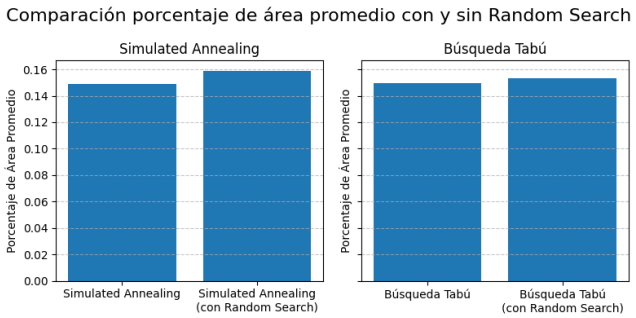


Figura 20: Comparación del porcentaje de área cubierta por Simulated Annealing y Búsqueda Tabú, con y sin Random Search. La figura de la izquierda presenta los resultados de Simulated Annealing; a la derecha, los de Tabú Search. En ambas figuras, la columna de la izquierda presenta el algoritmo sin aplicar Random Search, y la de la derecha aplicándolo.

En la Figura 20 se puede apreciar la notable mejoría, sobre todo en Simulated Annealing, de aplicar Random Search previamente a ambos algoritmos. Por lo tanto, se confirman las hipótesis de que aplicar Random Search inicialmente mejora el rendimiento de los algoritmos.

Finalmente, la Figura 21 nos permite observar que, efectivamente, el rendimiento de los algoritmos mejora aplicando Random Search antes.

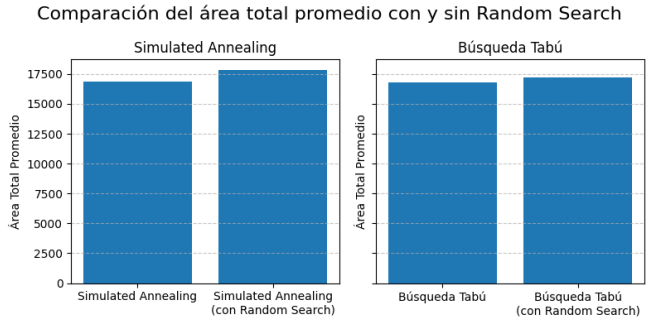


Figura 21: Comparación de la media de área total cubierta por Simulated Annealing y Búsqueda Tabú, son y sin Random Search. La columna de la izquierda de cada gráfica representa el algoritmo sin aplicar Random Search; la de la derecha, aplicándolo.

4.3. COMPARACIÓN DE TODOS LOS ALGORITMOS

En este apartado se evaluará cómo ha mejorado Random Search los algoritmos, teniendo como referencia acercarse lo máximo posible al Algoritmo Genético. Para ello, no se mostrará el tiempo que ha tardado, si no únicamente el porcentaje de área y el área total (Figura 22).

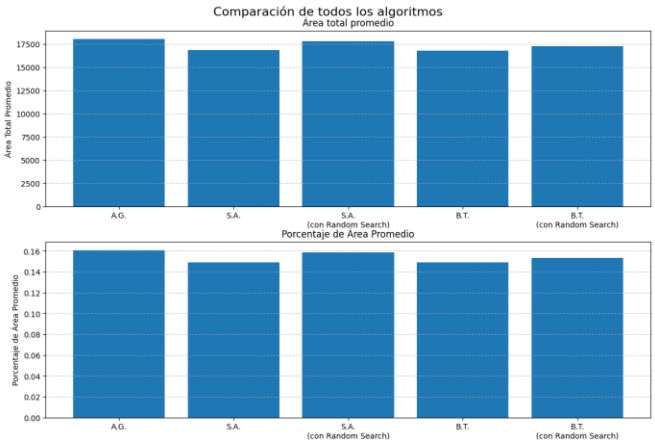


Figura 22: Comparación del área y porcentaje del área promedios iluminados. Las columnas representan, de izquieda a derecha: Algoritmo Genético, Simulated Annealing, Simulated Annealing con Random Search, Búsqueda Tabú y Búsqueda Tabú con Random Search.

En esta figura se puede apreciar la notable mejoría que Random Search introduce a los algoritmos, pues consigue dos cosas:

- Con mucho menos tiempo de computación, alcanzar resultados muy similares a los que alcanza el algoritmo genético.
- No hay que olvidar que los parámetros están seleccionados para su funcionamiento con una inicialización aleatoria, por lo que puede ser que optimizarlos para su inicialización con Random Search mejore incluso al Algoritmo Genético.

Por lo tanto, se puede concluir que incluir Random Search para *sesgar* los algoritmos de Simulated Annealing y Búsqueda Tabú ha sido un gran acierto, pues sin cambiar demasiado el tiempo de

computación, ha conseguido acercarse mucho a los resultados del Algoritmo Genético.

4.4. INFLUENCIA DEL ALCANCE EN EL TIEMPO DE EJECUCIÓN

Por la naturaleza del problema, es fácil determinar una condición de parada instantánea: si el *fitness* es igual al área del círculo de radio el alcance d, tenemos una solución óptima. Del mismo modo, si el área es igual a toda el área del polígono, tenemos también una solución óptima. Por lo tanto, se puede pensar que cuanto mayor sea el alcance, más tiempo tardará en encontrarse la solución óptima, pues donde antes se podía encontrar una circunferencia de radio d, si aumentamos el alcance, puede que esa circunferencia de radio d^+ tenga intersección con el polígono. Veamos experimentalmente cómo influye (Figura 23).

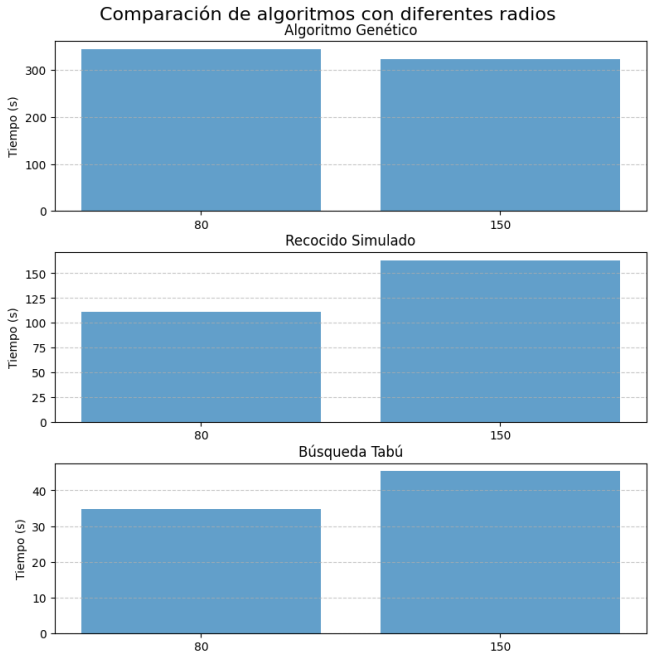


Figura 23: Comparación del tiempo de ejecución en función del parámetro radio.

Salvo en el Algoritmo Genético, se ve como, efectivamente, el tiempo de ejecución aumenta cuando aumenta el radio. Esto se debe a que es más difícil encontrar un círculo que pare por completo la ejecución del algoritmo.

El Algoritmo Genético no tiene la condición de parada anterior, pues se ha querido dejar que la población converja mediante mutaciones y cruces a la solución óptima. El tiempo de ejecución es menor porque, en ciertos polígonos, cuanto mayor es el alcance, menos puntos de intersección existen entre la circunferencia de radio aumentado y el polígono. Por lo tanto, menos cálculos hay que realizar, reduciendo el tiempo total que tarda en ejecutar.

5. APLICACIÓN INTERACTIVA

En este breve apartado, se explicará cuáles han sido los requisitos HW y SW para el desarrollo de la aplicación interactiva que se ha implementado en este trabajo y que permite visualizar todo lo explicado anteriormente.

Como equipo de Hardware, se ha hecho uso del portátil OMEN Laptop 15-ek0xxx, con un sistema tipo PC basado en x64. Se ha usado el procesador Intel® Core™ i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 …. Tiene una memoria física instalada de 16,0GB (RAM) y la tarjeta gráfica NVIDIA GeForce RTX 2060.

Para el desarrollo de la aplicación se ha hecho uso del lenguaje de programación python, con la ayuda de distintas librerías: para la interfaz gráfica se ha usado la librería pygame; para los cálculos se ha empleado la librería por excelencia de álgebra lineal NumPy; para las gráficas se ha utilizado Matplotlib; para algunas operaciones como la determinación del cierre convexo, se ha hecho uso de scipy. Las versiones de Python y de cada librería se presentan a continuación:

- Python 3.12.2
- Pygame 2.6.1
- Numpy 2.2.2
- Matplotlib 3.10.1
- Scipy 1.15.3

Se ha intentado que la aplicación sea bastante completa, por lo que tratará de dar una explicación lo más breve posible acerca de su estructura y funcionamiento.

La aplicación se divide en tres grandes módulos, que son `src`, `data` y `results`. Las dos últimas carpetas contienen las bases de datos de polígonos y los resultados de los experimentos realizados con los algoritmos, respectivamente. La carpeta `src` contiene todos los scripts necesarios para la ejecución del programa.

Se divide en dos grandes submódulos, que son `algorithms` y `scenes`. La primera carpeta contiene todos los scripts relacionados con la ejecución de los algoritmos; mientras que la segunda carpeta contiene todas las clases que permiten la visualización gráfica de la aplicación, así como ejecutar y correr los algoritmos para mostrar sus resultados en pantalla. En Documentación Aplicación se puede ver con más detalle los distintos scripts que conforman la aplicación.

La incorporación de una aplicación interactiva da al usuario una explicación gráfica y dinámica sobre el funcionamiento de los algoritmos, y cómo atacan el problema. Además, se ofrecen distintas opciones para determinar el polígono que va a servir como base del problema: se puede dibujar, cargar de una base de datos y generar aleatoriamente, preseleccionando el número de vértices deseado (Figura 24).

Para los algoritmos Simulated Annealing y Búsqueda Tabú, se ofrece también la posibilidad de aplicar previamente Random Search. Así, el usuario puede ver que funciona mucho mejor que sin hacer uso de ello previamente.

Se incluye, además, una opción que no es para resolver el algoritmo, si no para experimentar y entender qué es el concepto de visibilidad con alcance limitado (Figura 25).

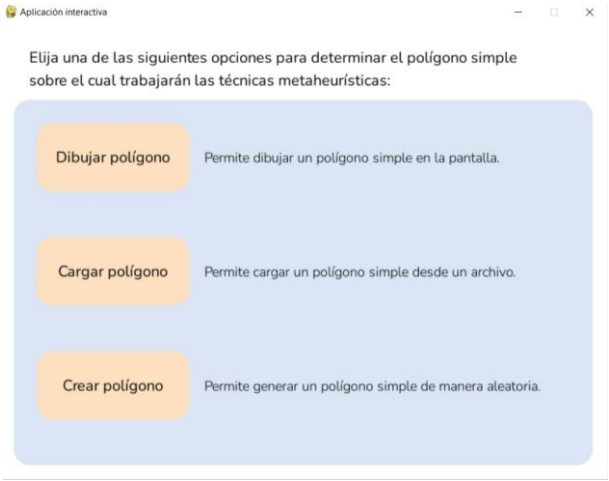


Figura 24: Menú de la aplicación interactiva.

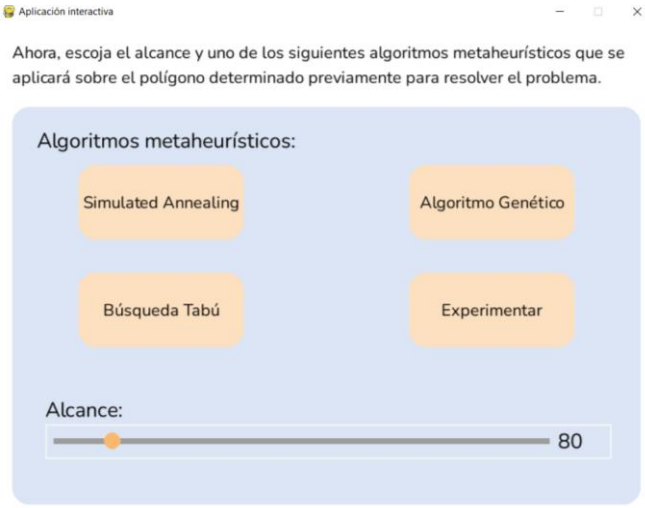


Figura 25: Elección del algoritmo o del método para experimentar. También se puede hacer uso de un *slider* a escala para determinar el alcance deseado para el problema en cuestión.

Tras elegir el algoritmo con el que se quiere resolver el problema (o la escena de experimentación), se abre una nueva escena que permite al usuario ver en directo la ejecución del algoritmo. En esta escena, se permite al usuario ejecutar una iteración del algoritmo, 10 iteraciones o visualizar directamente cómo se resuelve el algoritmo, viendo cómo cambia el área abarcada por la solución que exista en ese momento (Figura 26).

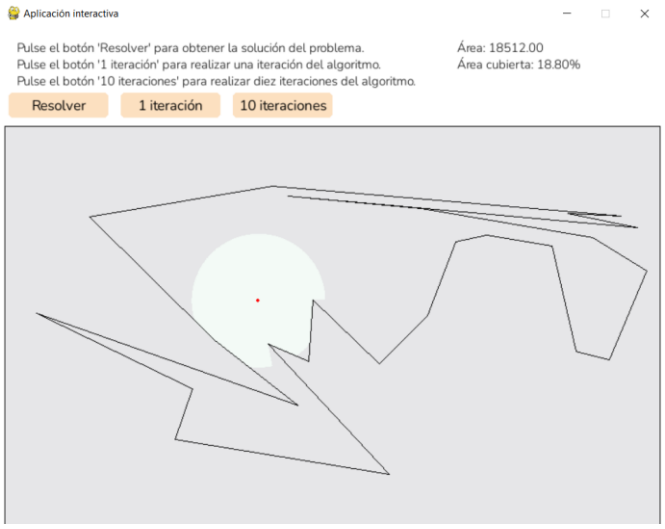


Figura 26: Ejemplo del panel de resolución del problema para Simulated Annealing.

6. CONCLUSIONES Y TRABAJOS FUTUROS

Si bien es cierto que en este proyecto se ha cubierto cómo solucionar el problema de la visibilidad con alcance limitado, es un problema más complejo y que presenta más variaciones que trabajar que el concepto de visibilidad con alcance ilimitado. En la visibilidad con alcance ilimitado es probable que muchos polígonos presenten una única solución. Incluso aunque presenten infinitas (polígonos que tengan núcleo), no depende de factores externos como el alcance. Añadir el alcance cambia el problema por completo, llegando a ser el caso que puntos que eran solución en visibilidad con alcance ilimitado no sean solución para alcance limitado (Figura 27).

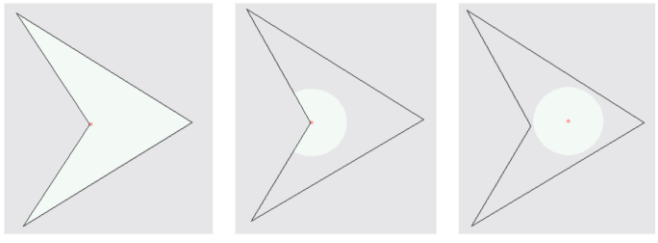


Figura 27: De izquierda a derecha. Una de las soluciones de visibilidad con alcance ilimitado; mismo punto, pero con alcance limitado; una de las soluciones para el alcance limitado.

Por otra parte, aunque el problema siga siendo el mismo, cambiar el alcance requiere del cómputo otra vez del problema desde cero, lo que resulta muy ineficiente a efectos prácticos. Sin embargo, resulta trivial que cambiar el alcance provoca que algunas regiones que previamente se descartaban para alojar una potencial solución puedan ser consideradas como nuevas regiones de interés para tener la solución. Esto es normal, ya que cambiar los parámetros de entrada de un problema conlleva a esto.

Con lo que respecta a los experimentos y los resultados obtenidos, se ha visto que Random Search como paso previo a algoritmos más potentes como Búsqueda Tabú y Simulated Annealing mejora notablemente cómo se desenvuelve el algoritmo en el problema. Como futuro trabajo, se puede realizar un estudio exhaustivo de parámetros iniciales, para maximizar el resultado. Esto, sin embargo, es una tarea muy delicada y que con bases de datos tan pequeñas como la que se ha tratado en esta memoria puede inducir un sesgo y que su rendimiento en otras bases de datos sea peor. Este es un problema muy común en lo que respecta a los modelos de Machine Learning, que sobre ajustan los parámetros a los datos, obteniendo resultados muy buenos en esos datos, pero extrapolando muy mal a datos con los que no se ha entrenado.

También, es probable que el tiempo de ejecución no aumente indefinidamente según aumente el radio. Explorar el comportamiento de los algoritmos en función de varios valores del radio es un trabajo futuro que puede ser un tema interesante, que se aleja del propósito de esta memoria.

Realizar estudios teóricos genéricos sobre la visibilidad con alcance limitado es una tarea compleja, pues los distintos tipos de polígonos que existen limitan mucho esta tarea. Hasta el momento, solo existen estudios sobre algunos tipos de algoritmos, como se ha explicado en la sección 2.

El desarrollo de la aplicación da pie a entender de manera más profunda el comportamiento de los algoritmos para distintos parámetros. Se puede

realizar un estudio del funcionamiento de los algoritmos en función de los distintos parámetros para comprobar su sensibilidad a estos, así como su rendimiento. Cada algoritmo tiene muchos parámetros que pueden ser modificados. En el caso del Algoritmo Genético, se pueden cambiar el tamaño de la población o la probabilidad de mutación, la probabilidad de cruce. Para Simulated Annealing se puede modificar cómo se reduce la temperatura, así como la temperatura inicial y la temperatura mínima. En la Búsqueda Tabú, también se pueden tocar otros parámetros y jugar más con la generación de vecinos, intercambiando tiempo computacional por precisión de búsqueda. Y en todos estos algoritmos se puede ver cómo se desenvuelven dejándoles más o menos tiempo corriendo, cambiando el número de iteraciones o siendo más estrictos con las nuevas soluciones propuestas.

Si bien el estudio realizado puede dar pie a extraer ciertas conclusiones acerca del comportamiento de los algoritmos, este estudio sigue estando hecho en 100 polígonos de 25 vértices. Para extraer mejores conclusiones se puede cambiar este número y emplear bases de datos con polígonos de más vértices. Sin embargo, esto requiere de una carga computacional más alta y con los recursos disponibles no ha sido posible realizarlo. Si se hace una comparación con otros polígonos de distinto número de vértices, se podría reforzar las conclusiones extraídas en los experimentos.

En cuando a la visibilidad por alcance limitado, todavía quedan muchas preguntas sin responder y que, a día de hoy, siguen siendo tema de debate, como, por ejemplo: dado un punto de inicio s y un punto final t, determinar la ruta tal que la visibilidad acumulada (área visible a lo largo del camino con alcance d) sea máxima. En esta memoria se ha presentado el tema y se ha tratado un único problema de los muchos que hay. Para futuros trabajos se pueden trabajar a raíz de los experimentos obtenidos en esta memoria algunos problemas como: ¿cuál es el número de luces mínimo que necesito para iluminar un polígono, si cada luz tiene un alcance fijo d?

Se pueden realizar estudios teóricos, y quizás obtener resultados, aunque no hay que olvidar que muchos de las preguntas planteadas en el tema de la visibilidad con alcance limitado son problemas de categoría **NP-hard**, por lo que las comprobaciones de las soluciones o de estos teoremas necesitan de nuevos algoritmos exactos que, por el momento, están más lejos que cerca de llegar.

7. BIBLIOGRAFÍA

[1] Chvátal, V. (1975). *A combinatorial theorem in plane geometry*. *Journal of Combinatorial Theory, Series B, 18*(1), 39–41. https://doi.org/10.1016/0095-8956(75)90061-1

[2] Ntafos, S. (1992). *Watchman routes under limited visibility*. *Computational Geometry: Theory and Applications, 1*(3), 149–170. https://doi.org/10.1016/0925-7721(92)90014-J

[3] Auer†, T., & Held†, M. (1996). Heuristics for the generation of random polygons. *Canadian Conference on Computational Geometry*, 38–43. https://doi.org/10.1515/9780773591134-009

[4] O'Rourke, J. (1994). "*Computational Geometry in C"*. Cambridge University Press

[5] Mikula, J., Kulich, M., Preučil, L., et al. (2024). *TřiVis: Versatile, reliable, and high-performance tool for computing visibility in polygonal environments*. arXiv.org. https://doi.org/10.48550/arXiv.2410.08752

[6] Bungiu, F., Hemmer, M., Hershberger, J., Huang, K., & Kröller, A. (2014). *Efficient computation of Visibility Polygons*. arXiv.org. https://arxiv.org/abs/1403.3905

[7] Canales Cano, S. (2004). *Métodos heurísticos en problemas geométricos: Visibilidad, iluminación y vigilancia* [Tesis doctoral, Universidad Politécnica de Madrid]. Facultad de Informática, Departamento de Matemática Aplicada.

[8] García, J. (1995). *Problemas algorítmicos-combinatorios de visibilidad* [Tesis doctoral, Universidad Politécnica de Madrid].

[9] Kim, S.-H., Park, J.-H., Choi, S.-H., Shin, S. Y., & Chwa, K.-Y. (1995). An optimal algorithm for finding the edge visibility

polygon under limited visibility. *Information Processing Letters, 53*(6), 359–365.

[10] Joe, B., & Simpson, R. B. (1987). *Corrections to Lee's visibility polygon algorithm*. *BIT Numerical Mathematics, 27*(4), 458–473. https://doi.org/10.1007/BF01937271

[11] Glover, F.; Laguna M. (1997): "*Tabu Search"*., Kluwer Academic Publishers.

[12] Pérez Bienzobas, D. (2023). *Maximización con Técnicas Metaheurísticas de la Región de Visibilidad de un Punto en un Polígono*. [Trabajo de fin de grado]. Universidad Pontificia de Comillas.

[13] Lee, D. T., & Lin, A. K. (1990). Computational complexity of art gallery problems. *Autonomous Robot Vehicles*, 303–309. https://doi.org/10.1007/978-1-4613-8997-2_23

[14] Kirkpatrick, S., Gelatt, C. D., Jr., & Vecchi, M. P. (1983). *Optimization by simulated annealing*. *Science, 220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

[15] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.

ANEXO

En este anexo se presenta la información necesaria acerca de la estructura de la aplicación, pseudocódigos e información sobre los parámetros usados. Se ha cambiado el número de columnas para poder apreciar mejor las imágenes.

DOCUMENTACIÓN APLICACIÓN

El repositorio se puede encontrar en el siguiente enlace: https://github.com/mariokroll/visibility-game.git. Esta aplicación se divide en las siguientes subcarpetas: `src`, `data`, `results` y `fonts`. El árbol que describe la aplicación se puede ver en el siguiente esquema:

```
visibility-game
├── README.md
├── colors.txt
├── data
├── fonts
├── results
└── src
    ├── __init__.py
    ├── main.py
    ├── polygon.py
    ├── testing_file.py
    ├── ui.py
    ├── utils.py
    ├── algorithms
    │   ├── fitness_algorithm.py
    │   ├── genetic_algorithm.py
    │   ├── random_search.py
    │   ├── simulated_annealing.py
    │   ├── tabu_search.py
    │   └── visibility_algorithm.py
    └── scenes
        ├── __init__.py
        ├── apply_random_search_scene.py
        ├── choose_algorithm.py
        ├── create_polygon_scene.py
        ├── draw_polygon_scene.py
        ├── genetic_algorithm_scene.py
        ├── load_polygon_scene.py
        ├── menu_scene.py
        ├── scene_manager.py
        ├── simulated_annealing_scene.py
        ├── solver_scene.py
        ├── start_scene.py
        ├── tabu_search_scene.py
        └── toying_with_concept_scene.py
```

Los archivos de las carpetas `data`, `results` y `fonts` no se han incluido en el árbol.

Los archivos de la carpeta `algorithms` tienen las siguientes funcionalidades:

- `fitness_algorithm.py`: aquí se encuentra el código necesario para calcular el área de la región de visibilidad con alcance limitado. Se encuentran implementados los dos algoritmos.
- `genetic_algorithm.py`: clase que permite correr el Algoritmo Genético.
- `random_search.py`: clase que permite correr el Random Search.
- `simulated_annealing.py`: clase que permite correr el algoritmo de Simulated Annealing.
- `tabu_search.py`: clase que permite correr el algoritmo de Búsqueda Tabú.
- `visibility_algorithm.py`: archivo que aloja el código necesario para correr el algoritmo de *ray casting* para calcular el polígono de visibilidad.

Los archivos de la carpeta `scenes` contienen el código necesario para correr cada escena de la aplicación. A continuación, se presenta una breve descripción de cada archivo, junto con una imagen de su funcionamiento:

- `apply_random_search_scene.py`: Pregunta al Usuario si quiere aplicar el algoritmo de Random Search o no.

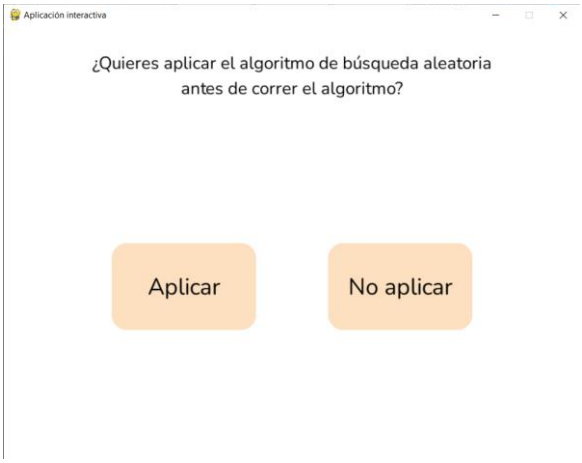


Figura 28: Escena para aplicar o no Random Search.

- `choose_algorithm.py`: Despliega la interfaz necesaria para preguntar al usuario por el algoritmo y el alcance deseado.

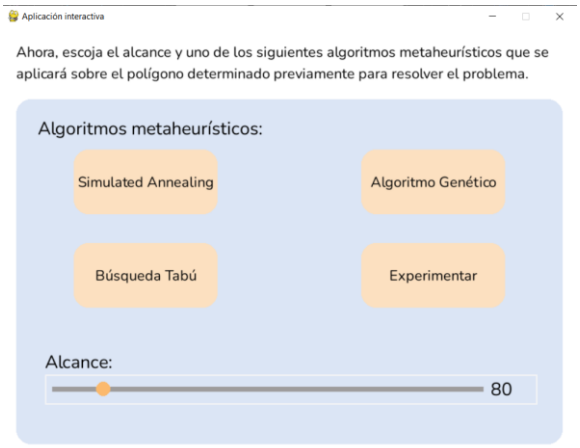


Figura 29: Visualización de la escena para elegir el algoritmo y el alcance.

- `create_polygon_scene.py`: Esta escena permite que el usuario escoja un número de vértices y hace uso del algoritmo Generador Aleatorio de Polígonos para crear un polígono aleatorio a partir de n puntos generados a partir de una distribución uniforme en el plano.

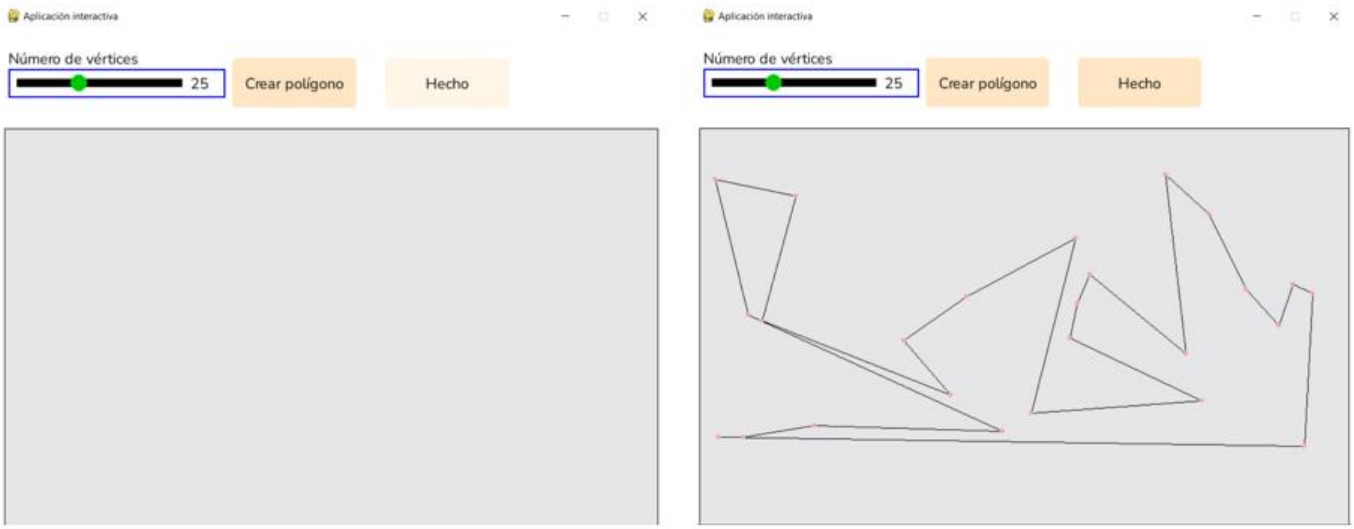


Figura 30: Escena para la generación aleatoria de polígono.

- `draw_polygon_scene.py`: En esta escena se le permite al usuario dibujar un polígono.

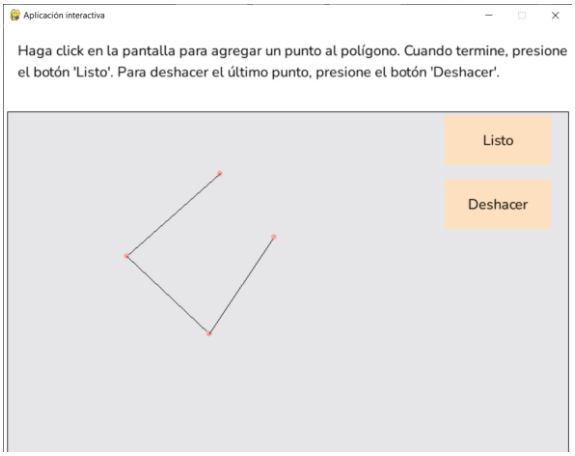


Figura 31: Interfaz para dibujar un polígono.

- `genetic_algorithm_scene.py`: Escena que permite mostrar cómo el Algoritmo Genético resuelve el problema y las iteraciones que realiza para resolverlo (ver Figura 13).
- `load_polygon_scene.py`: Escena que permite cargar un polígono de una base de datos de polígonos de 25 vértices o de 50 vértices.

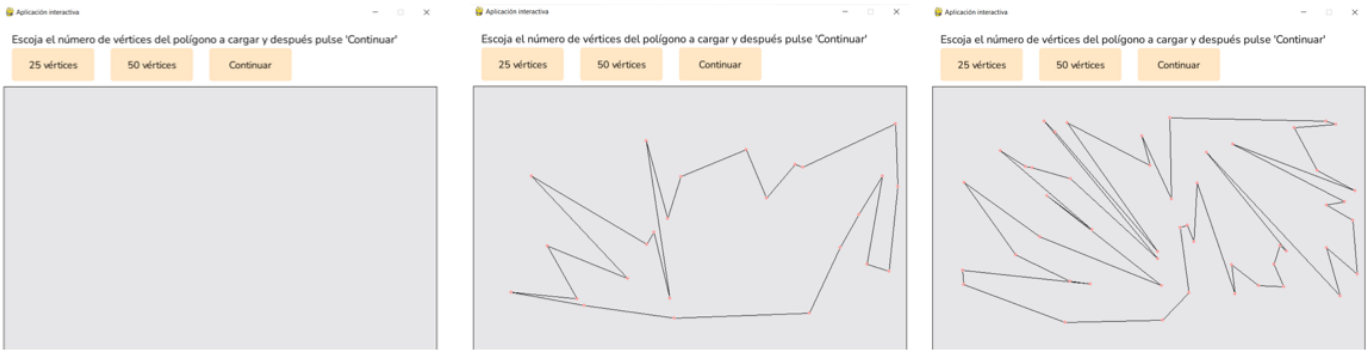


Figura 32: Distintas capturas de la escena que permite cargar un polígono. De derecha a izquierda: antes de cargar polígono; polígono cargado de 25 vértices; polígono cargado de 50 vértices.

- `menu_scene.py`: Segunda escena que permite escoger al usuario la manera de determinar el polígono con el que se trabajará el problema.

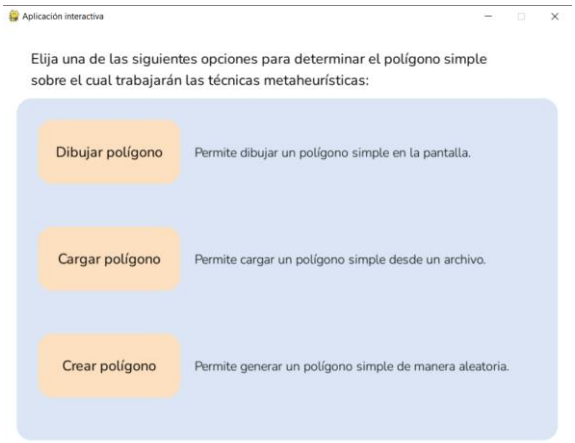


Figura 33: Escena de menú.

- `scene_manager.py`: Esta escena no despliega ninguna interfaz, pero se encarga del funcionamiento completo de la aplicación, manejando los cambios de escena y las interacciones del usuario.
- `simulated_annealing_scene.py`: Escena que permite mostrar cómo el algoritmo de Simulated Annealing resuelve el problema y las iteraciones que realiza para resolverlo (ver Figura 12).
- `solver_scene.py`: Esta escena no despliega ninguna interfaz, si no que sirve de padre para las escenas de los algoritmos. Incluye métodos comunes para la resolución de los algoritmos, que en algunos casos se sobreescriben para llevar a cabo un comportamiento concreto y específico para un algoritmo.
- `start_scene.py`: Primera escena de la aplicación interactiva. Ofrece una breve descripción de la funcionalidad de la aplicación.

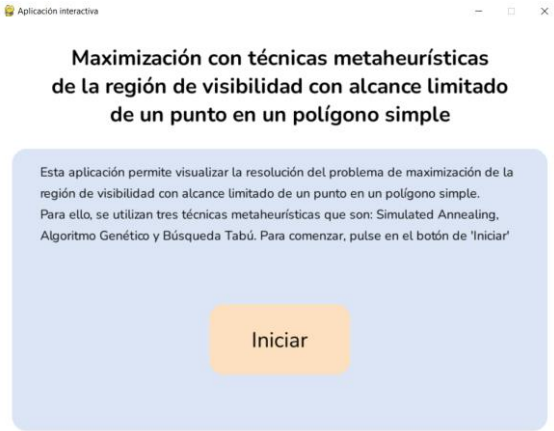


Figura 34: Escena inicial.

- `tabu_search_scene.py`: Escena que permite mostrar cómo el algoritmo de Búsqueda Tabú resuelve el problema y las iteraciones que realiza para resolverlo (ver Figura 15).

- `toying_with_concept_scene.py`: Esta escena permite que el usuario experimente con el concepto de visibilidad de alcance limitado, permitiéndole mover un punto por el polígono y viendo cuál es la región de visibilidad con alcance limitado.

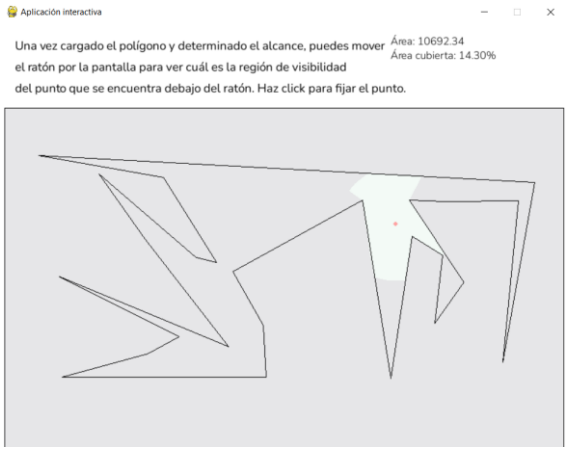


Figura 35: Escena para experimentar con un polígono.

PSEUDOCÓDIGOS

ALGORITMOS REGIÓN DE VISIBILIDAD CON ALCANCE LIMITADO

El siguiente pseudocódigo es común para ambos algoritmos.

```
P ← poligono
p ← punto
d ← alcance
VP ← poligono_visibilidad(P,p)
IP,VPI ← intersección_poligono_circunferencia(VP,p,d)
si IP = ∅ :
      si circunferencia dentro de VP do:
            return π * r^2
      else do:
            return area_poligono(VP)
```

```
function calcular_poligono(q_i,q_{i+1},VPI,p,i):
      polygon ← [q_i]
      k ← i + 1
      while ||VPI_k − q_{k+1}|| ≠ 0 do:
            polygon.append(VPI_k)
            k+= 1
      polygon.append(VPI_k)
      polygon.append(p)
return polygon
```

Algoritmo I

```
area ← 0
para i = 0 hasta len(IP) do:
      q_i ← IP_i
      q_{i+1} ← IP_{i+1}
      MP ← punto_medio(q_i,q_{i+1})
      angle ← angulo(q_i − p, q_{i+1} − p)
      if angle ≤ π do:
            MP ← 2p − MP
      t ← calcular_interseccion_semirrecta_poligono([p,MP],VPI)
      if ||IP − p|| < r do:
            poly += calcular_poligono(q_i,q_{i+1},VPI,p, i)
            area+= area_poligono(poly)
      else:
            area+= calcular_sector_circular(r,angle)
return area
```

Algoritmo II

```
area ← 0
i ← 0
para j = 0 hasta len(IP) do:
      q_i ← VPI_i
      v ← VPI_{i+1}
      if ||v − IP_{j+1}|| = 0 do:
            area+= area_poligono([q_i, v, p])
      else if ||v − p|| < r do:
            incrementar i hasta que ||VPI_i − IP_{j+1}|| = 0
            poly ← calcular_poligono(q_i, IP_{j+1}, VPI, p, i)
            area+= area_poligono(poly)
      else do:
            area+= calcular_sector_circular(r, angle)
return area
```

SIMULATED ANNEALING

```
s_actual ← solución inicial
s_best ← s_actual
T ← T_0  #Dato de entrada
α ← factor de reducción de T
σ ← desviación típica perturbación
para k_0 hasta k_max do:
      perturb ← N(0, σ)
      s_actual ← s_actual + perturb
      δ ← s_actual − s_best
      Si δ < 0 ó rand(0,1) < exp(δ/T) do:
            s_best ← s_actual
      T ← T * α
Return s_best
```

ALGORITMO GENÉTICO

```
poblacion ← puntos aleatorios en el polígono
para k_0 hasta k_max do:
      poblacion_nueva ← poblacion
      para i hasta n do:
            padre_1 ← operador_seleccion(poblacion)
            padre_2 ← operador_seleccion(poblacion)
            hijo ← operador_cruce(padre_1, padre_2)
            hijo ← operador_mutacion(hijo)
            poblacion_nueva_i ← hijo
      poblacion ← poblacion_nueva
Return max(poblacion)
```

BÚSQUEDA TABÚ

```
s_actual ← solución inicial
lista_tabu ← emtpy list
para k_0 hasta k_max do:
      neighbors ← generar_vecinos(s_actual)
      para cada neighbor en neighbors do:
            si neighbors en lista_tabu do:
                  neighbors.remove(neighbor)
      si neighbors vacía:
            return s_actual
      best_neighbor ← obtener_mejor(neighbors)
      Añadir s_actual a lista_tabu
      Recortar lista_tabu, si es necesario
      s_actual ← best_neighbor
return s_actual
```

RANDOM SEARCH

```
s_actual ← punto_aleatorio()
para k_0 hasta n do:
      solucion_potencial ← punto_aleatorio()
      si solucion_potencial > s_actual do
            s_actual ← solucion_potencial
return s_actual
```