TESIS DOCTORAL

Geometría computacional aplicada al reconocimiento de regiones de interés en visión por computador

Rubén Molano Gómez

Programa de Doctorado en Tecnologías Informáticas (TIN)

Con la conformidad de los directores:

Andrés Caro Lindo **Pablo García Rodriguez**

Esta tesis cuenta con la autorización del director y coodirector de la misma y de la Comisión Académica del programa. Dichas autorizaciones constan en el Servicio de la Escuela Internacional de Doctorado de la Universidad de Extremadura.

2024

TESIS DOCTORAL

Geometría computacional aplicada al reconocimiento de regiones de interés en visión por computador

Rubén Molano Gómez

Programa de Doctorado en Tecnologías Informáticas (TIN)

2024

A mi mujer, Carmen y a mis hijas, Clara, Ana y Carmen

Agradecimientos

Al finalizar mi Tesis Doctoral, me gustaría expresar mi agradecimiento a todas las personas que me han acompañado durante este camino. Su ayuda ha sido fundamental en cada etapa de este proceso.

En primer lugar, quiero expresar mi agradecimiento a mi mujer, Carmen. Su apoyo ha ido más allá de las palabras. Ha cuidado de mí, me ha impulsado a ser mejor persona y me ha animado en los momentos difíciles cuando los resultados no eran los esperados. Ha estado simplemente cada día a mi lado. A mis hijas, Clara, Ana y Carmen, les agradezco su comprensión, cariño y paciencia mientras trabajaba en esta tesis. Su alegría por cada pequeño logro mío ha sido un constante recordatorio de por qué es importante seguir adelante.

A mis directores, Andrés Caro y Pablo García, por la confianza que depositaron en mí durante estos años de trabajo. Sus consejos, orientaciones y apoyo constante fueron cruciales para llevar a cabo este proyecto. También quiero agradecer a Mar y a Tote por su dedicación y su aliento para que esta tesis llegase a su final.

Y finalmente, a mis padres por haberme dado una educación y por transmitirme unos valores y principios que han sido fundamentales en mi formación personal. También quiero expresar mi profundo agradecimiento a mi hermano Luis, cuya generosidad ha sido una fuente constante de apoyo y motivación a lo largo de este viaje.

Espero que este modesto agradecimiento refleje mi profunda gratitud hacia cada uno de ellos.

Gracias.

Resumen

La Geometría Computacional (GC) ha ido en constante crecimiento debido a su aplicación en un gran número de campos, incluyendo visión por computador, robótica, medicina o ingeniería. Esta disciplina se centra en el desarrollo de algoritmos para resolver problemas geométricos utilizando computadoras y métodos matemáticos. La optimización geométrica, que ha surgido como una importante línea de investigación dentro del campo de la Geometría Computacional, se ocupa de problemas en los que el objetivo es encontrar la mejor de todas las soluciones factibles. De manera formal, se utiliza para calcular la solución óptima con el valor máximo o mínimo en un espacio de búsqueda que contiene todas las soluciones posibles.

Una Región de Interés (ROI) definida como una sección específica de una imagen, es muy útil en visión por computador para el reconocimiento de patrones, análisis de vídeos o en realidad aumentada. Además, se pueden aplicar a diferentes campos, como en medicina para el diagnóstico médico, monitorización de una enfermedad, investigación médica o para la cirugía guiada por imágenes. En robótica, son importantes para el reconocimiento de objetos, seguimiento de objetos o mapas de navegación.

En el ámbito de la Geometría Computacional, las ROIs son áreas específicas que destacan debido a su importancia en problemas geométricos. Suelen estar compuestas por elementos básicos, siendo los polígonos las formas más usadas debido a su simplicidad, lo que, a su vez, mejora la precisión en los análisis mediante técnicas de optimización geométrica. El cálculo de áreas y perímetros dentro de estos polígonos es esencial para la resolución de problemas geométricos.

Uno de los problemas más interesantes es el cálculo del área o el perímetro de polígonos cuando el número de lados es variable, es decir, polígonos de k lados. A pesar de la existencia de numerosos artículos que estudian este problema en polígonos específicos, como triángulos, cuadriláteros, rectángulos o paralelogramos, aún no se ha elaborado una solución general basada en un único algoritmo, que permita calcular el mayor área o perímetro de cualquier polígono de k lados, independientemente de si es simple o convexo.

En este contexto, esta Tesis Doctoral resuelve el problema anterior al desarrollar una solución algorítmica que calcula el polígono simple o convexo de k lados de mayor o menor área o perímetro que está contenido o que contiene a una ROI. A lo largo de este estudio, se han creado numerosas variantes del algoritmo para resolver situaciones complejas. Estas modificaciones permiten calcular estas soluciones incluso cuando la ROI presenta zonas con irregularidades, como puntos, segmentos o agujeros.

Durante esta investigación, se han analizado tres categorías de problemas: los de inclusión, denominados $Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$; los que tratan la separabilidad, conocidos como $Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu)$; y aquellos que se centran en la envolvente, $Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$. La solución algorítmica previamente mencionada representa una unificación de estas tres clases de problemas.

En líneas generales, el algoritmo desarrollado tiene la capacidad de calcular una amplia gama de soluciones para polígonos de k lados. Es posible optimizar tanto el área como el perímetro del polígono, así como calcular las soluciones tanto dentro del polígono, ya sea simple o convexo, como en su exterior. Además, se puede optar por una implementación iterativa o recursiva, y es especialmente adecuado para resolver problemas relacionadas con la inclusión, la separabilidad o la envolvente.

Para demostrar la utilidad del algoritmo, se ha llevado a cabo una serie de aplicaciones prácticas que no solo sirven para ilustrar, sino que también confirman su capacidad para resolver una amplia variedad de problemas específicos en la vida real. El algoritmo fue desarrollado inicialmente en pseudocódigo y posteriormente implementado en los lenguajes de programación Python y Java. De esta forma, cualquier investigador ante un problema en particular puede personalizar el código para adaptarlo a sus necesidades.

Finalmente, se ha resuelto un problema relacionado con la optimización de volúmenes en el contexto de Volúmenes de Interés (VOIs): cálculo del paralelepípedo de mayor volumen y orientación arbitraria contenido en VOI. Este avance amplía aún más las posibles implicaciones de la investigación.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

> *"Computational geometry is a branch of computer science devoted to the study of algorithms which can be stated in terms of geometry."*
>
> —M. I. Shamos, *Computational geometry*

Los problemas matemáticos han demostrado ser una herramienta fundamental en una amplia variedad de situaciones. Aunque algunos pueden parecer abstractos y teóricos [9, 35, 82, 109, 191, 196], muchos son de gran utilidad en aplicaciones prácticas del mundo real, abarcando disciplinas tan diversas como la física [26, 139, 150, 190], la ingeniería [147, 168], la medicina [83, 96, 148], la biología [21, 59, 60] y las ciencias de la computación [10, 77, 167, 185]. Una característica esencial compartida por varios de estos problemas es su conexión con la geometría, lo que los convierte en un puente entre la abstracción matemática y el mundo físico.

Desde la creación de las primeras máquinas programables por Blaise Pascal [154] en 1642 y Gottfried Wilhelm Leibniz [72] en 1671, hasta la llegada de los primeros ordenadores [111, 153, 197], se han desarrollado nuevas aplicaciones en el ámbito de la geometría computacional, donde la geometría, las matemáticas y las ciencias de la computación se unen para resolver problemas geométricos de manera eficiente y precisa.

Esta Tesis Doctoral tiene como objetivo principal resolver una variedad de problemas matemáticos que surgen de la intersección entre la geometría, las matemáticas y las tecnologías informáticas. Su propósito es demostrar la estrecha colaboración entre la matemática clásica y la computacional en la resolución de estos problemas.

Dentro de este capítulo inicial, se establecen los problemas que serán objeto de estudio (Sección 1.1). Estos problemas serán tratados mediante la aplicación de herramientas de la geometría computacional. Para ello, se define qué es la geometría computacional y

cómo está relacionada con las Regiones de Interés (ROIs) y los Volúmenes de Interés (VOIs). El capítulo finaliza al exponer los objetivos que se proponen conseguir en esta Tesis Doctoral, así como la estructura que se seguirá y las publicaciones que se han obtenido a lo largo de toda la investigación.

1.1. Problemas geométricos propuestos

Se consideran los siguientes problemas que se pueden tratar en diferentes áreas:

Arquitectura. En las afueras de la ciudad, se ha edificado una casa en un terreno plano. Si todavía queda una porción de este espacio disponible, el objetivo es construir una piscina rectangular con el mayor área posible.

Industria fotovoltaica. Existe un terreno parcialmente ocupado por paneles solares y se desea cubrirlo en su totalidad. ¿Es posible calcular la forma poligonal de mayor área y cualquier número de lados que nos permita realizar dicho recubrimiento?

Industria metalúrgica. Una pieza de acero tiene múltiples zonas defectuosas debido a un proceso de fabricación incorrecto. El objetivo es encontrar la forma poligonal de mayor área que pueda ser reciclada y reutilizada, con el número de lados deseado.

Plan urbanístico. En el contexto de la planificación urbana de una ciudad, los edificios y monumentos son representados mediante diferentes colores. El problema consiste en diseñar un parque urbano de cualquier forma poligonal y mayor área, que incluya una amplia variedad de monumentos.

Medicina. Se ha diagnosticado un tumor en un tejido e identificado las células tumorales mediante sus coordenadas. Además, se han registrado también las coordenadas de las células sanas. ¿Cómo se puede lograr una separación precisa entre las células tumorales y las sanas para llevar a cabo una intervención quirúrgica con la mínima extirpación requerida?

Industria textil. Se dispone de una pieza de tela y se desea cortar un patrón específico de cualquier número de lados, ¿cómo se podría hacer de manera eficiente de tal forma que se minimice el área utilizada en el proceso de corte, permitiendo de esta forma reducir el desperdicio de tela?. Si además, ahora la pieza de tela tiene defectos o imperfecciones, tales como errores en su textura, color, forma o tamaño que afectan a su calidad o apariencia, ¿cómo se podría resolver el problema?

Industria de la piedra natural. Se cuenta con un bloque de granito de dimensiones conocidas y se pretende utilizarlo para fines decorativos. Si se busca obtener el máximo rendimiento del material, ¿cómo se podría calcular el paralelepípedo de mayor volumen que puede ser tallado a partir de este bloque?

Los problemas mencionados con anterioridad se encuentran dentro de la disciplina conocida como Geometría Computacional (GC). Al analizar nuevamente las cuestiones previas, pero esta vez desde la perspectiva de esta área de estudio, se obtiene:

Problema 1. Calcular el rectángulo de mayor área y orientación arbitraria contenido en una Región de Interés (ROI) [123].

Arquitectura

Problema 2. Dado un valor k fijo, calcular el polígono simple de k lados de mayor área o perímetro contenido en una ROI con obstáculos arbitrarios, como puntos, segmentos y agujeros [118–121].

Industria fotovoltaica
Industria metalúrgica

Problema 3. Dado un valor k fijo y un conjunto O contenido en una ROI con obstáculos arbitrarios, calcular el polígono simple de k lados de mayor o menor área o perímetro, que contenga a O y esté dentro de la ROI [92, 169].

Plan urbanístico
Medicina

Problema 4. Dado un valor k fijo, calcular el polígono simple de k lados de menor área o perímetro que contenga a una ROI entre obstáculos arbitrarios [62].

Industria textil

Problema 5. Calcular el paralelepípedo de mayor volumen y cualquier orientación contenido en un Volumen de Interés (VOI) [122].

Industria de la piedra natural

Todos estos problemas se centran en la optimización de alguna medida geométrica, como el área, perímetro o volumen, de una forma en una ROI o VOI, ya sea para calcular el mayor o menor área o perímetro, o para calcular el mayor volumen que satisfaga ciertas restricciones predefinidas. Además, cada uno de estos problemas proporciona numerosas aplicaciones prácticas concretas en diferentes campos y por tanto, requieren

de la optimización de formas geométricas mediante la geometría computacional para resolver problemas específicos.

En el marco de esta Tesis Doctoral, la geometría computacional desempeña un papel fundamental debido a su aplicación en la resolución de problemas relacionados con las ROIs y VOIs. A través de la implementación de técnicas y herramientas de la geometría computacional, es posible optimizar formas geométricas y encontrar soluciones eficientes a problemas específicos en diferentes campos.

A continuación, se define qué es la geometría computacional y su importancia en relación con las Regiones de Interés (ROIs) y Volúmenes de Interés (VOIs) en el contexto de nuestra investigación.

1.2. Geometría computacional (GC)

La geometría es una de las ramas más antigua de las matemáticas que se centra en el estudio de las propiedades de las figuras en el plano o en el espacio, y en sus orígenes estudiaba fundamentalmente problemas relativos a la medida como el perímetro, el área y el volumen. Incluye conceptos como puntos, segmentos, ángulos, polígonos, poliedros, etc. Una de las primeras culturas en incorporar el estudio de la geometría fue la civilización babilónica, seguida por la egipcia y posteriormente por la geometría griega [179], donde esta ciencia comenzó a tener un marco teórico más formal a través del matemático griego Euclides (330 a.C.–275 a.C.) por su obra *Los Elementos* [80]; tratado matemático y geométrico compuesto por trece libros donde se recopilaba los conocimientos matemáticos de la antigüedad y que constituyó la geometría euclídea. Sin embargo, dicha geometría fue incapaz de resolver tres famosos problemas geométricos usando únicamente regla y compás: la trisección de un ángulo, la duplicidad del volumen de un cubo y la cuadratura del círculo. Los dos primeros fueron demostrados como irresolubles por P. L. Wantzel en 1837 [192], utilizando en su demostración la *Teoría de Galois* [175]. El último, se debe al matemático F. Lindemann quien probó en 1882 la trascendencia de π [106], y por tanto, la irresolubilidad del tercer problema.

Durante la Edad Media, apenas hubo aportaciones por parte de los matemáticos hindúes y árabes. En el Renacimiento, R. Descartes [55] contribuyó a la geometría al introducir el uso de la notación algebraica para describir figuras geométricas y crear la geometría analítica. Durante los siglos XIX y XX, nuevas teorías aparecieron, como la geometría diferencial, cuyas mayores aportaciones fueron realizadas por C. F. Gauss [70] y la geometría no euclídea, iniciada por I. Kant [91]. En el momento de la aparición de los ordenadores, surge una nueva disciplina llamada *Geometría Computacional (GC)*.

El término "Geometría Computacional" data de 1975 y fue empleado por primera vez por M. I. Shamos en el título de su tesis doctoral [165] y dirigida por F. P. Preparata. Años más tarde, en 1985, publicaron el libro *Computational geometry: an introduction* [145], donde el núcleo central era la tesis anterior, y se demostraba que existía un nexo de conexión entre las matemáticas y las ciencias de la computación. Aunque la geometría computacional ha progresado significativamente, podemos ver que representa una de las primeras áreas geométricas que se aplica para resolver problemas prácticos de la vida cotidiana. De esta forma, la geometría computacional combina las matemáticas, las ciencias de la computación y la geometría para resolver problemas geométricos y visualizar soluciones.

El coste de complejidad es un factor importante a tener en cuenta en la geometría computacional, ya que los algoritmos propuestos deben ser capaces de manejar grandes cantidades de datos en un tiempo razonable. En aplicaciones prácticas como el modelado 3D, gemelos digitales, robótica y visión por computador, es crucial que los algoritmos sean eficientes y puedan procesar información en tiempo real o en tiempos aceptables.

Como la geometría computacional proporciona soluciones a problemas geométricos, existen un gran número de áreas de aplicación, como en visión por computador [141, 182], reconocimiento de patrones [30, 180], robótica [162], sistemas de información geográfica (SIG) [53, 138] o diseño asistido por computador (CAD) [61].

1.2.1. Clasificación de problemas

Dependiendo de la naturaleza de los objetos implicados, se puede clasificar toda la colección de problemas en cinco grupos: proximidad, búsqueda geométrica, intersección, convexidad y optimización geométrica [103, 145].

- **Proximidad.** Tienen como objetivo determinar la distancia mínima o la relación más cercana entre dos objetos en un espacio dado. Se suelen utilizar para evitar colisiones, planificar trayectorias o en la búsqueda de la ruta más corta. Ejemplos que se pueden encontrar son de la forma: dados n puntos en el plano, buscar dos puntos cuya distancia sea la menor posible [20] y dados n puntos en el plano, calcular el vecino más próximo de cada uno [34].

- **Búsqueda geométrica.** Se utilizan para encontrar o posicionar objetos en un espacio geométrico según ciertos criterios. Se emplea para la detección de objetos en imágenes o la búsqueda de puntos o regiones específicas en un mapa. Ejemplos de este tipo son: dados n puntos en el plano, calcular el número de puntos que

se encuentran en un determinado rectángulo [52] y dados n puntos en el plano, encontrar todos los puntos que están dentro de un polígono de k lados [58].

- **Intersección.** Determinan la detección y el cálculo de la intersección de dos o más estructuras geométricas. Se aplican para la detección de colisiones en videojuegos o de contornos en imágenes. Algunos ejemplos de problemas que se pueden encontrar incluyen calcular la intersección entre dos polígonos convexos [136, 166]; calcular la intersección de dos poliedros convexos [40]; dados n segmentos, determinar el número de parejas que intersectan [186], y dados n rectángulos isotéticos, calcular los k pares que intersecan [58].

- **Convexidad.** Abarcan la resolución de estructuras convexas en el espacio. Pueden ser aplicados para la identificación de formas convexas en imágenes, encontrar el mínimo o el máximo de una función en un espacio convexo o para determinar la seguridad de una trayectoria para un robot en un espacio tridimensional. El problema principal es convex hull o envolvente convexa (Figura 1.1), definido de la siguiente forma:

  Dado un conjunto S de n puntos en el plano, se llama convex hull de S, y se denota por $\mathcal{CH}(S)$, al menor polígono convexo P en el que cada punto de S está en la frontera de P o en su interior [74].

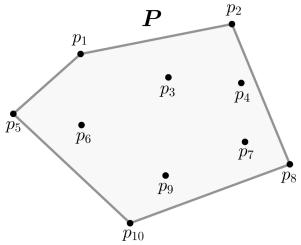


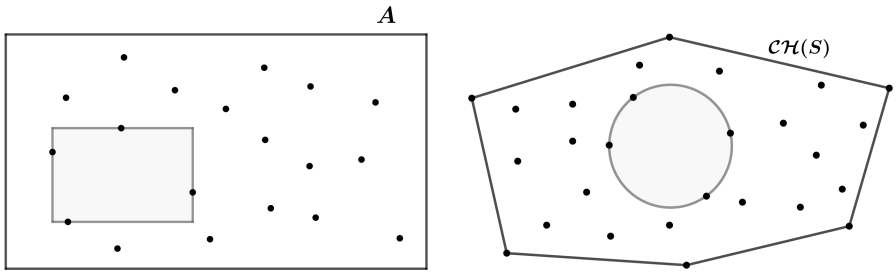
Figura 1.1: Convex hull (CH)

Otros problemas relacionados son: unión de dos polígonos convexos disjuntos y convex hull en dos y tres dimensiones [144] y convex hull de un polígono simple [110].

- **Optimización geométrica.** Se encargan del cálculo de la solución óptima, ya sea maximizando o minimizando un valor, en un espacio de búsqueda que incluye todas las posibles soluciones. Utilizan figuras geométricas fácilmente reconocibles como círculos, esferas, triángulos, rectángulos, paralelogramos o cuadrados. Implican el uso de algoritmos y técnicas de computación para encontrar soluciones óptimas a

problemas geométricos específicos, como la optimización de formas, la planificación de trayectorias o la modelación de superficies.

Algunos casos de aplicación incluyen: dado un rectángulo A y un conjunto S de n puntos de A, calcular el rectángulo vacío de área máxima (MER problem) que esté totalmente contenido en A [39, 127] (Figura 1.2a); calcular el cuadrado vacío de área máxima con cualquier orientación [16]; encontrar el rectángulo isotético vacío de área máxima dados n segmentos de orientación arbitraria que no intersectan entre sí [128] y dado un conjunto S de n puntos en el plano, calcular el círculo vacío de área maxima (LEC problem) cuyo centro se encuentre dentro del convex hull de S [161] (Figura 1.2b).



(a) Maximum Empty Rectangle (MER problem)

(b) Largest Empty Circle (LEC problem)

Figura 1.2: Clasificación de problemas: optimización geométrica

En esta Tesis Doctoral, el objetivo fundamental es resolver problemas relacionados con la optimización geométrica. En este campo, se busca calcular la solución óptima dentro del conjunto de todas las alternativas posibles.

1.3. Región de interés (ROI)

Una región de interés (ROI) es una sección específica de una imagen o un frame de un vídeo que se desea analizar, aumentando así la eficacia y velocidad de los algoritmos. Son muy útiles en visión por computador para la *detección de objetos*, para reconocer patrones en las imágenes y así, poder diferenciar objetos, separarlos y clasificarlos [204]; *análisis de vídeos*, para la identificación de objetos en movimiento o el análisis de patrones en el tiempo [32]; *realidad aumentada (RA)*, superponiendo información adicional sobre un objeto del mundo real [159]. Además, específicamente las ROIs se pueden aplicar en diferentes campos o áreas, como por ejemplo:

- **Medicina.** *Diagnóstico médico*, para identificar cualquier enfermedad a través de imágenes médicas, como imágenes de rayos X, tomografía computarizada (TC) o resonancia magnética nuclear (RMN) [143]; *investigación médica*, para ayudar al diagnóstico, tratamiento y prevención de enfermedades a través del análisis de imágenes [57]; *monitorización*, para seguir el desarrollo de una enfermedad a lo largo del tiempo [69]; *cirugía guiada por imágenes*, permite a los cirujanos comparar imágenes en tiempo real con imágenes previas [183].

- **Robótica.** *Reconocimiento de objetos*, para identificar y distinguir objetos [171]; *seguimiento de objetos*, para detectar y ubicar un objeto a lo largo del tiempo [184]; *mapas de navegación*, permiten generar mapas detallados para que los robots planifiquen sus trayectorias y puedan evitar colisiones con el entorno [131].

- **Sistemas de seguridad y vigilancia.** *Detección de movimiento*, permite conocer cualquier movimiento en una zona específica [115]; *reconocimiento facial*, identificar de forma única a una persona a través de su rostro [45].

- **Sistemas de información geográfica (SIG).** *Análisis de datos geográficos*, para seleccionar áreas específicas para su posterior análisis [137]; *meteorología*, crear modelos climáticos en áreas específicas [38].

- **Tecnología de los alimentos.** *Predicción*, para predecir y mejorar la calidad de los alimentos [140]; *control de calidad*, para asegurar la seguridad y los estándares de calidad [177].

En el procesamiento de imágenes, es común encontrarnos con ROIs irregulares o con partes que no están bien definidas. Estas zonas no disponibles se conocen como *agujeros* (H_i) y son fundamentales para la interpretación de las ROIs. Por otro lado, también pueden aparecer zonas de baja intensidad, difíciles de detectar y que se denominan *puntos* (q_i). Estos puntos pueden estar dispersos o formar parte de *segmentos* (S_i), agrupaciones finitas de puntos alineados. La presencia de estas irregularidades puede dificultar la interpretación de la imagen y su análisis posterior. Es importante tener en cuenta estas irregularidades en las ROIs al realizar cualquier tipo de procesamiento de imágenes y considerar estrategias para minimizar su impacto en la interpretación de los resultados obtenidos.

La Figura 1.3 proporciona un ejemplo de estos conceptos y resalta la importancia de comprender las irregularidades en los ROIs para el procesamiento y análisis de imágenes.

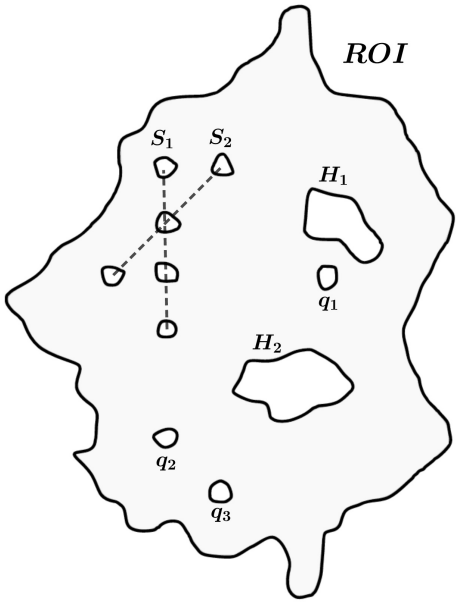


Figura 1.3: Región de interés con puntos (q_1, q_2, q_3), segmentos (S_1, S_2) y agujeros (H_1, H_2)

1.3.1. Polígonos

Definicion 1.3.1. *Una poligonal es una secuencia finita ordenada de puntos del plano que llamamos vértices de la poligonal, es decir,* $P = \{p_1 \dots p_n\}$. *Se definen los lados o aristas, como los segmentos de recta determinados por parejas de vértices consecutivos, es decir, de la forma* $\overline{p_i p_{i+1}}$ *para* $i = 1 \dots n-1$. *Un polígono es una poligonal cerrada, es decir, poligonal* $P = \{p_1 \dots p_n\}$ *tal que* $p_1 = p_n$.

Definicion 1.3.2. *Un polígono simple es un polígono cuyos lados no adyacentes no pueden intersectarse dentro del polígono. Diremos que un polígono es convexo, si todos los segmentos que unen dos puntos del polígono, quedan dentro del polígono. En caso contrario, se dice que es cóncavo (Figura 1.4).*

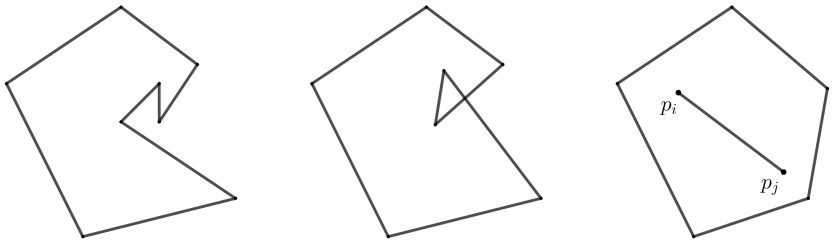


Figura 1.4: Polígono simple, no simple y convexo

Las ROIs son estructuras que suelen estar compuestas por elementos básicos, siendo los polígonos una de las formas más usadas debido a su simplicidad. Estos polígonos se han utilizado en diversos estudios y campos de investigación. En el procesamiento de imágenes de edificios, Sun et al. [178] optimizaron la extracción de información en el proceso de segmentación, mientras que Zhao et al. [203] aplicaron un método mejorado basado en PolyMapper [105] para predecir el contorno de edificios en formato vectorial.

En el campo biomédico, los polígonos se han empleado para el reconocimiento del iris [146], una técnica común en sistemas de identificación y autenticación biométrico. Además, también se han utilizado para evaluar microexpresiones faciales [200]. En la detección precoz del cáncer, el uso de polígonos ha sido fundamental en el análisis de imágenes médicas y la identificación de las mejores áreas de interés para el diagnóstico y tratamiento. De hecho, se han desarrollado estudios que se enfocan en la determinación de la mejor ROI para el análisis de imágenes de cáncer, lo que ha demostrado ser una herramienta importante en la prevención y tratamiento de la enfermedad [12, 22].

Dentro del campo de la robótica, los polígonos también se han aplicado para resolver problemas de planificación de movimiento y navegación. Se han desarrollado algoritmos que dividen un polígono en un conjunto de polígonos más pequeños con una determinada área, lo que permite la planificación de rutas y movimientos precisos [19, 81]. Además, se han propuesto estrategias que permiten a los robots móviles explorar polígonos simples desconocidos, lo que resulta útil en aplicaciones de exploración autónoma [193].

1.4. Volumen de interés (VOI)

Un volumen de interés (VOI) es una región específica en una imagen tridimensional que se ha identificado para su estudio, permitiendo reducir la complejidad del análisis de la imagen en su totalidad. En visión por computador se emplean para la *segmentación de imágenes*, para separar diferentes estructuras dentro de un volumen 3D [198]; *control de calidad en la producción de objetos*, para evaluar la calidad de una sección de un objeto [199]; *reconocimiento de objetos*, para identificar y clasificar objetos [67, 176]. Los VOIs tienen una amplia gama de aplicaciones en diferentes campos, como la medicina, robótica, geología y muchos otros.

- **Medicina.** *Almacenamiento en bases de datos*, para la recuperación de imágenes y su posterior clasificación [97]; *aplicaciones interactivas en telemedicina*, para mejorar la precisión y la calidad de las evaluaciones médicas y diagnósticos a distancia [158]; *ecografía robótica*, para la detección temprana de enfermedades y la planificación del tratamiento [76].

- **Robótica.** *Planificación de rutas de cobertura*, para encontrar una trayectoria que atraviese todos los puntos de un área o volumen de interés [68]; *zonas de exclusión*, para definir el área donde el robot no pueda operar [174]; *ejecución de tareas en entornos complejos*, para realizar una tarea específica en un entorno peligroso [84].
- **Geología.** *Evaluación de riesgos geotécnicos*, para permitir un análisis más preciso y detallado del riesgo dentro de un área geológica [201]; *identificación de recursos minerales*, para detectar la presencia de minerales [173].
- **Tecnología de los alimentos.** *Microbiología alimentaria*, para detectar, analizar y controlar la presencia de microorganismos que pueden provocar enfermedades graves [129]; *microtomografía*, para ayudar en el estudio de la microestructura de los alimentos [130].

1.4.1. Poliedros

Definicion 1.4.1. *Un poliedro es un cuerpo geométrico limitado por caras que son polígonos. Los prismas son poliedros que tienen dos caras paralelas e iguales llamadas bases y caras laterales formadas por paralelogramos. Si las bases son paralelogramos, el cuerpo geométrico se llama paralelepípedo y si además, todas las caras son rectángulos, se habla de ortoedro (Figura 1.5).*

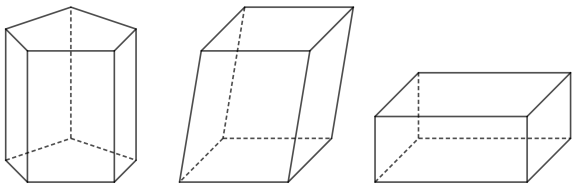


Figura 1.5: Prisma pentagonal, paralelepípedo y ortoedro

Una forma común de aproximar el VOI es mediante la construcción de un poliedro que se ajuste lo mejor posible a la forma del objeto. Esto se puede hacer mediante la unión de varios polígonos planos para formar un poliedro, o mediante la creación de un modelo 3D del objeto recurriendo a software de modelado. Aunque esta aproximación no es la más precisa para todas las formas de objetos, puede ser una herramienta útil para obtener una estimación rápida y confiable de la imagen en ciertos casos.

Los poliedros han sido ampliamente estudiados en diferentes campos. En medicina, se han usado mosaicos triangulares para reconstruir la superficie de órganos a partir de información de contorno generada por tomografía computarizada (TC), ecografías y exámenes de medicina nuclear [47]. También, a modo de ejemplo, se ha desarrollado

una metodología para evaluar el movimiento tridimensional (3D) global del ventrículo izquierdo humano a partir de datos de resonancia magnética, representándolo como una superficie poliédrica [65].

En el campo de la robótica, se ha presentado un algoritmo que permite a un robot móvil autónomo (AMR) generar un modelo tridimensional de un terreno inexplorado con obstáculos, compuesto por un número variable de poliedros [149]. Este avance es importante para aplicaciones en exploración, búsqueda y rescate, ya que permite a los robots mapear y navegar en terrenos desconocidos de manera autónoma. Además, se ha desarrollado una técnica para localizar objetos poliédricos en una mano robótica mediante la integración de datos visuales y táctiles [27].

Por último, en el campo de la teledetección, se ha desarrollado un nuevo método para la extracción de edificios en áreas urbanas a partir de datos LIDAR (Light Detection and Ranging) de alta resolución aplicando modelos poliédricos tridimensionales [157].

En general, estos avances demuestran la versatilidad y la importancia de los poliedros en una amplia variedad de aplicaciones en campos tan diversos como la medicina, la robótica y la teledetección.

1.5. Relación entre la GC, ROI y VOI

El uso de ROI y VOI es una herramienta fundamental en la geometría computacional, como se ha demostrado en diversos estudios. La delimitación del análisis a una región específica de una imagen permite evitar cálculos innecesarios y acelerar la solución del problema. Además, la representación de estas regiones y volúmenes mediante polígonos y poliedros permite una manipulación más sencilla de la geometría, lo que contribuye a una mayor precisión y rapidez en los análisis mediante la optimización geométrica. En consecuencia, la utilización de una ROI y un VOI junto con la representación de estas regiones y volúmenes mediante polígonos y poliedros es esencial para la optimización geométrica en la geometría computacional.

En esta Tesis Doctoral, se adoptará el método ilustrado en la Figura 1.6. El proceso comienza con la selección de una ROI de una imagen (Figura 1.6a), a partir de la cual se calcula el polígono inscrito que mejor se adapte a su forma (Figura 1.6b). Posteriormente, se aplica la optimización geométrica para obtener la solución óptima dentro del conjunto de todas las soluciones factibles. Se presentan tres ejemplos de soluciones: un rectángulo (Figura 1.6c), un hexágono simple (Figura 1.6d) y un hexágono convexo (Figura 1.6d) para el cálculo del mayor área. Como se puede observar, la solución final puede tener un área y perímetro muy diferentes según si se emplea un polígono simple o convexo.

(a) Selección de la ROI

(b) Polígono inscrito en la ROI

(c) Rectángulo

(d) Hexágono simple

(e) Hexágono convexo

Figura 1.6: Aplicación práctica real y soluciones

1.6. Objetivos de la Tesis

El objetivo principal de esta Tesis Doctoral es resolver los cinco problemas planteados en la Sección 1.1 mediante la utilización de técnicas de optimización geométrica, con el fin de validar su aplicabilidad en diversos campos prácticos y demostrar que las posibles aplicaciones son numerosas y variadas.

Para lograr este objetivo principal, es necesario desarrollar una serie de algoritmos y establecer los siguientes objetivos parciales:

Objetivo 1. Calcular el rectángulo de mayor área y orientación arbitraria contenido en una Región de Interés (ROI).

Objetivo 2. Dado un valor k fijo, calcular el polígono simple de k lados de mayor área o perímetro contenido en una ROI con obstáculos arbitrarios, como puntos, segmentos y agujeros.

Objetivo 3. Dado un valor k fijo y un conjunto O contenido en una ROI con obstáculos arbitrarios, calcular el polígono simple de k lados de mayor o menor área o perímetro, que contenga a O y esté dentro de la ROI.

Objetivo 4. Dado un valor k fijo, calcular el polígono simple de k lados de menor área o perímetro que contenga a una ROI entre obstáculos arbitrarios, como puntos, segmentos y agujeros.

Objetivo 5. Desarrollar el pseudocódigo y el código fuente de los algoritmos utilizados anteriormente en Python y Java para que cualquier investigador pueda examinar el código en detalle, comprender el funcionamiento de los algoritmos y realizar modificaciones o mejoras en el código.

Objetivo 6. Calcular el paralelepípedo de mayor volumen y cualquier orientación que está contenido en un Volumen de Interés (VOI).

El objetivo parcial 1 representa una solución específica para el cálculo del rectángulo de área máxima en contraste con el objetivo parcial 2, que ofrece una solución más general. Este último propone un método genérico para calcular el k-gon simple de área o perímetro máximo que puede ser inscrito en cualquier contorno cerrado con obstáculos arbitrarios. Además, si esta solución puede inscribir alguno de los obstáculos con el fin de maximizar o minimizar el área o el perímetro del polígono seleccionado, siempre y cuando se cumpla la restricción de que el polígono esté contenido dentro de la ROI, se estaría tratando con el objetivo parcial 3.

Por otro lado, el objetivo parcial 4 ofrece la capacidad de calcular el k-gon simple con el menor área o perímetro posible. Finalmente, el objetivo parcial 5 desarrolla todo el código fuente, mientras que el objetivo parcial 6 representa una solución para el cálculo del paralelepípedo de mayor volumen contenido en un Volumen de Interés (VOI).

Es importante destacar que esta solución es única y diferenciada en el sentido de que ningún otro documento ha propuesto una solución genérica para cualquier tipo de polígono en un contorno cerrado con obstáculos. La mayoría de los problemas planteados se centran en soluciones muy específicas y no reconfigurables, lo que significa que solo pueden manejar casos particulares y no pueden adaptarse a diferentes situaciones. En contraste, la solución presentada es muy flexible y puede aplicarse a cualquier tipo de contorno cerrado y cualquier número de lados.

Por lo tanto, esta solución es una herramienta valiosa para cualquier investigador que necesite encontrar un polígono inscrito o circunscrito óptimo, adaptado a sus necesidades específicas, sin importar la presencia de obstáculos en el contorno cerrado.

1.7. Estructura de la Tesis

En el primer capítulo de esta Tesis Doctoral, se detallan las aplicaciones prácticas y los problemas que se resuelven a través de la optimización geométrica. Se ofrece una introducción a la geometría computacional y se explica cómo las Regiones de Interés (ROIs) y los Volúmenes de Interés (VOIs), basados en figuras geométricas básicas como polígonos y poliedros, pueden ser aplicados en diversos campos.

El Capítulo 2 realiza una revisión de la literatura disponible hasta la fecha para el cálculo de los polígonos simples o convexos de mayor o menor área o perímetro, con o sin obstáculos arbitrarios, como puntos, segmentos y agujeros, que están contenidos o que contienen un polígono en general. Además, se examina la literatura para el cálculo de poliedros de mayor volumen que puedan encontrarse completamente dentro de otro poliedro o que lo contengan en su totalidad. El objetivo es proporcionar una comprensión completa del estado actual de la investigación en estos temas, para permitir un análisis y discusión rigurosos en los siguientes capítulos.

En el Capítulo 3 se desarrolla el cálculo de la matriz de adyacencia de un polígono reticulado a partir de una Región de Interés (ROI) con obstáculos arbitrarios. Se definen los conceptos de polígono reticulado y matriz de adyacencia, y se desarrollan diferentes algoritmos para determinar la posición de puntos en un polígono, calcular intersecciones entre segmentos y verificar la posibilidad de intersección entre un segmento y un polígono. Por último, se describe el algoritmo empleado para calcular la matriz de adyacencia.

En el Capítulo 4 se resuelve el problema inclusión $Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$, es decir, cálculo del polígono simple de mayor área o perímetro contenido en un polígono reticulado o en una Región de Interés (ROI) con obstáculos arbitrarios. Los algoritmos desarrollados, iterativo y recursivo, proporcionan soluciones específicas basadas en el valor k elegido por el usuario. La razón detrás de este método surge de la escasez de algoritmos que traten de manera completa el cálculo de polígonos simples con obstáculos arbitrarios, lo que contribuye al logro de conseguir el objetivo parcial 2 y, por consiguiente, del objetivo parcial 1. Modificando el algoritmo principal, de cada una de las versiones, es posible resolver el problema para el cálculo del rectángulo o polígono convexo de k lados de mayor área o perímetro.

En el Capítulo 5 se considera el problema de separabilidad $Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu)$. El objetivo es calcular el polígono simple de k lados de mayor o menor área o perímetro que contenga un conjunto dado y esté contenido en un polígono reticulado o en una región de interés (ROI) con obstáculos arbitrarios. Los algoritmos desarrollados, tanto en su versión iterativa como recursiva, contribuyen a alcanzar el objetivo parcial 3, que se deriva de dos problemas planteados inicialmente en esta Tesis Doctoral: diseño de un parque urbano que contenga varios monumentos y separación precisa de células tumorales y sanas en medicina. Al igual que en el Capítulo 4, es posible resolver el problema para el cálculo del rectángulo o polígono convexo de k lados realizando ajustes en el algoritmo principal.

El Capítulo 6 resuelve dos tipos de problemas. En primer lugar, se ocupa de resolver el problema de la envolvente $Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$, es decir, cálculo del polígono simple de k lados de menor área o perímetro que contiene a otro polígono simple entre obstáculos arbitrarios. El segundo problema, consiste en ampliar el resultado anterior cuando el polígono simple de k lados contenga a una ROI entre obstáculos arbitrarios. Para resolver estos problemas, se recurre al procedimiento realizado en el capítulo anterior, logrando así alcanzar el objetivo parcial 4. De manera similar a los dos capítulos precedentes, realizando modificaciones en el algoritmo principal es posible resolver los problemas cuando las soluciones finales sean el cálculo del rectángulo o el polígono convexo de k lados.

El Capítulo 7 desarrolla los dos algoritmos principales de esta Tesis Doctoral, el Algoritmo 7.1 y el Algoritmo 7.2, que permiten unificar los tres tipos de problemas que se han visto en los Capítulos 4, 5 y 6. El primero está centrado en polígonos simples y convexos, mientras que el Algoritmo 7.2 se aplica específicamente a rectángulos.

En el Capítulo 8 se demuestra la utilidad práctica de los algoritmos descritos en los Capítulos 4 al 7. Se exponen una serie de aplicaciones concretas que pueden surgir a partir de los problemas planteados en la Sección 1.1. Para calcular las diversas soluciones, se ha desarrollado inicialmente el código en pseudocódigo y posteriormente se ha traducido a los lenguajes de programación Python y Java, consiguiendo de esta forma el objetivo parcial 5.

El Capítulo 9 presenta un resultado relacionado con los Volúmenes de Interés (VOIs), que proporciona una comprensión fundamental sobre cómo optimizar el uso del espacio en el contexto de los VOIs. Se trata del cálculo del paralelepípedo de mayor volumen contenido en un VOI. La obtención de este resultado permite cumplir con el último de los objetivos parciales establecidos, en concreto, el objetivo parcial 6.

En el Capítulo 10, se describen todos los algoritmos que se han desarrollado a lo largo de esta Tesis Doctoral durante los capítulos 3 al 9.

Para finalizar, el Capítulo 11 expone las conclusiones más destacadas que surgen de este trabajo de investigación doctoral. Además de las conclusiones, se proponen una serie de mejoras que podrían ampliar aún más esta Tesis Doctoral.

1.8. Publicaciones generadas durante la investigación

A continuación, se enumeran los artículos que han sido publicados, los que han sido aceptados para su publicación y aquellos que actualmente se encuentran en proceso de revisión. También se incluyen los congresos internacionales en los que se ha participado. Todos estos trabajos guardan una estrecha relación con algunos de los objetivos parciales previamente descritos.

⌊**1**⌋ T. Perez-Palacios, T. Antequera, R. Molano, P. G. Rodríguez, and R. Palacios, *Sensory traits prediction in dry-cured hams from fresh product via mri and lipid composition*, Journal of Food Engineering, 101 (2010), pp. 152–157.

DOI: https://doi.org/10.1016/j.jfoodeng.2010.06.015

JCR-2022 = 5,5

Category: $\left[\begin{array}{llll} (26/142) & Q1 & \text{Engineering, Chemical} \\ (30/142) & Q1 & \text{Food Science \& Technology} \end{array}\right.$

Objetivo parcial 1

⌊**2**⌋ R. Molano, P. G. Rodríguez, A. Caro, and M. L. Durán, *Finding the largest area rectangle of arbitrary orientation in a closed contour*, Applied Mathematics and Computation, 218 (2012), pp. 9866–9874.

DOI: https://doi.org/10.1016/j.amc.2012.03.063

JCR-2022 = 4

Category: ⌈ (10/267) $Q1$ Mathematics, Applied

Objetivo parcial 1

⌊**3**⌋ R. Molano, D. Caballero, P. G. Rodríguez, M. Ávila, J. P. Torres, M. L. Durán, J. C. Sancho, and A. Caro, *Finding the largest volume parallelepipedon of arbitrary orientation in a solid*, IEEE Access, 9 (2021), pp. 103600–103609.

DOI: 10.1109/access.2021.3098234

JCR-2022 = 3,9

Category: ⌈ (73/158) $Q2$ Computer Science, Information Systems
(41/88) $Q2$ Telecommunications
(100/275) $Q2$ Engineering, Electrical & Electronic

Objetivo parcial 6

⌊**4**⌋ R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *An algorithm to compute any simple k-gon of a maximum area or perimeter inscribed in a region of interest*, SIAM Journal on Imaging Sciences, 15 (2022), pp. 1808–1832.

DOI: https://doi.org/10.1137/22m1482676

JCR-2022 = 2,1

Category: ⌈ (59/267) $Q1$ Mathematics, Applied
(61/108) $Q3$ Computer Science, Software Engineering
(21/28) $Q3$ Imaging Science & Photographic Technology
(109/145) $Q4$ Computer Science, Artificial Intelligence

Objetivos parciales 2 y 5 (ROI sin obstáculos arbitrarios)

⌊**5**⌋ R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *Post-processing of closed contours to obtain inscribed k-sided polygons*, in International Conference on Computing, Intelligence and Data Analytics, 2023.

Objetivos parciales 2 y 5 (ROI sin obstáculos arbitrarios)

⌊**6**⌋ R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *Obtaining the user-defined inside a closed contour with holes*, The Visual Computer (aceptado), (2023).

JCR-2022 = 3,5

Category: ⌈ (34/108) $Q2$ Computer Science, Software Engineering

Objetivos parciales 2 y 5 (ROI con agujeros)

⌊**7**⌋ R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *An efficient method for obtaining the maximum k-gon in an arbitrary polygon with obstacles*, Pattern Recognition (en revisión), (2023).

JCR-2022 = 8

Category: ⌈ (25/145) $Q1$ Computer Science, Artificial Intelligence
(30/275) $Q1$ Engineering, Electrical & Electronic

Objetivos parciales 2 y 5 (ROI con puntos y segmentos)

Capítulo 2

Avances y desarrollos recientes

La revisión de la literatura desempeña un papel fundamental al proporcionar una base sólida que posibilita un análisis riguroso de los temas tratados en cada uno de los capítulos. En el contexto de la optimización geométrica de la geometría computacional, esta revisión es particularmente importante debido a la naturaleza interdisciplinaria de este campo. Implica la aplicación de técnicas de optimización matemática y computacional para resolver problemas geométricos complejos en una variedad de campos, como la medicina o la robótica.

Esta revisión puede ser empleada para identificar deficiencias en el conocimiento previo y para establecer la justificación para la investigación adicional en el campo de la optimización geométrica. Por ejemplo, puede haber áreas en las que se necesite más investigación para mejorar la eficacia de los algoritmos actuales, o bien, para aplicar técnicas existentes a nuevos problemas y escenarios.

Para llevar a cabo esta Tesis Doctoral, se realizará una revisión detallada de las investigaciones previas relacionadas con polígonos (Sección 2.1). Dicha revisión permitirá resolver los objetivos parciales 1 a 5, que se describen en la Sección 1.6. Además, también se llevará a cabo una revisión detallada de los estudios previos sobre poliedros (Sección 2.2), y se centrará específicamente en el objetivo parcial 6, también descrito en la Sección 1.6. De esta manera, se podrá lograr una comprensión más profunda de los problemas geométricos analizados y se contribuirá al desarrollo de soluciones eficaces a través de técnicas de optimización geométrica.

2.1. Polígonos

Dado el carácter general de los objetivos parciales 2, 3 y 4, y la posible aproximación y representación de las ROIs mediante polígonos, es importante tener en cuenta los siguientes criterios o condiciones durante la revisión:

Condición 1. Tipo de polígono inicial: simple o convexo.

Condición 2. Tipo de polígono final: simple o convexo.

Condición 3. Cálculo del área o perímetro máximo o área o perímetro mínimo.

Condición 4. Presencia de obstáculos arbitrarios, como puntos, segmentos y agujeros.

Condición 5. Complejidad temporal del algoritmo propuesto.

Las tres primeras condiciones están relacionadas con dos problemas fundamentales en la optimización geométrica, como son, *problemas de inclusión (inclusion problems)* y *problemas de la envolvente (enclosure problems)*, definidos de la siguiente forma:

Definicion 2.1.1. *Sean $\mathcal{P}$ y $\mathcal{Q}$ dos familias de polígonos y μ una función real respecto al área o al perímetro, tal que:*

$$\forall Q, Q' \in \mathcal{Q},\ \ Q' \subseteq Q \Rightarrow \mu(Q') \leq \mu(Q)$$

Entonces, se define:

Inclusion problems. *$Inc(\mathcal{P}, \mathcal{Q}, \mu)$: Dado $P \in \mathcal{P}$, calcular el polígono $Q \in \mathcal{Q}$ más grande en términos de μ, que esté incluido en P.*

Enclosure problems. *$Enc(\mathcal{P}, \mathcal{Q}, \mu)$: Dado $P \in \mathcal{P}$, calcular el polígono $Q \in \mathcal{Q}$ más pequeño en términos de μ, que contiene a P.*

Las clases de problemas de inclusión y envolvente en geometría computacional se consideran "duales" en cierto sentido, ya que están relacionadas con la inclusión y la envolvente de polígonos, respectivamente. Sin embargo, no existe una manera sistemática conocida para transformar un algoritmo de un problema en uno para su dual. Esto se debe a que cada problema requiere una técnica de resolución específica, y a menudo, las propiedades y características que hacen que un algoritmo sea eficiente en un problema no se aplique al otro. La complejidad y riqueza de la geometría computacional se hacen evidentes a través de la diversidad de técnicas y enfoques que se utilizan para resolver

diferentes problemas. Por ejemplo, el problema *convex skull* se considera el dual del problema *convex hull*, pero hasta la fecha no se ha encontrado una forma sistemática para transformar los algoritmos de uno en el otro.

Para facilitar la referencia a conjuntos particulares durante la revisión, se empleará una serie de notaciones. Así, las siguientes expresiones se usarán para designar conjuntos específicos: $\mathcal{P}_{sim}$, $\mathcal{P}_{conv}$, $\mathcal{P}_k$ y $\mathcal{P}_{k\,conv}$ representarán el conjunto de todos los polígonos simples, convexos, simples de k lados y convexos de k lados, respectivamente. De manera similar, $\mathcal{P}_3$, $\mathcal{P}_{rect}$, $\mathcal{P}_{paral}$ y $\mathcal{P}_4$ se referirán al conjunto de todos los triángulos, rectángulos, paralelogramos y cuadriláteros, respectivamente. En particular, $\mathcal{P}_{rect\,alig}$ se usará para denotar el conjunto de todos los rectángulos cuyos lados son paralelos a los ejes de coordenadas. De esta forma, se podrá identificar fácilmente los conjuntos que se emplean en la revisión y simplificar la discusión.

El objetivo parcial 3 tiene especial relevancia dentro de esta Tesis Doctoral, ya que se ocupa de la relación entre los problemas de inclusión y envolvente. Por una parte, nos encontramos con una ROI con obstáculos arbitrarios y un conjunto de obstáculos O que son seleccionados. El problema consiste en calcular, dado un valor k establecido por el usuario, el k-gon de mayor o menor área o perímetro que contenga al conjunto O y esté contenido en la ROI. A esta clase de problemas se les denomina *problemas de separabilidad (separability problems)*.

Definicion 2.1.2. *Sean $\mathcal{P}$ y $\mathcal{Q}$ dos familias de polígonos, $\mathcal{O}$ una familia de obstáculos y μ una función real respecto al área o al perímetro. Entonces, se define:*

Separability problems. *$Sep(\mathcal{P}, \mathcal{Q}, \mathcal{O}, \mu)$: Dados $P \in \mathcal{P}$ y $O \in \mathcal{O}$, calcular el polígono $Q \in \mathcal{Q}$ más grande o más pequeño en términos de μ, que contenga a O y esté incluido en P.*

La revisión de la bibliografía se centrará en los objetivos parciales y su relación con la clase de problemas a la que pertenecen. Se analizarán detenidamente los artículos de mayor importancia en el estudio de figuras geométricas, como triángulos, rectángulos, paralelogramos, cuadriláteros y polígonos en general. Se examinarán los problemas de separabilidad relacionados con estos tipos de figuras. Esta revisión permitirá profundizar en el conocimiento del problema y sus soluciones, lo que ayudará a establecer unos cimientos sólidos para el desarrollo de soluciones novedosas y eficientes en el futuro.

2.1.1. Triángulos

2.1.1.1. Triángulo de mayor área o perímetro contenido en un polígono

$\boldsymbol{Inc(\mathcal{P}_{sim}, \mathcal{P}_3, \mu)}$

En 1979, Dobkin y Snyder [56] presentaron un algoritmo para calcular el k-gon de mayor área que se puede inscribir en un polígono convexo de n vértices, con un coste computacional de $O(n)$. Sin embargo, estudios posteriores refutaron este algoritmo como incorrecto para el caso $k = 3$ [95, 187]. Boyce et al. [28] ampliaron este resultado para incluir tanto el cálculo del triángulo de mayor área como el de mayor perímetro, con un coste computacional de $O(n \log n)$. Posteriormente, otros autores lograron desarrollar algoritmos lineales para polígonos convexos [88, 90]. El paso de polígonos convexos a polígonos simples fue desarrollado por Melissaratos [112, 113], aunque esto supuso un incremento notable en el coste computacional, elevándolo a $O(n^3)$. Recientemente, Lee et al. [104] han estudiado el problema de encontrar el triángulo de área máxima con diversas restricciones en un polígono simple o convexo, posiblemente con agujeros, y han propuesto un algoritmo con coste computacional en el mejor caso de $O(n^2 \log n)$.

2.1.1.2. Triángulo de menor área o perímetro que contiene a un polígono

$\boldsymbol{Enc(\mathcal{P}_{sim}, \mathcal{P}_3, \mu)}$

El problema de calcular el triángulo de área o perímetro mínimo que contiene a un polígono es una tarea fundamental en el campo de la geometría computacional. A lo largo de los años, ha sido objeto de estudio por parte de numerosos investigadores, y se han propuesto varios algoritmos para resolverlo.

Entre los primeros algoritmos propuestos para calcular el triángulo de área mínima se encuentra el de Klee y Laskowski [99], que logró una complejidad computacional de $O(n \log^2 n)$. Posteriormente, O'Rourke et al. [135] desarrollaron un algoritmo óptimo de tiempo lineal para este mismo problema. La determinación del triángulo de perímetro mínimo resultó ser más difícil y se han encontrado muy pocos resultados al respecto. De Pano [54] propuso un algoritmo $O(n^3)$ para este problema, mejorado posteriormente por Chang y Yap [36]. Aggarwal y Park [4] aplicaron la técnica de búsqueda matricial para reducir la complejidad del problema a $O(n \log n)$, y finalmente, Bhattacharya y Mukhopadhyay [23] lograron un coste computacional $O(n)$. Este algoritmo es el más eficaz para resolver el problema del triángulo de perímetro mínimo.

La Tabla 2.1 proporciona una visión general de los estudios relevantes sobre el cálculo de triángulos contenidos o que contienen a un polígono. Si bien la investigación hasta la actualidad ha logrado avances significativos en el cálculo del área máxima de un polígono, hay aspectos que aún necesitan ser exploradas en profundidad. Por ejemplo, se han encontrado dificultades en la determinación del triángulo de perímetro mínimo que encierra un polígono, lo que ha llevado a la escasez de resultados en este ámbito. Además, la presencia de obstáculos en un polígono plantea un desafío adicional para el cálculo del triángulo de área o perímetro máximo o mínimo. Por tanto, existe una necesidad crítica de investigar más en estas áreas y desarrollar algoritmos eficientes que permitan solucionar estos problemas de manera exitosa.

Tabla 2.1: Triángulos

Sección	Autor	Condición 1–3	Condición 4	Condición 5
2.1.1.1	[104]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_3, \text{área})$	agujeros	$O(n^2 \log n)$
	[28]	$Inc(\mathcal{P}_{conv}, \mathcal{P}_3, \text{perímetro})$	——	$O(n \log n)$
2.1.1.2	[135]	$Enc(\mathcal{P}_{conv}, \mathcal{P}_3, \text{área})$	——	$O(n)$
	[23]	$Enc(\mathcal{P}_{conv}, \mathcal{P}_3, \text{perímetro})$	——	$O(n)$

2.1.2. Rectángulos

2.1.2.1. Maximum Empty Rectangle (MER problem)

$\boldsymbol{Inc(\mathcal{P}_{rect}, \mathcal{P}_{rect}, \mu)}$

En 1984, Naamad et al. [127] propusieron el problema conocido como *"Rectángulo Vacío de Área Máxima"* (MER, por sus siglas en inglés): Dado un rectángulo A y un conjunto S de n puntos de A, calcular el rectángulo de área máxima contenido en A, no contenga ningún punto de S en su interior, y cuyos lados sean paralelos a las aristas de A. El problema se resolvió en $O(min(n^2, R \log n))$, donde R es el número de rectángulos que cumplen las condiciones anteriores. Posteriormente, Orlowski [133] mejoró el coste computacional a $O(R + n \log n)$. En [5, 41] se presentaron otros procedimientos que no requieren el cálculo de todos los MERs, con complejidades $O(n \log^2 n)$ y $O(n \log^3 n)$, respectivamente. Además, el primero resolvió el problema para el cálculo del rectángulo de perímetro máximo en $O(n \log n)$. Eliminando la condición de ejes alineados, [39, 125] demostraron que el mayor rectángulo vacío de orientación arbitraria para un conjunto dado de n puntos en el plano puede calcularse en $O(n^3)$.

Si el conjunto S está formado por n segmentos de orientación arbitraria que no intersectan entre sí, Nandy et al. [128] resolvieron el problema con coste computacional $O(n \log^2 n)$ para el cálculo del rectángulo de mayor área.

2.1.2.2. Rectángulo de mayor área o perímetro contenido en un polígono

$\boldsymbol{Inc(\mathcal{P}_{sim}, \mathcal{P}_{rect}, \mu)}$

Alt et al. [7] propusieron un algoritmo para calcular el rectángulo de mayor área con lados paralelos a los ejes x, y dentro de un polígono convexo de n vértices, con un coste computacional de $O(\log^3 n)$. Posteriormente, Fischer y Höffgen [63] y Alt et al. [8] redujeron este coste a $O(\log^2 n)$ y $O(\log n)$, respectivamente. Cuando se eliminó la restricción para polígonos convexos, Daniels et al. [50] resolvieron el problema, válido también para polígonos con agujeros, en un tiempo de $O(n \log^2 n)$. Boland y Urrutia [24] presentaron un algoritmo en un tiempo de $O(n \log n)$. Al considerar algoritmos de aproximación y eliminar la condición de lados paralelos a los ejes, Knauer et al. [100] demostraron que el rectángulo de mayor área en un polígono convexo podía calcularse en un tiempo de $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \log n)$. Para polígonos simples, Molano et al. [123] resolvieron el problema en un tiempo $O(n^3)$, mientras que Choi et al. [44] demostraron cómo resolver el problema para polígonos con agujeros en $O(n^3 \log n)$, incluido el cálculo del rectángulo de mayor perímetro.

2.1.2.3. Rectángulo de menor área o perímetro que contiene a un polígono

$\boldsymbol{Enc(\mathcal{P}_{sim}, \mathcal{P}_{rect}, \mu)}$

En 1975, Freeman y Shapira [64] propusieron un algoritmo con coste computacional $O(n^2)$ para calcular el rectángulo de menor área que contiene una curva cerrada, lo que equivale a un polígono simple de n vértices. Aunque, Arnon y Gieselmann [11] y Toussaint [181] mejoraron el algoritmo reduciendo su complejidad a $O(n)$, éste sólo era aplicable a polígonos convexos.

2.1.3. Paralelogramos

2.1.3.1. Paralelogramo de mayor área o perímetro contenido en un polígono

$\boldsymbol{Inc(\mathcal{P}_{sim}, \mathcal{P}_{paral}, \mu)}$

Jin y Matulef [89] trataron el problema de calcular el paralelogramo de mayor área dentro de un polígono convexo de n vértices. Su algoritmo tenía un coste computacional

de $O(n^2)$. Posteriormente, Jin [87] mejoró el método anterior para lograr un coste de $O(n \log^2 n)$. El paso de un polígono inicial convexo a uno simple fue realizado por Molano et al. [122] con un coste computacional de $O(n^3)$. Sin embargo, ninguno de los artículos anteriores menciona si es posible calcular el paralelogramo con el mayor perímetro o en presencia de obstáculos.

2.1.3.2. Paralelogramo de menor área o perímetro que contiene a un polígono

$\boldsymbol{Enc(\mathcal{P}_{sim}, \mathcal{P}_{paral}, \mu)}$

Schwarz et al. [164] presentaron un algoritmo con complejidad $O(n)$ para calcular el paralelogramo de menor área que contiene un polígono convexo de n vértices. En una investigación posterior [163], describieron cómo esta técnica puede ser aplicada en el procesamiento digital de imágenes. Más tarde, Rote [156] obtuvo el mismo resultado. De la misma forma que en la sección anterior, no existen artículos relacionados para el cálculo del paralelogramo de menor perímetro o en presencia de obstáculos. Aunque Schwarz et al. [163, 164] y Rote [156] consideraron como polígono inicial un polígono convexo en su trabajo, este sigue siendo un problema de menor dificultad en comparación con polígonos simples.

2.1.4. Cuadriláteros

2.1.4.1. Cuadrilátero de mayor área o perímetro contenido en un polígono

$\boldsymbol{Inc(\mathcal{P}_{sim}, \mathcal{P}_4, \mu)}$

El algoritmo propuesto por Dobkin y Sneider [56] para el cálculo del mayor k-gon contenido en un polígono convexo de n vértices, fue demostrado incorrecto para el caso $k = 4$, mediante un contraejemplo publicado por Keikha et al. [94]. Más tarde, Rote [156] logró resolver el problema con un algoritmo que tiene una complejidad computacional de $O(n)$.

2.1.4.2. Cuadrilátero de menor área o perímetro que contiene a un polígono

$\boldsymbol{Enc(\mathcal{P}_{sim}, \mathcal{P}_4, \mu)}$

Hasta el momento, no se han publicado artículos que presenten algoritmos para calcular el cuadrilátero de menor área o perímetro que contenga un polígono simple o convexo.

La Tabla 2.2 muestra los artículos más destacados sobre el cálculo de rectángulos, paralelogramos y cuadriláteros contenidos o que contienen polígonos simples o convexos. La literatura actual es limitada en cuanto al cálculo de soluciones en relación al perímetro o en presencia de obstáculos, lo que pone de manifiesto la necesidad de explorar áreas de investigación que aún no han sido suficientemente analizadas. Resulta crucial seguir investigando en estos aspectos para ampliar nuestro conocimiento y avanzar en este campo de estudio.

Tabla 2.2: Cuadriláteros

Sección	Autor	Condición 1-3	Cond. 4	Cond. 5
2.1.2.1	[39, 125]	$Inc(\mathcal{P}_{rect}, \mathcal{P}_{rect}, \text{área})$	puntos	$O(n^3)$
	[5]	$Inc(\mathcal{P}_{rect}, \mathcal{P}_{rect}, \text{perímetro})$	puntos	$O(n \log n)$
	[128]	$Inc(\mathcal{P}_{rect}, \mathcal{P}_{rect\ alig}, \text{área})$	segmentos	$O(n \log^2 n)$
2.1.2.2	[123]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{rect}, \text{área})$	——	$O(n^3)$
	[44]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{rect}, \text{perímetro})$	agujeros	$O(n^3 \log n)$
2.1.2.3	[64]	$Enc(\mathcal{P}_{sim}, \mathcal{P}_{rect}, \text{área})$	——	$O(n^2)$
2.1.3.1	[122]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{paral}, \text{área})$	——	$O(n^3)$
2.1.3.2	[156, 163, 164]	$Enc(\mathcal{P}_{conv}, \mathcal{P}_{paral}, \text{área})$	——	$O(n)$
2.1.4.1	[156]	$Inc(\mathcal{P}_{conv}, \mathcal{P}_4, \text{área})$	——	$O(n)$

2.1.5. Polígonos en general

2.1.5.1. Convex skull (Potato peeling problem) (CS) $\boxed{Inc(\mathcal{P}_{sim}, \mathcal{P}_{conv}, \mu)}$

"Potato peeling problem", también conocido como *"convex skull"*, es un problema de geometría computacional que consiste en calcular el polígono convexo de mayor área que está contenido dentro de un polígono dado de n vértices, el cual puede tener agujeros. Fue propuesto por Goodman [73] y resuelto por Chang y Yap [37] con un coste computacional alto: $O(n^7)$ para el cálculo del mayor área y $O(n^6)$ para el cálculo del mayor perímetro. Posteriormente, Hall-Holt et al. [78] y Cabello et al. [33] desarrollaron algoritmos de aproximación más eficientes para el cálculo del mayor área, con un coste computacional menor, obteniendo $O(n \log n)$ y $O(n \log^2 n)$, respectivamente.

2.1.5.2. k–gon de mayor área o perímetro contenido en un polígono

$\boldsymbol{Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)}$

Boyce et al. [28] demostraron que, para un polígono convexo de n vértices, el k-gon convexo de mayor área o perímetro contenido en el n-gon puede ser encontrado con un coste computacional de $O(kn \log n + n \log^2 n)$. Más tarde, Aggarwal et al. [3] mejoraron este algoritmo reduciendo su complejidad a $O(kn+n \log n)$. Ambos artículos consideraron el problema de inclusión $Inc(\mathcal{P}_{conv}, \mathcal{P}_{k\,conv}, \text{área})$. Cuando se extendió el problema a polígonos simples, Molano et al. [118] desarrollaron un algoritmo para calcular el k-gon de mayor área o perímetro, con una complejidad computacional de $O(n^5 k)$. Este algoritmo fue implementado en Python y Java. En trabajos posteriores, los autores lograron ampliar su solución para incluir polígonos simples con agujeros [120] y puntos y segmentos [119], sin aumentar la complejidad computacional, que se mantiene en $O(n^5 k)$, al desarrollar un algoritmo iterativo.

2.1.5.3. Convex hull (CH)

$\boldsymbol{Enc(\mathcal{P}_{sim}, \mathcal{P}_{conv}, \mu)}$

Se llama *"convex hull"* de un conjunto S de n puntos en el plano, y se denota por $\mathcal{CH}(S)$, al menor polígono convexo P en el que cada punto de S está en la frontera de P o en su interior. Existen dos algoritmos principales que resuelven el problema. El primero, propuesto por Graham [74], tiene un coste computacional de $O(n \log n)$. El segundo algoritmo, propuesto por Jarvis [86], $O(nh)$, donde h es el número de vértices del *"convex hull"*. Posteriormente, Kirkpatrick y Seidel [98] desarrollaron un algoritmo que mejoró estos costes a $O(n \log h)$.

Cuando los n puntos forman los vértices de un polígono simple, el problema es más sencillo de resolver. Así, diversos autores [71, 75, 102, 110, 132, 172] a lo largo de los años han probado que el cálculo del *convex hull* de un polígono simple puede realizarse con un coste computacional de $O(n)$.

2.1.5.4. k–gon de menor área o perímetro que contiene a un polígono

$\boldsymbol{Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)}$

En 1984, Chang y Yap [36] resolvieron el problema de calcular el k-gon convexo de menor área que encierra un polígono simple de n vértices con un coste computacional de $O(n^3 \log k)$. Posteriormente, Aggarwal et al. [2] lograron mejorar este algoritmo en un artículo publicado en 1985, reduciendo el coste a $O(n^2 \log n \log k)$. Otro problema geométrico importante es encontrar el polígono de mínimo perímetro que encierra un

polígono dado. Para resolver este problema, Mitchell y Polishchuk [117] propusieron un algoritmo con coste computacional de $O(nk \log k)$, donde k es el número de vértices del polígono de mínimo perímetro.

La Tabla 2.3 muestra los principales trabajos sobre el cálculo de polígonos simples o convexos contenidos o que contienen a otros polígonos. A pesar de los avances logrados en este campo, todavía existen muchas áreas de estudio importantes que requieren ser exploradas, especialmente en términos de calcular soluciones en presencia de obstáculos o en relación al perímetro. Además, se necesita más investigación sobre la adaptación de técnicas existentes para tratar con polígonos simples, ya que la mayoría de los métodos actuales se han centrado en polígonos convexos. En general, se requiere de mayores esfuerzos para desarrollar técnicas eficientes para el cálculo de polígonos contenidos en otros polígonos, y resolver problemas más complejos en este campo.

Tabla 2.3: Polígonos

Sección	Autor	Condición 1–3	Condición 4	Condición 5
2.1.5.1	[37]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{conv}$, área)	agujeros	$O(n^7)$
	[37]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{conv}$, perímetro)	agujeros	$O(n^6)$
	[78]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{conv}$, área)	——	$O(n \log n)$
2.1.5.2	[118]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{k}, \mu)$	——	$O(n^5 k)$
	[120]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{k}, \mu)$	agujeros	$O(n^5 k)$
	[119]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{k}, \mu)$	puntos segmentos	$O(n^5 k)$
2.1.5.3	[98]	$Inc(S, \mathcal{P}_{conv}$, área)	puntos	$O(n \log h)$
	[71, 75, 102] [110, 132, 172]	$Inc(\mathcal{P}_{sim}, \mathcal{P}_{conv}$, área)	——	$O(n)$
2.1.5.4	[2]	$Enc(\mathcal{P}_{sim}, \mathcal{P}_{k\,conv}$, área)	——	$O(n^2 \log n \log k)$
	[117]	$Enc(\mathcal{P}_{conv}, \mathcal{P}_{k\,conv}$, perímetro)	——	$O(nk \log k)$

2.1.6. Separabilidad

2.1.6.1. Conteniendo un punto, $O = \{p\}$

Dado un rectángulo A y un conjunto S de n puntos de A, el problema consiste en calcular el rectángulo de área máxima que esté completamente contenido en A, sus lados sean paralelos a las aristas de A y contenga en su interior un único punto p de S (Figura 2.1). Este problema es una versión mejorada del problema *"Rectángulo vacío de área máxima" (MER)*, visto en la Sección 2.1.2.1. Investigaciones realizadas por [13] y [92] han logrado resolver este problema con un coste computacional de $O(\log n)$ y $O(\log^4 n)$.

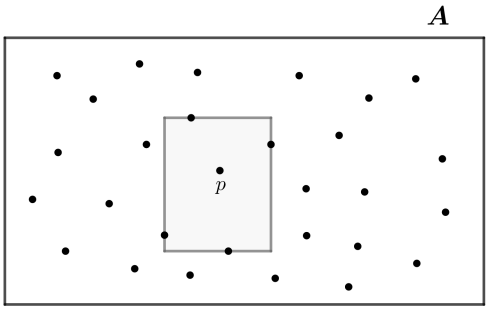


Figura 2.1: Separabilidad, $O = \{p\}$

2.1.6.2. Conteniendo un conjunto de puntos, $O = \{p_1, \ldots, p_r\}$

Sea A un rectángulo y $S = S_r \cup S_b$ un conjunto de puntos de A, donde S_r y S_b son subconjuntos disjuntos de S con tamaños r y b, respectivamente. El objetivo es encontrar un polígono P de área máxima tal que los puntos de S_r se encuentren en su interior o en su frontera, mientras que los puntos de S_b estén en el exterior de P.

Liu y Nediak [107] presentaron un algoritmo para calcular el rectángulo de mayor área con lados paralelos a los ejes x, y en $O(r^2 \log r + rb + b \log b)$. Backer y Keil [15] mejoraron el algoritmo con coste computacional de $O((r+b)\log^3(r+b))$, y propusieron otro para calcular el cuadrado de mayor área con lados paralelos en $O((r+b)\log(r+b))$. Si se elimina la restricción de lados paralelos a los ejes, Bandyapadhyay y Banik [17], demostraron que el rectángulo de mayor área puede calcularse en $O(g(r,b)\log(r+b)+r^2)$ donde $g(r,b) \in O(b^2(r+b))$. Más recientemente, Acharyya et al. [1] mejoraron aún más el algoritmo con un coste computacional de $O(b(r+b)(b\sqrt{r} + r\log r + b\log b)$. Por otro lado, si se considera como polígono inicial un polígono simple, Sheikhi y Alipour [170] demostraron que es posible calcular el triángulo de mayor área con coste computacional de $O(n^{4/3}\log^3 n)$, donde $n = r + b$.

La Tabla 2.4 presenta las publicaciones más relevantes en esta sección. Hasta la fecha, no se ha encontrado una referencia específica que resuelva el problema de calcular el polígono de mayor perímetro. Además, se observa una evidente ausencia de trabajos relacionados con la resolución de este problema para polígonos simples o convexos.

Tabla 2.4: Separabilidad

Sección	Autor	Condición 1-3	Cond. 4	Cond. 5
2.1.6.1	[13]	$Sep(\mathcal{P}_{rect}, \mathcal{P}_{rect\ alig}, \text{punto}, \text{área})$	puntos	$O(\log n)$
2.1.6.2	[170]	$Sep(\mathcal{P}_{sim}, \mathcal{P}_3, \text{puntos}, \text{área})$	puntos	$O(n^{4/3}\log^3 n),\ n = r + b$

2.1.7. Resumen bibliográfico sobre polígonos

Después de realizar la revisión de la literatura disponible sobre polígonos, se han obtenido algunas conclusiones importantes que permitirán mejorar la investigación.

1. Existe una gran cantidad de artículos que resuelven problemas particulares en relación a ciertos tipos de polígonos, como triángulos, rectángulos, paralelogramos o cuadriláteros. No obstante, aún no se ha logrado encontrar una generalización que permita resolver todos estos casos de manera simultánea, lo que constituye una barrera en este campo.

2. La literatura existente sobre el cálculo de soluciones relacionadas con el perímetro o en presencia de obstáculos es limitada en comparación con el cálculo de soluciones respecto al área.

3. La mayoría de los artículos revisados se centran en el uso de polígonos convexos como polígono inicial o final para calcular las soluciones. Esta táctica no siempre produce soluciones óptimas en comparación con el uso de polígonos simples, lo que implica una oportunidad para investigar métodos más eficientes para resolver este tipo de problemas.

A partir de estas observaciones, esta Tesis Doctoral propone una estrategia para resolver de manera simultánea estos problemas para cualquier valor k definido por el usuario. Este método permitirá superar las limitaciones identificadas en la literatura existente y aportar soluciones más eficientes a los problemas relacionados con polígonos.

Para concluir, el objetivo parcial 1, que consiste en el cálculo del rectángulo de mayor área y orientación arbitraria contenido en una ROI [123], se tratará como un caso particular del objetivo parcial 2, y por tanto, no se llevará a cabo el desarrollo del algoritmo. Además, en cada capítulo se presentará el código fuente en pseudocódigo de los objetivos parciales 2, 3 y 4.

En la Tabla 2.5, se resumen las contribuciones específicas que se realizarán en esta Tesis Doctoral. En la última columna, se detallan dos tipos de costes computacionales relacionados con los dos algoritmos que se desarrollarán en los capítulos siguientes: uno a través de una versión iterativa y otro mediante una versión recursiva.

Tabla 2.5: Contribución a través de los objetivos parciales 1 a 5

Capítulo	Objetivo parcial	Condición 1-3	Condición 4	Condición 5
4	1, 2, 5	$Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$	puntos agujeros segmentos	$O(n^5k)$, Iterativo $O(n^k)$, Recursivo
5	3, 5	$Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu)$		
6	4, 5	$Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$		

2.2. Poliedros

La revisión que se llevará a cabo estará orientada a alcanzar el objetivo parcial 6 (Sección 1.6) de esta Tesis Doctoral, que consiste en el cálculo del paralelepípedo de mayor volumen contenido en un Volumen de Interés (VOI). Dado que los VOIs pueden aproximarse mediante poliedros, la revisión se centrará en la identificación y análisis de las publicaciones científicas más relevantes en el campo de la geometría y la optimización geométrica, específicamente en el cálculo del volumen de poliedros.

2.2.1. Ortoedro

O'Rourke [134] demostró cómo calcular el ortoedro de menor volumen que contiene un conjunto de n puntos en $\mathbb{R}^3$ con un coste computacional de $O(n^3)$. Martin y Stephenson [108] desarrollaron una serie de algoritmos para resolver problemas específicos, como determinar si un objeto puede caber dentro de una caja rectangular o calcular el ortoedro de menor volumen que circunscribe un objeto, tanto en dos como en tres dimensiones. Es decir, ampliaron el problema resuelto previamente por Freeman y Shapira [64] al caso tridimensional. Posteriormente, Barequet y Har-Peled [18] presentaron el primer algoritmo de aproximación para el cálculo del ortoedro de menor volumen con un coste computacional $O(n \log n + n/\epsilon^3)$ donde $0 < \epsilon \leq 1$ es el factor control de la precisión.

2.2.2. Poliedros en general

Los problemas *convex skull* y *convex hull*, mencionados en la sección 2.1.5.1 y 2.1.5.3, respectivamente, se pueden extender al espacio tridimensional. En relación al primer problema, investigadores como Borgefors y Strand [25] han trabajado en el cálculo del poliedro convexo de mayor volumen dentro de un objeto digital en tres dimensiones. En un estudio posterior, Karmakar y Biswas [93], lograron resolver el problema con una complejidad computacional de $O(n \log n)$. En cuanto al segundo problema, Preparata y Hong [144] demostraron cómo calcular el convex hull de un conjunto finito de puntos en dos y tres dimensiones con un coste computacional de $O(n \log n)$. Además, Day [51] ofrece una descripción detallada de un algoritmo para el caso tridimensional.

Otros problemas relacionados con poliedros incluyen el cálculo del mayor tetraedro inscrito en un poliedro convexo [155], resuelto con un coste computacional de $O(n \log n)$; cálculo del volumen de un poliedro [46], y cálculo del paralelepípedo de menor volumen que contiene un conjunto de n puntos, resuelto por Vivien y Wicker [189] con un coste computacional de $O(n^6)$. El cálculo del paralelepípedo de mayor volumen contenido en un sólido, fue tratado recientemente por Molano et al. [122] y se resolvió con un coste computacional de $O(n^3)$.

2.2.3. Cuerpos de revolución

En diversos artículos se ha investigado la relación entre poliedros y los cuerpos de revolución, tales como la esfera, cilindro y elipsoide. En el caso de las esferas, Mutoh [126] analizó dos tipos de poliedros. El primero corresponde a aquellos poliedros con n vértices que tienen el volumen máximo y se inscriben en la esfera unitaria de $\mathbb{R}^3$. El segundo tipo se refiere a los poliedros con n vértices que tienen el volumen mínimo y están circunscritos en la esfera unitaria de $\mathbb{R}^3$. Su aportación fue construir tales poliedros para $n \leq 30$. Posteriormente, Chien y Nakazato [43] mostraron cómo resolver un problema de programación lineal relativo a la esfera circunscrita de un poliedro convexo.

En relación a los cilindros y elipsoides, existen algoritmos desarrollados para calcular el cilindro y el elipsoide de menor volumen que circunscribe un conjunto de n puntos en $\mathbb{R}^3$. En particular, Schömer et al. [160] resolvió el problema para cilindros con un coste computacional de $O(n^4 \log n)$, mientras que Welzl [194] desarrolló un algoritmo para elipsoides con un coste computacional de $O(n)$.

2.2.4. Resumen bibliográfico sobre poliedros

Diversas conclusiones se pueden extraer de la revisión llevada a cabo en relación al cálculo de volumen de poliedros. En primer lugar, destaca la notable abundancia de artículos relacionados con este tema, lo que indica que el cálculo de volumen de poliedros es un problema de gran interés en el ámbito matemático y de la computación. Sin embargo, también se puede observar que la información encontrada está muy dispersa, lo que dificulta la tarea de recopilar toda la información necesaria para atender el problema de manera completa.

En este contexto, esta Tesis Doctoral se centra en resolver un problema en particular, cálculo del paralelepípedo de mayor volumen y cualquier orientación contenido en un VOI.

Capítulo 3

Cálculo de la matriz de adyacencia de un polígono reticulado

Este capítulo se centra en el cálculo de la matriz de adyacencia de un polígono reticulado a partir de una Región de Interés (ROI), o contorno cerrado C, con obstáculos arbitrarios, como puntos, segmentos y agujeros (Figura 3.1).

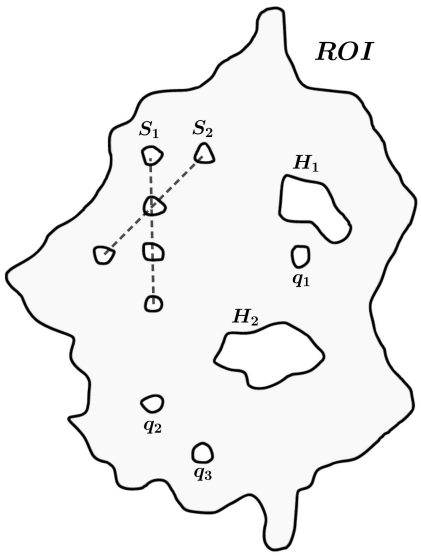


Figura 3.1: Región de interés con puntos (q_1, q_2, q_3), segmentos (S_1, S_2) y agujeros (H_1, H_2)

El capítulo comienza definiendo los conceptos de polígono reticulado y matriz de adyacencia. A continuación, se presentan diversos algoritmos, como determinar si un punto está dentro de un polígono, la intersección entre dos segmentos, calcular el punto de intersección entre dos segmentos y la intersección de un segmento con un polígono. Para finalizar, se describe el algoritmo utilizado para calcular la matriz de adyacencia de un polígono reticulado.

3.1. Polígono reticulado

3.1.1. Definición

Sea $R_{min} = [a, b] \times [c, d]$, $a, b, c, d \in \mathbb{Z}$ el rectángulo de menor área que contiene al contorno cerrado C [64]. Se llama *partición regular* $\Pi = \Pi_x \times \Pi_y$ de orden $r \times s$, a dos colecciones ordenadas de $r + 1$, $s + 1$ puntos igualmente espaciados que verifican:

$$\begin{aligned} \Pi_x &= \{a = x_0 < x_1 < \ldots < x_r = b\} \\ \Pi_y &= \{c = y_0 < y_1 < \ldots < y_s = d\} \end{aligned}$$

Sea $G_L = \{(x_i, y_j) : 0 \leq i \leq r,\ 0 \leq j \leq s\}$, la cuadrícula formada por los puntos de la partición Π. Se define *tamaño de partición*, $L = |x_{i+1} - x_i| = |y_{j+1} - y_j|$, a la longitud del lado de cada cuadrado formado por la cuadrícula. Además, se dice que la partición $\dot{\Pi}$ es más fina que la partición Π, si todos los puntos de Π pertenecen a $\dot{\Pi}$, y se denota como $\Pi \preceq \dot{\Pi}$.

En estas condiciones, se define *polígono reticulado* P como un polígono cuyos puntos pertenecen a la cuadrícula G_L para una partición regular Π. Las conexiones entre vértices consecutivos no se establecen necesariamente en las ocho direcciones, $\pi k/4, k = 0, \ldots, 7$ y además, las aristas del polígono no intersectan, excepto en sus vértices.

Las Figuras 3.2 y 3.3 muestran el polígono reticulado P obtenido a partir del problema inicial (Figura 3.1). Este polígono se forma al unir los puntos (x_i, y_j) de la cuadrícula G_L con el objetivo de obtener el polígono de mayor área que se encuentra completamente dentro de C. Para los agujeros, se busca obtener el polígono de menor área que contenga tanto a H_1 como a H_2, sin incluir puntos adicionales o intersectar y contener segmentos de los obstáculos arbitrarios del problema inicial.

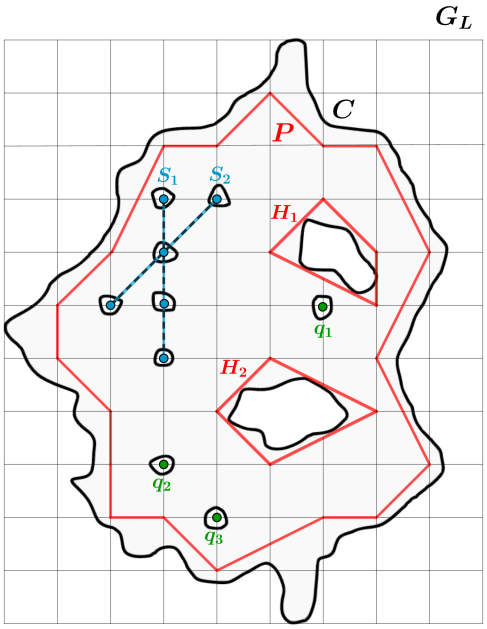


Figura 3.2: Polígono reticulado P sobre una partición regular con tamaño de partición L

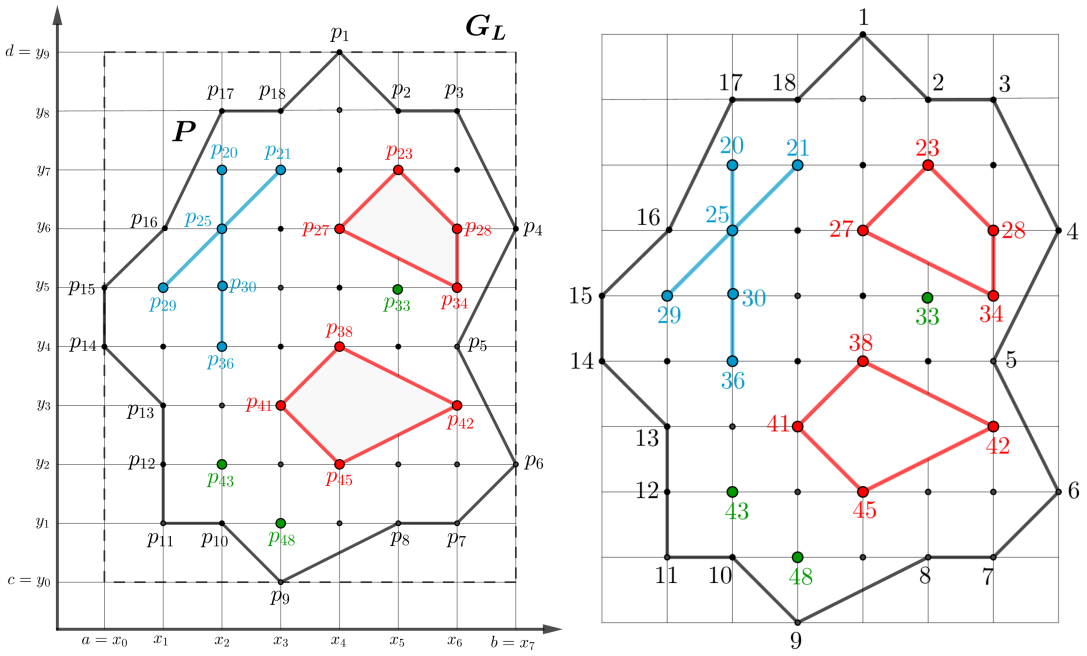


Figura 3.3: Polígono reticulado P sobre una partición regular de orden 7×9

3.1.2. Área de un polígono reticulado

Sea P un polígono reticulado que contiene r–puntos, t–segmentos y h–agujeros. Si $\#$ representa la cardinalidad de un conjunto, se define:

$$\begin{cases} PointsP = \{q_1, \ldots, q_r\} \\ SegmentsP = \{S_1, \ldots, S_t\}, \text{ donde } \#(S_i) = s_i,\ 1 \leq i \leq t,\ s = max\{s_1, \ldots, s_t\} \\ HolesP = \{H_1, \ldots, H_h\} \end{cases}$$

Sea ∂P la familia consistente en los puntos frontera de P, y su complementario $\imath P$ los puntos que pertenecen al interior, es decir, $P = \partial P \cup \imath P$, con $\#(\partial P) = n$, $\#(\imath P) = o$, $\#(P) = N = n + o \simeq \lambda n, \lambda \in \mathbb{N}$. De la misma forma, si $H_i = \partial H_i \cup \imath H_i$ con $\#(\partial H_i) = m_i$, $\#(\imath H_i) = n_i$, entonces $\#(H_i) = m_i + n_i \simeq \mu m_i, \mu \in \mathbb{N}$, $1 \leq i \leq h$. Además, sea $m = max\{m_1, \ldots, m_h\}$.

Para el polígono reticulado representado en la Figura 3.3,

$$\begin{cases} PointsP = \{q_1, q_2, q_3\} = \{p_{33}, p_{43}, p_{48}\} \\ SegmentsP = \{S_1, S_2\} \text{ , donde } S_1 = \{p_{20}, p_{25}, p_{30}, p_{36}\} \text{ y } S_2 = \{p_{21}, p_{25}, p_{29}\} \\ HolesP = \{H_1, H_2\} \text{ , donde } \partial H_1 = \{p_{23}, p_{28}, p_{34}, p_{27}\} \text{ y } \partial H_2 = \{p_{38}, p_{42}, p_{45}, p_{41}\} \end{cases}$$

Si $A(P)$ denota el área del polígono reticulado P, $A(P_0)$ el área del polígono reticulado exterior P_0 y $A(H_i)$ el área de cada agujero con $1 \leq i \leq h$, entonces:

$$A(P) = A(P_0) - \sum_{i=1}^{h} A(H_i) \tag{3.1}$$

con $A(q_i) = 0$, $1 \leq i \leq r$ y $A(S_j) = 0$, $1 \leq j \leq t$.

Por el teorema de Pick [188],

$$A(P) \quad = \quad \left(I + \frac{B}{2} + h - 1\right) \cdot L^2 \quad \text{con:} \quad \begin{cases} I = I_0 - \sum_{i=1}^{h} (I_i + B_i) \\ B = B_0 + \sum_{i=1}^{h} B_i \end{cases} \tag{3.2}$$

donde I y B son el número de puntos interior ($\imath P$) y frontera (∂P) de $P = \partial P \cup \imath P$, I_0 y B_0 el número de puntos interior ($\imath P_0$) y frontera (∂P_0) de $P_0 = \partial P_0 \cup \imath P_0$ e I_i y B_i el número de puntos interior ($\imath H_i$) y frontera (∂H_i) de cada agujero $H_i = \partial H_i \cup \imath H_i$, $1 \leq i \leq h$.

Las ecuaciones 3.1 y 3.2 proporcionan un método para calcular el área de un polígono reticulado. Sin embargo, hay una diferencia clave entre ellas. En el caso de la segunda ecuación, solo es necesario conocer el número de puntos que se encuentran tanto en el interior como en la frontera de los conjuntos seleccionados, es decir, P, P_0 y H_i, $1 \leq i \leq h$.

Así, para la Figura 3.3,

$$\left.\begin{array}{l} I = I_0 - \sum_{i=1}^{2} (I_i + B_i) = 23 \\ B = B_0 + \sum_{i=1}^{2} B_i = 18 + 4 + 4 = 26 \end{array}\right] \Rightarrow A(P) = \left(23 + \frac{26}{2} + 2 - 1\right) L^2 = 37L^2$$

Teniendo en cuenta todo lo mencionado anteriormente, se utilizarán los siguientes conjuntos para calcular la matriz de adyacencia de un polígono reticulado:

$$\left[\begin{array}{l} Points = P = P_0 - \bigcup_{i=1}^{h} (\imath H_i) = \{p_1, \ldots, p_{49}\} \\ Polygon = \partial P_0 = \{p_1, \ldots, p_{18}\} \\ PointsP = \{p_{33}, p_{43}, p_{48}\} \\ SegmentsP = \{S_1, S_2\} \text{ , donde } S_1 = \{p_{20}, p_{25}, p_{30}, p_{36}\} \text{ y } S_2 = \{p_{21}, p_{25}, p_{29}\} \\ HolesP = \{H_1, H_2\} \text{ , donde } \partial H_1 = \{p_{23}, p_{28}, p_{34}, p_{27}\} \text{ y } \partial H_2 = \{p_{38}, p_{42}, p_{45}, p_{41}\} \end{array}\right.$$

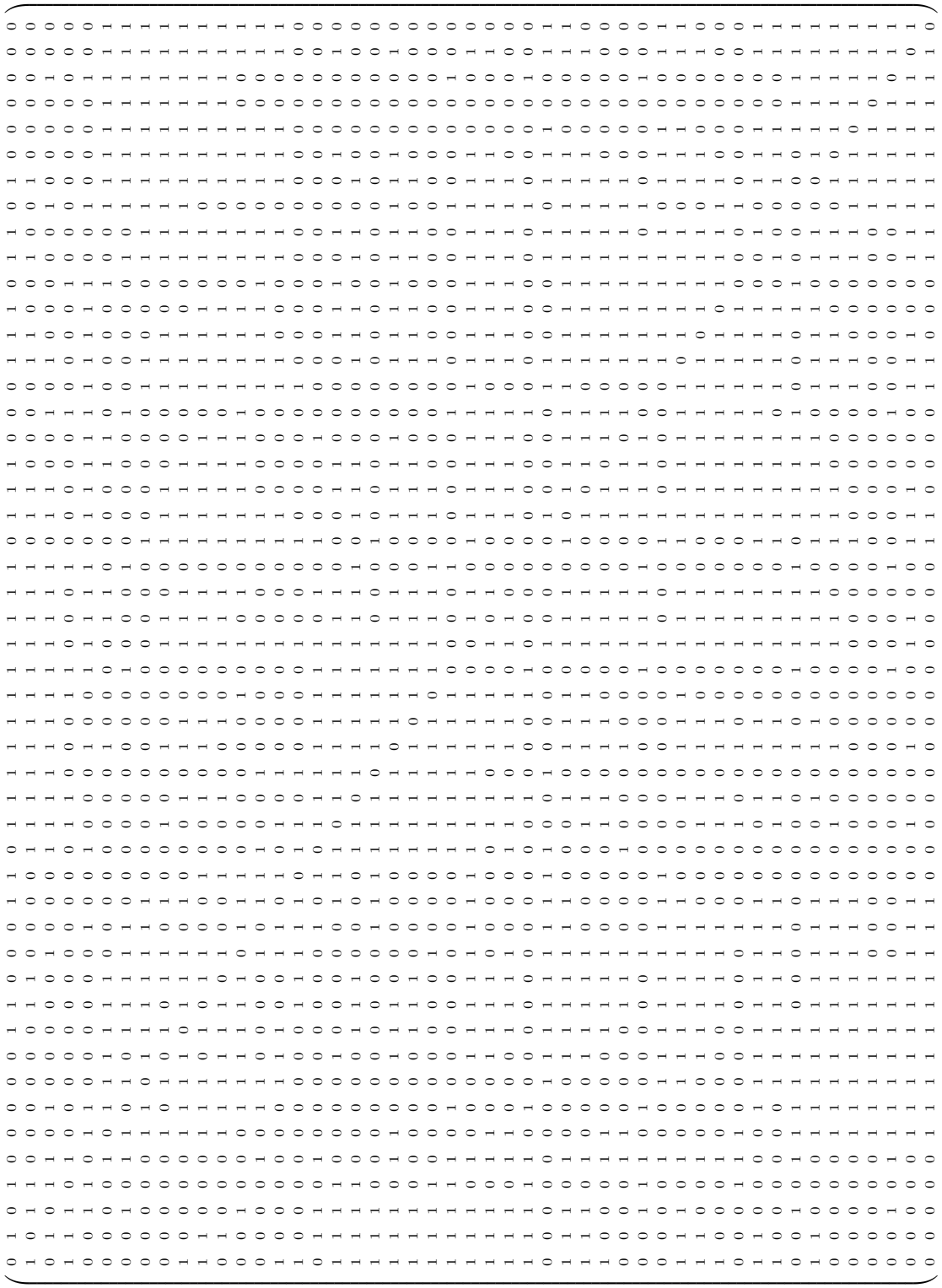
3.2. Matriz de adyacencia de un polígono reticulado

Sea P un polígono reticulado con r–puntos, t–segmentos y h–agujeros, con $\#(P) = N$. Se llama *matriz de adyacencia* $A = (a_{ij}), i, j \in \{1, \ldots N\}$ a una matriz cuadrada simétrica de orden N que verifica:

$$a_{ij} = \begin{cases} 1 & \text{si existe una arista entre } i \text{ y } j \text{ y contenida en } P \\ 0 & \text{en caso contrario} \end{cases}$$

Para la Figura 3.3 la matriz de adyacencia A es una matriz cuadrada de orden 49. Dado que la matriz puede ser de cualquier orden, es necesario desarrollar un algoritmo para calcularla.

$$A =$$



3.3. Cálculo de la matriz de adyacencia

En la Figura 3.4, se presentan los algoritmos que se desarrollarán para calcular la matriz de adyacencia de un polígono reticulado con r–puntos, t–segmentos y h–agujeros. Estos algoritmos se han dividido en cinco subsecciones (3.3.1 - 3.3.5) para facilitar una comprensión más completa en la obtención de la matriz de adyacencia.

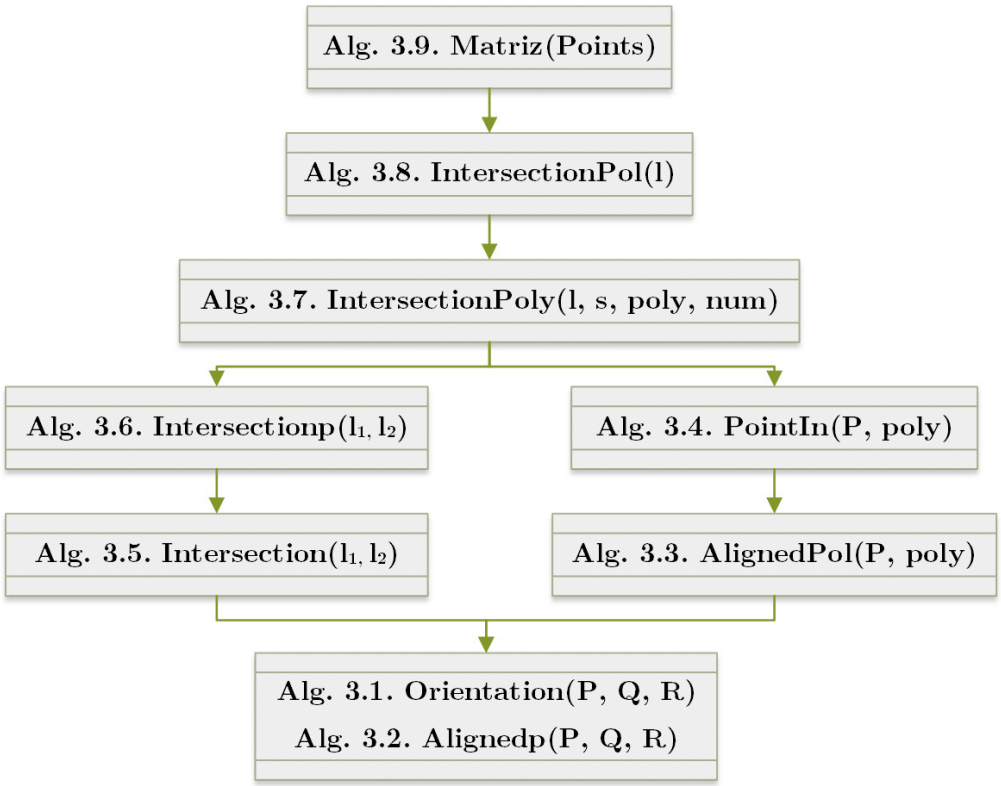


Figura 3.4: Matriz de adyacencia: esquema general de los algoritmos utilizados

3.3.1. Punto dentro de un polígono (Algoritmo 3.4)

Un problema muy natural en el campo de la geometría computacional es determinar si un punto se encuentra dentro o en la frontera de un polígono. Implementar este algoritmo puede resultar muy útil en aplicaciones como en sistemas de información geográfica (SIG) [42] o en el procesamiento de gráficos [202], donde se necesita especificar la ubicación relativa de un punto con respecto a un polígono de forma eficiente y precisa.

El cálculo de este algoritmo se divide en cuatro apartados (3.3.1.1–3.3.1.4), en los cuales cada uno realiza un paso específico para resolver el problema.

Algoritmo 3.1. Orientation(P, Q, R)
Algoritmo 3.2. Alignedp(P, Q, R)
Algoritmo 3.3. AlignedPol(P, poly)
Algoritmo 3.4. PointIn(P, poly)

3.3.1.1. Orientación de puntos en el plano (Algoritmo 3.1)

Definicion 3.3.1. *Se llama producto vectorial de los vectores $\vec{u}$ y $\vec{v}$ a otro vector $\vec{u} \times \vec{v}$, definido de la siguiente forma (3.3) y con las siguientes características (Figura 3.5):*

$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix} \tag{3.3}$$

- *Módulo:* $|\vec{u} \times \vec{v}| = |\vec{u}| \cdot |\vec{v}| \cdot \operatorname{sen} \alpha$.
- *Dirección:* perpendicular a los vectores $\vec{u}$ y $\vec{v}$.
- *Sentido:* el del avance de un sacacorchos que gira de $\vec{u}$ a $\vec{v}$.

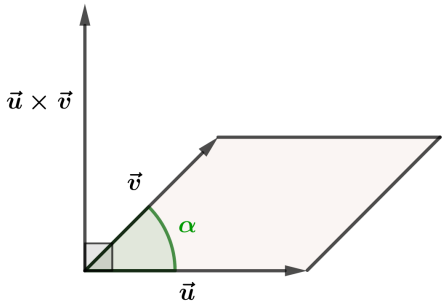


Figura 3.5: Producto vectorial de vectores

Si los vectores están definidos en el plano, $\vec{u} = (u_1, u_2)$ y $\vec{v} = (v_1, v_2)$, entonces:

$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_1 & u_2 & 0 \\ v_1 & v_2 & 0 \end{vmatrix} = \begin{vmatrix} u_1 & u_2 \\ v_1 & v_2 \end{vmatrix} \cdot \vec{k} = (u_1 \cdot v_2 - u_2 \cdot v_1) \cdot \vec{k}$$

Si el producto vectorial $\vec{u} \times \vec{v} > 0$, implica que $\vec{u}$ se encuentra en el sentido de las agujas del reloj desde $\vec{v}$ con respecto al origen de coordenadas $(0, 0)$; en sentido contrario a las agujas del reloj si $\vec{u} \times \vec{v} < 0$ y en el caso de que el producto sea nulo, indica que los vectores son colineales.

Supongamos que se tienen tres puntos, $P = (p_1, p_2), Q = (q_1, q_2)$ y $R = (r_1, r_2)$ y se quiere determinar su orientación. Siguiendo el procedimiento anterior,

$$\overrightarrow{QP} \times \overrightarrow{QR} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ p_1 - q_1 & p_2 - q_2 & 0 \\ r_1 - q_1 & r_2 - q_2 & 0 \end{vmatrix} = \begin{vmatrix} p_1 - q_1 & p_2 - q_2 \\ r_1 - q_1 & r_2 - q_2 \end{vmatrix} \cdot \vec{k}$$

El Algoritmo 3.1 presenta un método para calcular la orientación de tres puntos en el plano, con coste computacional constante $O(1)$.

Algoritmo 3.1 Orientation(P, Q, R)

Input: P, Q, R: puntos

Output: Orientación de los puntos P, Q y R

// $P = (p_1, p_2) = (P[1], P[2])$

1 dir $= (P[1] - Q[1]) \cdot (R[2] - Q[2]) - (P[2] - Q[2]) \cdot (R[1] - Q[1])$

2 **return** dir

3.3.1.2. Punto dentro de un segmento (Algoritmo 3.2)

El Algoritmo 3.2 ofrece una solución para determinar, con coste computacional constante $O(1)$, si un punto R que es colineal con un segmento $\overline{PQ}$ se encuentra dentro de dicho segmento.

Algoritmo 3.2 Alignedp(P, Q, R)

Input: R: punto, $\overline{PQ}$: segmento

Output: *true* si $P \leq R \leq Q$ o $Q \leq R \leq P$

1 alig $= min\,(P[1], Q[1]) \leq R[1] \leq max\,(P[1], Q[1])$ **and** $min\,(P[2], Q[2]) \leq R[2] \leq max\,(P[2], Q[2])$

2 **return** alig

3.3.1.3. Punto contenido en algún lado de un polígono (Algoritmo 3.3)

El Algoritmo 3.3 se utiliza para comprobar si un punto P se encuentra en uno de los lados de un polígono. Se basa en los algoritmos anteriores, Algoritmo 3.1 y Algoritmo 3.2. Para llevar a cabo la comprobación, el algoritmo recorre secuencialmente todos los puntos consecutivos del polígono *pol* (líneas 4–8) utilizando los algoritmos previos.

El coste computacional está definido por la línea 4, con una complejidad de $O(n)$ para el polígono principal *Polygon* y en $O(m)$ para cada uno de los agujeros del polígono $H_i = HolesP[i]$, donde $m = max\{m_1, \ldots, m_h\}$.

Algoritmo 3.3 AlignedPol(P, poly)

Input: P: punto, poly: polígono: *Polygon* or *HolesP[i]*

Output: *true* si el punto P pertenece a uno de los lados de poly

```
1 alinpol ← false
2 pol ← poly
3 Insert(pol, poly[1])
4 for i ← 1 to Length(pol)-1 do
5     if Orientation(Points[pol[i]], Points[pol[i+1]], P] = 0 then
6         if Alignedp(Points[pol[i]], Points[pol[i+1]], P) = true then
7             alinpol ← true
8             break
9 return alinpol
```

3.3.1.4. Algoritmo 3.4: PointIn(P, poly)

El Algoritmo 3.4 determina si un punto P está dentro o sobre uno de los lados de un polígono. Permite realizar la comprobación tanto para el polígono principal *Polygon* como para los agujeros *HolesP[i]*, $1 \leq i \leq h$.

El algoritmo comienza comprobando si el punto P se encuentra sobre uno de los lados del polígono *poly* (línea 1) (Algoritmo 3.3). En caso de que no esté sobre ningún lado (línea 3), se procede a recorrer ordenadamente todos los puntos frontera del polígono, calculando los ángulos formados por los puntos y las líneas que los conectan con el punto P. La suma de todos estos ángulos puede tener dos posibles valores, 0 y $\pm 2\pi$. Estos valores determinan la posición del punto en relación al polígono, siendo 0 si el punto está fuera del polígono y 2π si está dentro (líneas 9–10).

El coste computacional del algoritmo presenta dos casos. En primer lugar, si se aplica al polígono principal *Polygon*, la complejidad es $O(n)$. Por otro lado, si se utiliza para cada uno de los agujeros del polígono *HolesP[i]*, el coste es $O(m)$, donde $m = max\{m_1, \ldots, m_h\}$. Además, se emplea la función *Angle(u, v: vectores)* que calcula el ángulo formado por los vectores $\vec{u}$ y $\vec{v}$, con coste computacional constante $O(1)$.

Algoritmo 3.4 PointIn(P, poly)

Input: P: punto, poly: polígono: *Polygon* or *HolesP[i]*

Output: *true* si el punto P está dentro o sobre un lado de *poly*

```
1  if AlignedPol(P, poly) = true then
2  |   point-in ← true
3  else
4  |   point-in ← false
5  |   pol ← poly
6  |   Insert(pol, poly[1])
7  |   for i ← 1 to Length(pol)-1 do
8  |   |   suma ← suma + Angle(Points[pol[i]] − P, Points[pol[i + 1]] − P)
9  |   if Abs(suma) = 2π then
10 |   |   point–in ← true
11 return point–in
```

3.3.2. Intersección de segmentos (Algoritmo 3.5)

Algoritmo 3.5. Intersection(l_1, l_2)

Dados los segmentos l_1 y l_2, el Algoritmo 3.5 se utiliza para determinar si intersectan. Este algoritmo, con coste computacional constante, se encuentra en el libro *"Introduction to Algorithms"* de Cormen et al. [48]. Se ha realizado una pequeña modificación en el algoritmo, añadiendo una nueva línea (línea 10) para eliminar el caso en el que los dos segmentos sean consecutivos.

Algoritmo 3.5 Intersection(l_1, l_2)

Input: l_1, l_2: segmentos

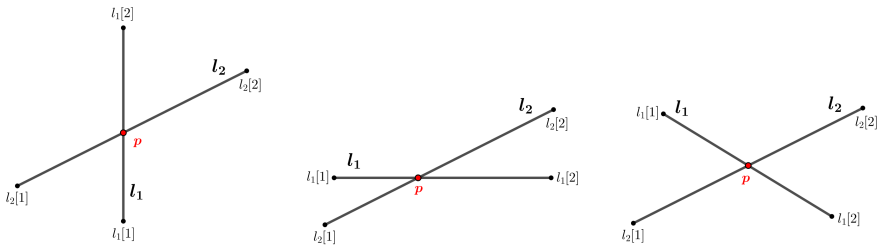
Output: *true* si los segmentos l_1 y l_2 son secantes

```
1  inter ← false
2  p1 ← l1[1]
3  p2 ← l1[2]
4  p3 ← l2[1]
5  p4 ← l2[2]
6  d1 ← Orientation(p3, p4, p1)
7  d2 ← Orientation(p3, p4, p2)
8  d3 ← Orientation(p1, p2, p3)
9  d4 ← Orientation(p1, p2, p4)
10 if p1 ≠ p3 and p1 ≠ p4 and p2 ≠ p3 and p2 ≠ p4 then
11     if ((d1 > 0 and d2 < 0) or (d1 < 0 and d2 > 0)) and
        ((d3 > 0 and d4 < 0) or (d3 < 0 and d2 > 0)) then
12         inter ← true
13     else if (d1 = 0 and Alignedp(p3, p4, p1) = true) then
14         inter ← true
15     else if (d2 = 0 and Alignedp(p3, p4, p2) = true) then
16         inter ← true
17     else if (d3 = 0 and Alignedp(p1, p2, p3) = true) then
18         inter ← true
19     else if (d4 = 0 and Alignedp(p1, p2, p4) = true) then
20         inter ← true
21     else
22         inter ← false
23 return inter
```

3.3.3. Punto de intersección de dos segmentos (Algoritmo 3.6)

Algoritmo 3.6. Intersectionp(l_1, l_2)

El Algoritmo 3.6 calcula el punto de intersección de dos segmentos no consecutivos, l_1 y l_2. Se consideran tres casos que dependen de la posición relativa de los segmentos (Figura 3.6).



(a) Caso 1: l_1 vertical (b) Caso 2: l_1 horizontal (c) Caso 3

Figura 3.6: Posición relativa de dos segmentos secantes no consecutivos en el plano

Sea el segmento $l_1 = \{P_1, Q_1\}$ definido de la siguiente forma:

$$\left[\begin{array}{l} P_1 = l_1[1] = (l_1[1][1],\, l_1[1][2]) = (x_1,\, y_1) \\ Q_1 = l_1[2] = (l_1[2][1],\, l_1[2][2]) = (x_2,\, y_2) \end{array}\right.$$

Si $\overrightarrow{u_1} = \overrightarrow{P_1Q_1} = (x_2 - x_1,\, y_2 - y_1)$ es el vector director de la recta r que pasa por el segmento l_1,

$$\begin{array}{rcl} r \equiv \dfrac{x - x_1}{x_2 - x_1} = \dfrac{y - y_1}{y_2 - y_1} & \Rightarrow & (y_2 - y_1) \cdot (x - x_1) = (x_2 - x_1) \cdot (y - y_1) \\ & \Rightarrow & (y_2 - y_1)x + (x_1 - x_2)y = x_1 \cdot y_2 - x_2 \cdot y_1 \\ & \Rightarrow & r \equiv AX + By = C \text{ , donde:} \end{array}$$

$$\left[\begin{array}{rcl} A & = & y_2 - y_1 = l_1[2][2] - l_1[1][2] \\ B & = & x_1 - x_2 = l_1[1][1] - l_1[2][1] \\ C & = & x_1 \cdot y_2 - x_2 \cdot y_1 = l_1[1][1] \cdot l_1[2][2] - l_1[2][1] \cdot l_1[1][2] \end{array}\right.$$

Haciendo el mismo proceso para el segmento $l_2 = \{P_2, Q_2\}$,

$$\left[\begin{array}{l} P_2 = l_2[1] = (l_2[1][1],\, l_2[1][2]) = (x_3,\, y_3) \\ Q_2 = l_2[2] = (l_2[2][1],\, l_2[2][2]) = (x_4,\, y_4) \end{array}\right.$$

se obtiene una recta s de ecuación, $s \equiv DX + Ey = F$, donde:

$$\left[\begin{array}{lcl} D & = & y_4 - y_3 = l_2[2][2] - l_2[1][2] \\ E & = & x_3 - x_4 = l_2[1][1] - l_2[2][1] \\ F & = & x_3 \cdot y_4 - x_4 \cdot y_3 = l_2[1][1] \cdot l_2[2][2] - l_2[2][1] \cdot l_2[1][2] \end{array}\right.$$

Por tanto, el cálculo del punto de intersección $p = (x, y)$ de los segmentos l_1 y l_2 requiere la resolución del siguiente sistema de ecuaciones:

$$\left.\begin{array}{rcrcl} Ax & + & By & = & C \\ Dx & + & Ey & = & F \end{array}\right\}$$

Por la regla de Cramer [49],

$$x = \frac{\begin{vmatrix} C & B \\ F & E \end{vmatrix}}{\begin{vmatrix} A & B \\ D & E \end{vmatrix}} = \frac{CE - BF}{AE - BD} \qquad y = \frac{\begin{vmatrix} A & C \\ D & F \end{vmatrix}}{\begin{vmatrix} A & B \\ D & E \end{vmatrix}} = \frac{AF - CD}{AE - BD}$$

Luego, $p = (x, y) = \left(\dfrac{CE - BF}{AE - BD}\,,\, \dfrac{AF - CD}{AE - BD}\right)$ con $BD - AE \neq 0$

Si se consideran ahora los casos de la Figura 3.6:

Caso 1: $l_1[1][1] = l_1[2][1] \Rightarrow x_1 = x_2 \Rightarrow B = 0 \Rightarrow p = \left(\dfrac{C}{A}\,,\, \dfrac{AF - CD}{AE}\right)$

Caso 2: $l_1[1][2] = l_1[2][2] \Rightarrow y_1 = y_2 \Rightarrow A = 0 \Rightarrow p = \left(\dfrac{BF - CE}{BD}\,,\, \dfrac{C}{B}\right)$

Caso 3: $A, B \neq 0 \Rightarrow p = \left(\dfrac{CE - BF}{AE - BD}\,,\, \dfrac{AF - CD}{AE - BD}\right)$

El Algoritmo 3.6 comienza calculando en primer lugar los vectores directores de las rectas que contienen a los segmentos l_1 y l_2 (líneas 1 y 2), siendo paralelos si $u_1[1] \cdot u_2[2] = u_1[2] \cdot u_2[1]$ (línea 3).

$$\left[\begin{array}{l} \overrightarrow{u_1} = (u_1[1], u_1[2]) = (l_1[2][1] - l_1[1][1], l_1[2][2] - l_1[1][2]) = (x_2 - x_1, y_2 - y_1) \\ \overrightarrow{u_2} = (u_2[1], u_2[2]) = (l_2[2][1] - l_2[1][1], l_2[2][2] - l_2[1][2]) = (x_4 - x_3, y_4 - y_3) \end{array}\right.$$

El algoritmo continúa suponiendo que los dos segmentos intersectan (línea 4). En ese caso, los segmentos pueden ser secantes o coincidentes. Si son secantes (línea 5), se calcula el punto de intersección p teniendo en cuenta los casos mostrados en la Figura 3.6: caso 1 (línea 12), caso 2 (línea 14) y caso 3 (línea 16). Si los segmentos son coincidentes (línea 18), se obtienen infinitos puntos de intersección y se toma, por ejemplo, $p = \text{Medio}(l_1[1], l_1[2])$ como punto de intersección.

El coste computacional del Algoritmo 3.6 es constante debido a que depende del Algoritmo 3.5. *Intersection*(l_1, l_2), y de la función *Medio*(P, Q), que calcula el punto medio de un segmento con extremos P y Q. Ambos tienen un coste computacional constante.

Algoritmo 3.6 Intersectionp(l_1, l_2)

Input: l_1, l_2: segmentos

Output: Punto de intersección de los segmentos l_1 and l_2

```
1  u1 ← (l1[2][1] − l1[1][1], l1[2][2] − l1[1][2])
2  u2 ← (l2[2][1] − l2[1][1], l2[2][2] − l2[1][2])
3  paral ← u1[1] · u2[2] = u1[2] · u2[1]
4  if Intersection(l1, l2) = true then
5      if paral = false then
6          A ← l1[2][2] − l1[1][2]
7          B ← l1[1][1] − l1[2][1]
8          C ← l1[1][1] · l1[2][2] − l1[2][1] · l1[1][2]
9          D ← l2[2][2] − l2[1][2]
10         E ← l2[1][1] − l2[2][1]
11         F ← l2[1][1] · l2[2][2] − l2[2][1] · l2[1][2]
12         if l1[1][1] = l1[2][1] then
13             p ← {C / A, (AF − CD) / AE}
14         else if l1[1][2] = l1[2][2] then
15             p ← {(BF − CE) / BD, C / B}
16         else
17             p ← {(BF − CE) / (BD − AE), (AF − CD) / (AE − BD)}
18     else
19         p ← Medio(l1[1], l1[2])
20 return p
```

3.3.4. Intersección segmento–polígono

Algoritmo 3.7. IntersectionPoly(l, s, poly, num)
Algoritmo 3.8. IntersectionPol(l)

3.3.4.1. Intersección segmento–lado (Algoritmo 3.7)

El Algoritmo 3.7 determina cuándo es posible la intersección entre un segmento l y un lado s de *Polygon* o un agujero de *HolesP*. Devuelve *true* si la intersección no está permitida y *false* en caso contrario.

Comienza suponiendo que la intersección existe (línea 2) y calcula con la ayuda del Algoritmo 3.6 (línea 3) el punto de intersección. Dado que existen tres posiciones relativas para la intersección de segmentos no consecutivos (Figura 3.6) (líneas 4, 7, 10), el algoritmo calcula los puntos m_1 y m_2 dentro de un entorno de radio epsilon (línea 1, $\epsilon \approx 0$). El cálculo de m_1 y m_2 se realiza a partir de la posición relativa del segmento l y el lado s teniendo en cuenta la Figura 3.6.

Caso 1: $l_1[1][1] = l_1[2][1] \Rightarrow \left[\begin{array}{l} m_1 = (l_1[1][1],\, p[2] - \epsilon) \\ m_2 = (l_1[1][1],\, p[2] + \epsilon) \end{array} \right.$

Caso 2: $l_1[1][2] = l_1[2][2] \Rightarrow \left[\begin{array}{l} m_1 = (p[1] - \epsilon,\, l_1[1][2]) \\ m_2 = (p[1] + \epsilon,\, l_1[1][2]) \end{array} \right.$

Caso 3: Sea $r \equiv Ax + By = C$ la recta que pasa por el segmento l. Si se selecciona un punto $p(x, y)$ y se establece un entorno de radio epsilon, $x \pm \epsilon$, se obtiene $A(x \pm \epsilon) + By = C$. Luego, $y = (C - A(x \pm \epsilon)\,/B$ y las coordenadas de los puntos m_1 y m_2 se pueden expresar de la siguiente forma:

$$m_1 = \left(p[1] - \epsilon,\, \frac{C - A(x - \epsilon)}{B} \right) \qquad m_2 = \left(p[1] + \epsilon,\, \frac{C - A(x + \epsilon)}{B} \right)$$

Además, el algoritmo usa el parámetro *num* para determinar si la intersección es para *Polygon* ($num = 0$) o para un agujero específico *HolesP[i]* de *HolesP* ($num = 1$).

En la Figura 3.7 se muestran dos intersecciones. La primera intersección, $l_1 \cap s_1$, no es posible para *Polygon* debido a que *PointIn(m_2, poly) = false* (línea 17). La segunda intersección, $l_2 \cap s_2$, no es posible para $HolesP[2]$ debido a que *PointIn(m_2, poly) = true* y *AlignedPol(m_2, poly) = false* (línea 20).

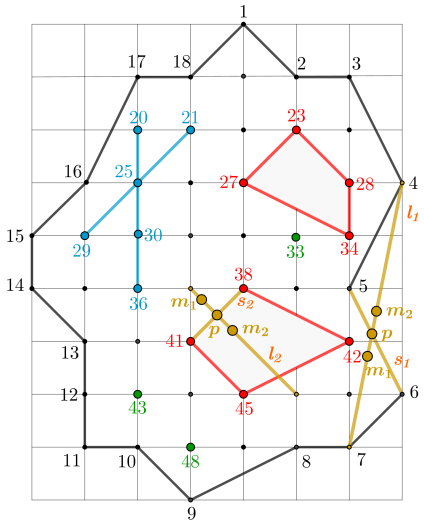


Figura 3.7: Intersección segmento–lado

El coste computacional del Algoritmo 3.7 viene determinado para *Polygon* (línea 16, $num = 0$) en $O(n)$ por la función *PointIn*, y para *HolesP[i]* (línea 19, $num = 1$) en tiempo $O(m^2)$ por las funciones *PointIn* en tiempo $O(m)$ y *AlignedPol* en tiempo $O(m)$.

3.3.4.2. Algoritmo 3.8: IntersectionPol(l)

El Algoritmo 3.8 calcula cuándo es posible la intersección entre un segmento l y los conjuntos *Polygon*, *SegmentsP* y *HolesP*. Si la intersección no es posible, devuelve el valor *true*; de lo contrario, devuelve *false*.

Para facilitar el seguimiento y comprensión del algoritmo, se ha dividido en tres bloques correspondientes a cada uno de los conjuntos mencionados anteriormente. Cada bloque explica la función del algoritmo y analiza el coste computacional asociado. Esta estructura proporciona una visión más completa y clara del algoritmo en su conjunto.

Bloque 1: Polygon (líneas 2–6). En este bloque, primero se calculan los lados de *Polygon* utilizando la función *SidesPol(Polygon)* en tiempo lineal $O(n)$ (línea 2). A continuación, se aplica el Algoritmo 3.7 (línea 4) para determinar si existe una intersección no permitida entre el segmento l y alguno de los lados de *Polygon*. Si es así, el algoritmo termina y devuelve *true* (líneas 5–6). El coste computacional de este bloque es $O(n^2)$ debido al bucle (línea 3) que se ejecuta en $O(n)$ y el Algoritmo 3.7 (línea 4) que tiene un tiempo de ejecución de $O(n)$.

Si todas las intersecciones son posibles (línea 7), se ejecuta el Bloque 2.

Algoritmo 3.7 IntersectionPoly(l, s, poly, num)

Input: l: segmento, s: lado de poly (*Polygon* o *Holes[i]*), num = $\{0, 1\}$

Output: *true* si la intersección entre el segmento l y el lado s no está permitida

```
1  inter ← false;  ε ≈ 0
2  if Intersection(l, s) = true then
3      p ← Intersectionp(l, s)                                   /* Alg.3.6 */
4      if l[1][1] = l[2][1] then
5          m1 ← {l[1][1], p[2] − ε}
6          m2 ← {l[1][1], p[2] + ε}
7      else if l[1][2] = l[2][2] then
8          m1 ← {p[1] − ε, l[1][2]}
9          m2 ← {p[1] + ε, l[1][2]}
10     else
11         A ← l[2][2] − l[1][2]
12         B ← l[1][1] − l[2][1]
13         C ← l[1][1] · l[2][2] − l[2][1] · l[1][2]
14         m1 ← {p[1] − ε, (C − A · (p[1] − ε)) / B}
15         m2 ← {p[1] + ε, (C − A · (p[1] + ε)) / B}
16     if num = 0 then
17         if (PointIn(m1, poly) = false or PointIn(m2, poly) = false) then
18             inter ← true
19     else
20         if (PointIn(m1, poly) = true and AlignedPol(m1, poly) = false) or
             (PointIn(m2, poly) = true and AlignedPol (m2, poly) = false) then
21             inter ← true
22 return inter
```

Bloque 2: SegmentsP (líneas 8–16). Se comprueba si existe una intersección entre el segmento l y alguno de los segmentos de $SegmentsP$ que tenga la forma como la que aparece en la Figura 3.6. Si se encuentra una intersección de este tipo, el algoritmo termina y devuelve *true* (líneas 9, 10 y 16). Dado que $SegmentsP = \{S_1, \ldots, S_t\}$, el coste computacional de este bloque es $O(t)$ debido al bucle de la línea 8.

Si todas las intersecciones son posibles (línea 17), se ejecuta el Bloque 3.

Bloque 3: HolesP (líneas 18–28). Se realiza una comparación del segmento l con los lados de los agujeros $HolesP[1], \ldots, HolesP[h]$, utilizando el mismo proceso que en el bloque 1. Si se encuentra alguna intersección no permitida con alguno de los agujeros, el algoritmo termina y devuelve *true* (línea 25), evitando la ejecución para los agujeros restantes: $i+1, \ldots, h$. El coste computacional de esta primera parte es $O(hm^4)$, ya que se tiene: bucle (línea 18) calculado en tiempo $O(h)$, la función *SidesPol(HolesP[i])* en tiempo $O(m)$, bucle (línea 22) en tiempo $O(m)$ y el Algoritmo 3.7 (línea 23) en tiempo $O(m^2)$.

Además, en el Bloque 3 se considera un escenario especial en el que el segmento l se encuentre completamente contenido dentro de algún agujero. Este caso específico se ilustra en las líneas 26 a 28, y se muestra en la Figura 3.8. Como ejemplo, si se toma el agujero $HolesP[1]$ y el segmento l_1 definido por los puntos $(27, 28)$, se observa que IntersectionPol$(l_1) = true$, debido a que el punto medio se encuentra dentro de H_1 y no está incluido en alguno de sus lados. De manera similar, esto se cumple para los segmentos formados por los puntos: $l_2 = (23, 34)$, $l_3 = (38, 45)$ y $l_4 = (41, 42)$.

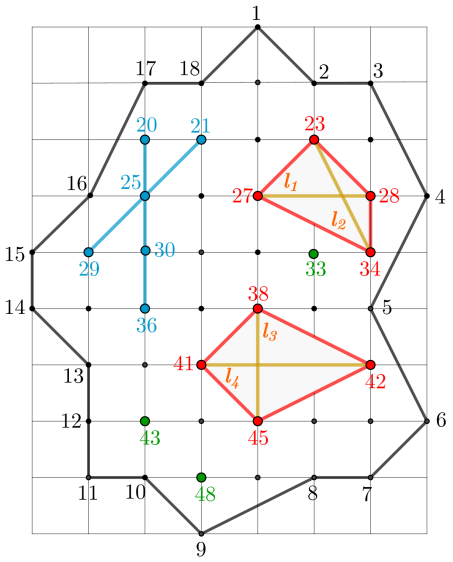


Figura 3.8: Segmento contenido en un agujero

El coste computacional para este escenario es $O(hm^2)$ debido al bucle (línea 18) calculado en tiempo $O(h)$, la función *PointIn (P, poly)* en tiempo $O(m)$ y la función *AlignedPol(P, poly)* en tiempo $O(m)$. Por tanto, el coste computacional total del Bloque 3 es $O\left(Max\left(hm^4, hm^2\right)\right) = O\left(hm^4\right)$.

Al combinar los costes computacionales de los Bloques 1, 2 y 3, se obtiene el coste computacional final del Algoritmo 3.8: $O\left(Max(n^2, t, hm^4)\right) = O\left(n^2\right)$ si se asume que $\#(P) = N \simeq n \gg m$.

Algoritmo 3.8 IntersectionPol(l)

Input: l : segmento

Output: *true* si la intersección del segmento con *Polygon*, *SegmentsP* o *HolesP* no está permitida

```
1  inter ← false
2  lados ← SidesPol(Polygon)
3  for i ← 1 to Length(lados) do
4      inter ← IntersectionPoly(l, lados[i], Polygon, 0)                    /* Alg.3.7 */
5      if inter = true then
6          break
7  if inter = false then
8      for i ← 1 to Length(SegmentsP) do
9          if inter = true then
10             break
11         n ← Length(SegmentsP[i])
12         s ← (Points[SegmentsP[i][1]], Points[SegmentsP[i][n]])
13         if Intersection(l, s) = true then
14             p ← Intersectionp(l, s)                                      /* Alg.3.6 */
15             if (Orientation(s[1], l[1], p) ≠ 0 and Orientation(s[1], l[2], p) ≠ 0 and
                Orientation(s[2], l[1], p) ≠ 0 and Orientation(s[2], l[2], p) ≠ 0) then
16                 inter ← true
17 if inter = false then
18     for i ← 1 to Length(HolesP) do
19         hol ← SidesPol(HolesP[i])
20         if inter = true then
21             break
22         for j ← 1 to Length(HolesP[i]) do
23             int ← IntersectionPoly(l, hol[j], HolesP[i], 1)              /* Alg.3.7 */
24             if int = true then
25                 break
26         m ← Medio(l[0], l[1])
27         if (PointIn(m, HolesP[i]) = true and AlignedPol(m, HolesP[i]) = false then
28             inter ← true)
29 return inter
```

3.3.5. Algoritmo 3.9: Matriz(Points)

Algoritmo 3.9. Matriz(Points)

El Algoritmo 3.9 es el algoritmo principal del Capítulo 3. Toma como valores de entrada los puntos (*Points*) de un polígono reticulado P con r–puntos, t–segmentos y h–agujeros y calcula la matriz de adyacencia.

El funcionamiento es el siguiente: primero se inicializa la matriz de adyacencia con todos sus términos iguales a 0 (línea 1). A continuación, se calculan todos los lados de *Polygon* (línea 2) y mediante dos bucles (líneas 3 y 4), se recorren todos los puntos del conjunto *Points*, con $j = i + 1$, para formar los lados l. Si el lado l pertenece a un lado de *Polygon* (línea 6), se asigna el valor 1 a los elementos de la matriz correspondientes a los puntos p_i y p_j. En caso contrario (línea 8), se usa el Algoritmo 3.8 (línea 9) para determinar si la intersección de l con *Polygon*, *SegmentsP* o *HolesP* está permitida o no. Si es así, se toma el punto medio del lado (línea 10), y si ese punto se encuentra dentro o sobre un lado de *Polygon* (línea 11), se asigna el valor 1 a los elementos correspondientes de la matriz de adyacencia. Finalmente, se realiza la asignación matriz(j, i) = matriz(i, j) (línea 13) debido a que la matriz de adyacencia es simétrica y, hasta este momento, solo se han calculado los valores por encima de la diagonal principal, formando así una matriz triangular superior.

El coste computacional del Algoritmo 3.9 se establece a través de las líneas 3 a 12, con un tiempo de ejecución de $O(n^5)$ debido al bucle (línea 3) que se ejecuta en $O(n)$, el bucle (línea 4) que se ejecuta en $O(n)$, el Algoritmo 3.8 (línea 9) que se ejecuta en $O(n^2)$ y la función *PointIn(P, poly)* (línea 11) que se ejecuta en $O(n)$. Por tanto, el coste computacional del Algoritmo 3.9 es $O(n^5)$.

3.4. Coste computacional: resumen

La Figura 3.9 presenta un esquema de los algoritmos que se han empleado. En este nuevo esquema, se incluye el coste computacional de cada algoritmo con el objetivo de analizar la contribución de cada uno al coste total del Algoritmo 3.9. Esta información nos permite evaluar la complejidad general del algoritmo y comprender cómo podemos optimizar su eficiencia, en caso de ser necesario.

Algoritmo 3.9 Matriz(Points)

Input: Points

Output: Matriz de adyacencia

```
1  Inicializar matriz, matriz(i,j)← 0
2  lados ← SidesPol(Polygon)
3  for i ← 1 to Length(Points)-1 do
4      for j ← i + 1 to Length(Points) do
5          l ← (Points[i], Points[j])
6          if Length(Union(lados, {l})) = Length(lados) then
7              matriz(i,j) ← 1
8          else
9              if IntersectionPol(l) = false then
10                 m ← Medio(Points[i], Points[j])
11                 if PointIn(m, Polygon) = true then
12                     matriz(i,j) ← 1
13 Matriz simétrica, matriz(j,i) ← matriz(i,j)
14 return matriz
```

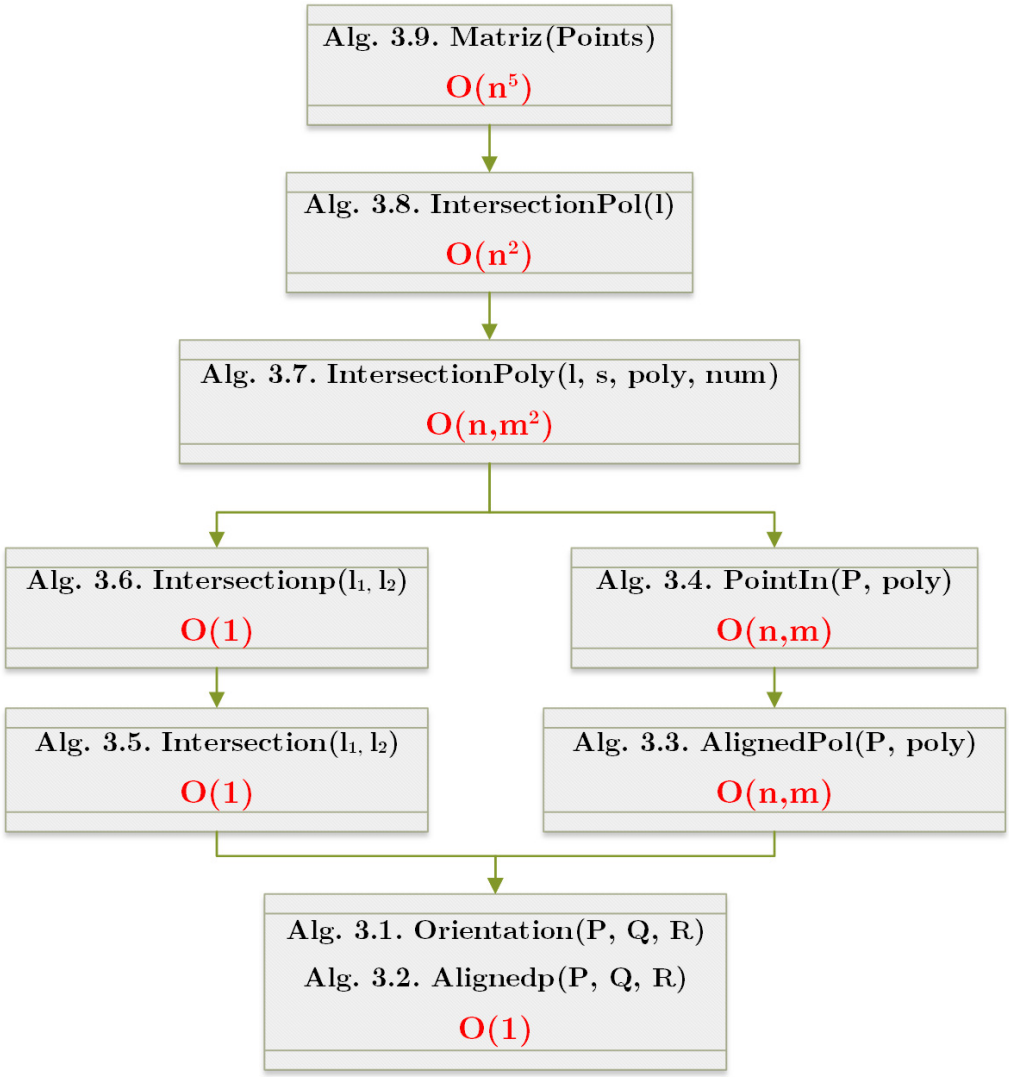


Figura 3.9: Matriz de adyacencia: coste computacional

Capítulo 4

Problemas de inclusión, $Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$

En este capítulo se resuelve el problema de calcular el polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado o en una Región de Interés (ROI) con obstáculos arbitrarios. Se hace referencia a una serie de artículos aceptados, publicados y en proceso de revisión [118–120, 123] y una ponencia presentada en un congreso [121]. Estos recursos son fundamentales para resolver el problema de inclusión $Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$, donde $\mathcal{P}_{sim}$ representa el conjunto de todos los polígonos simples, $\mathcal{P}_k$ el conjunto de todos los polígonos simples de k lados y μ es una función real respecto al área o al perímetro.

[**118**] R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *An algorithm to compute any simple k-gon of a maximum area or perimeter inscribed in a region of interest*, SIAM Journal on Imaging Sciences, 15 (2022), pp. 1808–1832.

[**119**] R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *An efficient method for obtaining the maximum k-gon in an arbitrary polygon with obstacles*, Pattern Recognition (en revisión), (2023).

[**120**] R. Molano, M. Ávila, J.C. Sancho, P.G. Rodríguez, and A. Caro, *Obtaining the user-defined polygons inside a closed contour with holes*, The Visual Computer (aceptado), (2023).

[**121**] R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *Post-processing of closed contours to obtain inscribed k-sided polygons*, in International Conference on Computing, Intelligence and Data Analytics, 2023.

[123] R. Molano, P. G. Rodríguez, A. Caro, and M. L. Durán, *Finding the largest area rectangle of arbitrary orientation in a closed contour*, Applied Mathematics and Computation, 218 (2012), pp. 9866–9874.

El algoritmo propuesto proporciona soluciones en función del valor k elegido por el usuario. Es decir, el usuario tiene la capacidad de definir qué polígono simple de k lados es relevante para sus investigaciones o propósito en general. Esto permite una mayor flexibilidad del algoritmo a diferentes necesidades y aplicaciones.

La motivación detrás de la creación de un algoritmo que resuelva todos los casos simultáneamente, a partir de la elección del valor k, para el cálculo de polígonos simples es una consecuencia de la Sección *2.1.7. Resumen bibliográfico sobre polígonos*. En dicha sección, se llegó a tres ideas fundamentales: en primer lugar, se observó que los algoritmos existentes se limitan a tratar con problemas particulares relacionados con ciertos tipos de polígonos, como triángulos, rectángulos, paralelogramos o cuadriláteros; en segundo lugar, se observó la escasez de bibliografía disponible sobre el cálculo del mayor perímetro o la determinación de polígonos con obstáculos arbitrarios; y por último, se comprobó que la mayoría de las soluciones propuestas se centran exclusivamente en polígonos convexos, ignorando la complejidad de los polígonos simples.

El algoritmo desarrollado ofrece soluciones al problema planteado en el Capítulo 3 (Figura 4.1). Además, permite cumplir satisfactoriamente con el objetivo parcial 2, junto con el objetivo parcial 1, ya que demuestra cómo, al realizar particiones cada vez más finas sobre una Región de Interés (ROI) o contorno cerrado (C), la solución generada a partir de un tamaño de partición dado converge de forma natural hacia la solución real.

Objetivo 1. Calcular el rectángulo de mayor área y orientación arbitraria contenido en una Región de Interés (ROI).

Objetivo 2. Dado un valor k fijo, calcular el polígono simple de k lados de mayor área o perímetro contenido en una ROI con obstáculos arbitrarios, como puntos, segmentos y agujeros.

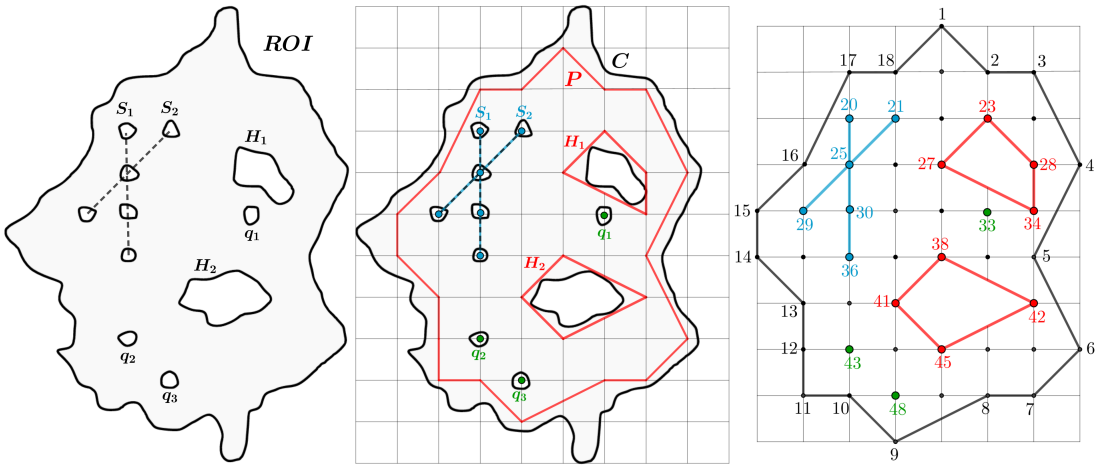


Figura 4.1: Región de interés con puntos, segmentos y agujeros, Polígono reticulado P

4.1. Descripción de los algoritmos utilizados (versión iterativa y recursiva)

A continuación, se presenta un esquema (Figura 4.2) de los algoritmos empleados en el Capítulo 4. Para resolver el problema planteado se han desarrollado dos variantes: una versión iterativa y una versión recursiva.

Para la versión iterativa (Figura 4.2a), el algoritmo principal denominado **Algoritmo 4.10. SolutionIncI(N, distancia, matriz, fun)** (Sección 4.3), se usa para calcular el polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado P con obstáculos arbitrarios. Este algoritmo consta de cuatro parámetros. El primer parámetro, N, representa el número de puntos del polígono reticulado. El segundo parámetro, k = distancia +1, define el número de lados del polígono que se desea calcular. La matriz de adyacencia se utiliza como tercer parámetro, y el cuarto parámetro, denominado fun, determina si se desea calcular el área ($fun = 0$) o el perímetro ($fun = 1$).

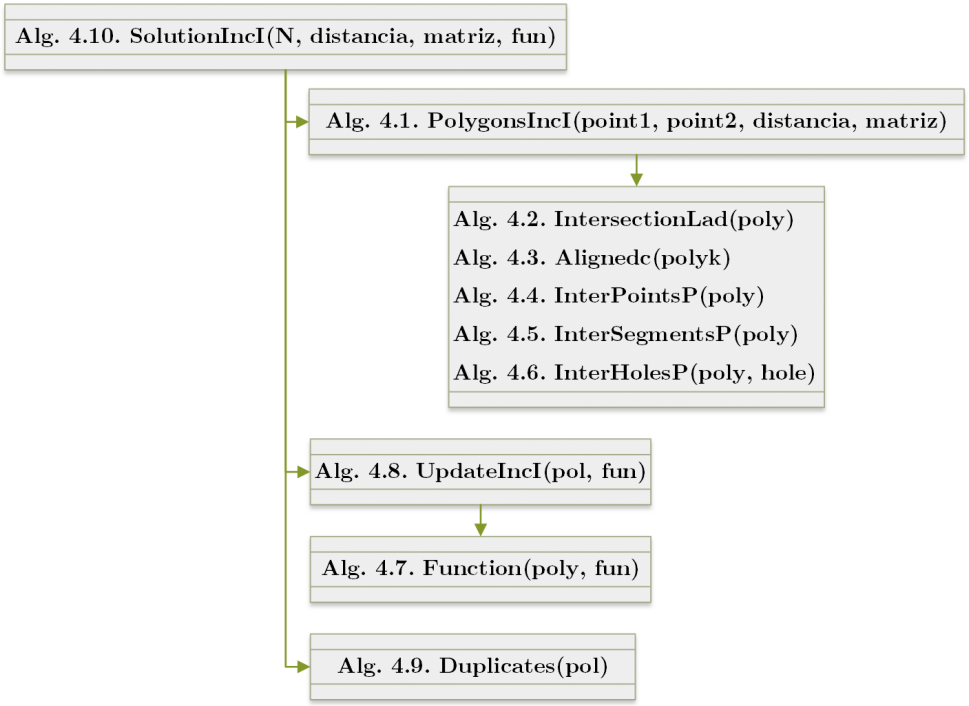
Además, se encuentra el **Algoritmo 4.1. PolygonsIncI(point1, point2, distancia, matriz)** (Sección 4.2), encargado de calcular el polígono simple de k lados contenido en un polígono reticulado entre dos puntos arbitrarios, *point1* y *point2*. Se incluyen otros algoritmos: el *Algoritmo 4.8. UpdateIncI(pol, fun)* (Sección 4.3.2), responsable de actualizar las soluciones, es decir, calcular siempre la mejor solución respecto al área o al perímetro; y por último, el *Algoritmo 4.9. Duplicates(pol)* (Sección 4.3.3), diseñado para eliminar soluciones repetidas.

La versión recursiva (Figura 4.2b) se basa en el **Algoritmo 4.18. SolutionIncR(N, distancia, matriz, fun)** (Sección 4.6) y consta de los mismos cuatro parámetros que la versión iterativa. Para su ejecución necesita tres algoritmos adicionales: el *Algoritmo 4.19. StartIncR(point1, point2, distancia, matriz, fun)*, que inicializa el proceso recursivo; el **Algoritmo 4.16. PolygonsIncR(point1, point2, first, iter, lados, distancia, matriz, fun)**, que calcula el polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado entre dos puntos arbitrarios, *point1* y *point2*; y por último, el *Algoritmo 4.17. UpdateIncR(poly, fun, distancia)*, que actualiza las soluciones.

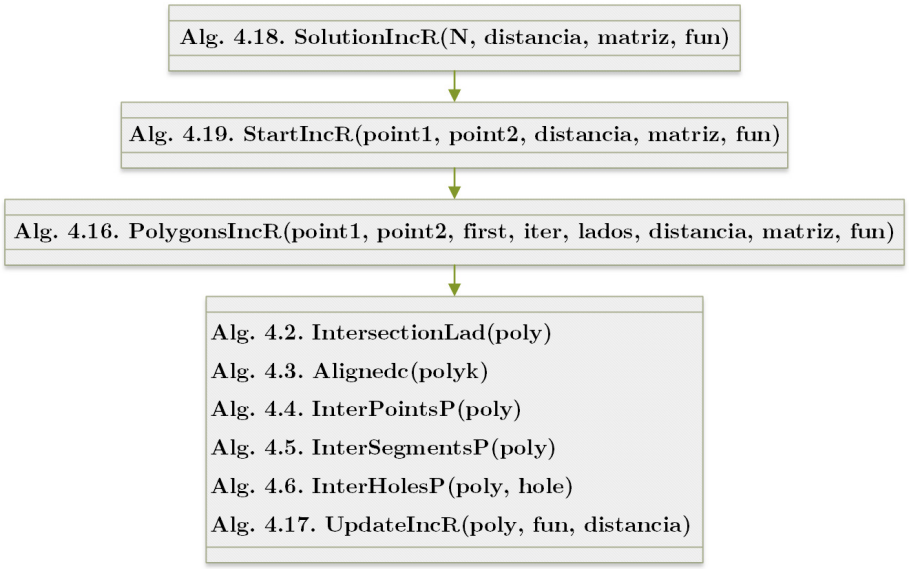
Además de estos algoritmos, en este capítulo se recurre a conjuntos específicos para calcular el polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado P con r–puntos, t–segmentos y h–agujeros. Dichos conjuntos son los siguientes:

$$\left[\begin{array}{l} Points = P = P_0 - \bigcup_{i=1}^{h}(\imath H_i) \\ Polygon = \partial P_0 \\ PointsP = \{q_1, \ldots, q_r\} \\ SegmentsP = \{S_1, \ldots, S_t\}, \text{ donde } \#(S_i) = s_i,\ 1 \leq i \leq t,\ s = max\{s_1, \ldots, s_t\} \\ HolesP = \{H_1, \ldots, H_h\}, \text{ donde: } \#(H_i) = m_i,\ 1 \leq i \leq h,\ m = max\{m_1, \ldots, m_h\} \end{array}\right.$$

El desarrollo de la versión iterativa comienza en la Sección 4.2, mientras que en la Sección 4.6 se introduce la versión recursiva. Finalmente, en la Sección 4.7 se detalla un algoritmo general que permite calcular cualquier tipo de solución en función de varios parámetros, tanto en la versión iterativa como en la recursiva.



(a) Versión iterativa



(b) Versión recursiva

Figura 4.2: Problemas de inclusión: esquema general de los algoritmos utilizados (versión iterativa y recursiva)

4.2. Cálculo entre dos puntos del polígono simple de k lados contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 4.1)

El desarrollo del Algoritmo 4.1 se divide en cinco subsecciones (4.2.1–4.2.5), que serán examinadas en detalle de manera individual más adelante. Cada una de estas subsecciones está diseñada para llevar a cabo un paso específico para la resolución del problema en cuestión.

Algoritmo 4.2. IntersectionLad(poly)
Algoritmo 4.3. Alignedc(polyk)
Algoritmo 4.4. InterPointsP(poly)
Algoritmo 4.5. InterSegmentsP(poly)
Algoritmo 4.6. InterHolesP(poly, hole)

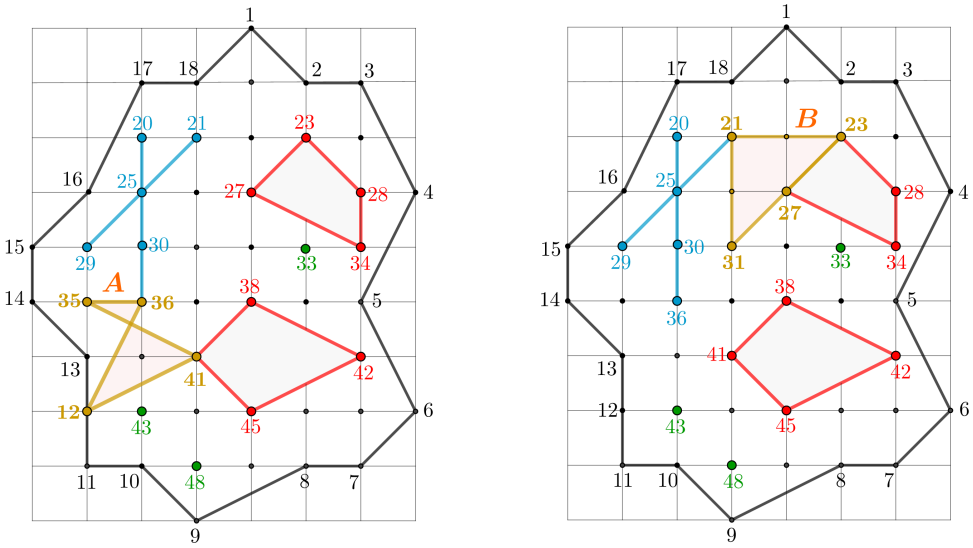
Con el objetivo de facilitar la comprensión del Algoritmo 4.1, se ha estructurado en cuatro bloques, en los cuales se explica la función sobre el algoritmo y se evalúa el coste computacional asociado.

Bloque 1: Aristas entre dos puntos (líneas 2–13). Se calculan todas las aristas que se encuentran a una distancia específica entre *point1* y *point2*. Primero, el algoritmo considera si los dos puntos están conectados (línea 2) mediante el Alg. 3.9. A continuación, se calculan secuencialmente todas las aristas que están a distancia 1 de *point1*, almacenando las soluciones en cada iteración hasta lograr la distancia deseada. El coste computacional viene determinado por los siguientes bucles: línea 3 calculada en $O(k)$, donde $k = distancia + 1$; línea 5 calculada en $O(n^2)$, ya que el número máximo de aristas es igual al número combinatorio $\binom{N}{2} = \frac{N \cdot (N-1)}{2} \simeq N^2 \simeq n^2$; línea 8 en tiempo $O(n)$, ya que la matriz de adyacencia es una matriz cuadrada simétrica de orden $N \simeq n$. Por tanto, el coste computacional del Bloque 1 es $O(n^3 k)$.

Bloque 2: Algoritmos 4.2–4.3 (líneas 14–17). Entre todas las soluciones presentes en *temp1* (línea 13), se seleccionan aquellas en las que el punto final coincide con *point2* (línea 16). Además, se aplican los algoritmos *IntersectionLad(poly)* y *Alignedc(polyk)*. El primero, determina si dos lados de un polígono intersectan, con un coste de complejidad $O(k^2)$, mientras que el segundo algoritmo, con un coste de complejidad $O(k)$, devuelve 0 si tres puntos consecutivos de *polyk* están alineados.

En la Figura 4.3a, se puede observar que el polígono $A = (35, 36, 12, 41)$ no puede formar un cuadrilátero. Del mismo modo, en la Figura 4.3b, se muestra que $B = (21, 23, 27, 31)$ tampoco puede ser un cuadrilátero, debido a que los puntos 23, 27 y 31 están alineados. En realidad, es un triángulo.

El coste computacional está definido por el bucle en la línea 15, con una complejidad de $O(n^2)$, y los algoritmos mencionadas anteriormente, obteniendo por tanto un coste computacional para el Bloque 2 de $O(n^2k^3)$.



(a) Intersección de los lados de un polígono

(b) Puntos consecutivos alineados

Figura 4.3: Algoritmo 4.2 IntersectionLad(poly), Algoritmo 4.3 Alignedc(polyk)

Bloque 3: Algoritmos 4.4–4.5 (líneas 18–20). Elimina todas las soluciones que no cumplan con las condiciones establecidas por los algoritmos *InterPointsP(poly)* y *InterSegmentsP(poly)*. El primero, devuelve *false* si todos los puntos de *PointsP* se encuentran fuera de un polígono de *temp2* o sobre alguno de los lados, mientras que el segundo, también devuelve *false* si no se produce ninguna intersección entre cualquiera de los segmentos de *SegmentsP* con un polígono de *temp2*. Estos algoritmos tienen un coste computacional de $O(k^2r)$ y $O(k^2st)$, respectivamente.

El coste computacional está establecido por el bucle de la línea 18, con una complejidad de $O(n^2)$, y los algoritmos anteriores. Por tanto, el coste total es de $O(n^2k^4rst)$.

Bloque 4: Algoritmo 4.6 (líneas 21–27). Para finalizar, el Bloque 4 se ocupa de garantizar que todas las soluciones temporalmente elegidas de *temp3* no corten a algún agujero de *HolesP*.

En la Figura 4.4, se presentan dos polígonos como ejemplo. El primero, $A = (22, 28, 26)$, no puede formar parte de la solución final ya que intersecta con el agujero H_1. Del mismo modo, el polígono $B = (37, 39, 6, 48)$ no puede ser una solución válida ya que en su interior se encuentra el agujero H_2.

El coste computacional del Bloque 4 está condicionado por los bucles presentes en las líneas 21 y 23, con una complejidad de $O(n^2)$ y $O(h)$, respectivamente. Además, también interviene el algoritmo *InterHolesP(poly, hole)* con coste computacional $O(k^2m)$. Por tanto, el coste computacional total del Bloque 4 es $O(n^2k^2hm)$.

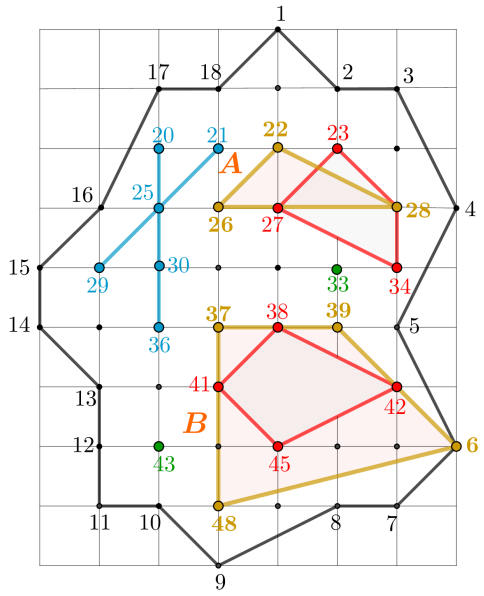


Figura 4.4: Algoritmo 4.6 InterHolesP(poly, hole)

Al combinar los costes computacionales de los Bloques 1 al 4 se obtiene el coste computacional final del Algoritmo 4.1: $O(Max(n^3k, n^2k^3, n^2k^4rst, n^2k^2hm)) = O(n^3k)$ si se asume que $\#(P) = N \simeq n \gg k$.

4.2. Cálculo entre dos puntos del polígono simple de k lados contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 4.1)

Algoritmo 4.1 PolygonsIncI(point1, point2, distancia, matriz)

Input: point1, point2 $\in$ *Points*, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia

Output: Polígonos simples de k lados entre *point1* y *point2* para una determinada *distancia*

```
1  temp1 ← {{point1}};  temp3, polig ← ∅
2  if matriz(point1, point2) = 1 then
3      for i ← 1 to distancia do
4          temp2 ← ∅
5          for j ← 1 to Length(temp1) do
6              last ← temp1[j][i]
7              if last ≠ point2 then
8                  for k ← 1 to N do
9                      if (matriz(last,k) = 1 and Length(Union(temp1[j],k)) ≠
                         Length(temp1[j])) then
10                         Insert(temp2, Insert(temp1[j],k))
11                         Delete(temp1[j], temp1[j][Length(temp1[j])])
12         temp1 ← ∅
13         temp1 ← temp2
14     temp2 ← ∅
15     for i ← 1 to Length(temp1) do
16         if (temp1[i][distancia + 1] = point2 and IntersectionLad(temp1[i]) = false and
             Alignedc(temp1[i]) ≠ 0) then
17             Insert(temp2, temp1[i])
18     for i ← 1 to Length(temp2) do
19         if (InterPointsP(temp2[i]) = false and InterSegmentsP(temp2[i]) = false) then
20             Insert(temp3, temp2[i])
21     for i ← 1 to Length(temp3) do
22         h ← 0
23         for j ← 1 to Length(HolesP) do
24             if InterHolesP(temp3[i], HolesP[j]) = false then
25                 h ← h+1
26         if h = Length(HolesP) then
27             Insert(polig, temp3[i])
28 return polig
```

A continuación, se presenta una descripción de cada uno de los algoritmos utilizados en la resolución del Algoritmo 4.1, que van desde el Algoritmo 4.2 hasta el 4.6.

4.2.1. Intersección de los lados de un polígono (Algoritmo 4.2)

El Algoritmo 4.2 comprueba si dos lados de un polígono intersectan. Comienza calculando los lados del polígono de k lados (*poly*) usando la función *SidesPol(poly)* (línea 2). Luego, realiza una comprobación para determinar si dos lados cualesquiera de *poly* intersectan o no (línea 7). Si se encuentra una intersección, el algoritmo finaliza y devuelve el valor *true*.

El coste computacional del Algoritmo 4.2 es $O(k^2)$, ya que para su ejecución emplea los bucles de las líneas 3 y 6, ambos con complejidad $O(k)$ y la función *Intersection(l_1, l_2)* (Algoritmo 3.5) calculada con coste computacional constante.

Algoritmo 4.2 IntersectionLad(poly)

Input: *poly*: polígono de k lados

Output: *false* si no existen intersecciones entre los lados de *poly*

```
1  inter ← false
2  lados ← SidesPol(poly)
3  for i ← 1 to Length(lados)-1 do
4      if inter = true then
5          break
6      for j ← i + 1 to Length(lados) do
7          inter = Intersection(lados[i], lados[j])        /* Alg.3.5 */
8          if inter = true then
9              break
10 return inter
```

4.2.2. Puntos consecutivos alineados (Algoritmo 4.3)

Este algoritmo determina si las k aristas formadas entre dos puntos pueden o no formar un polígono de k lados. Emplea el Algoritmo 3.1, *Orientation(P, Q, R)*, para comprobar si tres puntos consecutivos de *polyk* están en algún momento alineados o no. Si lo están, el algoritmo finaliza y esas k aristas no formarían un polígono de k lados.

El coste computacional total es $O(k)$, debido al bucle de la línea 4 en $O(k)$ y el Algoritmo 3.1, que tiene un coste computacional constante.

Algoritmo 4.3 Alignedc(polyk)

Input: *polyk*: k aristas entre dos puntos

Output: 0 si tres puntos consecutivos de *polyk* están alineados

```
1 polk ← polyk
2 Insert(polk, polyk[1])
3 Insert(polk, polyk[2])
4 for i ← 1 to Length(polk)-2 do
5     alinc ← Orientation(Points[polk[i]], Points[polk[i+2]], Points[polk[i+1]]) /* Alg.3.1 */
6     if alinc = 0 then
7         break
8 return alinc
```

4.2.3. Posición puntos–polígono (Algoritmo 4.4)

El Algoritmo 4.4 se usa para comprobar si todos los puntos de $PointsP$ están situados fuera del polígono *poly* o sobre alguno de sus lados. Utiliza los algoritmos *AlignedPol(P, poly)* (Algoritmo 3.3) y *PointIn(P, poly)* (Algoritmo 3.4), que determinan si algún punto de $PointsP$ se encuentra dentro de *poly*, pero no sobre uno de sus lados. En caso de que se cumpla esta condición, el algoritmo finaliza y devuelve el valor *true*.

El coste computacional total del algoritmo es $O(k^2 r)$. Esta complejidad se debe al bucle de la línea 2, ejecutado en $O(r)$, y los algoritmos anteriores con costes $O(k)$.

Algoritmo 4.4 InterPointsP(poly)

Input: *poly*: polígono de k lados

Output: *false*, si todos los puntos de $PointsP$ están fuera de *poly* o sobre uno de sus lados

```
1 ppoint ← false
2 for i ← 1 to Length(PointsP) do
3     if PointIn(Points[PointsP[i]], poly) = true then
4         if AlignedPol(Points[PointsP[i]], poly) = false then
5             ppoint ← true
6             break
7 return ppoint
```

4.2.4. Posición segmentos–polígono (Algoritmo 4.5)

El Algoritmo 4.5 determina si existe alguna intersección entre los segmentos de *SegmentsP* y el polígono *poly*. Para lograr esto, el algoritmo examina cada segmento (línea 2) y verifica si el punto medio de alguna sección de ese segmento se encuentra exclusivamente en el interior de *poly* (líneas 7–9). Si se cumple esta condición para algún segmento, el algoritmo termina y devuelve *true* (línea 10).

El coste computacional total es $O(k^2st)$ debido a los bucles de las líneas 2 y 7 ejecutados en $O(t)$ y $O(s)$, respectivamente, y los algoritmos *PointIn(P, poly)* (Algoritmo 3.4) y *AlignedPol(P, poly)* (Algoritmo 3.3) computados en $O(k)$.

Algoritmo 4.5 InterSegmentsP(poly)

Input: *poly*: polígono de k lados

Output: *false*, si todos los segmentos de *SegmentsP* están fuera de *poly* o sobre uno de sus lados

```
1  seg ← false;
2  for i ← 1 to Length(SegmentsP) do
3      if seg = true then
4          break
5      Segm ← SegmentsP[i]
6      Insert(Segm, SegmentsP[1])
7      for j ← 1 to Length(SegmP[i])-1 do
8          m ← Medio(Points[SegmP[i][j]], Points[SegmP[i][j+1]])
9          if (PointIn(m, poly) = true and AlignedPol(m, poly) = false) then
10             seg ← true
11             break
12 return seg
```

En la Figura 4.5, se ha construido un polígono A con los vértices (30, 31, 36, 35). Al analizar la figura, podemos observar que el polígono A intersecta al segmento S_1. Al evaluar la función *InterSegmentsP(A)*, se obtiene el valor *true*, ya que el punto medio m está en el interior del polígono A pero no en su frontera.

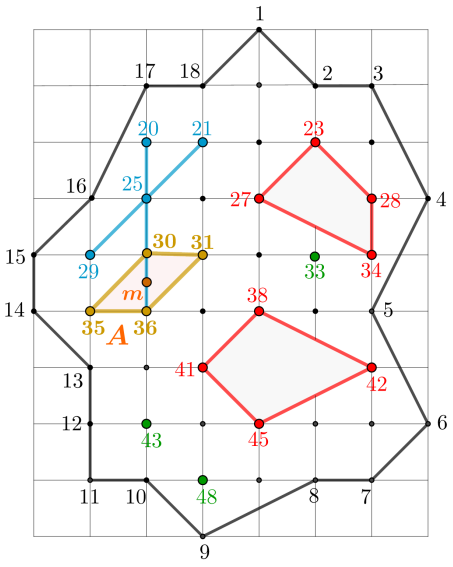


Figura 4.5: Algoritmo 4.5 InterSegmentsP(poly)

4.2.5. Posición agujeros–polígono (Algoritmo 4.6)

El Algoritmo 4.6 comprueba si un agujero *hole* está contenido o intersecta con el polígono *poly*. Para ello, calcula el punto medio de cada lado y determina si ese valor está dentro de *poly*, pero no en su frontera. Si es así, el algoritmo finaliza y devuelve *true*.

El coste computacional total es $O(k^2m)$ por el bucle de la línea 4 ejecutado en $O(m)$ y los algoritmos *PointIn(P, poly)* (Algoritmo 3.4) y *AlignedPol(P, poly)* (Algoritmo 3.3) computados en $O(k)$ cada una de ellas.

Algoritmo 4.6 InterHolesP(poly, hole)

Input: *poly*: polígono de k lados, *hole*: agujero de $HolesP$
Output: *false* si *hole* no está contenido o no intersecta con *poly*

```
1 int ← false
2 hol ← hole
3 Insert(hol, hole[1])
4 for i ← 1 to Length(hol)-1 do
5 |   m ← Medio(Points[hol[i]], Points[hol[i+1]])
6 |   if (PointIn(m, poly) = true and AlignedPol(m, poly) = false) then
7 |   |   int ← true
8 |   |   break
9 return int
```

4.3. Cálculo del polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 4.10)

Se ha demostrado mediante el Algoritmo 4.1 cómo calcular entre dos puntos el polígono simple de k lados contenido en un polígono reticulado con obstáculos arbitrarios. Ahora, el objetivo final es claro: calcular dicho polígono simple con el área o perímetro máximo posible. Para lograr esto, se desarrollará un algoritmo, denominado Algoritmo 4.10. Este proceso se llevará a cabo empleando los algoritmos que se detallan a continuación, cada uno con un propósito bien definido.

Algoritmo 4.1. PolygonsIncI(point1, point2, distancia, matriz)
Algoritmo 4.7. Function(poly, fun)
Algoritmo 4.8. UpdateIncI(pol, fun)
Algoritmo 4.9. Duplicates(pol)

4.3.1. Área o perímetro de un polígono reticulado (Algoritmo 4.7)

El Algoritmo 4.7, con coste computacional $O(k)$, se emplea para calcular el área [29, 205] o el perímetro de un polígono simple de k lados, dependiendo del valor del parámetro *fun*. Si *fun* es igual a 0, se calcula el área; si *fun* es igual a 1, se calcula el perímetro.

4.3.2. Actualización de soluciones (Algoritmo 4.8)

Mediante el Algoritmo 4.8 se calcula el polígono o conjunto de polígonos con el mayor área o perímetro posible, según el valor *fun*. El coste computacional es $O(n^2k)$ debido al bucle de la línea 2 computado en $O(n^2)$ y al Algoritmo 4.7 en la línea 3 con una complejidad de $O(k)$.

Algoritmo 4.7 Function(poly, fun)

Input: *poly*: polígono de k lados, *fun*: área (fun = 0) o perímetro (fun = 1)
Output: Área o perímetro de *poly*

```
1  polig ← poly
2  Insert(polig, poly[1])
3  switch fun do
4      case 0 do
5          for i ← 1 to Length(polig)-1 do
6              f ← f + Det(Points[polig[i]], Points[polig[i+1]])
7          f ← Abs(0,5 · f)
8      case 1 do
9          for i ← 1 to Length(polig)-1 do
10             a ← Points[poly[i]]; b ← Points[poly[i + 1]]
11             f ← f + √((a[1] − b[1])² + (a[2] − b[2])²)
12 return f
```

Algoritmo 4.8 UpdateIncI(pol, fun)

Input: *pol*: polígonos de k lados, *fun*: área (fun = 0) o perímetro (fun = 1)
Output: Polígonos con mayor área o perímetro

```
1  update ← ∅;  max ← 0
2  for i ← 1 to Length(pol) do
3      f ← Function(pol[i], fun)                          /* Alg.4.7 */
4      if f > max then
5          max ← f;  update ← pol[i]
6      else
7          Insert(update, pol[i])
8  return update
```

4.3.3. Eliminación de soluciones duplicadas (Algoritmo 4.9)

La Figura 4.6 muestra la solución del cuadrilátero de mayor área, $A = (5, 38, 11, 21)$. Otras soluciones podrían haber sido: (38, 11, 21, 5) y (11, 21, 5, 38). Así, para cada solución de un polígono simple de k lados, existen $2k$ soluciones con el mismo área.

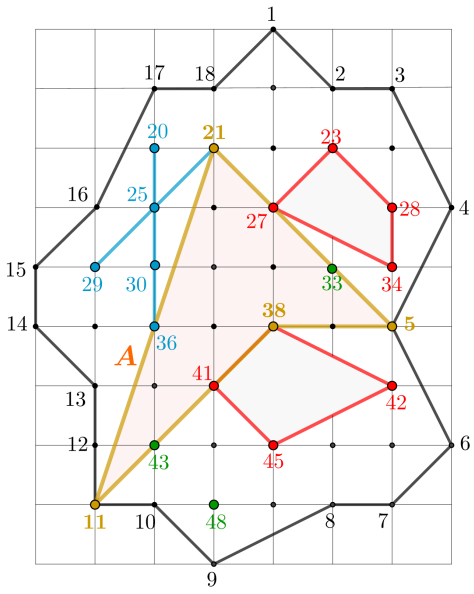


Figura 4.6: Algoritmo 4.9 Duplicates(pol)

El Algoritmo 4.9 tiene la función de eliminar soluciones repetidas. El coste total de este algoritmo es $O(n^4 k \log k)$. Este coste se deriva de dos bucles en la línea 2 y la línea 4, cada uno de ellos con una complejidad de $O(n^2)$, y un algoritmo de ordenación en la línea 5, que puede ser ejecutado en un tiempo de $O(k \log k)$ [48].

4.3.4. Algoritmo 4.10. SolutionIncI(N, distancia, matriz, fun)

El Algoritmo 4.10 es el algoritmo principal. En primer lugar, se calculan todos los polígonos simples de k lados (líneas 1 a 5). Este cálculo tiene un coste computacional de $O(n^5 k)$, debido a los bucles de las líneas 2 y 3, que se ejecutan en $O(n^2)$, y al Algoritmo 4.1, que se efectúa en $O(n^3 k)$. Posteriormente, se recurre a los algoritmos *UpdateIncI(pol, fun)* y *Duplicates(pol)* para obtener la solución requerida. Por tanto, el coste computacional del Algoritmo 4.10 es $O(n^5 k)$, ya que $Max(n^5 k, n^2 k, n^4 k \log k) = n^5 k$.

Algoritmo 4.9 Duplicates(pol)

Input: *pol*: polígonos de k lados

Output: Polígonos no duplicados

```
1  aux1, aux2, dupl ← ∅
2  for i ← 1 to Length(pol)-1 do
3      Insert(aux2, i)
4      for j ← i + 1 to Length(pol) do
5          if Sort(pol[i]) = Sort(pol[j]) then
6              Insert(aux1, j)
7  Insert(aux2, Length(pol))
8  Sort(DeleteDuplicates(aux1))
9  aux3 ← Complement(aux2, aux1)
10 for i ← 1 to Length(aux3) do
11     Insert(dupl, pol[aux3[i]])
12 return dupl
```

Algoritmo 4.10 SolutionIncI(N, distancia, matriz, fun)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1)

Output: Polígonos simples de k lados de mayor área o perímetro contenido en P

```
1  polig, solution ← ∅
2  for i ← 1 to N-1 do
3      for j ← i + 1 to N do
4          if PolygonsIncI(i, j, distancia, matriz) ≠ ∅ then
5              Insert(polig, PolygonsIncI(i, j, distancia, matriz))
6  solution ← Duplicates(UpdateIncI(polig, fun))
7  return solution
```

4.4. Coste computacional: resumen (versión iterativa)

En la Figura 4.7, se muestra un análisis detallado del coste computacional de todos los algoritmos que se han desarrollado en las secciones 4.2 y 4.3.

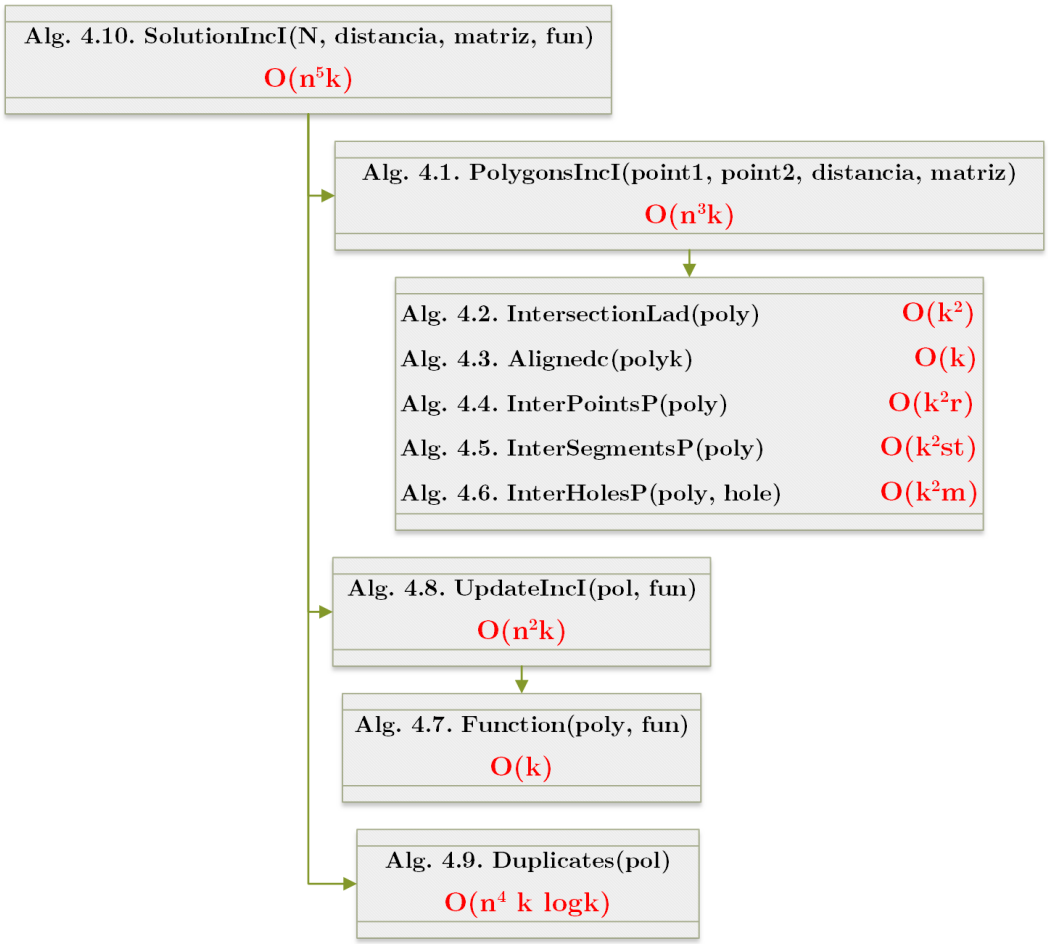


Figura 4.7: Problemas de inclusión: coste computacional (versión iterativa)

4.5. Cálculo del rectángulo y polígono convexo de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios

En esta sección, se pone a prueba la flexibilidad del Algoritmo 4.10, al poder calcular una amplia variedad de soluciones con cambios mínimos en el código. Esto constituye un avance para los investigadores, ya que les permite explorar nuevas posibilidades en el cálculo de formas geométricas. Ahora, en lugar de estar restringidos exclusivamente al cálculo de polígonos simples, se podrán analizar problemas relacionados con rectángulos y polígonos convexos.

4.5.1. Rectángulo de mayor área o perímetro

El Algoritmo 4.11 verifica si el polígono *poly* es un rectángulo. Calcula el ángulo formado por tres vértices consecutivos. Si alguno de los ángulos es distinto de $\pi/2$, el algoritmo finaliza y devuelve false. El coste computacional es constante.

El Algoritmo 4.12 calcula el rectángulo de mayor área o perímetro contenido en un polígono reticulado P con obstáculos arbitrarios. En primer lugar, el algoritmo calcula todos los cuadriláteros (líneas 2 a 5) y luego selecciona aquellos que son rectángulos mediante el algoritmo anterior (línea 7). Para obtener la solución deseada, se utilizan los algoritmos *UpdateIncI(pol, fun)* y *Duplicates(pol)*. El coste computacional del Algoritmo 4.12 es $O(n^5)$.

Algoritmo 4.11 Rect(poly)

Input: *poly*: cuadrilátero

Output: *true* si *poly* es un rectángulo

```
1  rect ← true
2  polig ← poly
3  Insert(polig, poly[1])
4  Insert(polig, poly[2])
5  for i ← 2 to Length(polig)-1 do
6      ang ← Angle(Points[polig[i+1]] - Points[polig[i]], Points[polig[i]] - Points[polig[i-1]])
7      if Abs(ang) ≠ π/2 then
8          rect ← false
9          break
10 return rect
```

Algoritmo 4.12 RectangleIncI(N, matriz, fun)

Input: N: número de puntos de P, matriz: matriz de adyacencia,
fun: área (fun = 0) o perímetro (fun = 1)
Output: Rectángulo de mayor área o perímetro contenido en P

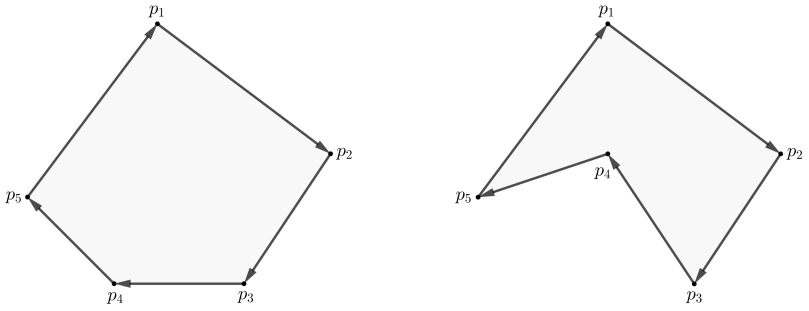
```
1  polig, pol, solution ← ∅
2  for i ← 1 to N-1 do
3      for j ← i + 1 to N do
4          if PolygonsIncI(i, j, 3, matriz) ≠ ∅ then
5              Insert(polig, PolygonsIncI(i, j, 3, matriz))
6  for i ← 1 to Length(polig) do
7      if Rect(polig[i]) = true then
8          Insert(pol, polig[i])
9  solution ← Duplicates(UpdateIncI(pol, fun))
10 return solution
```

4.5.2. Polígono convexo de k lados de mayor área o perímetro

Para resolver el cálculo del polígono convexo de k lados, se ha dividido el problema en tres algoritmos con el objetivo de proporcionar una explicación más clara y comprensible.

□ **Algoritmo 4.13. Convex(poly).** Se usa para determinar si un polígono (*poly*) es convexo. Para ello, recorre todos los vértices del polígono y calcula la orientación (línea 7, Algoritmo 3.1) formada por tres puntos consecutivos. Si ocurre que la orientación actual de los tres puntos (*act*) tiene signo distinto a la orientación anterior (*ant*), $ant \cdot act < 0$, eso indica que el polígono no es convexo. En ese caso, el algoritmo finaliza y devuelve el valor *false*. El coste computacional del algoritmo es $O(k)$ debido al bucle en la línea 6.

En la Figura 4.8, se presentan dos polígonos, uno convexo y otro cóncavo. Si se recorren los vértices de los polígonos en sentido horario, en el primer caso (Figura 4.8a), la orientación de todos los puntos es positiva. Por otro lado, en el segundo caso (Figura 4.8b), la orientación cambia de sentido. Específicamente, Orientation(p_2, p_3, p_4) > 0, mientras que Orientation(p_3, p_4, p_5) < 0.



(a) Polígono convexo (b) Polígono cóncavo

Figura 4.8: Algoritmo 4.13 Convex(poly)

Algoritmo 4.13 Convex(poly)

Input: *poly*: polígono de k lados

Output: *true* si *poly* es un polígono convexo

```
1  convex ← true
2  ant, act ← 0
3  polig ← poly
4  Insert(polig, poly[1])
5  Insert(polig, poly[2])
6  for i ← 2 to Length(polig)-1 do
7      act ← Orientation(Points[polig[i-1]], Points[polig[i]], Points[polig[i+1]])
8      if act ≠ 0 then
9          if ant · act < 0 then
10             convex ← false
11             break
12         else
13             ant ← act
14 return convex
```

□ **Algoritmo 4.14. PolygonsConvexIncI(point1, point2, distancia, matriz).** Calcula entre dos puntos los polígonos convexos de k lados. Se obtiene al realizar una modificación del Algoritmo 4.1, al incorporar la función *Convex(poly)*. A continuación, utiliza los mismos procedimientos que en el algoritmo anterior. El coste computacional es equivalente al del Algoritmo 4.1, es decir, $O(n^3k)$.

Algoritmo 4.14 PolygonsConvexIncI(point1, point2, distancia, matriz)

Input: point1, point2 $\in Points$, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia

Output: Polígonos convexos de k lados entre *point1* y *point2* para una determinada distancia

```
temp1 ← {point1};  temp3, polig ← ∅
if matriz(point1, point2) = 1 then
    ...
    ...
    temp2 ← ∅
    for i ← 1 to Length(temp1) do
        if Convex(temp1[i]) = true then
            Insert(temp2, temp1[i])
    ...
    ...
return polig
```

□ **Algoritmo 4.15. SolutionConvexIncI(N, distancia, matriz, fun).** Calcula el polígono convexo de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios. El funcionamiento del algoritmo es similar al Algoritmo 4.10 utilizado para calcular polígonos simples, pero en este caso se emplea el Algoritmo 4.14 en lugar del Algoritmo 4.1. El coste computacional del algoritmo es el mismo que el del Algoritmo 4.10, $O(n^5k)$.

Algoritmo 4.15 SolutionConvexIncI(N, distancia, matriz, fun)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1)

Output: Polígonos convexos de k lados de mayor área o perímetro contenido en P

```
1 polig, solution ← ∅
2 for i ← 1 to N-1 do
3     for j ← i + 1 to N do
4         if PolygonsConvexIncI(i, j, distancia, matriz) ≠ ∅ then
5             Insert(polig, PolygonsConvexIncI(i, j, distancia, matriz))
6 solution ← Duplicates(UpdateIncI(polig, fun))
7 return solution
```

4.6. Algoritmo recursivo

Para calcular la solución del polígono simple de k lados entre dos puntos contenido en un polígono reticulado con obstáculos arbitrarios, se recurre al **Algoritmo 4.1. PolygonsIncI(point1, point2, distancia, matriz)** con coste computacional $O(n^3k)$. Este algoritmo inicia su ejecución en el punto *point1* y explora todos los puntos adyacentes antes de avanzar a los puntos del siguiente nivel. Una vez se han calculado todas las posibles soluciones, verificando ciertas restricciones, se aplican los algoritmos del 4.2 al 4.6 para obtener la solución deseada.

Para la versión recursiva, se ha desarrollado el **Algoritmo 4.16 PolygonsIncR(point1, point2, first, iter, lados, distancia, matriz, fun**). Este algoritmo realiza una exploración ordenada de los puntos, dando prioridad a aquellos que están más cercanos a los recién visitados. De esta forma, el algoritmo se aleja progresivamente del punto inicial *point1*, hasta encontrar la mejor solución para el problema.

Para crear la solución general en su versión recursiva, siguiendo una estructura similar a la implementada en el **Algoritmo 4.10: SolutionIncI(N, distancia, matriz, fun)**, se utilizan cuatro algoritmos:

Algoritmo 4.16. PolygonsIncR(point1, point2, first, iter, lados, distancia, matriz, fun)
Algoritmo 4.17. UpdateIncR(poly, fun, distancia)
Algoritmo 4.18. SolutionIncR(N, distancia, matriz, fun)
Algoritmo 4.19. StartIncR(point1, point2, distancia, matriz, fun)

4.6.1. Cálculo entre dos puntos del polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 4.16)

Se utiliza para calcular de forma recursiva el polígono simple de k lados de mayor área o perímetro entre dos puntos, teniendo en cuenta una *distancia* determinada.

El algoritmo, además de tomar como valores de entrada los mismos parámetros que en el Algoritmo 4.1, utiliza dos números naturales, *first* e *iter*, así como un conjunto de aristas denominado *lados*. La llamada recursiva (línea 9) ocurre en función de las variables *iter* y *distancia*, y esta recursión continúa hasta que *iter* alcanza el valor *distancia* (línea 10).

Algoritmo 4.16 PolygonsIncR(point1, point2, first, iter, lados, distancia, matriz, fun)

Input: point1, point2 ∈ *Points*, first, iter ∈ ℕ, lados: aristas, distancia ∈ ℕ, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1)

Output: Polígonos simples de k lados de mayor área o perímetro entre *point1* y *point2* para una determinada *distancia*

```
1  temp1 ← lados
2  Insert(temp1, first)
3  temp2 ← ∅
4  for i ← 1 to N do
5      if (i ≠ point1 and i ≠ first) then
6          last ← temp1[Length(temp1)]
7          if (matriz(last, i) = 1 and Length(Union(temp1,i)) ≠ Length(temp1)) then
8              if iter < distancia then
9                  PolygonsIncR(point1, point2, i, iter+1, temp1, distancia, matriz, fun)
10             else
11                 Insert(temp1, i)
12                 if (temp1[distancia + 1] = point2 and IntersectionLad(temp1) = false and Alig-
                       nedc(temp1) ≠ 0) then
13                     h ← 0
14                     for j ← 1 to Length(HolesP) do
15                         if InterHolesP(temp1, HolesP[j]) = false then
16                             h ← h+1
17                     if h = Length(HolesP) then
18                         if (InterPointsP(temp1) = false and InterSegmentsP(temp1) = false
                               then
19                             temp2 ← temp1
20                             UpdateIncR(temp2, fun, distancia)
21                 Delete(temp1, temp1[Length(temp1)])
```

El coste computacional de este algoritmo recursivo está determinado por el bucle en la línea 4, que tiene una complejidad de $O(n)$. Dentro de este bucle, en cada iteración, se realiza una llamada recursiva a la función *PolygonsIncR* en función de k, donde $k = distancia + 1$. Por tanto, el número total de llamadas recursivas depende de n y de cómo k disminuye en cada iteración. Dado que k se reduce en 1 en cada llamada recursiva, el número total de llamadas recursivas se puede estimar como $O(n^k)$.

En forma general, la complejidad temporal para este algoritmo se puede expresar de la siguiente forma:

$$T(k) = \begin{cases} 1 & \text{si} \quad k = 1 \\ 1 + n\,T(k-1) & \text{si} \quad k > 1 \end{cases}$$

$$T(k) = 1 + n\,T(k-1) = 1 + n\,(1 + n\,T(k-2)) = 1 + n + n^2\,T(k-2) =$$
$$= 1 + n + n^2 + \ldots + n^{k-1}\,T(1) = \sum_{i=0}^{k-1} n^i \in O(n^k)$$

4.6.2. Actualización de soluciones (Algoritmo 4.17)

El Algoritmo 4.17, con coste computacional $O(k)$, actualiza el conjunto de las soluciones con el objetivo de maximizar el área o perímetro. El coste computacional está relacionado con el Algoritmo 4.7, con complejidad $O(k)$.

Algoritmo 4.17 UpdateIncR(poly, fun, distancia)

Input: *poly*: polígono de k lados, *fun*: área (fun = 0) o perímetro (fun = 1), distancia $\in \mathbb{N}$

Output: Actualización de soluciones

```
1  max1, max2 ← 0
2  if MejorSol ≠ ∅ then
3      if (Length(poly) = Length(MejorSol[1]) and Length(poly) = distancia + 1) then
4          max1 ← Function(poly, fun)                          /* Alg.4.7 */
5          max2 ← Function(MejorSol[1], fun)                   /* Alg.4.7 */
6          if max1 = max2 then
7              Insert(MejorSol, poly)
8          if max1 > max2 then
9              MejorSol ← ∅
10             Insert(MejorSol, poly)
11 else
12     Insert(MejorSol, poly)
```

4.6.3. Cálculo del polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 4.18)

El Algoritmo 4.18 es el algoritmo principal en su versión recursiva. En primer lugar, se calculan los polígonos simples de k lados de mayor área o perímetro (líneas 1 a 4), aplicando el Algoritmo 4.19, que inicia el proceso recursivo. El coste computacional es inicialmente $O(n^{k+2})$ debido a los bucles de las líneas 2 y 3, que operan en $O(n^2)$, y al algoritmo anterior que se ejecuta en $O(n^k)$ como consecuencia de la llamada al Algoritmo 4.16. A continuación, de todas las soluciones que se obtuvieron en el proceso anterior, elimina aquellas soluciones repetidas para obtener la mejor solución. Por tanto, el coste del Algoritmo 4.18 es $O(n^k)$, ya que $O(Max(n^{k+2}, n^4 k \log k)) = O(n^{k+2}) = O(n^k)$.

Algoritmo 4.18 SolutionIncR(N, distancia, matriz, fun)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1)

Output: Polígonos simples de k lados de mayor área o perímetro contenido en P

```
1 polig, solution ← ∅
2 for i ← 1 to N-1 do
3 |   for j ← i + 1 to N do
4 |   |   StartIncR(i, j, distancia, matriz, fun)                /* Alg.4.19 */
5 polig ← Duplicates(MejorSol)                                   /* Alg.4.9 */
6 for i ← 1 to Length(polig) do
7 |   if Length(polig[i]) = distancia +1 then
8 |   |   Insert(solution, polig[i])
9 return solution
```

Algoritmo 4.19 StartIncR(point1, point2, distancia, matriz, fun)

Input: point1, point2 $\in Points$, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1)

Output: Inicio del proceso recursivo

```
1 lados ← ∅
2 if matriz(point1, point2) = 1 then
3 |   PolygonsIncR(point1, point2, point1, 1, lados, distancia, matriz, fun)
```

En la versión recursiva también es posible determinar el rectángulo y el polígono convexo de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios, con coste computacional $O(n^6)$ y $O(n^k)$, respectivamente. El pseudocódigo es similar a los propuestos en el Algoritmo 4.12 y Algoritmo 4.15, pero ahora se incluye el Algoritmo 4.16. PolygonsIncR(point1, point2, first, iter, lados, distancia, matriz, fun).

Algoritmo 4.20. RectangleIncR(N, matriz, fun)
Algoritmo 4.21. SolutionConvexIncR(N, distancia, matriz, fun)

4.6.4. Coste computacional: resumen (versión recursiva)

En la Figura 4.9 se representa un esquema que describe el coste computacional de los algoritmos empleados en la resolución de la versión recursiva.

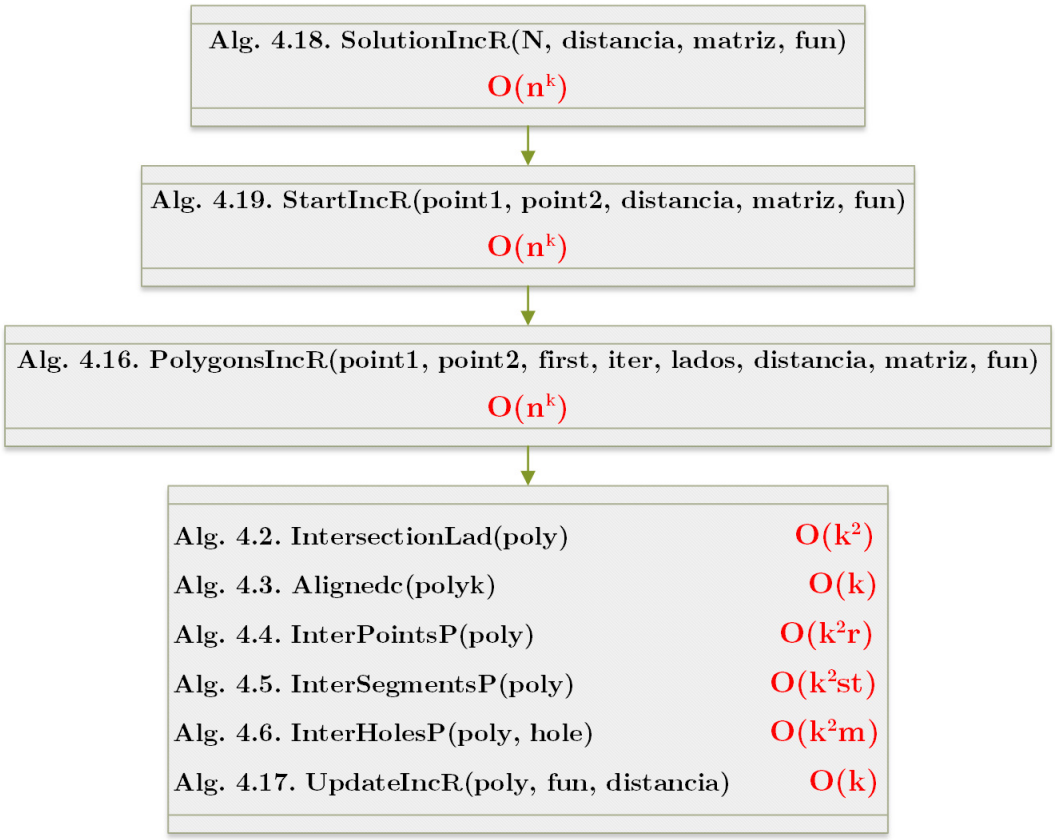


Figura 4.9: Problemas de inclusión: coste computacional (versión recursiva)

4.7. Solución general

El Algoritmo 4.22 unifica todas las versiones *Solution* en un solo procedimiento, englobando tanto la iteración como la recursión, con coste computacional $O(n^5k)$ y $O(n^k)$, respectivamente. Se define a través seis parámetros, de los cuales los cuatro primeros ya han sido previamente establecidos. Los dos últimos parámetros se emplean para calcular la solución de un polígono, ya sea simple (*conv = 0*) o convexo (*conv = 1*), ya sea en su versión iterativa (*ver = 0*) o en su versión recursiva (*ver = 1*). Esto implica que a través de un único algoritmo, se puedan resolver hasta ocho variaciones diferentes del problema.

Algoritmo 4.22 SolutionInc(N, distancia, matriz, fun, conv, ver)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia,
fun: área (fun = 0) o perímetro (fun = 1), *conv*: simple (conv = 0) o convexo (conv = 1),
ver: iterativo (ver = 0) o recursivo (ver = 1)
Output: Polígono simple o convexo de k lados de mayor área o perímetro contenido en P

```
1  solution ← ∅
2  sum ← 4fun + 2conv + ver
3  switch sum do
4      case 0 do
5          solution ← SolutionIncI(N, distancia, matriz, 0)          /* Alg.4.10 */
6      case 1 do
7          solution ← SolutionIncR(N, distancia, matriz, 0)          /* Alg.4.18 */
8      case 2 do
9          solution ← SolutionConvexIncI(N, distancia, matriz, 0)    /* Alg.4.15 */
10     case 3 do
11         solution ← SolutionConvexIncR(N, distancia, matriz, 0)    /* Alg.4.21 */
12     case 4 do
13         solution ← SolutionIncI(N, distancia, matriz, 1)          /* Alg.4.10 */
14     case 5 do
15         solution ← SolutionIncR(N, distancia, matriz, 1)          /* Alg.4.18 */
16     case 6 do
17         solution ← SolutionConvexIncI(N, distancia, matriz, 1)    /* Alg.4.15 */
18     case 7 do
19         solution ← SolutionConvexIncR(N, distancia, matriz, 1)    /* Alg.4.21 */
20 return solution
```

El Algoritmo 4.23 introduce un nuevo parámetro denominado (*ver*), en comparación con los dos algoritmos anteriores respecto al cálculo del rectángulo (Algoritmo 4.12 y Algoritmo 4.20). Este parámetro adicional se ha incorporado con el objetivo de distinguir entre las dos versiones principales: la versión iterativa (*ver = 0*) con coste computacional $O(n^5)$, y la versión recursiva (*ver = 1*) con coste computacional $O(n^6)$.

Algoritmo 4.23 RectangleInc(N, matriz, fun, ver)

Input: N: número de puntos de P, matriz: matriz de adyacencia,
fun: área (fun = 0) o perímetro (fun = 1), *ver*: iterativo (ver = 0) o recursivo (ver = 1)
Output: Rectángulo de mayor área o perímetro contenido en P

```
1  rect ← ∅
2  sum ← 2fun + ver
3  switch sum do
4      case 0 do
5          rect ← RectangleIncI(N, matriz, 0)                    /* Alg.4.12 */
6      case 1 do
7          rect ← RectangleIncR(N, matriz, 0)                    /* Alg.4.20 */
8      case 2 do
9          rect ← RectangleIncI(N, matriz, 1)                    /* Alg.4.12 */
10     case 3 do
11         rect ← RectangleIncR(N, matriz, 1)                    /* Alg.4.20 */
12 return rect
```

Tanto el Algoritmo 4.22 como el Algoritmo 4.23 poseen múltiples características destacadas que pueden tener un impacto significativo en el cálculo de áreas y perímetros de polígonos, ya sean simples o convexos. Entre estas características se incluyen:

Simplicidad de uso. Los algoritmos son más fáciles de usar, ya que solo requieren unos pocos parámetros, dependiendo del algoritmo, para funcionar.

Facilita la experimentación. Permite experimentar con diferentes configuraciones de manera más eficiente. Se puede cambiar fácilmente entre cálculos de área y perímetro, probar distintos tipos de polígonos y evaluar el rendimiento de algoritmos iterativos y recursivos.

Documentación clara. Al contar con un solo algoritmo, la documentación del código se simplifica, permitiendo una comprensión rápida de su uso y la posibilidad de personalizarlo según las necesidades.

4.8. Solución del caso propuesto

A continuación, se presentan las soluciones para el problema inicial (Figura 4.10) al aplicar cada uno de los dos algoritmos recursivos. Estas soluciones se detallan en tablas (Tabla 4.1 a Tabla 4.3), cada una compuesta por tres columnas. En la primera columna, se especifica la *distancia*; la segunda columna, indica el polígono a calcular; y finalmente, la tercera columna, presenta las soluciones encontradas.

Algoritmo 4.22. SolutionInc(N, distancia, matriz, fun, conv, ver)
Algoritmo 4.23. RectangleInc(N, matriz, fun, ver)

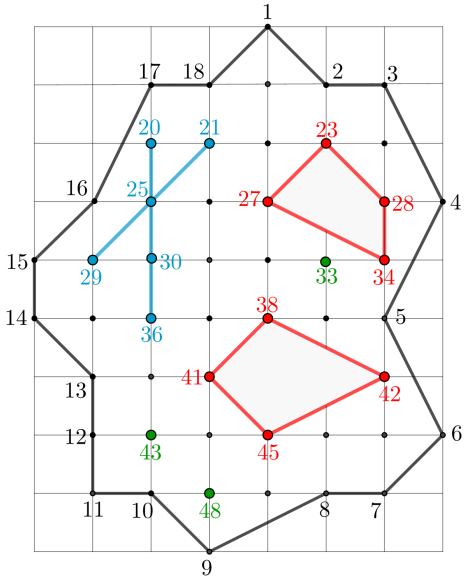


Figura 4.10: Problema inicial

Además de estos algoritmos para calcular las soluciones deseadas, es necesario emplear los siguientes conjuntos:

$$\begin{cases} Points = P = \{p_1, \ldots, p_{49}\} \\ Polygon = \partial P_0 = \{p_1, \ldots, p_{18}\} \\ PointsP = \{p_{33}, p_{43}, p_{48}\} \\ SegmentsP = \{S_1, S_2\}, \text{ donde } S_1 = \{p_{20}, p_{25}, p_{30}, p_{36}\}, S_2 = \{p_{21}, p_{25}, p_{29}\} \\ HolesP = \{H_1, H_2\}, \text{ donde } H_1 = \{p_{23}, p_{28}, p_{34}, p_{27}\}, H_2 = \{p_{38}, p_{42}, p_{45}, p_{41}\} \end{cases}$$

Por último, se llevará a cabo una comparación entre las soluciones generadas para los polígonos simples y los polígonos convexos (Tabla 4.4, Figuras 4.11 y 4.12). Para el cálculo del área y el perímetro, se ha usado un tamaño de partición $L = 1$.

Al analizar los resultados presentados en la Tabla 4.4, se observa un incremento tanto en el área como en el perímetro de los polígonos simples a medida que aumenta el número de lados a calcular. Sin embargo, en ciertos casos de polígonos convexos no se lograron encontrar soluciones. Esta situación se ilustra en la Tabla 4.3 para el cálculo del heptágono y el octógono.

4.8.1. Polígono simple de *k* lados de mayor área o perímetro

Tabla 4.1: Solución: Polígono simple de k lados de mayor área o perímetro

distancia	***k*-gon**	**SolutionInc(49, distancia, matriz, 0, 0, ver)**
2	Triángulo	(11,33,21)
3	Cuadrilátero	(5,38,11,21)
4	Pentágono	(6,43,33,21,11), (7,43,33,21,11)
5	Hexágono	(6,43,33,21,11,7)
6	Heptágono	(5,38,43,6,7,11,21), (6,21,11,38,42,10,7)
7	Octágono	(5,21,11,38,42,10,7,6)
distancia	***k*-gon**	**SolutionInc(49, distancia, matriz, 1, 0, ver)**
2	Triángulo	(1,11,2)
3	Cuadrilátero	(1,40,2,11)
4	Pentágono	(1,11,19,43,2)
5	Hexágono	(1,40,2,43,6,11)
6	Heptágono	(1,11,19,40,22,43,2)
7	Octágono	(1,40,19,31,2,43,6,11)

4.8.2. Rectángulo de mayor área o perímetro

Tabla 4.2: Solución: Rectángulo de mayor área o perímetro

RectangleInc(49, matriz, 0, ver)
(14,48,45,29), (13,27,33,43)
RectangleInc(49, matriz, 1, ver)
(10,25,26,48), (12,11,7,47)

4.8.3. Polígono convexo de k lados de mayor área o perímetro

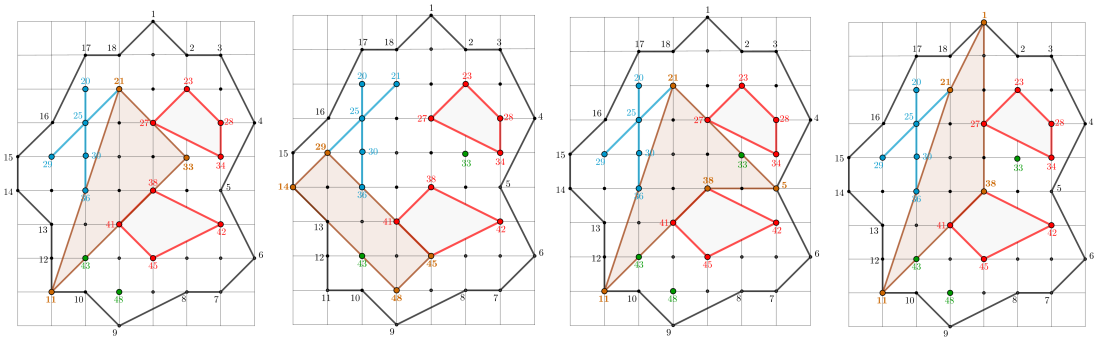
Tabla 4.3: Solución: Polígono convexo de k lados de mayor área o perímetro

distancia	***k*-gon**	**SolutionInc(49, distancia, matriz, 0, 1, ver)**
3	Cuadrilátero	(1,38,11,21)
4	Pentágono	(1,21,11,43,2), (1,32,41,11,21)
5	Hexágono	(1,27,41,10,36,21), (1,32,41,43,36,21)
		(1,38,41,40,36,21), (8,49,43,14,15,29), (10,41,27,22,21,25)
		(10,41,27,19,21,30), (14,48,49,45,29,15), (19,32,41,43,30,21)
		(19,38,41,40,30,21), (21,25,40,41,38,22), (21,25,43,41,32,22)
6	Heptágono	$\emptyset$
7	Octágono	$\emptyset$
distancia	***k*-gon**	**SolutionInc(49, distancia, matriz, 1, 1, ver)**
3	Cuadrilátero	(1,11,43,2)
4	Pentágono	(1,21,11,43,2)
5	Hexágono	(1,22,37,43,11,21)
6	Heptágono	$\emptyset$
7	Octágono	$\emptyset$

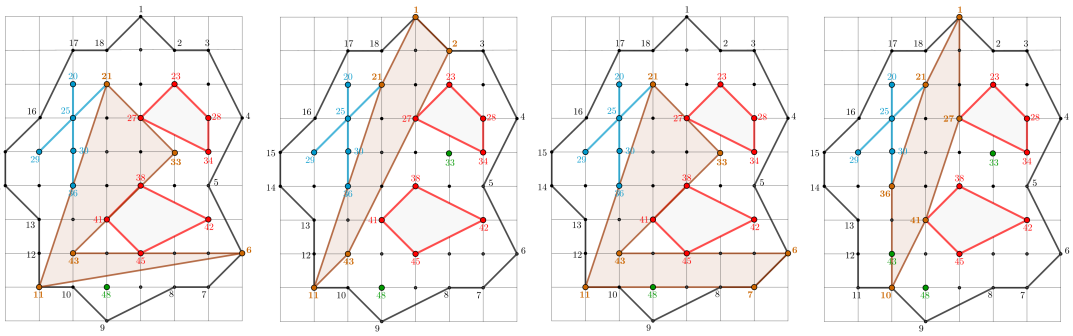
4.8.4. Comparación entre las soluciones para polígonos simples y polígonos convexos

Tabla 4.4: Comparativa entre el área y el perímetro de polígonos simples y convexos

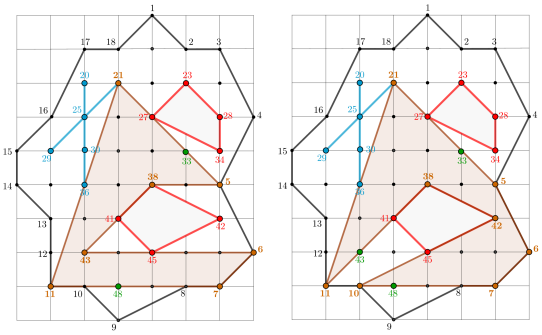
***k*-gon**		**3**	**4**	**5**	**6**	**7**	**8**	**Rectángulo**
Área	Simple	8	9	10,50	13	14	15,50	6
	Convexo		8,5	8	7	–	–	
Perímetro	Simple	18,01	28,75	30,63	38,48	39,55	44,84	12
	Convexo		18,07	18,09	17,35	–	–	



(a) Triángulo (b) Rectángulo (c) Cuadrilátero (d) Cuadrilátero convexo



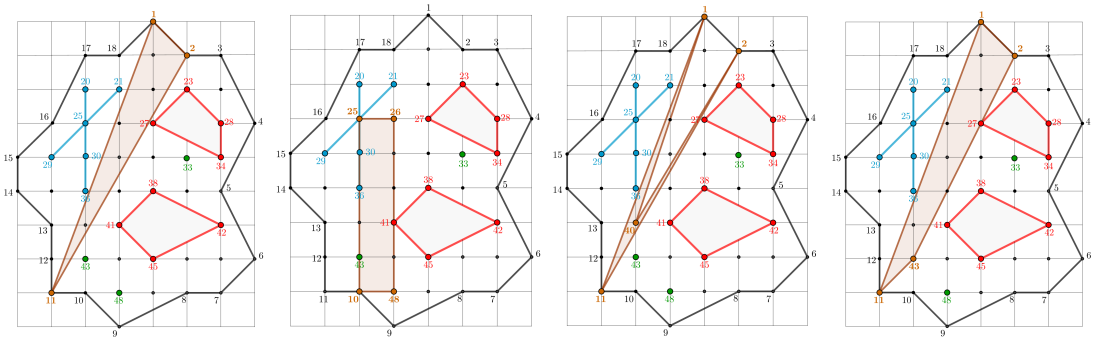
(e) Pentágono (f) Pentágono convexo (g) Hexágono (h) Hexágono convexo



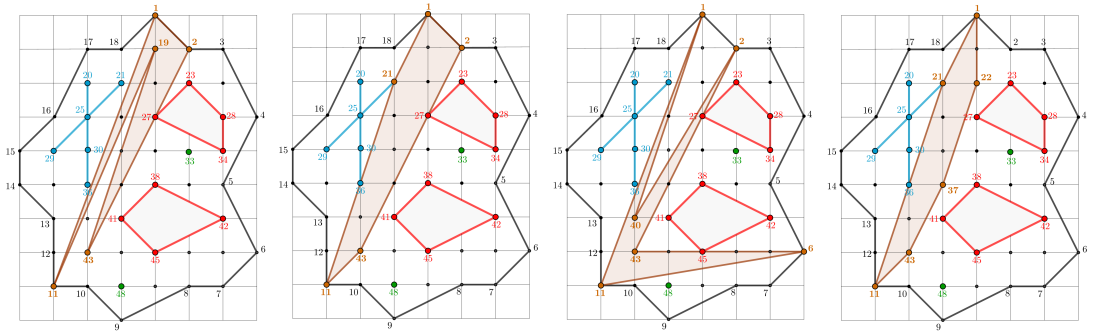
(i) Heptágono (j) Octógono

Figura 4.11: Comparación geométrica entre el área de polígonos simples y convexos

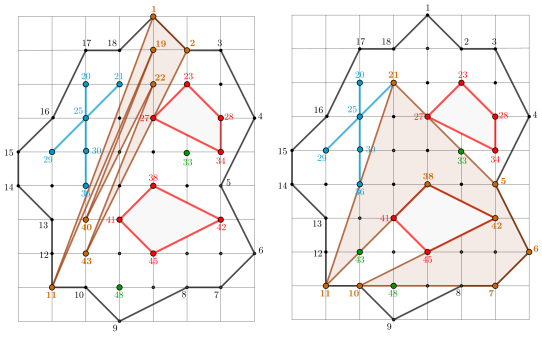
Para finalizar, esta sección se completa al calcular las soluciones relacionadas con el perímetro.



(a) Triángulo (b) Rectángulo (c) Cuadrilátero (d) Cuadrilátero convexo



(e) Pentágono (f) Pentágono convexo (g) Hexágono (h) Hexágono convexo



(i) Heptágono (j) Octógono

Figura 4.12: Comparación geométrica entre el perímetro de polígonos simples y convexos

4.9. Cálculo del polígono simple de k lados de mayor área o perímetro contenido en una ROI con obstáculos arbitrarios

Sea C una región de interés o contorno cerrado con r–puntos, t–segmentos y h–agujeros y C_0 el contorno exterior a C. Además, sea R_{min} el rectángulo de menor área que contiene al contorno cerrado C_0 [64] y Π una partición regular de R_{min} con *tamaño de partición* L.

Se llama *área inferior* $\underline{A}(C_0, \Pi)$ el área del polígono reticulado P_0 de mayor área contenido en C_0 y *área superior* $\overline{A}(H_j, \Pi)$ el área del polígono reticulado Q_j de menor área que contiene a H_j, $1 \leq j \leq h$, y ambos construidos por puntos de G_L. Por el Teorema de Pick [188],

$$\begin{aligned} \underline{A}(C_0, \Pi) &= \left(\#(\imath P_0) + \frac{\#(\partial P_0)}{2} - 1 \right) \cdot L^2 \\ \overline{A}(H_j, \Pi) &= \left(\#(\imath Q_j) + \frac{\#(\partial Q_j)}{2} - 1 \right) \cdot L^2 \end{aligned}$$

Teorema 4.9.1. *Sea C un contorno cerrado con r–puntos, t–segmentos y h–agujeros, $A(C)$ su área y C_0 el contorno exterior a C. Entonces, existe una sucesión de particiones regulares $\{\Pi_n\}_{n\in\mathbb{N}}$ with $\Pi_i \preceq \Pi_{i+1}$ para todo i tal que:*

$$\lim_{n\to\infty} \left(\underline{A}(C_0, \Pi_n) - \sum_{j=1}^{h} \overline{A}(H_j, \Pi_n) \right) = A(C)$$

Demostración. Consideremos una partición regular $\dot{\Pi}$ más fina que Π. Entonces, $\underline{A}(C_0, \Pi) \leq \underline{A}(C_0, \dot{\Pi})$ y $\overline{A}(H_j, \Pi) \geq \overline{A}(H_j, \dot{\Pi})$ con $1 \leq j \leq h$. Por tanto, $\underline{A}(C_0, \Pi) - \sum_{j=1}^{h} \overline{A}(H_j, \Pi) \leq \underline{A}(C_0, \dot{\Pi}) - \sum_{j=1}^{h} \overline{A}(H_j, \dot{\Pi})$. Luego, existe una sucesión de particiones regulares $\{\Pi_n\}_{n\in\mathbb{N}}$ con $\Pi_i \preceq \Pi_{i+1}$ para todo i tal que $\lim_{n\to\infty} \left(\underline{A}(C_0, \Pi_n) - \sum_{j=1}^{h} \overline{A}(H_j, \Pi_n) \right) = A(C)$. □

El Teorema 4.9.1 establece que es posible calcular el área de un contorno cerrado C a partir del contorno exterior C_0 y los agujeros H_j, a través de la construcción de particiones regulares cada vez más finas, $L_{i+1} = L_i/2$ con $\Pi_i \preceq \Pi_{i+1}$ para todo i. Además, el Algoritmo 4.22 demuestra cómo calcular el polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado P. Por tanto, combinando el Teorema 4.9.1 y el Algoritmo 4.22, se logra el objetivo final: el polígono simple de k

lados contenido en P es del mismo modo el máximo en área o en perímetro dentro del contorno cerrado C.

En particular, se muestran tres particiones para el mismo contorno cerrado C con obstáculos arbitrarios (Figura 4.13), con tamaños de partición: $L_1 = 1$, $L_2 = 1/2$, $L_3 = 1/4$, número de puntos: 49, 197, 804 y matriz de adyacencia: A, $\dot{A}$, $\ddot{A}$.

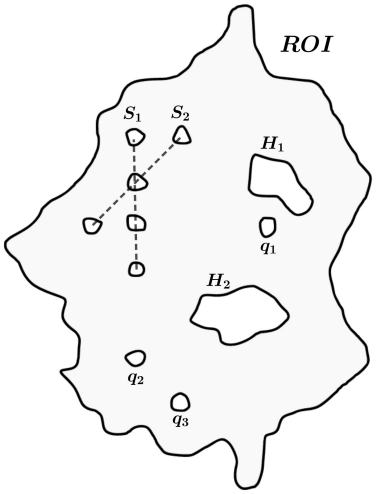


Figura 4.13: Contorno cerrado con obstáculos arbitrarios

Las soluciones obtenidas al ejecutar los Algoritmos 4.22 y 4.23 se representan en las Figuras 4.14 a 4.17. Se puede observar que, a medida que se realiza una partición más fina, el área de los polígonos simples o convexos es igual o mayor y el área de los rectángulos aumenta. Adicionalmente, la Tabla 4.5 resume las áreas que se obtienen al aplicar los algoritmos previamente mencionados.

Tabla 4.5: Área con diferentes tamaños de partición

Figura	**Algoritmo**	***k*-gon**	L_1	L_2	L_3
4.14.	Alg. 4.22	Triángulo	8	9,63	9,63
4.15.	Alg. 4.22	Cuadrilátero simple	9	11,25	11,38
4.16.	Alg. 4.22	Cuadrilátero convexo	8,50	11,25	11,38
4.17.	Alg. 4.23	Rectángulo	6	8	8,50

(a) SolutionInc(49, 2, A, 0, 0, ver), Área = 8

(b) SolutionInc(197, 2, $\dot{A}$, 0, 0, ver), Área = 9,63

(c) SolutionInc(804, 2, $\ddot{A}$, 0, 0, ver), Área = 9,63

Figura 4.14: Triángulo de mayor área con diferentes tamaños de partición

(a) SolutionInc(49, 3, A, 0, 0, ver), Área = 9

(b) SolutionInc(197, 3, $\dot{A}$, 0, 0, ver), Área = 11,25

(c) SolutionInc(804, 3, $\ddot{A}$, 0, 0, ver), Área = 11,38

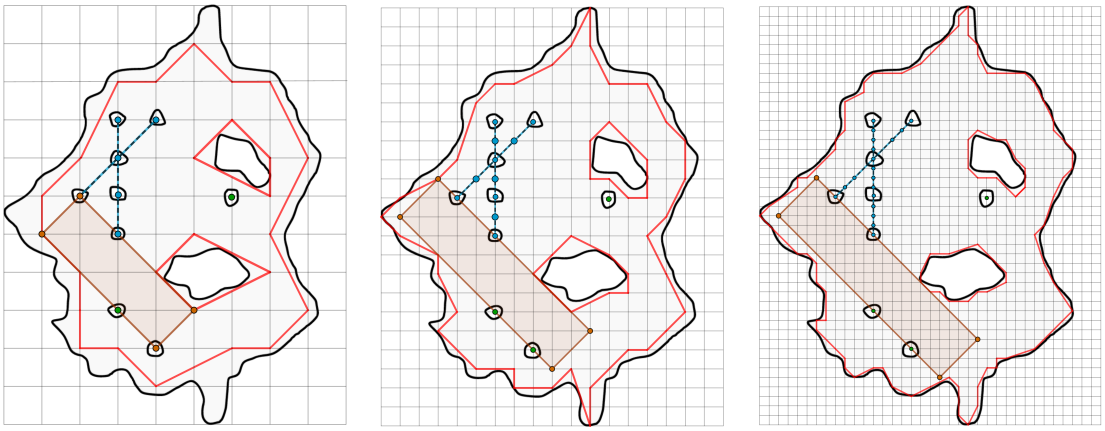
Figura 4.15: Cuadrilátero simple de mayor área con diferentes tamaños de partición

(a) SolutionInc(49, 3, A, 0, 1, ver), Área = 8,50

(b) SolutionInc(197, 3, $\dot{A}$, 0, 1, ver), Área = 11,25

(c) SolutionInc(804, 3, $\ddot{A}$, 0, 1, ver), Área = 11,38

Figura 4.16: Cuadrilátero convexo de mayor área con diferentes tamaños de partición



(a) RectangleInc(49, A, 0, ver), Área = 6

(b) RectangleInc(197, $\dot{A}$, 0, ver), Área = 8

(c) RectangleInc(804, $\ddot{A}$, 0, ver), Área = 8,50

Figura 4.17: Rectángulo de mayor área con diferentes tamaños de partición

Capítulo 5

Problemas de separabilidad, $Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu)$

Este capítulo resuelve el problema de separabilidad $Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu)$. En otras palabras, se busca determinar, dado un valor k fijo y un conjunto O contenido en un polígono reticulado con obstáculos arbitrarios, calcular el polígono simple de k lados de mayor o menor área o perímetro, que contenga a O y esté contenido en el polígono reticulado.

Se tratan dos variantes del problema, cálculo del área o perímetro máximo, o bien, del área o perímetro mínimo. Esta distinción se establece con el propósito de resolver los problemas que se plantearon en la Sección 1.1:

Plan urbanístico. En el contexto de la planificación urbana de una ciudad, los edificios y monumentos son representados mediante diferentes colores. El problema consiste en diseñar un parque urbano de cualquier forma poligonal y mayor área, que incluya una amplia variedad de monumentos.

Medicina. Se ha diagnosticado un tumor en un tejido e identificado las células tumorales mediante sus coordenadas. Además, se han registrado también las coordenadas de las células sanas. ¿Cómo se puede lograr una separación precisa entre las células tumorales y las sanas para llevar a cabo una intervención quirúrgica con la mínima extirpación requerida?

Este algoritmo contribuye a la consecución del objetivo parcial 3 (ver Sección 1.6), ya que al realizar particiones cada vez más finas sobre una Región de Interés (ROI), la solución obtenida converge de manera natural hacia la solución real.

Objetivo 3. Dado un valor k fijo y un conjunto O contenido en una ROI con obstáculos arbitrarios, calcular el polígono simple de k lados de mayor o menor área o perímetro, que contenga a O y esté dentro de la ROI.

Este capítulo presenta un algoritmo diseñado para ofrecer soluciones basadas en el valor k seleccionado por el usuario. De esta forma, el algoritmo se vuelve altamente adaptable y ajustable a diferentes escenarios. Esta flexibilidad es especialmente valiosa, ya que las necesidades de investigación y los objetivos pueden variar significativamente de un caso a otro. Por ejemplo, en el contexto de la planificación urbana, un usuario puede estar interesado en calcular el polígono simple de k lados con el mayor área para determinar la mejor ubicación de un parque urbano, conteniendo obstáculos como edificios o monumentos. Por otro lado, otro usuario podría estar interesado en calcular el polígono simple de k lados de menor área que contenga las células malignas de un tumor cerebral, como paso previo para realizar una posible extirpación o un tratamiento con radiación.

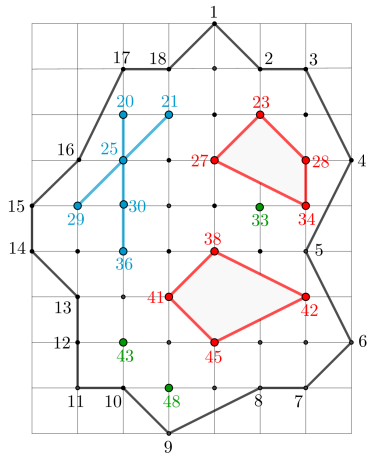
Esta capacidad de adaptar el algoritmo a diferentes valores de k también permite explorar distintas configuraciones. Así por ejemplo, se pueden identificar patrones en la forma y distribución de los polígonos resultantes. Esto facilita el análisis y la toma de decisiones en diversas áreas, como la planificación urbana, la medicina y otras muchas disciplinas.

Como ejemplo práctico, se usa el mismo caso de estudio que se ha desarrollado a lo largo de los Capítulos 3 y 4 (Figura 5.1a). A partir de la Figura 5.1b, el objetivo planteado es calcular el polígono simple de k lados de mayor o menor área o perímetro que contenga al conjunto de obstáculos arbitrarios $O = \{43, 48, H_2\}$.

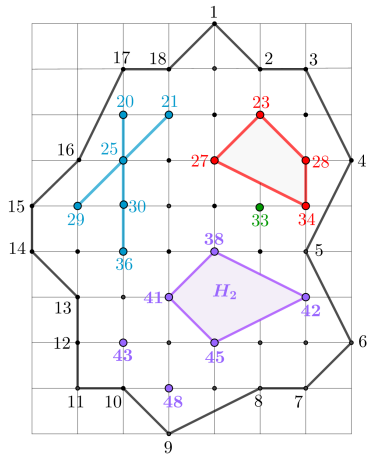
5.1. Descripción de los algoritmos utilizados (versión iterativa)

De la misma forma que se hizo en el Capítulo 4, se proporciona un esquema (Figura 5.2) de los algoritmos empleados. Asimismo, se han desarrollado dos versiones para resolver el problema: una versión iterativa, que se detalla a partir de esta Sección 5.1, y una versión recursiva, que se presenta en la Sección 5.5.

La versión iterativa tiene como algoritmo principal ***Algoritmo 5.6. SolutionSepI(N, distancia, matriz, fun, upd)*** (Sección 5.3), y se ocupa de calcular el polígono simple de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado P con obstáculos arbitrarios.



(a) Polígono reticulado



(b) Obstáculos arbitrarios $O = \{43, 48, H_2\}$

Figura 5.1: Problema inicial: polígono reticulado, conjunto de obstáculos arbitrarios

Cuenta con cinco parámetros: N, que representa el número de puntos del polígono reticulado; $k = distancia + 1$ y define el número de lados del polígono a calcular; matriz, que es la matriz de adyacencia; fun, que determina si calcular el área ($fun = 0$) o el perímetro ($fun = 1$); y finalmente, upd, que indica si hay que computar el máximo ($upd = 0$) o el mínimo ($upd = 1$).

También se encuentra el ***Algoritmo 5.1. PolygonsSepI(point1, point2, distancia, matriz)*** (Sección 5.2), responsable de calcular el polígono simple de k lados entre dos puntos que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

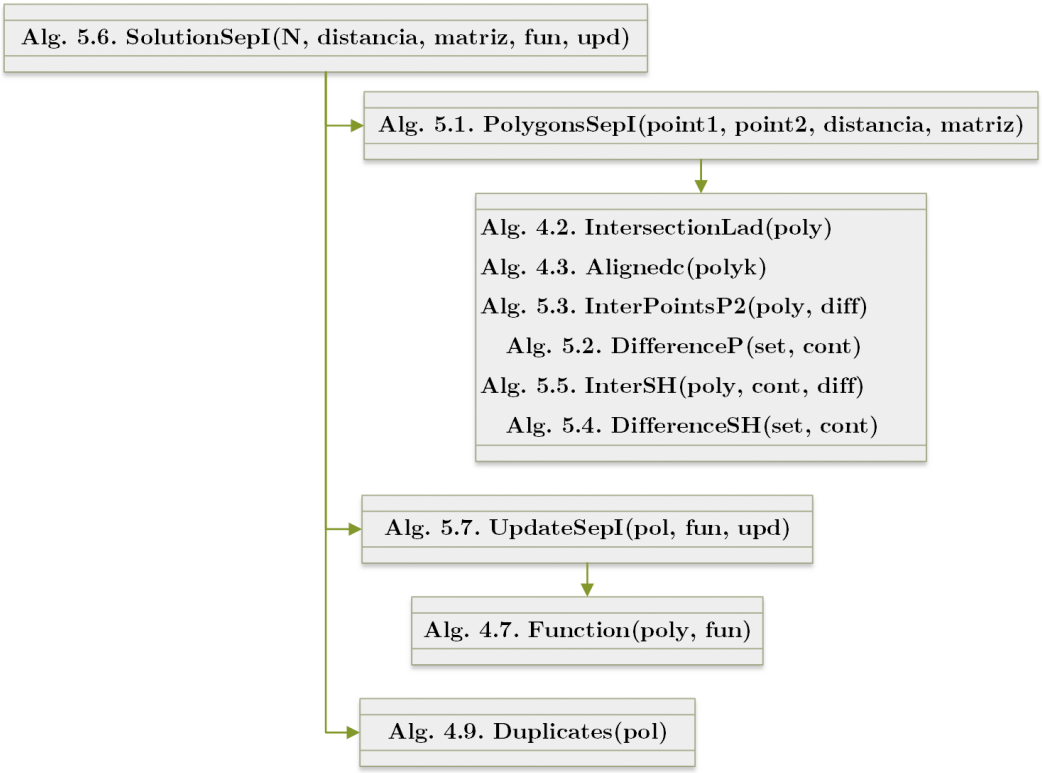


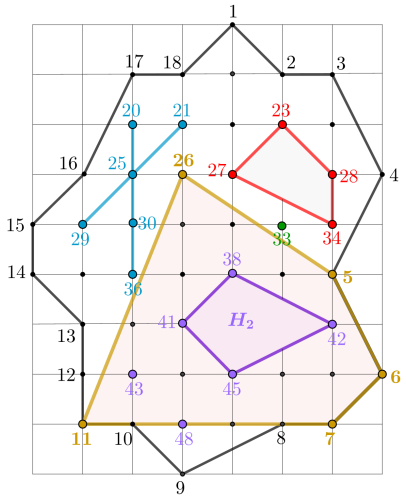
Figura 5.2: Problemas de separabilidad: esquema general de los algoritmos utilizados (versión iterativa)

Para calcular la solución de un polígono reticulado P con r–puntos, t–segmentos y h–agujeros, se emplean los siguientes conjuntos:

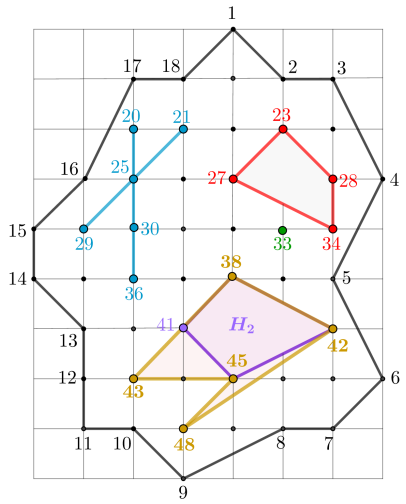
$$\left[\begin{array}{l} Points = P = P_0 - \bigcup_{i=1}^{h}(\imath H_i) \\ Polygon = \partial P_0 \\ PointsP = \{q_1, \ldots, q_r\} \\ SegmentsP = \{S_1, \ldots, S_t\}, \text{ donde } \#(S_i) = s_i,\ 1 \leq i \leq t,\ s = max\{s_1, \ldots, s_t\} \\ HolesP = \{H_1, \ldots, H_h\}, \text{ donde: } \#(H_i) = m_i,\ 1 \leq i \leq h,\ m = max\{m_1, \ldots, m_h\} \\ O = \{ContP,\ ContS,\ ContH\}, \text{ donde:} \\ \quad ContP \subseteq PointsP,\ ContS \subseteq SegmentsP,\ ContH \subseteq HolesP \\ \quad \#(ContP) = \overline{r},\ \#(ContS) = \overline{t},\ \#(ContH) = \overline{h} \end{array}\right.$$

El conjunto de obstáculos arbitrarios O se define como $O = \{ContP,\ ContS,\ ContH\}$. Esto significa que O está compuesto por un subconjunto de puntos, segmentos y agujeros de $PointsP$, $SegmentsP$ y $HolesP$, respectivamente. A modo de ejemplo, en la Figura 5.3, el conjunto O está formado por los puntos 43, 48 y el agujero H_2.

Se dice que un polígono de k lados es una solución del problema para un conjunto O si todos los elementos de O se encuentran en el interior o en la frontera del polígono solución. Además, los puntos, segmentos y agujeros que no forman parte del conjunto O deben estar en el exterior de la solución. En la Figura 5.3, se presentan dos soluciones para el cálculo del pentágono simple: una con el área máxima (Figura 5.3a) y otra con el área mínima (Figura 5.3b).



(a) Pentágono simple de mayor área: $(5, 26, 11, 7, 6)$



(b) Pentágono simple de menor área: $(38, 43, 45, 48, 42)$

Figura 5.3: Pentágono simple que contiene al conjunto $O = \{43, 48, H_2\}$

5.2. Cálculo entre dos puntos del polígono simple de k lados que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 5.1)

El desarrollo del Algoritmo 5.1 se divide en dos subsecciones, 5.2.1 y 5.2.2, que corresponden a los Algoritmos 5.2 al 5.5. Los Algoritmos 4.2 y 4.3 ya fueron analizados en el Capítulo 4, por lo que no se volverá a realizar un análisis detallado de ellos.

Algoritmo 4.2. IntersectionLad(poly)
Algoritmo 4.3. Alignedc(polyk)
Algoritmo 5.3. InterPointsP2(poly, diff) $\longrightarrow$ **Algoritmo 5.2.** DifferenceP(set, cont)
Algoritmo 5.5. InterSH(poly, cont, diff) $\longrightarrow$ **Algoritmo 5.4.** DifferenceSH(set, cont)

Para comprender el funcionamiento del Algoritmo 5.1, se ha estructurado en bloques, con el propósito de ofrecer una presentación más clara.

Bloque 1: Aristas entre dos puntos (líneas 2–13). Analizado en el Capítulo 4. El coste computacional es $O(n^3k)$.

Bloque 2: Algoritmos 4.2–4.3 (líneas 14–17). Analizado en el Capítulo 4. El coste computacional es $O(n^2k^3)$.

Bloque 3: Algoritmos 5.2–5.5 (líneas 18–23). El Bloque 3 realiza inicialmente el cálculo de los conjuntos *DiffP*, *DiffS* y *DiffH* (líneas 18–20). Estos conjuntos son el resultado de la diferencia entre *PointsP* y *ContP*, *SegmentsP* y *ContS*, y *HolesP* y *ContH*, respectivamente. Para llevar a cabo esta operación, se usan los siguientes algoritmos: Algoritmo 5.2, con coste computacional $O(r)$, y el Algoritmo 5.4, cuyo coste computacional es $O(t^2)$ cuando se aplica a los segmentos y $O(h^2)$ cuando se aplica a los agujeros.

A continuación, se procede a eliminar todas las soluciones que no cumplen las condiciones definidas por los algoritmos *InterPointsP2(poly, diff)* y *InterSH(poly, cont, diff)* (línea 22). Estas funciones devuelven *false* si todos los elementos de O se encuentran dentro o en la frontera de un polígono de *temp2*, mientras que los demás puntos, segmentos y agujeros que no pertenecen al conjunto O se encuentran en el exterior. El coste computacional de la primera función es $O(kr)$, mientras que el de la segunda función es $O(kst)$ cuando se aplica a segmentos y $O(kmh)$ cuando se aplica a agujeros. De esta manera, el coste computacional del Bloque 3 es $O(n^2k^3rsthm)$ debido al bucle de la línea 21 y las funciones anteriores.

Por tanto, el coste computacional del Algoritmo 5.1 es $O(n^3k)$, ya que $O(Max(n^3k, n^2k^3, n^2k^3rsthm)) = O(n^3k)$ si se asume que $\#(P) = N \simeq n \gg k$.

Algoritmo 5.1 PolygonsSepI(point1, point2, distancia, matriz)

Input: point1, point2 ∈ *Points*, distancia ∈ ℕ, matriz: matriz de adyacencia

Output: Polígonos simples de k lados entre *point1* y *point2* que contienen al conjunto O para una determinada distancia

```
1  temp1 ← {{point1}};   polig ← ∅
2  if matriz(point1, point2) = 1 then
3      for i ← 1 to distancia do
4          temp2 ← ∅
5          for j ← 1 to Length(temp1) do
6              last ← temp1[j][i]
7              if last ≠ point2 then
8                  for k ← 1 to N do
9                      if (matriz(last,k) = 1 and Length(Union(temp1[j],k)) ≠ Length(temp1[j]))
                        then
10                         Insert(temp2, Insert(temp1[j],k))
11                         Delete(temp1[j], temp1[j][Length(temp1[j])])
12         temp1 ← ∅
13         temp1 ← temp2
14     temp2 ← ∅
15     for i ← 1 to Length(temp1) do
16         if (temp1[i][distancia + 1] = point2 and IntersectionLad(temp1[i]) = false and
            Alignedc(temp1[i]) ≠ 0) then
17             Insert(temp2, temp1[i])
18     DiffP ← DifferenceP(PointsP, ContP)                          /* Alg.5.2 */
19     DiffS ← DifferenceSH(SegmentsP, ContS)                       /* Alg.5.4 */
20     DiffH ← DifferenceSH(HolesP, ContH)                          /* Alg.5.4 */
21     for i ← 1 to Length(temp2) do
22         if (InterPointsP2(temp2[i], DiffP) = false and
            InterSH(temp2[i], ContS, DiffS) = false and
            InterSH(temp2[i], ContH, DiffH) = false) then
23             Insert(polig, temp2[i])
24 return polig
```

5.2.1. Posición puntos–polígono (Algoritmo 5.3)

En esta sección, se realiza un análisis de dos algoritmos: Algoritmo 5.2 y Algoritmo 5.3. El primero de ellos calcula la diferencia entre los conjuntos de puntos de *set* y *cont*. El resultado de esta operación es un nuevo conjunto llamado *diff*, que se pasa como parámetro al Algoritmo 5.3. Inicialmente, el coste computacional es $O(\overline{r})$ debido al bucle en la línea 2, pero finalmente se reduce a $O(r)$, ya que $\overline{r} \leq r$.

El segundo algoritmo tiene como objetivo verificar si todos los puntos del conjunto *ContP* se encuentran dentro del polígono *poly*, y si todos los puntos del conjunto *diff* están en el exterior de *poly*. En caso de que se cumpla esta condición, el algoritmo devuelve *false*. Se utilizan dos bucles (líneas 2 y 6) y el algoritmo *PointIn(P, poly)* para realizar las comprobaciones anteriores. El coste computacional de estas operaciones es $O(kr)$.

La Figura 5.4 muestra un polígono con coordenadas $(33, 43, 9, 42)$. Si se aplica el Algoritmo 5.3 el resultado es *true*, ya que aunque los puntos 43 y 48 se localizan en el interior del polígono, el punto 33 debería estar en el exterior, pero se encuentra en la frontera.

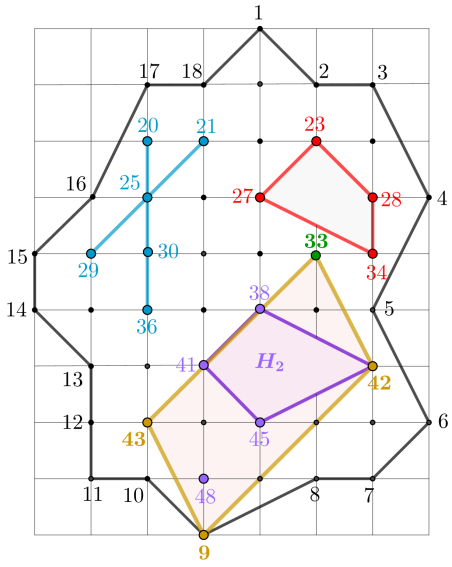


Figura 5.4: Algoritmo 5.3 InterPointsP2(poly, diff)

Algoritmo 5.2 DifferenceP(set, cont)

Input: *cont*: conjunto de puntos de *set*

Output: Conjunto diferencia entre *set* y *cont*

```
1 diff ← set
2 for i ← 1 to Length(cont) do
3 |   if Contains(set, cont[i]) then
4 |   |   Remove(diff, cont[i])
5 return diff
```

Algoritmo 5.3 InterPointsP2(poly, diff)

Input: *poly*: polígono de k lados, *diff*: conjunto diferencia

Output: *false*, si todos los puntos de *ContP* están dentro de *poly* y todos los puntos de *diff* están fuera de *poly*

```
1 int1 ← true;  int2 ← false
2 for i ← 1 to Length(ContP) do
3 |   int1 ← PointIn(Points[ContP[i]], poly) = true
4 |   if int1 = false then
5 |   |   break
6 for i ← 1 to Length(diff) do
7 |   int2 ← PointIn(Points[diff[i]], poly) = true
8 |   if int2 = true then
9 |   |   break
10 int3 ← (int1 and ! int2)
11 return ! int3
```

5.2.2. Posición segmentos–polígono, agujeros–polígono (Algoritmo 5.5)

Se introducen dos algoritmos: Algoritmo 5.4 y Algoritmo 5.5. El objetivo del primer algoritmo es calcular la diferencia entre los conjuntos de segmentos o agujeros de *set* y *cont*. El resultado se almacena en el conjunto *diff* y se utiliza como parámetro en el Algoritmo 5.5. El coste computacional de este algoritmo varía según si se calcula la diferencia entre segmentos o agujeros. Si se realiza la diferencia entre segmentos, el coste es $O(t^2)$ debido a los bucles en las líneas 2 y 4, que se ejecutan en $O(t)$ y $O(\bar{t})$, respectivamente. Dado que $\bar{t} \leq t$, se puede considerar el coste como $O(t)$. Si se calcula la diferencia para agujeros, el razonamiento es similar y el coste computacional es $O(h^2)$.

Algoritmo 5.4 DifferenceSH(set, cont)

Input: *cont*: conjunto de segmentos o agujeros de *set*
Output: Conjunto diferencia entre *set* y *cont*

```
1  diff ← ∅
2  for i ← 1 to Length(set) do
3      con ← 0
4      for j ← 1 to Length(cont) do
5          if set[i] = cont[j] then
6              con ← con + 1
7              break
8      if con = 0 then
9          Insert(diff, set[i])
10 return diff
```

El segundo algoritmo se divide de dos partes. En la primera parte, se realiza una comprobación para determinar si todos los segmentos o agujeros se encuentran dentro de *poly* (líneas 3–20). En la segunda parte, se comprueba si todos los segmentos o agujeros de *diff* se encuentran fuera de *poly* (líneas 21–27). Si ambas condiciones se cumplen, el algoritmo devuelve *false*. El coste computacional del algoritmo varía dependiendo si se están evaluando segmentos o agujeros. En el caso de los segmentos, el tiempo de ejecución es $O(kst)$, mientras que para los agujeros es $O(kmh)$.

En la Figura 5.5, se representa un polígono con coordenadas $(30, 42, 9, 43)$. Al aplicar el Algoritmo 5.5, el resultado es *true*, ya que los puntos 30 y 36 del segmento S_1 se encuentran ubicados dentro del polígono y deberían situarse en el exterior.

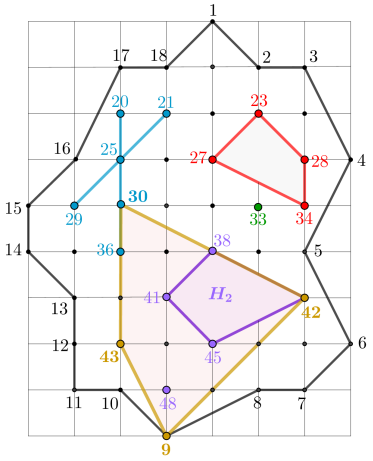


Figura 5.5: Algoritmo 5.5 InterSH(poly, cont, diff)

Algoritmo 5.5 InterSH(poly, cont, diff)

Input: *poly*: polígono de k lados, *cont*: conjunto de segmentos o agujeros, *diff*: conjunto diferencia

Output: *false*, si todos los segmentos o agujeros de *cont* están dentro de *poly* o en su frontera y todos los segmentos o agujeros de *diff* están fuera de *poly*

```
1  int1 ← true
2  int2 ← false
3  for i ← 1 to Length(cont) do
4      if int1 = false then
5          break
6      for j ← 1 to Length(cont[i]) do
7          if PointIn(Points[cont[i][j]], poly) = false then
8              int1 ← false
9  for i ← 1 to Length(cont)-1 do
10     if int1 = false then
11         break
12     n ← Length(cont[i])
13     for j ← 1 to n-1 do
14         m ← Medio(Points[cont[i][j]], Points[cont[i][j+1]])
15         if PointIn(m, poly) = false then
16             int1 ← false
17     if Function(cont[i], 0) ≠ 0 then
18         m ← Medio(Points[cont[i][n]], Points[cont[i][1]])
19         if PointIn(m, poly) = false then
20             int1 ← false
21 for i ← 1 to Length(diff) do
22     if int2 = true then
23         break
24     for j ← 1 to Length(diff[i]) do
25         if PointIn(Points[diff[i][j]], poly) = true then
26             int2 ← true
27             break
28 int3 ← (int1 and ! int2)
29 return ! int3
```

5.3. Cálculo del polígono simple de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 5.6)

El Algoritmo 5.6 es el algoritmo principal, y su obtención involucra una serie de algoritmos que contribuyen a su desarrollo y funcionalidad.

Algoritmo 5.1. PolygonsSepI(point1, point2, distancia, matriz)
Algoritmo 5.7. UpdateSepI(pol, fun, upd)
Algoritmo 4.7. Function(poly, fun)
Algoritmo 4.9. Duplicates(pol)

El funcionamiento del Algoritmo 5.6 se describe de la siguiente forma: en primer lugar, se calculan todos los polígonos simples de k lados (líneas 1–5), con un coste computacional de $O(n^5k)$. Posteriormente, se hacen uso de los Algoritmos 5.7, 4.7 y 4.9, con el objetivo de obtener el polígono simple de k lados de mayor o menor área o perímetro. El coste computacional total resultante es $O(n^5k)$.

Algoritmo 5.6 SolutionSepI(N, distancia, matriz, fun, upd)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1)
Output: Polígonos simples de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en P

```
1 polig, solution ← ∅
2 for i ← 1 to N-1 do
3     for j ← i + 1 to N do
4         if PolygonsSepI(i, j, distancia, matriz) ≠ ∅ then
5             Insert(polig, PolygonsSepI(i, j, distancia, matriz))
6 solution ← Duplicates(UpdateSepI(polig, fun, upd))
7 return solution
```

La tarea de determinar el máximo o mínimo valor en relación al área o al perímetro se lleva a cabo mediante el *Algoritmo 5.7. UpdateSepI(pol, fun, upd)*. Los parámetros *fun* y *upd* deciden el tipo de polígono a calcular, ya sea el de mayor área, menor área, mayor perímetro o menor perímetro. El coste computacional es $O(n^2k)$ debido a los bucles de las líneas 5 y 13 computados en $O(n^2)$ y el Algoritmo 4.7 con coste de complejidad $O(k)$.

Algoritmo 5.7 UpdateSepI(pol, fun, upd)

Input: *pol*: polígonos de k lados, *fun*: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1)

Output: Polígonos con mayor o menor área o perímetro

```
1  update ← ∅
2  switch upd do
3      case 0 do
4          max ← 0
5          for i ← 1 to Length(pol) do
6              f ← Function(pol[i], fun)                    /* Alg.4.7 */
7              if f > max then
8                  max ← f;  update ← pol[i]
9              else
10                 Insert(update, pol[i])
11     case 1 do
12         min ← +∞
13         for i ← 1 to Length(pol) do
14             f ← Function(pol[i], fun)                    /* Alg.4.7 */
15             if f < min then
16                 min ← f;  update ← pol[i]
17             else
18                 Insert(update, pol[i])
19 return update
```

Siguiendo el proceso utilizado en el Capítulo 4, es posible calcular el rectángulo y el polígono convexo de k lados de mayor o menor área. El procedimiento para obtener estos resultados se basa en una adaptación de los algoritmos necesarios para el cálculo del rectángulo y polígono convexo de k lados del capítulo anterior, combinados con los algoritmos actuales de este capítulo. El coste computacional total para la implementación de ambos algoritmos continúa siendo $O(n^5)$ y $O(n^5k)$, respectivamente.

Algoritmo 5.8. RectangleSepI(N, distancia, matriz, fun, upd)
Algoritmo 5.9. SolutionConvexSepI(N, distancia, matriz, fun, upd)

5.4. Coste computacional: resumen (versión iterativa)

En la Figura 5.6, se presenta el análisis del coste computacional de los algoritmos utilizados para la versión iterativa. En el centro de estos cálculos se halla el Algoritmo 5.6, responsable de calcular el polígono simple de k lados con el mayor o menor área o perímetro. Este polígono debe contener un conjunto O y estar contenido dentro de un polígono reticulado con obstáculos arbitrarios.

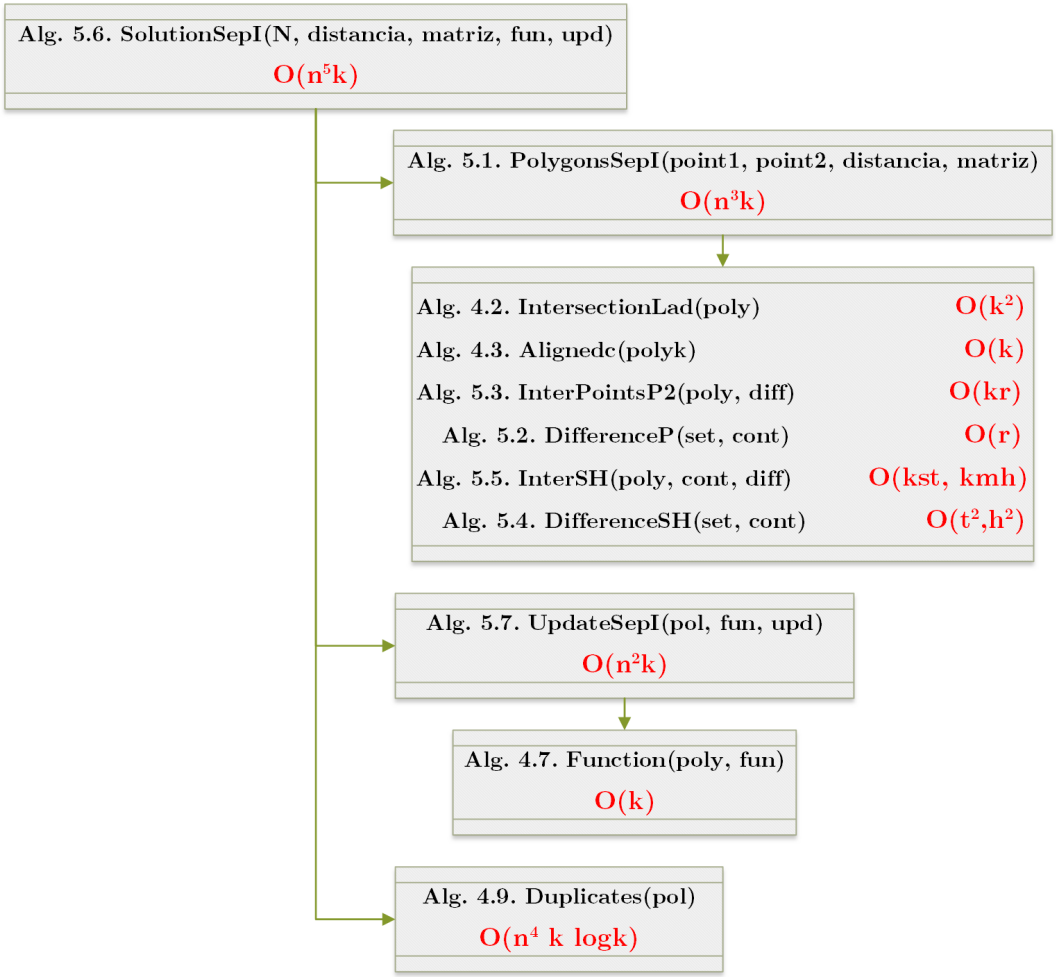


Figura 5.6: Problemas de separabilidad: coste computacional (versión iterativa)

5.5. Algoritmo recursivo

El Algoritmo recursivo **Algoritmo 5.12: SolutionSepR(N, distancia, matriz, fun, upd)**, elaborado en esta sección, sigue una metodología similar a la presentada en el Capítulo 4. Es el encargado de calcular el polígono simple de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios. Para su funcionamiento, se basa principalmente en cuatro algoritmos, además de otros que se trataron en el capítulo anterior (Figura 5.7):

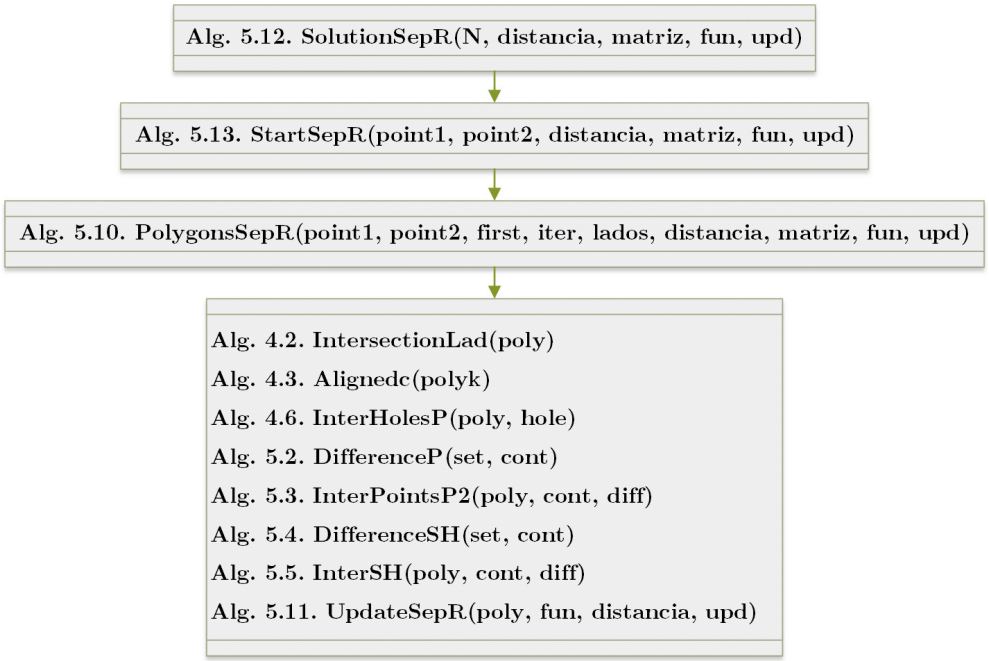


Figura 5.7: Problemas de separabilidad: esquema general de los algoritmos utilizados (versión recursiva)

5.5.1. Cálculo entre dos puntos del polígono simple de k lados de mayor o menor área o perímetro que contiene al conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 5.10)

El Algoritmo 5.10 comparte los mismos parámetros de entrada que el **Algoritmo 4.16. PolygonsIncR(point1, point2, first, iter, lados, distancia, matriz, fun)**. Se ha introducido un nuevo parámetro llamado *upd*, que permite controlar si se busca maximizar o minimizar el área o el perímetro.

Algoritmo 5.10 PolygonsSepR(point1, point2, first, iter, lados, distancia, matriz, fun, upd)

Input: point1, point2 $\in$ *Points*, inter, iter $\in \mathbb{N}$, lados: aristas, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1)

Output: Polígonos simples de k lados de mayor o menor área o perímetro entre *point1* y *point2* que contienen al conjunto O para una determinada distancia

```
1  temp1 ← lados
2  Insert(temp1, first)
3  temp2 ← ∅
4  for i ← 1 to N do
5      if (i ≠ point1 and i ≠ inter) then
6          last ← temp1[Length(temp1)]
7          if (matriz(last, i) = 1 and Length(Union(temp1,i)) ≠ Length(temp1)) then
8              if iter < distancia then
9                  PolygonsSepR(point1, point2, i, iter+1, temp1, distancia, matriz, fun)
10             else
11                 Insert(temp1, i)
12                 if (temp1[distancia + 1] = point2 and IntersectionLad(temp1) = false
                    and Alignedc(temp1) ≠ 0) then
13                     DiffP ← DifferenceP(PointsP, ContP)              /* Alg.5.2 */
14                     DiffS ← DifferenceSH(SegmentsP, ContS)           /* Alg.5.4 */
15                     DiffH ← DifferenceSH(HolesP, ContH)              /* Alg.5.4 */
16                     h ← 0
17                     for j ← 1 to Length(DiffH) do
18                         if InterHolesP(temp1, DiffH[j]) = false then
19                             h ← h+1
20                     if h = Length(DiffH) then
21                         if (InterPointsP2(temp1, DiffP) = false and
                            InterSH(temp1, ContS, DiffS) = false and
                            InterSH(temp1, ContH, DiffH) = false) then
22                             temp2 ← temp1
23                             UpdateSepR(temp2, fun, distancia, upd)
24                 Delete(temp1, temp1[Length(temp1)])
```

El coste computacional total de este algoritmo es equivalente al que se muestra en el Algoritmo 4.16. Este coste se estima como $O(n^k)$ debido al bucle en la línea 4, que tiene una complejidad de $O(n)$, y a las n llamadas recursivas que ocurren a lo largo del algoritmo a medida que el valor de k disminuye una unidad progresivamente.

5.5.2. Actualización de soluciones (Algoritmo 5.11)

El Algoritmo 5.11 actualiza la mejor solución, permitiendo realizar tanto el mayor como el menor área o perímetro, a través del parámetro *upd*. El coste computacional es $O(k)$, ya que hace uso del Algoritmo 4.7 para realizar los cálculos.

Algoritmo 5.11 UpdateSepR(poly, fun, distancia, upd)

Input: *poly*: polígono de k lados, *fun*: área (fun = 0) o perímetro (fun = 1), distancia $\in \mathbb{N}$, *upd*: máximo (upd = 0) o mínimo (upd = 1)
Output: Actualización de soluciones

```
1  max1, max2 ← 0
2  if MejorSol ≠ ∅ then
3      if (Length(poly) = Length(MejorSol[1]) and Length(poly) = distancia + 1) then
4          max1 ← Function(poly, fun)                                  /* Alg.4.7 */
5          max2 ← Function(MejorSol[1], fun)                           /* Alg.4.7 */
6          if max1 = max2 then
7              Insert(MejorSol, poly)
8          switch upd do
9              case 0 do
10                 if max1 > max2 then
11                     MejorSol ← ∅
12                     Insert(MejorSol, poly)
13             case 1 do
14                 if max1 < max2 then
15                     MejorSol ← ∅
16                     Insert(MejorSol, poly)
17 else
18     Insert(MejorSol, poly)
```

5.5.3. Cálculo del polígono simple de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios (Algoritmo 5.12)

Para la versión recursiva, el Algoritmo 5.12 es el algoritmo principal. El algoritmo comienza calculando inicialmente los polígonos simples de k lados de mayor o menor área o perímetro (líneas 1 a 4) en $O(n^{k+2})$, debido al Algoritmo 5.13 computado en $O(n^k)$ y a los bucles de las líneas 2 y 3. Luego, de todas las soluciones encontradas, se procede a eliminar las soluciones duplicadas a través del Algoritmo 4.9 (línea 5) en $O(n^4 k \log k)$. El coste computacional es $O(n^k)$, ya que $O(Max(n^{k+2}, n^4 k \log k)) = O(n^{k+2}) = O(n^k)$.

Algoritmo 5.12 SolutionSepR(N, distancia, matriz, fun, upd)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1)

Output: Polígonos simples de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en P

```
1 polig, solution ← ∅
2 for i ← 1 to N-1 do
3 |   for j ← i + 1 to N do
4 |   |   StartSepR(i, j, distancia, matriz, fun, upd)          /* Alg.5.13 */
5 polig ← Duplicates(MejorSol)                                  /* Alg.4.9 */
6 for i ← 1 to Length(polig) do
7 |   if Length(polig[i]) = distancia +1 then
8 |   |   Insert(solution, polig[i])
9 return solution
```

Algoritmo 5.13 StartSepR(point1, point2, distancia, matriz, fun, upd)

Input: point1, point2 $\in$ *Points*, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1)

Output: Inicio del proceso recursivo

```
1 lados ← ∅
2 if matriz(point1, point2) = 1 then
3 |   PolygonsSepR(point1, point2, point1, 1, lados, distancia, matriz, fun, upd)
```

En la versión recursiva, también se puede calcular tanto el rectángulo como el polígono convexo de k lados de mayor o menor área o perímetro. El coste computacional asociado a cada uno de los algoritmos es $O(n^6)$ y $O(n^k)$, respectivamente.

Algoritmo 5.14. RectangleSepR(N, matriz, fun, upd)
Algoritmo 5.15. SolutionConvexSepR(N, distancia, matriz, fun, upd)

5.5.4. Coste computacional: resumen (versión recursiva)

En la Figura 5.8 se muestra el análisis del coste computacional de los algoritmos que se han explicado y desarrollado para la versión recursiva.

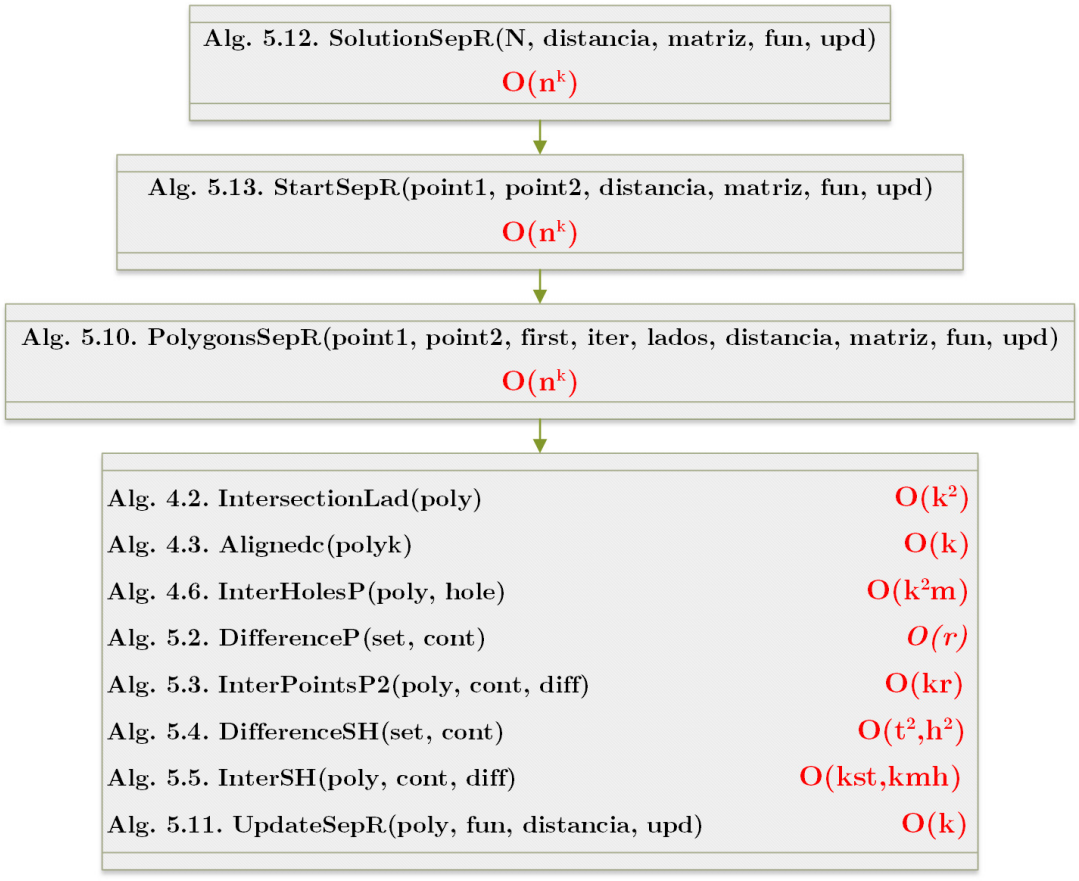


Figura 5.8: Problemas de separabilidad: coste computacional (versión recursiva)

5.6. Solución general

El Algoritmo 5.16 proporciona una solución que engloba todas las versiones anteriores relacionadas con el cálculo de un polígono, ya sea simple o convexo, según los valores de los últimos cuatro parámetros: *fun* indica si se debe calcular el área (*fun = 0*) o el perímetro (*fun = 1*); *upd* permite seleccionar el valor máximo (*upd = 0*) o mínimo (*upd = 1*); *conv* ofrece la opción de obtener un polígono simple (*conv = 0*) o convexo (*conv = 1*); y por último, *ver* permite elegir entre la versión iterativa (*ver = 0*) o recursiva (*ver = 1*). En consecuencia, este algoritmo tiene la capacidad de resolver hasta 16 versiones distintas del problema de manera eficiente. El coste computacional para la versión iterativa es $O(n^5k)$, mientras que para la versión recursiva es $O(n^k)$.

Algoritmo 5.16 SolutionSep(N, distancia, matriz, fun, upd, conv, ver)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia,
fun: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1),
conv: simple (conv = 0) o convexo (conv = 1),
ver: iterativo (ver = 0) o recursivo (ver = 1)

Output: Polígono simple o convexo de k lados de mayor área o perímetro que contiene a un conjunto O y está contenido en P

```
1  solution ← ∅
2  sum ← 8fun + 4upd + 2conv + ver
3  switch sum do
4      case 0 do
5          solution ← SolutionSepI(N, distancia, matriz, 0, 0)          /* Alg.5.6 */
6      case 1 do
7          solution ← SolutionSepR(N, distancia, matriz, 0, 0)          /* Alg.5.12 */
8      case 2 do
9          solution ← SolutionConvexSepI(N, distancia, matriz, 0, 0)    /* Alg.5.9 */
10     case 3 do
11         solution ← SolutionConvexSepR(N, distancia, matriz, 0, 0)    /* Alg.5.15 */
12     case 4 do
13         solution ← SolutionSepI(N, distancia, matriz, 0, 1)          /* Alg.5.6 */
14     case 5 do
15         solution ← SolutionSepR(N, distancia, matriz, 0, 1)          /* Alg.5.12 */
```

Algoritmo 5.16 SolutionSep(N, distancia, matriz, fun, upd, conv, ver) (cont.)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia,
fun: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1),
conv: simple (conv = 0) o convexo (conv = 1),
ver: iterativo (ver = 0) o recursivo (ver = 1)

Output: Polígono simple o convexo de k lados de mayor área o perímetro que contiene a un conjunto O y está contenido en P

```
1  solution ← ∅
2  sum ← 8fun + 4upd + 2conv + ver

   ...
   ...
   ...
   switch sum do
16     case 6 do
17         solution ← SolutionConvexSepI(N, distancia, matriz, 0, 1)      /* Alg.5.9 */
18     case 7 do
19         solution ← SolutionConvexSepR(N, distancia, matriz, 0, 1)     /* Alg.5.15 */
20     case 8 do
21         solution ← SolutionSepI(N, distancia, matriz, 1, 0)            /* Alg.5.6 */
22     case 9 do
23         solution ← SolutionSepR(N, distancia, matriz, 1, 0)           /* Alg.5.12 */
24     case 10 do
25         solution ← SolutionConvexSepI(N, distancia, matriz, 1, 0)      /* Alg.5.9 */
26     case 11 do
27         solution ← SolutionConvexSepR(N, distancia, matriz, 1, 0)     /* Alg.5.15 */
28     case 12 do
29         solution ← SolutionSepI(N, distancia, matriz, 1, 1)            /* Alg.5.6 */
30     case 13 do
31         solution ← SolutionSepR(N, distancia, matriz, 1, 1)           /* Alg.5.12 */
32     case 14 do
33         solution ← SolutionConvexSepI(N, distancia, matriz, 1, 1)      /* Alg.5.9 */
34     case 15 do
35         solution ← SolutionConvexSepR(N, distancia, matriz, 1, 1)     /* Alg.5.15 */
36 return solution
```

El Algoritmo 5.17 integra, respecto al parámetro *ver*, en un único algoritmo todas las versiones anteriores, Algoritmo 5.8 y Algoritmo 5.14,. Esta incorporación tiene como finalidad diferenciar entre las dos modalidades anteriores: la iterativa (*ver = 0*) con coste computacional $O(n^5)$ y la recursiva (*ver = 1*) con coste computacional $O(n^6)$.

Algoritmo 5.17 RectangleSep(N, matriz, fun, upd, ver)

Input: N: número de puntos de P, matriz: matriz de adyacencia,
fun: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1),
ver: iterativo (ver = 0) o recursivo (ver = 1)

Output: Rectángulo de mayor área o perímetro que contiene a un conjunto O y está contenido en P

```
1  rect ← ∅
2  sum ← 4fun + 2upd + ver
3  switch sum do
4      case 0 do
5          rect ← RectangleSepI(N, matriz, 0, 0)          /* Alg.5.8 */
6      case 1 do
7          rect ← RectangleSepR(N, matriz, 0, 0)          /* Alg.5.14 */
8      case 2 do
9          rect ← RectangleSepI(N, matriz, 0, 1)          /* Alg.5.8 */
10     case 3 do
11         rect ← RectangleSepR(N, matriz, 0, 1)          /* Alg.5.14 */
12     case 4 do
13         rect ← RectangleSepI(N, matriz, 1, 0)          /* Alg.5.8 */
14     case 5 do
15         rect ← RectangleSepR(N, matriz, 1, 0)          /* Alg.5.14 */
16     case 6 do
17         rect ← RectangleSepI(N, matriz, 1, 1)          /* Alg.5.8 */
18     case 7 do
19         rect ← RectangleSepR(N, matriz, 1, 1)          /* Alg.5.14 */
20 return rect
```

5.7. Solución del caso propuesto

A continuación, se exponen varios casos prácticos relacionados con el problema que se ha desarrollado en los capítulos anteriores (Figura 5.9). Las soluciones finales se obtienen a través del Algoritmo 5.16 y Algoritmo 5.17 y se ilustran de forma gráfica mediante figuras.

Algoritmo 5.16. SolutionSep(N, distancia, matriz, fun, upd, conv, ver)
Algoritmo 5.17. RectangleSep(N, matriz, fun, upd, ver)

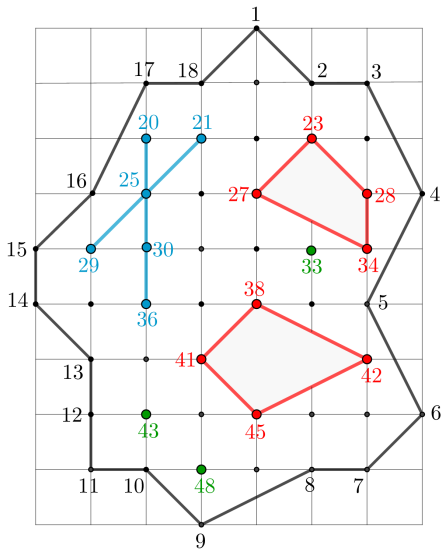


Figura 5.9: Problema inicial

Para emplear estos algoritmos, es imprescindible utilizar los conjuntos descritos a continuación y establecer el conjunto de obstáculos sobre el que se calculará la solución:

$$\begin{bmatrix} Points = P = \{p_1, \ldots, p_{49}\} \\ Polygon = \partial P_0 = \{p_1, \ldots, p_{18}\} \\ PointsP = \{p_{33}, p_{43}, p_{48}\} \\ SegmentsP = \{S_1, S_2\}, \text{ donde } S_1 = \{p_{20}, p_{25}, p_{30}, p_{36}\}, S_2 = \{p_{21}, p_{25}, p_{29}\} \\ HolesP = \{H_1, H_2\}, \text{ donde } H_1 = \{p_{23}, p_{28}, p_{34}, p_{27}\}, H_2 = \{p_{38}, p_{42}, p_{45}, p_{41}\} \end{bmatrix}$$

5.7.1. $O = \{43, 48, H_2\}$ $ContP = \{43, 48\}$, $ContS = \{\emptyset\}$, $ContH = \{H_2\}$

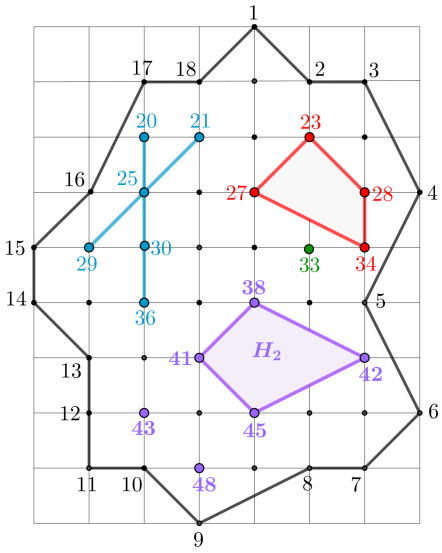
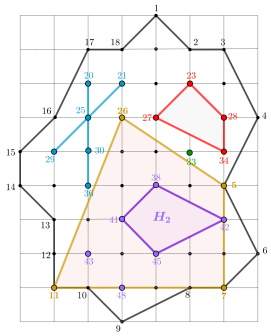
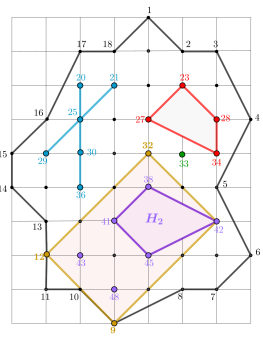


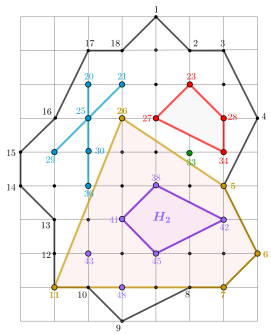
Figura 5.10: Conjunto $O = \{43, 48, H_2\}$



(a) Cuadrilátero simple, convexo

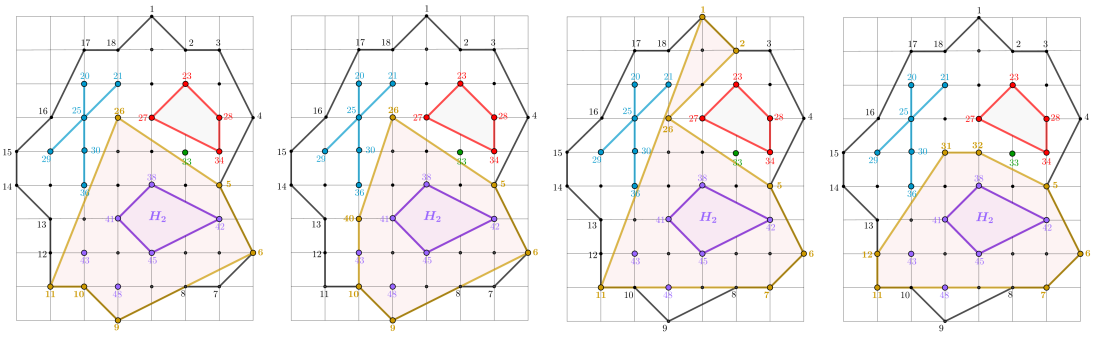


(b) Rectángulo



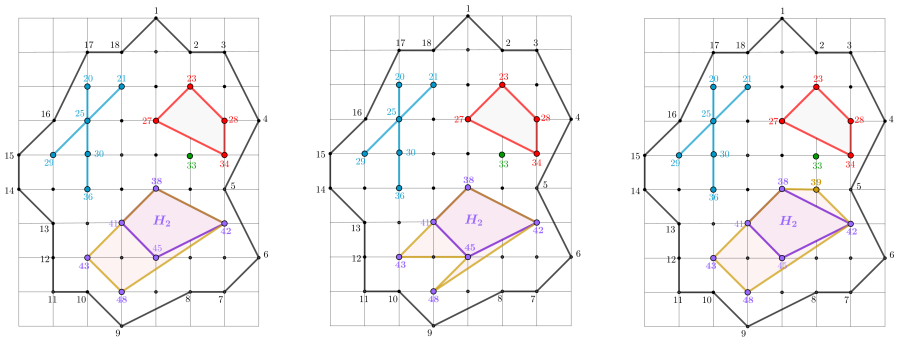
(c) Pentágono simple, convexo

Figura 5.11: Solución mayor área, $O = \{43, 48, H_2\}$

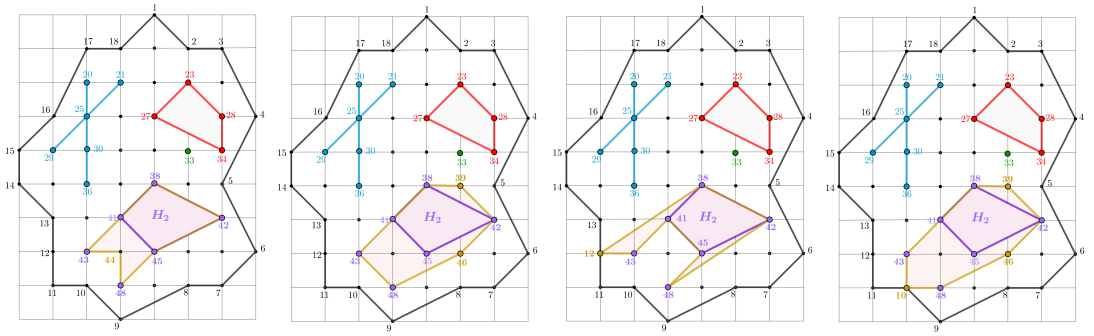


(d) Hexágono (e) Hexágono convexo (f) Heptágono (g) Heptágono convexo

Figura 5.11: Solución mayor área, $O = \{43, 48, H_2\}$ (cont.)



(a) Cuadrilátero simple, convexo (b) Pentágono (c) Pentágono convexo



(d) Hexágono (e) Hexágono convexo (f) Heptágono (g) Heptágono convexo

Figura 5.12: Solución menor área, $O = \{43, 48, H_2\}$

5.7.2. $O = \{33, H_1\}$ $ContP = \{33\}$, $ContS = \{\emptyset\}$, $ContH = \{H_1\}$

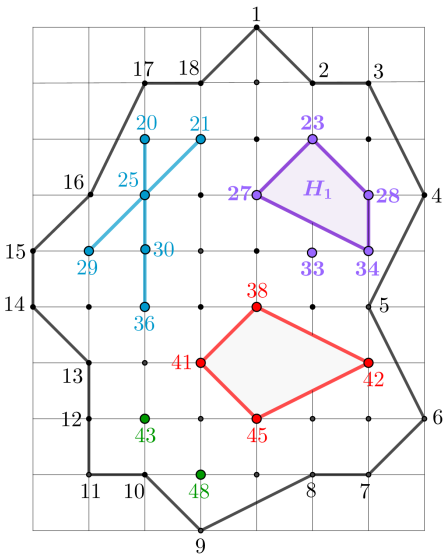
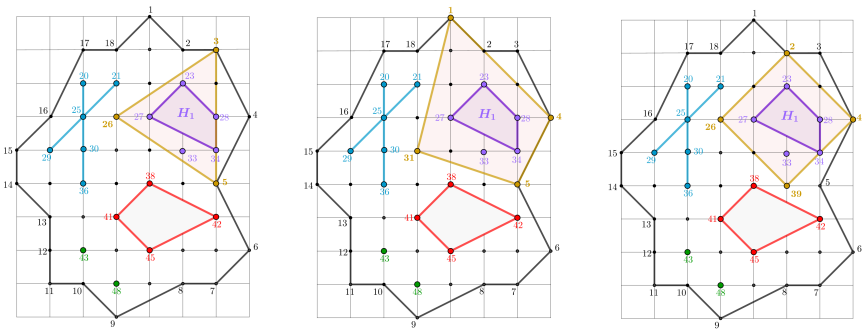
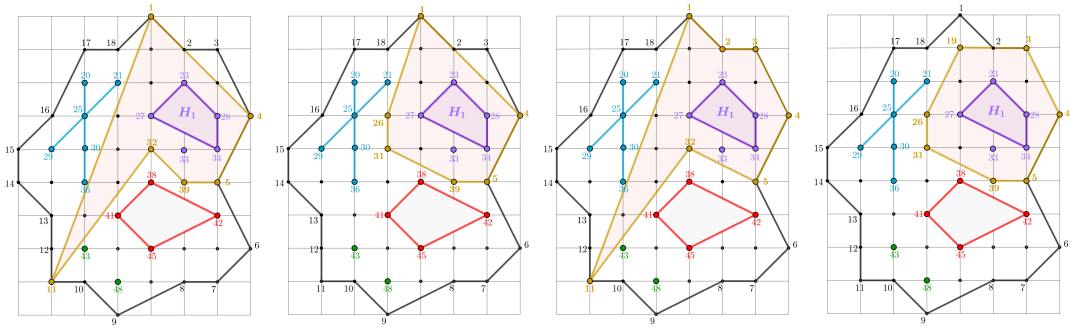


Figura 5.13: Conjunto $O = \{33, H_1\}$



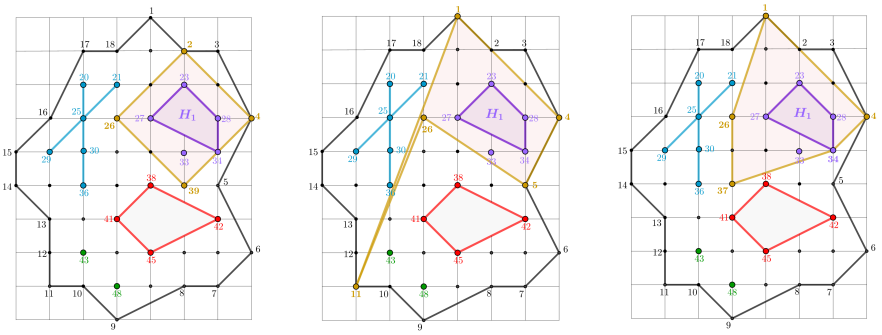
(a) Triángulo (b) Cuadrilátero simple, convexo (c) Rectángulo

Figura 5.14: Solución mayor área, $O = \{33, H_1\}$

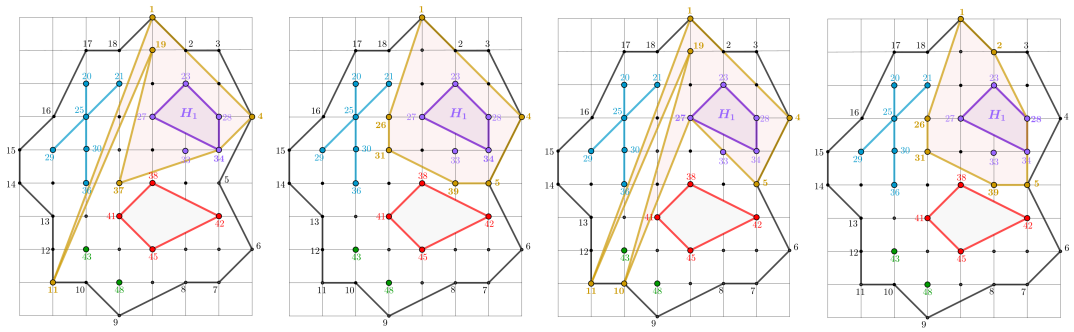


(d) Hexágono (e) Hexágono convexo (f) Heptágono (g) Heptágono convexo

Figura 5.14: Solución mayor área, $O = \{33, H_1\}$ (cont.)



(a) Rectángulo (b) Pentágono (c) Pentágono convexo



(d) Hexágono (e) Hexágono convexo (f) Heptágono (g) Heptágono convexo

Figura 5.15: Solución mayor perímetro, $O = \{33, H_1\}$

5.7.3. $O = \{33, H_1, H_2\}$ $ContP = \{33\}$, $ContS = \{\emptyset\}$, $ContH = \{H_1, H_2\}$

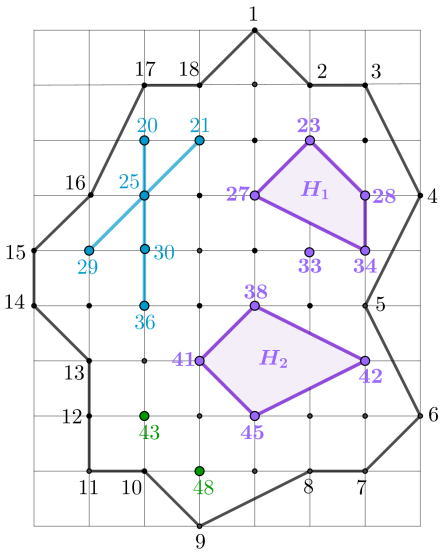
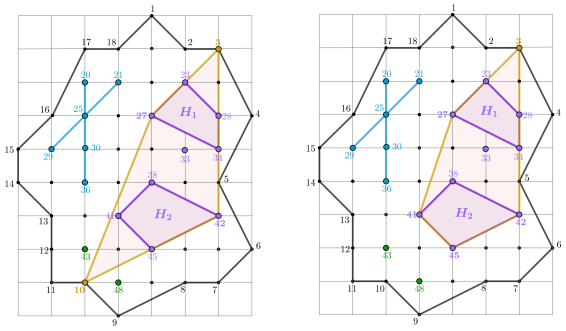


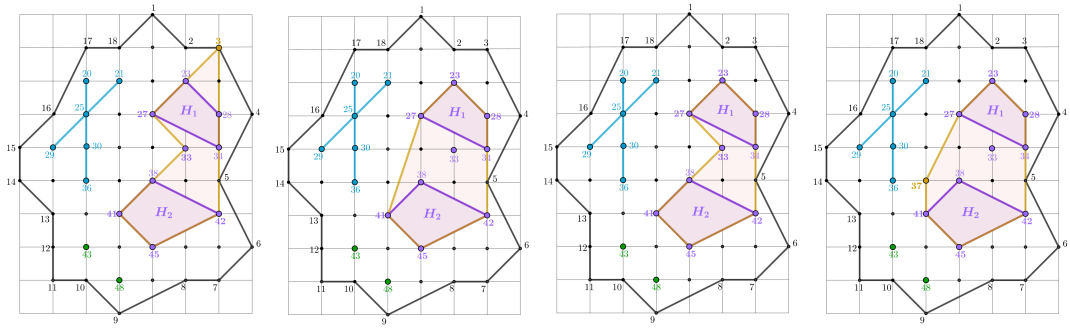
Figura 5.16: Conjunto $O = \{33, H_1, H_2\}$



(a) Cuadrilátero simple, convexo

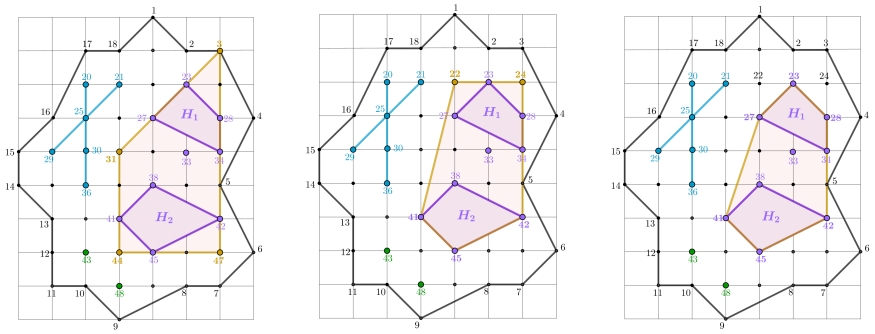
(b) Pentágono simple, convexo

Figura 5.17: Solución menor área, $O = \{33, H_1, H_2\}$

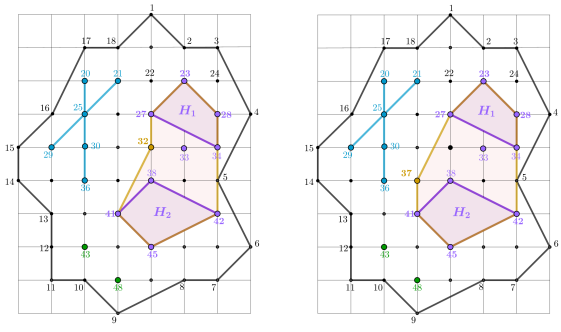


(c) Hexágono (d) Hexágono convexo (e) Heptágono (f) Heptágono convexo

Figura 5.17: Solución menor área, $O = \{33, H_1, H_2\}$ (cont.)



(a) cuadrilátero simple, convexo (b) Pentágono simple, convexo (c) Hexágono simple, convexo



(d) Heptágono (e) Heptágono convexo

Figura 5.18: Solución menor perímetro, $O = \{33, H_1, H_2\}$

5.7.4. $O = \{43, S_1, S_2\}$ $ContP = \{43\}$, $ContS = \{S_1, S_2\}$, $ContP = \{\emptyset\}$

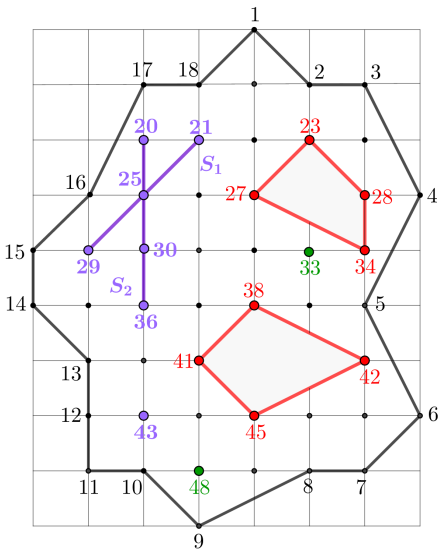
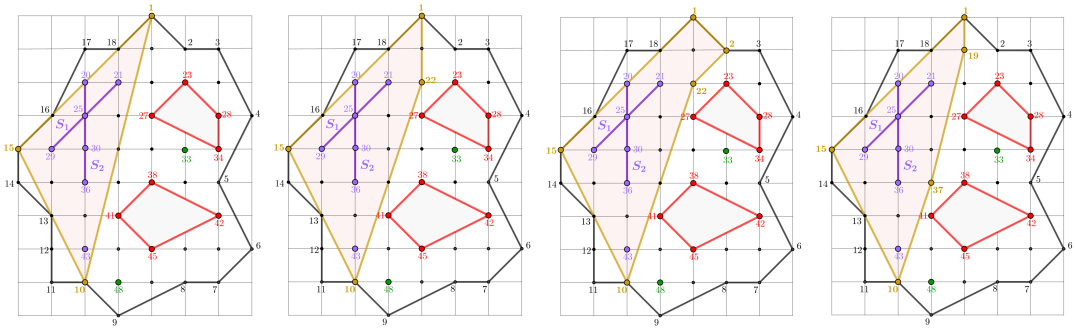
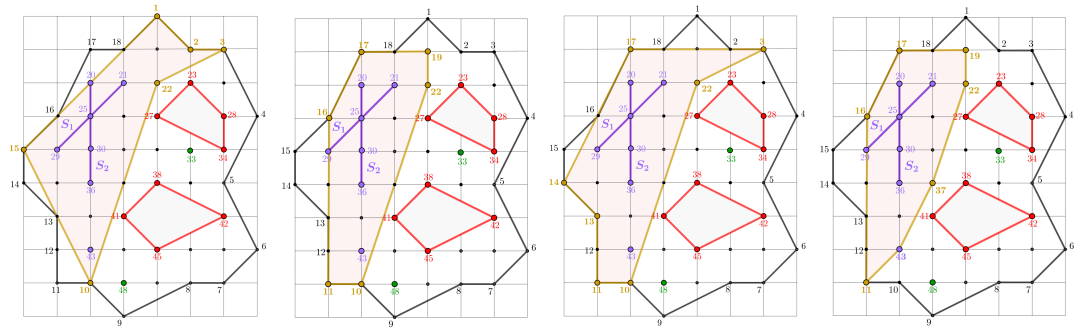


Figura 5.19: Conjunto $O = \{43, S_1, S_2\}$



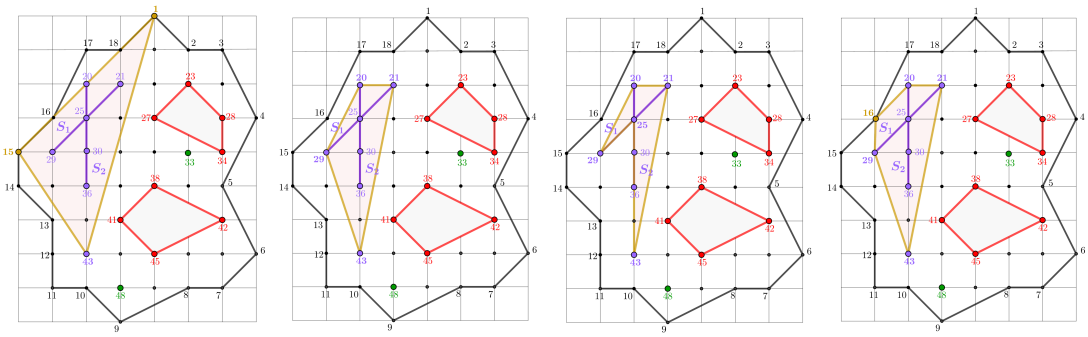
(a) Triángulo (b) Cuadrilátero simple, convexo (c) Pentágono (d) Pentágono convexo

Figura 5.20: Solución mayor área, $O = \{43, S_1, S_2\}$

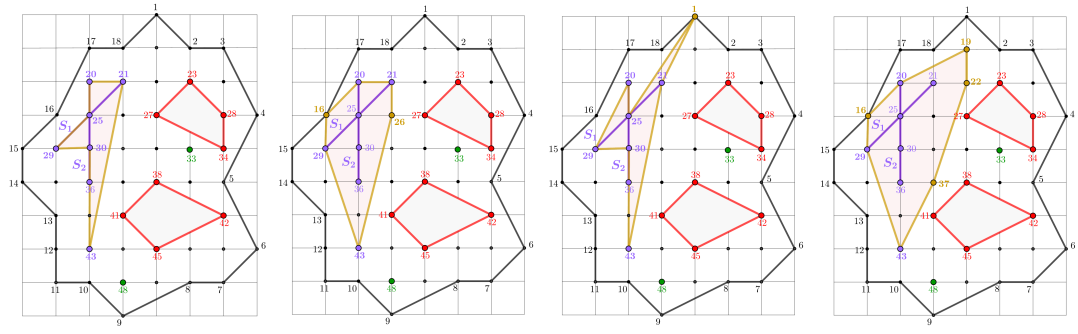


(e) Hexágono (f) Hexágono convexo (g) Heptágono (h) Heptágono convexo

Figura 5.20: Solución mayor área, $O = \{43, S_1, S_2\}$ (cont.)



(a) Triángulo (b) Cuadrilátero simple, convexo (c) Pentágono (d) Pentágono convexo



(e) Hexágono (f) Hexágono convexo (g) Heptágono (h) Heptágono convexo

Figura 5.21: Solución menor área, $O = \{43, S_1, S_2\}$

5.8. Cálculo del polígono simple de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en una ROI con obstáculos arbitrarios

En la Sección 4.9, se describe un procedimiento para calcular el polígono simple de k lados de mayor área o perímetro contenido en una ROI con obstáculos arbitrarios. Este cálculo se lleva a cabo mediante la creación de una sucesión de particiones regulares cada vez más finas, $L_{i+1} = L_i/2$ y $\Pi_i \preceq \Pi_{i+1}$ para todo i. En particular, se generan tres particiones con tamaños de partición: $L_1 = 1$, $L_2 = 1/2$, $L_3 = 1/4$, número de puntos: 49, 197, 804 y matriz de adyacencia: A, $\dot{A}$, $\ddot{A}$. Posteriormente, se aplican los Algoritmos 4.22 y 4.23 para calcular varios tipos de soluciones relacionadas con el polígono de mayor área o perímetro.

Algoritmo 4.22. SolutionInc(N, distancia, matriz, fun, conv, ver)
Algoritmo 4.23. RectangleInc(N, matriz, fun, ver)

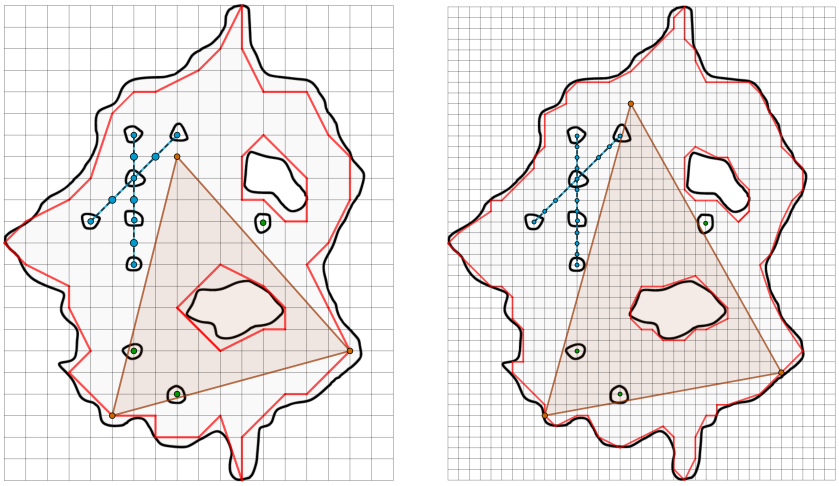
Si se aplica un procedimiento similar al caso en el que el polígono simple de k lados a calcular pueda ser con el menor área o perímetro que contenga a un conjunto dado O, se obtienen las soluciones que se pueden encontrar en la Sección 5.8.1 para el cálculo del mayor área y la Sección 5.8.2 para el cálculo del menor área.

Además, la Sección 5.8.3 proporciona un resumen en la Tabla 5.1 donde se comparan las áreas obtenidas al aplicar los Algoritmos 5.16 y 5.17. Esta tabla permite observar cómo, al realizar particiones regulares cada vez más finas, el área máxima de los polígonos simples o convexos tiende a aumentar, mientras que el área mínima tiende a disminuir.

Algoritmo 5.16. SolutionSep(N, distancia, matriz, fun, upd, conv, ver)
Algoritmo 5.17. RectangleSep(N, matriz, fun, upd, ver)

Los resultados obtenidos tienen implicaciones importantes cuando se trata de buscar polígonos con diferentes números de lados, como triángulos, cuadriláteros, pentágonos y otros, en una Región de Interés (ROI) con obstáculos arbitrarios. La capacidad de ajustar la precisión de la aproximación mediante particiones regulares proporciona una herramienta poderosa para resolver problemas geométricos en la ROI con obstáculos arbitrarios, sin importar la forma o el número de lados del polígono buscado.

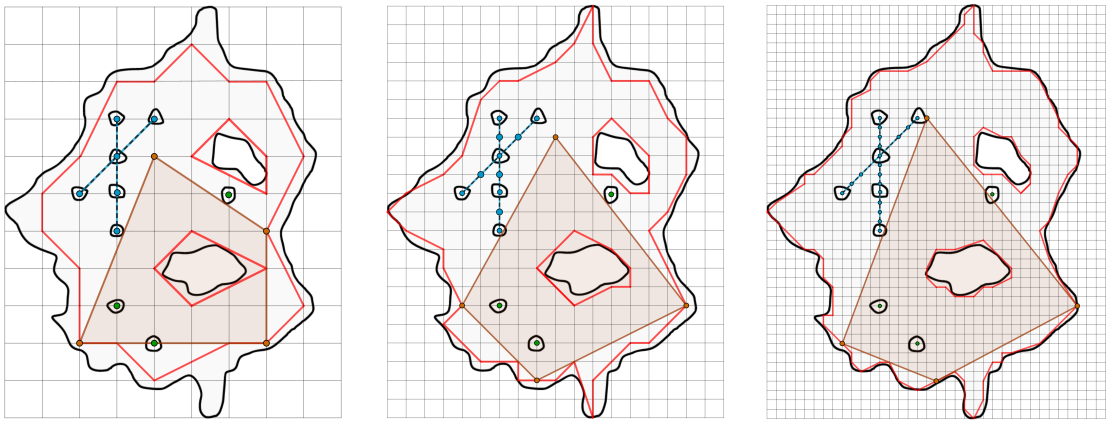
5.8.1. $O = \{43, 48, H_2\}$. Mayor área



(a) SolutionSep(197, 2, $\dot{A}$, 0, 0, conv, ver), Área = 15,38

(b) SolutionSep(804, 2, $\ddot{A}$, 0, 0, conv, ver), Área = 18,94

Figura 5.22: $O = \{43, 48, H_2\}$. Triángulo de mayor área con diferentes tamaños de partición

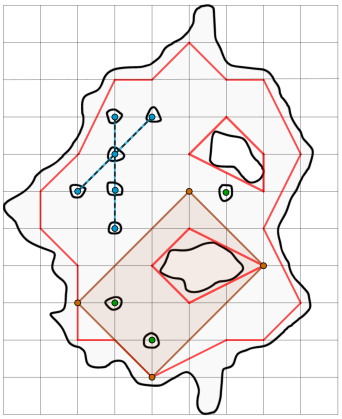


(a) SolutionSep(49, 3, A, 0, 0, conv, ver), Área = 17

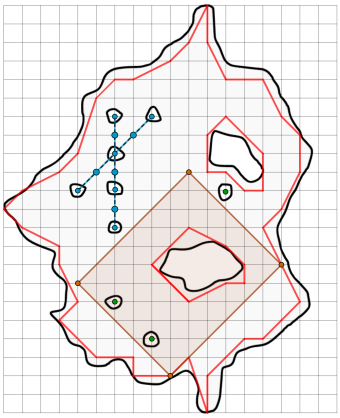
(b) SolutionSep(197, 3, $\dot{A}$, 0, 0, conv, ver), Área = 19,50

(c) SolutionSep(804, 3, $\ddot{A}$, 0, 0, conv, ver), Área = 22

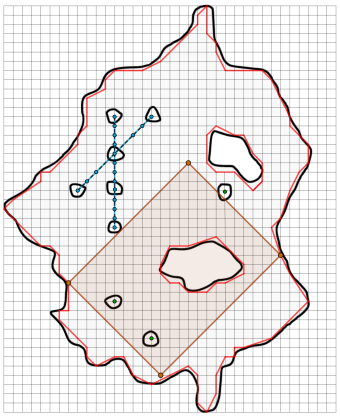
Figura 5.23: $O = \{43, 48, H_2\}$. Cuadrilátero simple y convexo de mayor área con diferentes tamaños de partición



(a) RectangleSep(49, A, 0, 0, ver), Área = 12



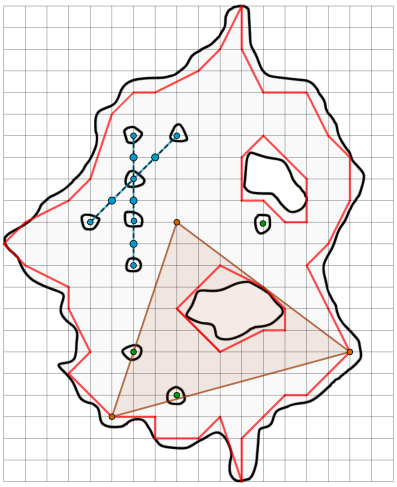
(b) RectangleSep(197, $\dot{A}$, 0, 0, ver), Área = 15



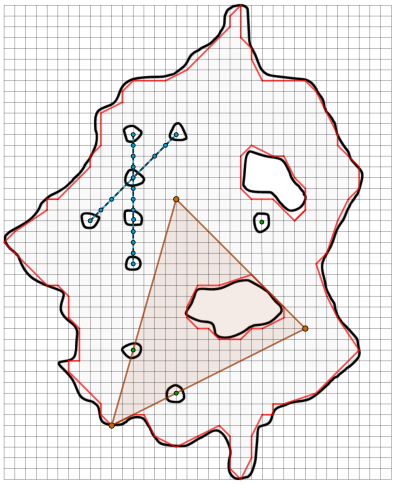
(c) RectangleSep(804, $\ddot{A}$, 0, 0, ver), Área = 16,25

Figura 5.24: $O = \{43, 48, H_2\}$. Rectángulo de mayor área con diferentes tamaños de partición

5.8.2. $O = \{43, 48, H_2\}$. Menor área

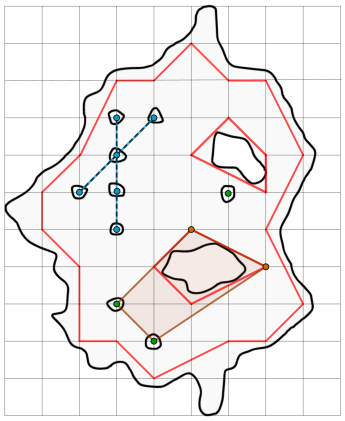


(a) SolutionSep(197, 2, $\dot{A}$, 0, 1, 0, ver), Área = 11,25

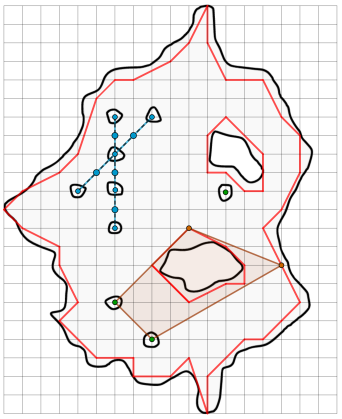


(b) SolutionSep(804, 2, $\ddot{A}$, 0, 1, 0, ver), Área = 10,13

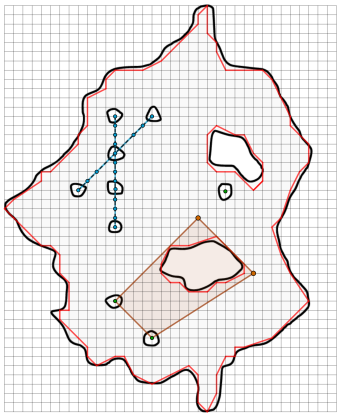
Figura 5.25: $O = \{43, 48, H_2\}$. Triángulo de menor área con diferentes tamaños de partición



(a) SolutionSep(49, 3, A, 0, 1, conv, ver), Área = 5,50

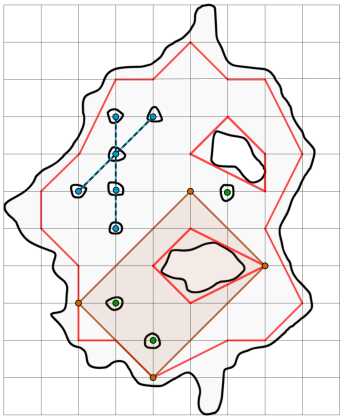


(b) SolutionSep(197, 3, $\dot{A}$, 0, 1, conv, ver), Área = 6,25

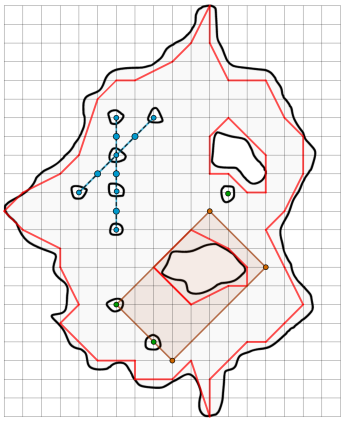


(c) SolutionSep(804, 3, $\ddot{A}$, 0, 1, conv, ver), Área = 5,63

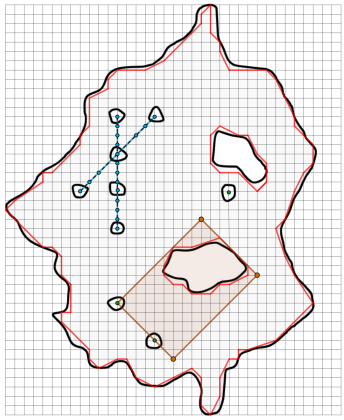
Figura 5.26: $O = \{43, 48, H_2\}$. Cuadrilátero simple y convexo de menor área con diferentes tamaños de partición



(a) RectangleSep(49, A, 0, 1, ver), Área = 12



(b) RectangleSep(197, $\dot{A}$, 0, 1, ver), Área = 7,50



(c) RectangleSep(804, $\ddot{A}$, 0, 1, ver), Área = 6,75

Figura 5.27: $O = \{43, 48, H_2\}$. Rectángulo de menor área con diferentes tamaños de partición

5.8.3. $O = \{43, 48, H_2\}$. Comparativa área con diferentes tamaños de partición

Tabla 5.1: $O = \{43, 48, H_2\}$. Comparativa mayor y menor área

Figura	Algoritmo	L_1	L_2	L_3
	Máximo área			
5.22	5.16. SolutionSep(N, 2, matriz, 0, 0, 0, ver)	—	15,38	18,94
5.23	5.16. SolutionSep(N, 3, matriz, 0, 0, 0, ver)	17	19,50	22
5.23	5.16. SolutionSep(N, 3, matriz, 0, 0, 1, ver)	17	19,50	22
5.24	5.17. RectangleSep(N, matriz, 0, 0, ver)	12	15	16,25
	Mínimo área			
5.25	5.16. SolutionSep(N, 2, matriz, 0, 1, 0, ver)	—	11,25	10,13
5.26	5.16. SolutionSep(N, 3, matriz, 0, 1, 0, ver)	5,50	6,25	5,63
5.26	5.16. SolutionSep(N, 3, matriz, 0, 1, 1, ver)	5,50	6,25	5,63
5.27	5.17. RectangleSep(N, matriz, 0, 1, ver)	12	7,50	6,75

Capítulo 6

Problemas de la envolvente, $Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$

Durante la revisión de la bibliografía del Capítulo 2, centrada en los problemas de *"Enclosure problems"*, se identificó una falta de investigaciones tratando estos problemas de manera completa. La mayoría de los estudios se limitaban a ciertos tipos de polígonos, como triángulos, rectángulos o paralelogramos, sin tener en cuenta formas o figuras más complejas. Además, se observó una notable ausencia de soluciones propuestas para calcular el perímetro mínimo o para el caso de obstáculos arbitrarios.

Este capítulo tiene dos objetivos principales. En primer lugar, resolver el problema de la envolvente $Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$. Este problema consiste en calcular el polígono simple de k lados de menor área o perímetro que contiene a otro polígono simple entre obstáculos arbitrarios como puntos, segmentos y agujeros. En segundo lugar, extender el caso en el que el polígono simple de k lados contenga una Región de Interés (ROI) entre obstáculos arbitrarios.

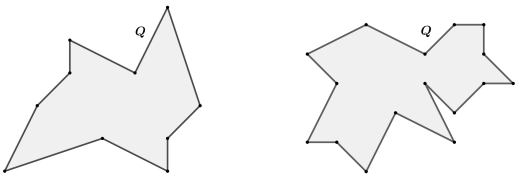
Al tratar este problema, se logra resolver de manera simultánea todos los casos a partir del valor k elegido por el usuario. Además, se ofrece una solución al problema planteado en la Sección 1.1, el cual está relacionado con el objetivo parcial 4.

Industria textil. Se dispone de una pieza de tela y se desea cortar un patrón específico de cualquier número de lados, ¿cómo se podría hacer de manera eficiente de tal forma que se minimice el área utilizada en el proceso de corte, permitiendo de esta forma reducir el desperdicio de tela?. Si además, ahora la pieza de tela tiene defectos o imperfecciones, tales como errores en su textura, color, forma o tamaño que afectan a su calidad o apariencia, ¿cómo se podría resolver el problema?

Objetivo 4. Dado un valor k fijo, calcular el polígono simple de k lados de menor área o perímetro que contenga a una ROI entre obstáculos arbitrarios.

6.1. Clasificación de problemas

En este capítulo, se resuelven dos tipos de problemas distintos pero interrelacionados. El primero de ellos, **Problema 4.1**, se centra en el cálculo del polígono simple de k lados de menor área o perímetro que contiene a un polígono reticulado (Q) (Figura 6.1). El objetivo es proporcionar una solución efectiva para este problema específico, que complemente la literatura presentada en el Capítulo 2 (Tabla 6.1).



(a) Ejemplo 1 (b) Ejemplo 2

Figura 6.1: Problema 4.1. Planteamiento

Tabla 6.1: Bibliografía: Enclosure problems

Autor	Condición 1–3	Condición 5
[135]	$Enc(\mathcal{P}_{conv}, \mathcal{P}_3, \text{área})$	$O(n)$
[23]	$Enc(\mathcal{P}_{conv}, \mathcal{P}_3, \text{perímetro})$	$O(n)$
[156, 163, 164]	$Enc(\mathcal{P}_{conv}, \mathcal{P}_{paral}, \text{área})$	$O(n)$
[64]	$Enc(\mathcal{P}_{sim}, \mathcal{P}_{rect}, \text{área})$	$O(n^2)$
[2]	$Enc(\mathcal{P}_{sim}, \mathcal{P}_{k\,conv}, \text{área})$	$O(n^2 \log n \log k)$
[117]	$Enc(\mathcal{P}_{conv}, \mathcal{P}_{k\,conv}, \text{perímetro})$	$O(nk \log k)$
Tesis Doctoral	$Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$	$O(n^5 k)$, Iterativo $O(n^k)$, Recursivo

Esta solución también sienta las bases para estudiar el segundo tipo de problema que se presenta en este capítulo. Este segundo problema, **Problema 4.2**, guarda relación con el tema tratado en la Sección 1.1: cálculo del polígono simple de k lados de menor área o perímetro que contiene a una ROI entre obstáculos arbitrarios (Figura 6.2). Para lograr este objetivo, se resolverá el problema planteado en la Sección 1.1, llamado **Industria textil**.

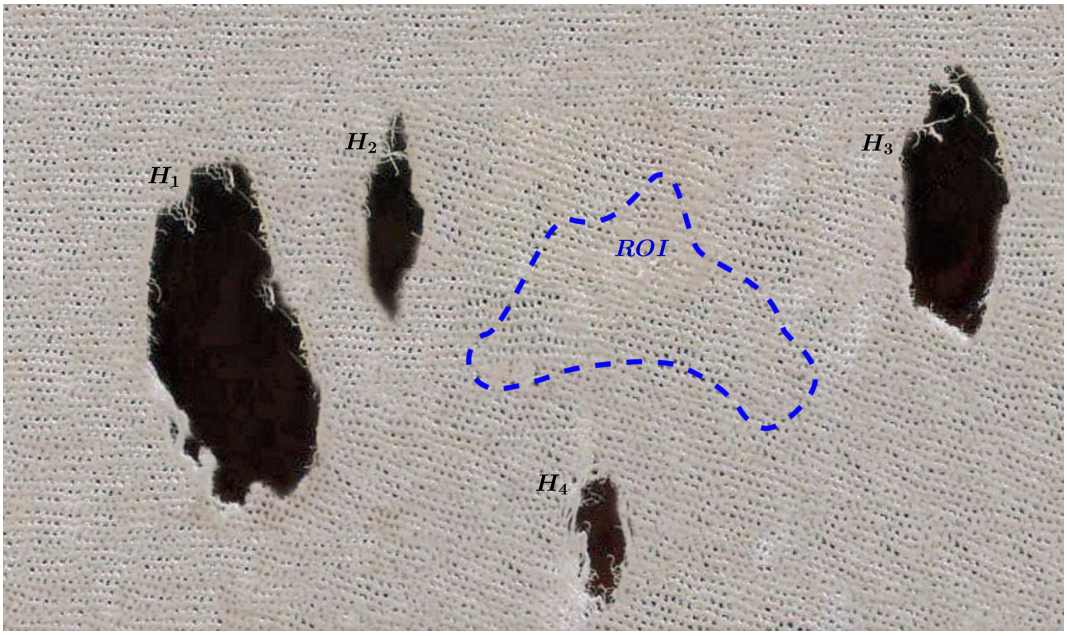


Figura 6.2: Problema 4.2. Planteamiento

6.2. Solución de los problemas

La solución de los Problemas 4.1 y 4.2 se obtienen directamente del problema resuelto en el Capítulo 5, establecido como objetivo parcial 3,

Objetivo 3. Dado un valor k fijo y un conjunto O contenido en una ROI con obstáculos arbitrarios, calcular el polígono simple de k lados de mayor o menor área o perímetro, que contenga a O y esté dentro de la ROI.

Así, cualquier problema relacionado con la envolvente es en realidad un problema de separabilidad. En consecuencia, se emplearán los algoritmos previamente presentados en el mencionado capítulo (ver *Sección 5.6. Solución general*).

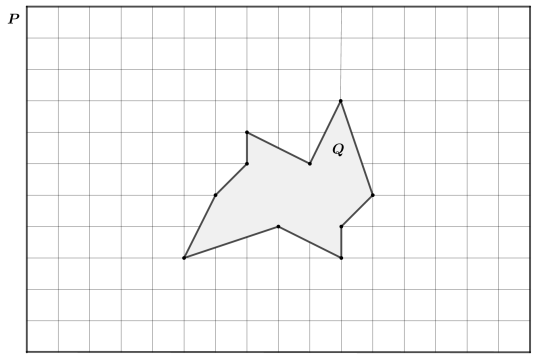
Algoritmo 5.16. SolutionSep(N, distancia, matriz, fun, 1, conv, ver)
Algoritmo 5.17. RectangleSep(N, matriz, fun, 1, ver)

El parámetro *upd* se ha establecido con el valor *upd = 1* ya que se busca calcular el menor área o perímetro.

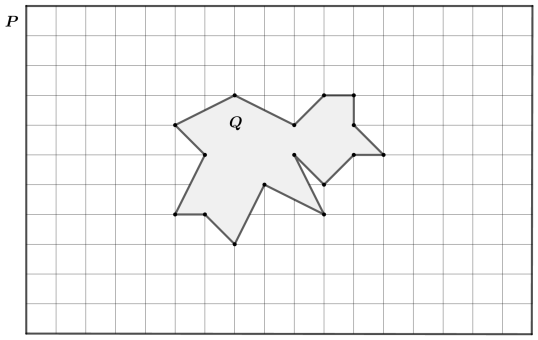
6.2.1. Problema 4.1

El proceso para determinar las soluciones se llevará a cabo siguiendo los pasos que se detallan a continuación:

1⌋ Construir el polígono reticulado P a partir de una partición regular $\Pi = \Pi_x \times \Pi_y$ con tamaño de partición L (Figura 6.3).



(a) Ejemplo 1



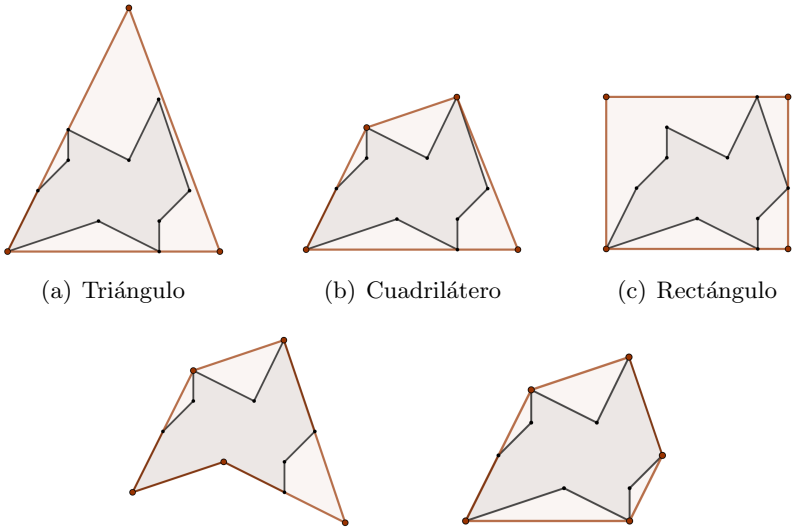
(b) Ejemplo 2

Figura 6.3: Problema 4.1: Polígono reticulado P

2⌋ Considerar los conjuntos (ver Sección 5.1),

$$\left[\begin{array}{l} Points = P = P_0 - \imath Q \\ Polygon = \partial P_0 \\ PointsP = \emptyset \\ SegmentsP = \emptyset \\ HolesP = Q \\ O = \{ContP,\ ContS,\ ContH\},\ \text{donde: } ContP = \emptyset,\ ContS = \emptyset,\ ContH = Q \end{array}\right.$$

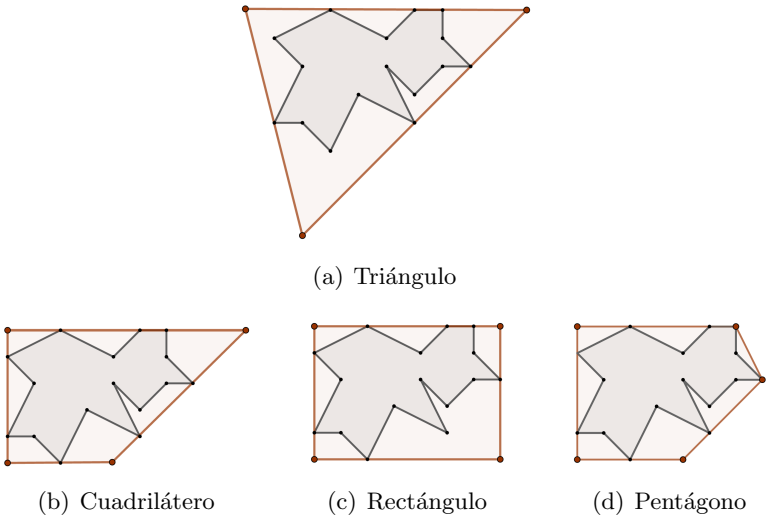
3⌋ Aplicar los Algoritmos 5.16 y 5.17.



(a) Triángulo (b) Cuadrilátero (c) Rectángulo

(d) Pentágono menor área (e) Pentágono menor perímetro

Figura 6.4: Problema 4.1: Ejemplo 1. Menor área y perímetro



(a) Triángulo

(b) Cuadrilátero (c) Rectángulo (d) Pentágono

Figura 6.5: Problema 4.1: Ejemplo 2. Menor área

6.2.2. Problema 4.2

Siguiendo un procedimiento semejante al realizado para el Problema 4.1, el proceso consta de las siguientes tres etapas.

1⌋ Construir el polígono reticulado P a partir de una partición regular $\Pi = \Pi_x \times \Pi_y$ con tamaño de partición L (Figura 6.6).

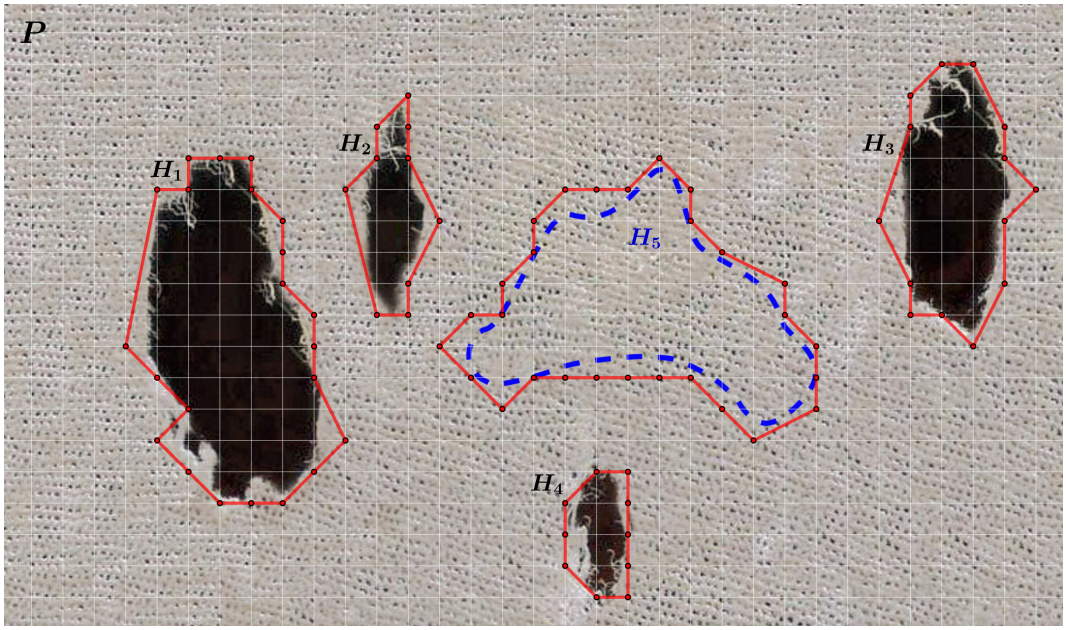


Figura 6.6: Problema 4.2: Polígono reticulado P

2⌋ Considerar los conjuntos (ver Sección 5.1),

$$\left[\begin{array}{l} Points = P = P_0 - \bigcup_{i=1}^{5}(\imath H_i) \\ Polygon = \partial P_0 \\ PointsP = \emptyset \\ SegmentsP = \emptyset \\ HolesP = \{H_1, H_2, H_3, H_4, ROI\} = \{H_1, H_2, H_3, H_4, H_5\} \\ O = \{ContP,\ ContS,\ ContH\},\ \text{donde: } ContP = \emptyset,\ ContS = \emptyset,\ ContH = H_5 \end{array}\right.$$

3⌋ Aplicar los Algoritmos 5.16 y 5.17.

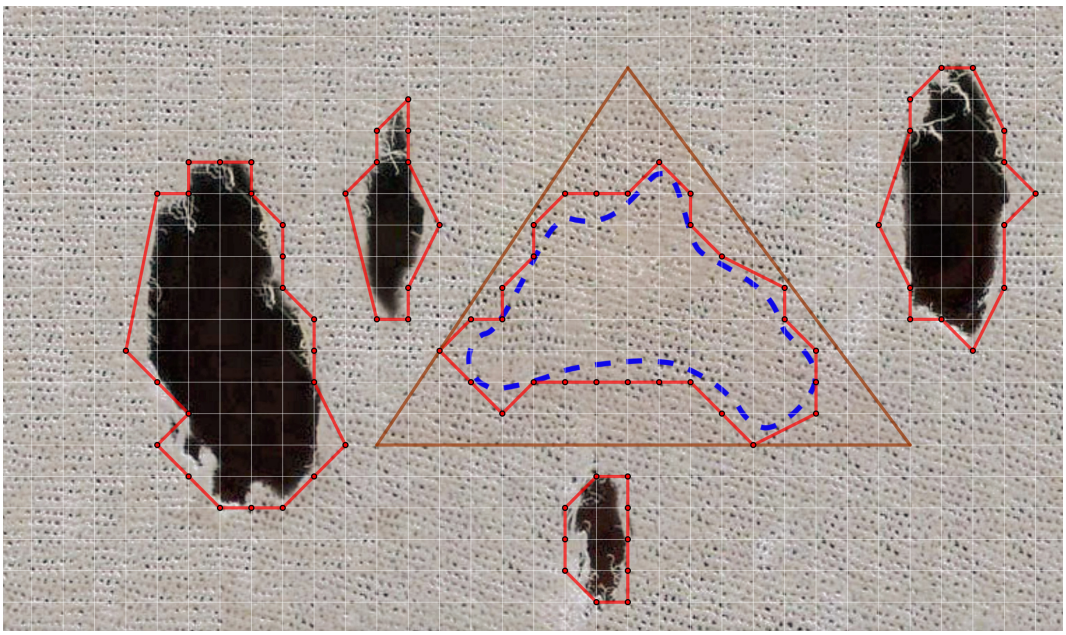


Figura 6.7: Problema 4.2: Triángulo menor área

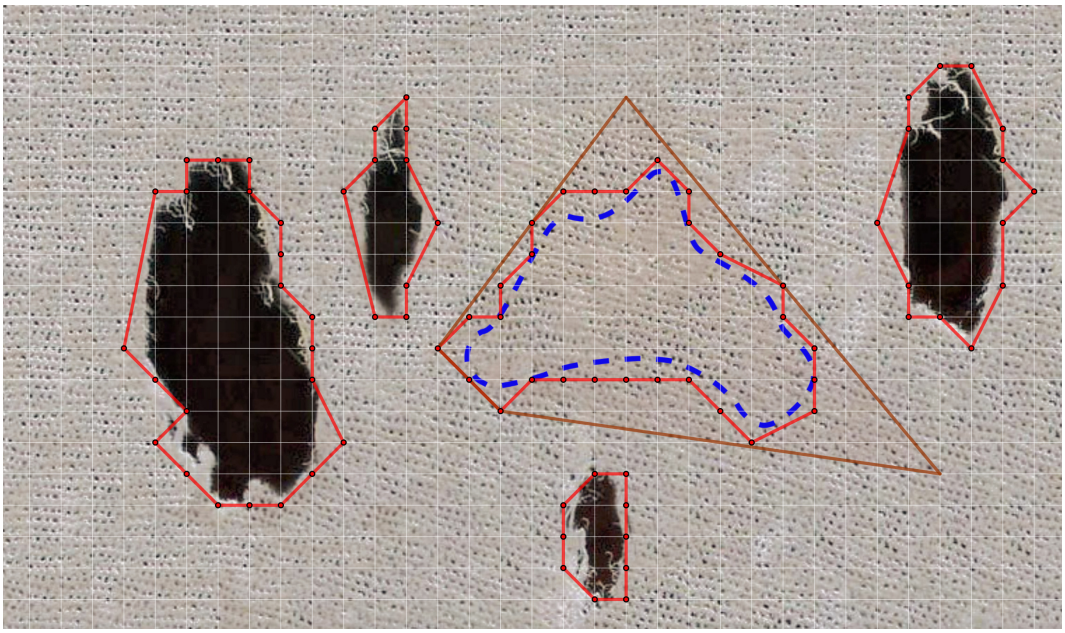


Figura 6.8: Problema 4.2: Cuadrilátero menor área

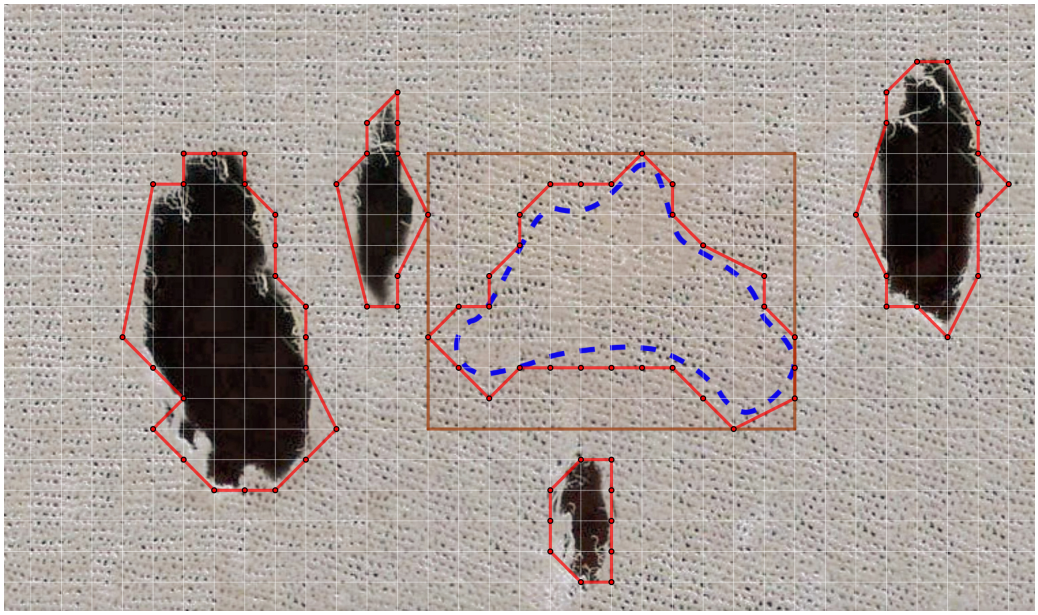


Figura 6.9: Problema 4.2: Rectángulo menor área

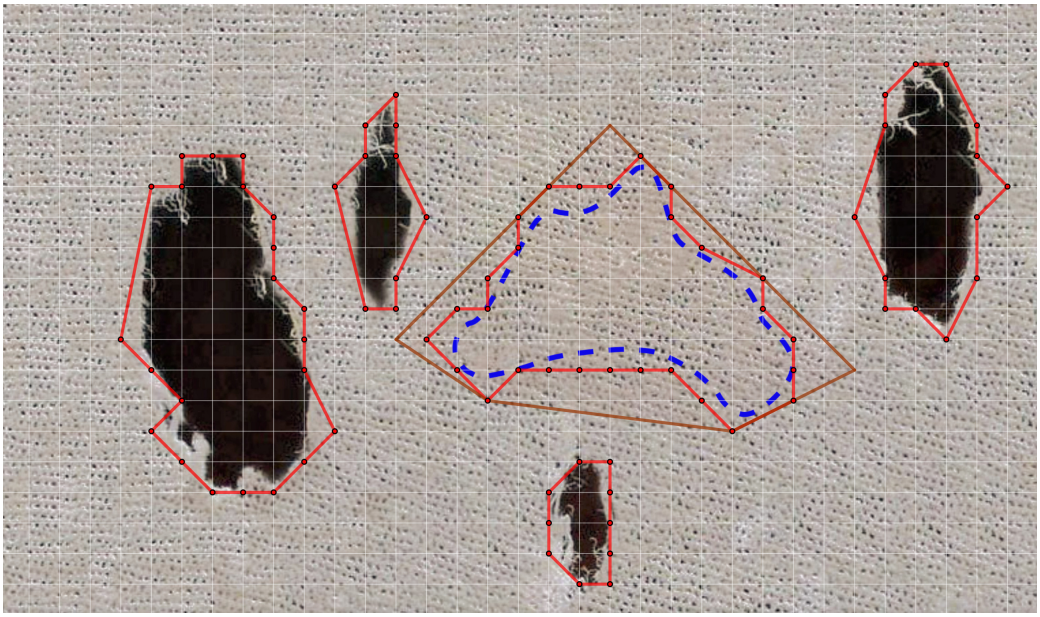


Figura 6.10: Problema 4.2: Pentágono menor área

Capítulo 7

Unificación de problemas: Inclusión, Separabilidad y Envolvente

En este capítulo, se introducen los dos algoritmos clave de esta Tesis Doctoral: el Algoritmo 7.1 y el Algoritmo 7.2. Estos algoritmos unifican en un solo procedimiento todas las soluciones tratadas en los capítulos anteriores relacionados con los problemas de inclusión, separabilidad y envolvente.

A través de estos algoritmos, es posible obtener soluciones como calcular el pentágono simple de mayor área contenido en un polígono reticulado, calcular el heptágono convexo de menor área que contiene a una región de interés o calcular el cuadrilátero de mayor perímetro contenido en un conjunto O y que contiene a un polígono reticulado. Cada uno de estos cálculos puede llevarse a cabo tanto en su versión iterativa como en la recursiva.

Como se demostró al resolver los Problemas 4.1 y 4.2 del Capítulo 6, cualquier problema de la envolvente puede considerarse un problema de separabilidad. Además, se puede afirmar que todo problema de inclusión se reduce a un problema de separabilidad. Basta con suponer que el conjunto $O = \emptyset$ y calcular el área o perímetro máximo, para transformar la solución de un problema planteado en las condiciones del Capítulo 5 en uno del Capítulo 4. En consecuencia, se establece la siguiente correspondencia entre problemas:

$$\left[\begin{array}{l} \text{Si } O = \emptyset,\ upd = 0, \text{ entonces } \ Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu) = Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu) \\ \\ \text{Si } O = \{Q\},\ upd = 1, \text{ entonces } \ Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu) = Env(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)) \end{array}\right.$$

7.1. Algoritmo 7.1. Solution(N, distancia, matriz, fun, upd, conv, ver, prob)

El **Algoritmo 7.1. Solution(N, distancia, matriz, fun, upd, conv, ver, prob)** calcula el polígono simple o convexo de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado P con obstáculos arbitrarios. Consta de ocho parámetros, los siete primeros comunes con el **Algoritmo 5.16. SolutionSep(N, distancia, matriz, fun, upd, conv, ver)**. El último, *prob*, se utiliza para determinar si se va a resolver un problema de inclusión ($prob = 0$) o un problema de separabilidad ($prob = 1$). El coste computacional para la versión iterativa y recursiva es $O(n^5k)$ y $O(n^k)$, respectivamente.

Los parámetros fun, upd, $conv$, ver y $prob$ permiten 32 variantes del problema. No obstante, en la práctica, se identifican 24 combinaciones únicas. Esto se debe a que cuando se decide resolver un problema de inclusión ($prob = 0$), la única opción válida es determinar si se calculará el máximo área o perímetro ($upd = 0$). Esta restricción reduce el número total de combinaciones posibles a 24, como se muestra en la Tabla 7.1.

Teniendo en cuenta lo anterior, cada valor decimal del parámetro *sum*, con $sum = 16fun + 8upd + 4conv + 2ver + prob$, se puede relacionar con un tipo de solución de los Algoritmos 4.22 y 5.16 (Tabla 7.2), lo que permite obtener cualquier solución para resolver un problema específico. Todas estas soluciones, que dependen de estos cinco parámetros, pueden resumirse en el pseudocódigo del Algoritmo 7.1.

Tabla 7.1: Algoritmo 7.1: Combinaciones posibles de las variables fun, upd, conv, ver, prob

fun	upd	conv	ver	prob	sum
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7

(Continúa)

fun	*upd*	*conv*	*ver*	*prob*	sum
0	1	0	0	1	9
0	1	0	1	1	11
0	1	1	0	1	13
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18
1	0	0	1	1	19
1	0	1	0	0	20
1	0	1	0	1	21
1	0	1	1	0	22
1	0	1	1	1	23
1	1	0	0	1	25
1	1	0	1	1	27
1	1	1	0	1	29
1	1	1	1	1	31

Tabla 7.2: Algoritmos de inclusión y separabilidad para polígonos simples y convexos

conv	**ver**	**Alg 4.22. SolutionInc(N, distancia, matriz, fun, conv, ver)**
0	0	Alg. 4.10. SolutionIncI(N, distancia, matriz, fun)
1	0	Alg. 4.15. SolutionConvexIncI(N, distancia, matriz, fun)
0	1	Alg. 4.18. SolutionIncR(N, distancia, matriz, fun)
1	1	Alg. 4.21. SolutionConvexIncR(N, distancia, matriz, fun)
conv	**ver**	**Alg. 5.16. SolutionSep(N, distancia, matriz, fun, upd, conv, ver)**
0	0	Alg. 5.6. SolutionSepI(N, distancia, matriz, fun, upd)
1	0	Alg. 5.9. SolutionConvexSepI(N, distancia, matriz, fun, upd)
0	1	Alg. 5.12. SolutionSepR(N, distancia, matriz, fun, upd)
1	1	Alg. 5.15. SolutionConvexSepR(N, distancia, matriz, fun, upd)

Algoritmo 7.1 Solution(N, distancia, matriz, fun, upd, conv, ver, prob)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1), *conv*: simple (conv = 0) o convexo (conv = 1), *ver*: iterativo (ver = 0) o recursivo (ver = 1), *prob*: inclusión, $O = \emptyset$ (prob = 0) o separabilidad, $O \neq \emptyset$ (prob = 1)

Output: Polígono simple o convexo de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en P

```
1  solution ← ∅
2  sum ← 16fun + 8upd + 4conv + 2ver + prob
3  switch sum do
4      case 0 do
5          solution ← SolutionIncI(N, distancia, matriz, 0)              /* Alg.4.10 */
6      case 1 do
7          solution ← SolutionSepI(N, distancia, matriz, 0, 0)           /* Alg.5.6 */
8      case 2 do
9          solution ← SolutionIncR(N, distancia, matriz, 0)              /* Alg.4.18 */
10     case 3 do
11         solution ← SolutionSepR(N, distancia, matriz, 0, 0)           /* Alg.5.12 */
12     case 4 do
13         solution ← SolutionConvexIncI(N, distancia, matriz, 0)        /* Alg.4.15 */
14     case 5 do
15         solution ← SolutionConvexSepI(N, distancia, matriz, 0, 0)     /* Alg.5.9 */
16     case 6 do
17         solution ← SolutionConvexIncR(N, distancia, matriz, 0)        /* Alg.4.21 */
18     case 7 do
19         solution ← SolutionConvexSepR(N, distancia, matriz, 0, 0)     /* Alg.5.15 */
20     case 9 do
21         solution ← SolutionSepI(N, distancia, matriz, 0, 1)           /* Alg.5.6 */
22     case 11 do
23         solution ← SolutionSepR(N, distancia, matriz, 0, 1)           /* Alg.5.12 */
24     case 13 do
25         solution ← SolutionConvexSepI(N, distancia, matriz, 0, 1)     /* Alg.5.9 */
26     case 15 do
27         solution ← SolutionConvexSepR(N, distancia, matriz, 0, 1)     /* Alg.5.15 */
```

Algoritmo 7.1 Solution(N, distancia, matriz, fun, upd, conv, ver, prob) (cont.)

Input: N: número de puntos de P, distancia $\in \mathbb{N}$, matriz: matriz de adyacencia, *fun*: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1), *conv*: simple (conv = 0) o convexo (conv = 1), *ver*: iterativo (ver = 0) o recursivo (ver = 1), *prob*: inclusión, $O = \emptyset$ (prob = 0) o separabilidad, $O \neq \emptyset$ (prob = 1)

Output: Polígono simple o convexo de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en P

```
...
switch sum do
    case 16 do
        solution ← SolutionIncI(N, distancia, matriz, 1)              /* Alg.4.10 */
    case 17 do
        solution ← SolutionSepI(N, distancia, matriz, 1, 0)            /* Alg.5.6 */
    case 18 do
        solution ← SolutionIncR(N, distancia, matriz, 1)              /* Alg.4.18 */
    case 19 do
        solution ← SolutionSepR(N, distancia, matriz, 1, 0)           /* Alg.5.12 */
    case 20 do
        solution ← SolutionConvexIncI(N, distancia, matriz, 1)        /* Alg.4.15 */
    case 21 do
        solution ← SolutionConvexSepI(N, distancia, matriz, 1, 0)      /* Alg.5.9 */
    case 22 do
        solution ← SolutionConvexIncR(N, distancia, matriz, 1)        /* Alg.4.21 */
    case 23 do
        solution ← SolutionConvexSepR(N, distancia, matriz, 1, 0)     /* Alg.5.15 */
    case 25 do
        solution ← SolutionSepI(N, distancia, matriz, 1, 1)            /* Alg.5.6 */
    case 27 do
        solution ← SolutionSepR(N, distancia, matriz, 1, 1)           /* Alg.5.12 */
    case 29 do
        solution ← SolutionConvexSepI(N, distancia, matriz, 1, 1)      /* Alg.5.9 */
    case 31 do
        solution ← SolutionConvexSepR(N, distancia, matriz, 1, 1)     /* Alg.5.15 */
return solution
```

7.2. Algoritmo 7.2. Rectangle(N, matriz, fun, upd, ver, prob)

El **Algoritmo 7.2. Rectangle(N, matriz, fun, upd, ver, prob)** calcula el rectángulo de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado P con obstáculos arbitrarios. Este algoritmo consta de seis parámetros, y, al igual que el Algoritmo 7.1, comparte los cinco primeros con el **Algoritmo 5.17. RectangleSep(N, matriz, fun, upd, ver)**. El parámetro final, *prob*, se emplea para decidir si se va a resolver un problema de inclusión ($prob = 0$) o un problema de separabilidad ($prob = 1$). El coste computacional es $O(n^5)$ para la versión iterativa y $O(n^6)$ para la versión recursiva.

Existen un total de 16 variantes que se pueden configurar mediante los parámetros fun, upd, ver y $prob$. Sin embargo, solo 12 de estas configuraciones son válidas. Además, cada valor decimal del parámetro sum, definido como $sum = 8fun + 4upd + 2ver + prob$, se puede relacionar con un tipo de solución de los Algoritmos 4.23 y 5.17 (Tabla 7.4), lo que permite obtener el código final del Algoritmo 7.2.

Tabla 7.3: Algoritmo 7.2: Combinaciones posibles de las variables fun, upd, ver, prob

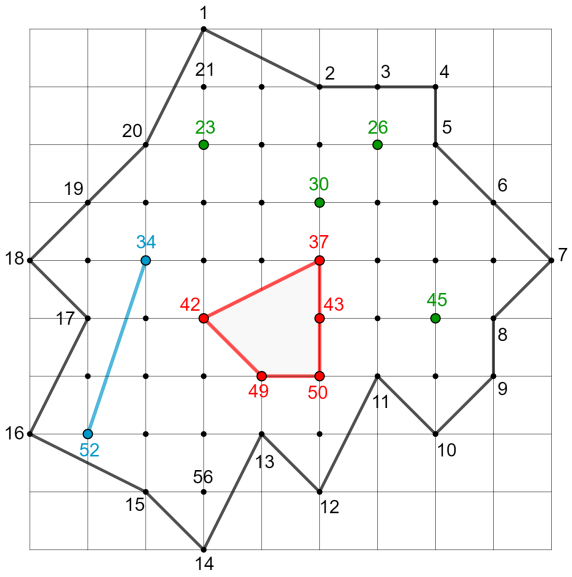
fun	upd	ver	prob	sum
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	1	5
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	1	13
1	1	1	1	15

Tabla 7.4: Algoritmos de inclusión y separabilidad para rectángulos

ver	**Alg. 4.23. RectangleInc(N, matriz, fun, ver)**
0	Alg. 4.12. RectangleIncI(N, matriz, fun)
1	Alg. 4.20. RectangleIncR(N, matriz, fun)
ver	**Alg. 5.17. RectangleSep(N, matriz, fun, upd, ver)**
0	Alg. 5.8. RectangleSepI(N, matriz, fun, upd)
1	Alg. 5.14. RectangleSepR(N, matriz, fun, upd)

7.3. Caso propuesto

A continuación, se realiza el cálculo de varias soluciones en relación con la Figura 7.1, utilizando los Algoritmos 7.1 y 7.2. Además de estos algoritmos, es necesario emplear los conjuntos que definen el polígono reticulado P, así como los puntos, segmentos y agujeros.



$$\begin{bmatrix} Points = P = \{p_1, \ldots, p_{56}\} \\ Polygon = \partial P_0 = \{p_1, \ldots, p_{20}\} \\ PointsP = \{p_{23}, p_{26}, p_{30}, p_{45}\} \\ SegmentsP = \{p_{34}, p_{52}\} \\ HolesP = \{p_{37}, p_{43}, p_{50}, p_{49}, p_{42}\} \end{bmatrix}$$

Figura 7.1: Problema inicial

Algoritmo 7.2 Rectangle(N, matriz, fun, upd, ver, prob)

Input: N: número de puntos de P, matriz: matriz de adyacencia,
fun: área (fun = 0) o perímetro (fun = 1), *upd*: máximo (upd = 0) o mínimo (upd = 1), *ver*: iterativo (ver = 0) o recursivo (ver = 1), *prob*: inclusión, $O = \emptyset$ (prob = 0) o separabilidad, $O \neq \emptyset$ (prob = 1)

Output: Rectángulo de mayor área o perímetro que contiene a un conjunto O y está contenido en P

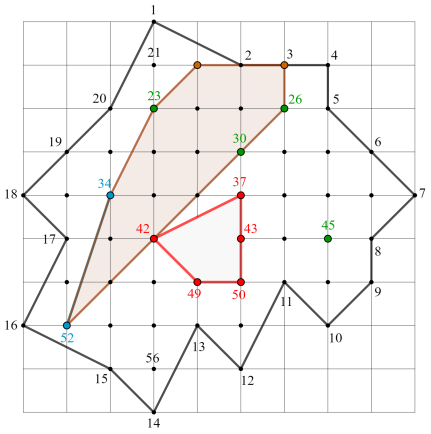
```
1  rect ← ∅
2  sum ← 8fun + 4upd + 2ver + prob
3  switch sum do
4      case 0 do
5          rect ← RectangleIncI(N, matriz, 0)                    /* Alg.4.12 */
6      case 1 do
7          rect ← RectangleSepI(N, matriz, 0, 0)                 /* Alg.5.8 */
8      case 2 do
9          rect ← RectangleIncR(N, matriz, 0)                    /* Alg.4.20 */
10     case 3 do
11         rect ← RectangleSepR(N, matriz, 0, 0)                 /* Alg.5.14 */
12     case 5 do
13         rect ← RectangleSepI(N, matriz, 0, 1)                 /* Alg.5.8 */
14     case 7 do
15         rect ← RectangleSepR(N, matriz, 0, 1)                 /* Alg.5.14 */
16     case 8 do
17         rect ← RectangleIncI(N, matriz, 1)                    /* Alg.4.12 */
18     case 9 do
19         rect ← RectangleSepI(N, matriz, 1, 0)                 /* Alg.5.8 */
20     case 10 do
21         rect ← RectangleIncR(N, matriz, 1)                    /* Alg.4.20 */
22     case 11 do
23         rect ← RectangleSepR(N, matriz, 1, 0)                 /* Alg.5.14 */
24     case 13 do
25         rect ← RectangleSepI(N, matriz, 1, 1)                 /* Alg.5.8 */
26     case 15 do
27         rect ← RectangleSepR(N, matriz, 1, 1)                 /* Alg.5.14 */
28 return rect
```

7.3.1. Solución del caso propuesto

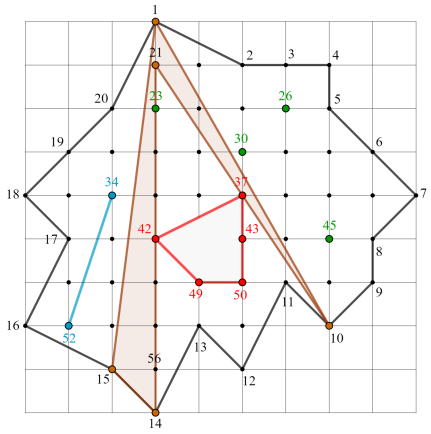
En la Tabla 7.5, se presentan las soluciones (Figura 7.2) que se buscan obtener mediante la aplicación de los Algoritmos 7.1 y 7.2. Como se puede apreciar, la versatilidad para calcular diversos tipos de soluciones es elevada, lo que convierte a estos algoritmos en una herramienta eficaz para que cualquier usuario pueda adaptar su problema a las diversas configuraciones que ofrecen dichos algoritmos.

Tabla 7.5: Solución

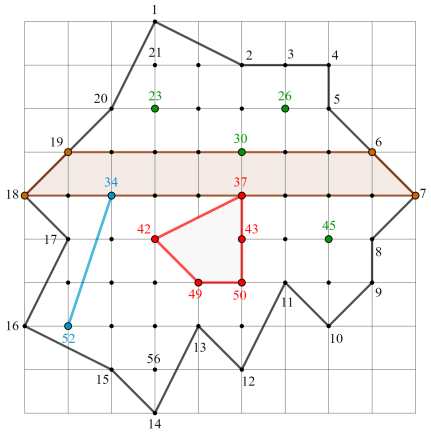
Fig. 7.2	**Alg. 7.1. Solution(N, distancia, matriz, fun, upd, conv, ver, prob)** **Alg. 7.2. Rectangle(N, matriz, fun, upd, ver, prob)**
a)	Solution(56, 5, A, 0, 0, 1, ver, 0) Hexágono convexo de mayor área contenido en P
b)	Solution(56, 4, A, 1, 0, 0, ver, 0) Pentágono simple de mayor perímetro contenido en P
c)	Solution(56, 3, A, 1, 0, 1, ver, 0) Cuadrilátero convexo de mayor perímetro contenido en P
d)	Rectangle(56, A, 0, 0, ver, 1) $ContP = \{26, 30\}$ Rectángulo de mayor área que contiene al conjunto O
e)	Solution(56, 5, A, 1, 0, 1, ver, 1) $ContP = \{26, 30\}$ Hexágono convexo de mayor perímetro que contiene al conjunto O
f)	Rectangle(56, A, 0, 0, ver, 1) $ContP = \{30, 45\}$, $ContH = HolesP$ Rectángulo de mayor área que contiene al conjunto O
g)	Rectangle(56, A, 0, 1, ver, 1) $ContP = \{30, 45\}$, $ContH = HolesP$ Rectángulo de menor área que contiene al conjunto O
h)	Solution(56, 5, A, 0, 1, 1, ver, 1) $ContP = \{30, 45\}$, $ContS = SegmentsP$ Hexágono convexo de menor área que contiene al conjunto O
i)	Solution(56, 4, A, 0, 0, 1, ver, 1) Pentágono convexo de mayor área que contiene a la $ROI = \{PointsP, SegmentsP, HolesP\}$



(a) Solution(56, 5, A, 0, 0, 1, ver, 0)

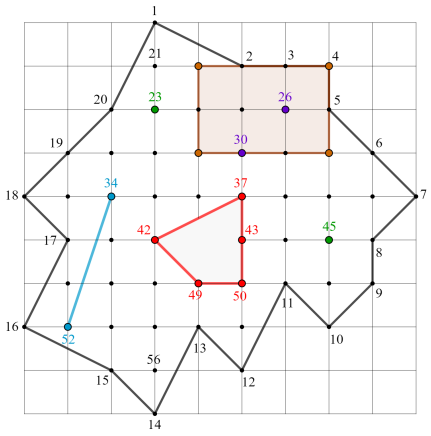


(b) Solution(56, 4, A, 1, 0, 0, ver, 0)

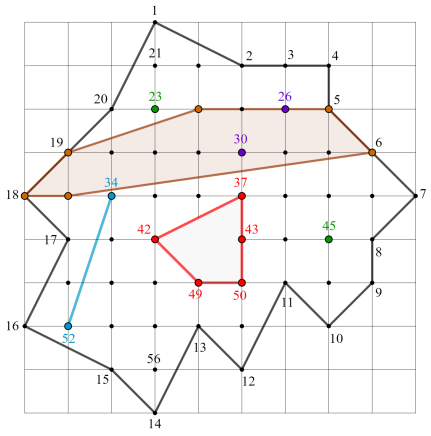


(c) Solution(56, 3, A, 1, 0, 1, ver, 0)

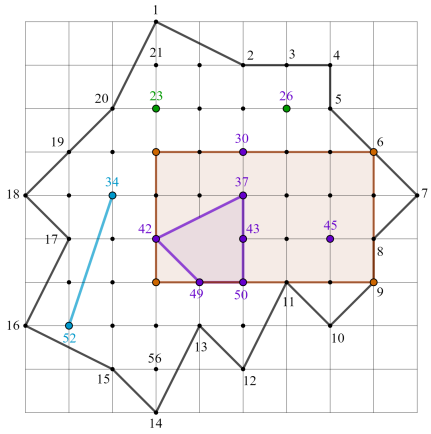
Figura 7.2: Solución del caso propuesto



(d) Rectangle(56, A, 0, 0, ver, 1)

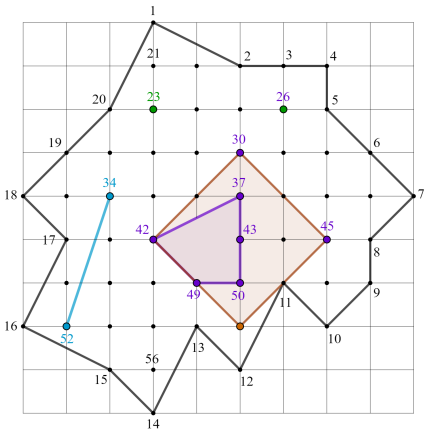


(e) Solution(56, 5, A, 1, 0, 1, ver, 1)

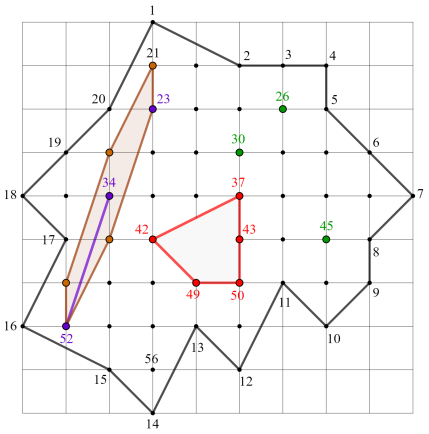


(f) Rectangle(56, A, 0, 0, ver, 1)

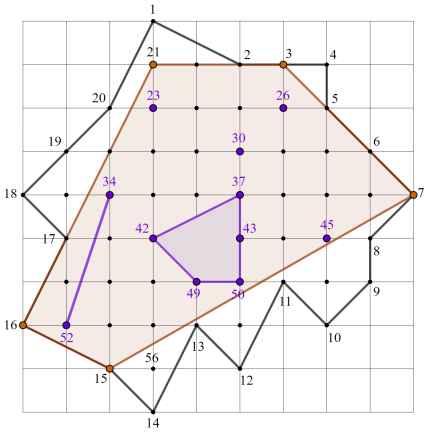
Figura 7.2: Solución del caso propuesto (cont1.)



(g) Rectangle(56, A, 0, 1, ver, 1)



(h) Solution(56, 5, A, 0, 1, 1, ver, 1)



(i) Solution(56, 4, A, 0, 0, 1, ver, 1)

Figura 7.2: Solución del caso propuesto (cont2.)

Capítulo 8

Resultados de la aplicación práctica

El Capítulo 8 demuestra, a través de diversos ejemplos, las aplicaciones prácticas que se pueden generar a partir de los problemas planteados en los Capítulos 4, 5 y 6. La resolución de cada una de estas aplicaciones se lleva a cabo empleando los algoritmos detallados en dichos capítulos, respaldados por las soluciones generales proporcionadas en el Capítulo 7, que constituyen el núcleo de esta Tesis Doctoral.

4. Problemas de inclusión, $Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$

Algoritmo 4.22. SolutionInc(N, distancia, matriz, fun, conv, ver)
Algoritmo 4.23. RectangleInc(N, matriz, fun, ver)

5. Problemas de separabilidad, $Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu)$
6. Problemas de la envolvente, $Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$

Algoritmo 5.16. SolutionSep(N, distancia, matriz, fun, upd, conv, ver)
Algoritmo 5.17. RectangleSep(N, matriz, fun, upd, ver)

7. Unificación de problemas: Inclusión, Separabilidad y Envolvente

Algoritmo 7.1. Solution(N, distancia, matriz, fun, upd, conv, ver, prob)
Algoritmo 7.2. Rectangle(N, matriz, fun, upd, ver, prob)

Para obtener las soluciones a partir de los algoritmos anteriores, se ha desarrollado el pseudocódigo, así como la implementación en los lenguajes de programación Python y Java, consiguiendo de esta forma el objetivo parcial 5.

8.1. Problemas de inclusión

8.1.1. Tumor cerebral (I)

La adquisición de imágenes médicas es un componente importante en el diagnóstico y tratamiento de diversas enfermedades, incluyendo tumores cerebrales [6, 114]. La Figura 8.1 muestra la imagen de un tumor cerebral, un meningioma [195]. Un meningioma es un tumor primario del sistema nervioso central (SNC) que se origina en el cerebro o la médula espinal. Crecen lentamente y son los tumores cerebrales primarios más comunes. Normalmente, la cirugía se presenta como la terapia más adecuada para tratar este tipo de tumor cerebral.

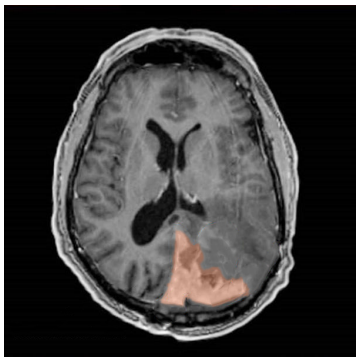
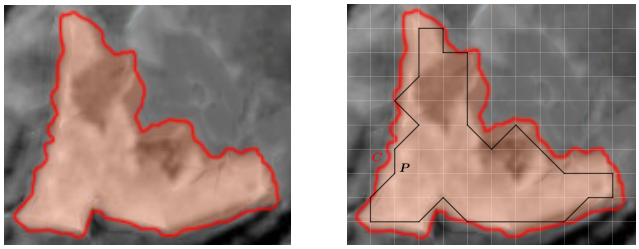


Figura 8.1: Tumor cerebral (I). Meningioma

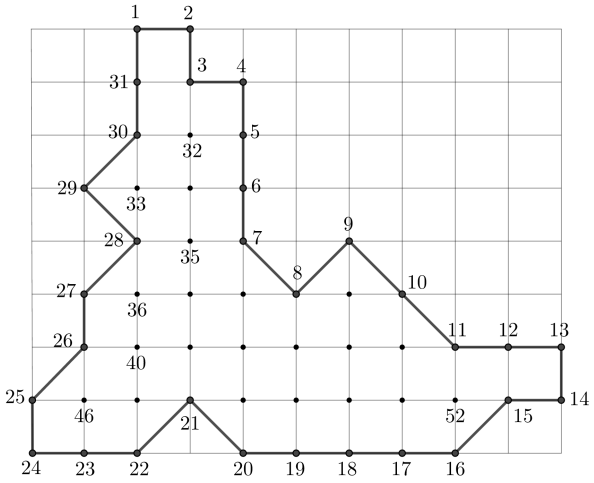
El objetivo para este primer problema consiste en calcular el k-gon de mayor área que está contenido dentro del tumor cerebral para una posible intervención quirúrgica. Para ello, se crea inicialmente el contorno cerrado C (Figura 8.2a), y a continuación, se construye el polígono reticulado P (Figura 8.2b). La característica del problema a resolver señala que es un problema de inclusión sin obstáculos arbitrarios.



(a) Contorno cerrado C (b) Polígono reticulado P

Figura 8.2: Tumor cerebral (I). Extracción del polígono reticulado

A continuación, se establecen las coordenadas del polígono reticulado, así como los conjuntos que serán empleados en la ejecución de los Algoritmos 7.1 y 7.2 (Figura 8.3).



$$\begin{cases} Points = \{p_1, \ldots, p_{52}\} \\ Polygon = \{p_1, \ldots, p_{31}\} \\ PointsP = \emptyset \\ SegmentsP = \emptyset \\ HolesP = \emptyset \end{cases}$$

Figura 8.3: Tumor cerebral (I). Definición del polígono reticulado

Las soluciones se presentan en tablas, Tabla 8.1 a Tabla 8.3, compuestas por tres columnas. En la primera columna, se muestra el polígono que hay que calcular; en la segunda columna, se exponen las soluciones encontradas y en la tercera columna, el valor del mayor área. Las ejemplos gráficos de las soluciones aparecen en la Figura 8.4.

Tabla 8.1: Tumor cerebral (I). Solución: Polígono simple de k lados de mayor área

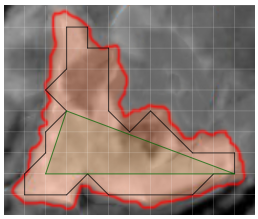
***k*-gon**	**Solution(52, distancia, matriz, 0, 0, 0, ver, 0)**	**Área**
Triángulo	(14,28,46), (15,35,25)	13,5
Cuadrilátero	(3,46,14,38), (16,30,40,20), (16,34,27,20), (16,34,24,21)	16
Pentágono	(1,47,15,9,43)	18,5
Hexágono	(1,43,9,15,20,40), (1,43,13,16,20,40), (4,42,9,15,19,26)	21
	(13,43,32,24,21,16)	
Heptágono	(1,40,20,15,9,42,5), (1,43,13,16,21,24,28), (1,43,9,15,16,20,40)	23
	(4,42,9,15,16,21,24), (4,42,9,15,17,21,24), (4,42,9,15,18,21,24)	
	(4,42,9,15,19,21,24), (4,42,9,15,20,21,24), (4,42,9,15,20,40,31)	
	(9,43,32,24,21,16,15), (9,43,32,24,21,17,15), (9,43,32,24,21,18,15)	
	(9,43,32,24,21,19,15), (9,43,32,24,21,20,15), (13,43,32,24,21,20,16)	

Tabla 8.2: Tumor cerebral (I). Solución: Rectángulo de mayor área

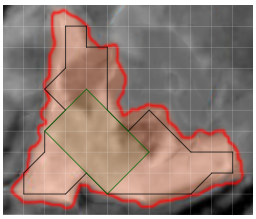
Rectangle(52, matriz, 0, 0, ver, 0)	**Área**
(4,31,47,48), (27,20,44,34), (10,27,46,51)	12

Tabla 8.3: Tumor cerebral (I). Solución: Polígono convexo de k lados de mayor área

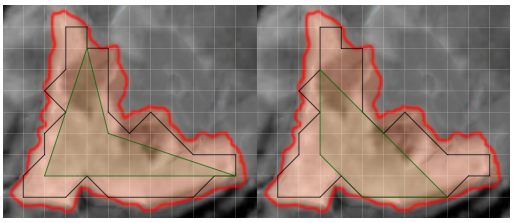
***k*-gon**	**Solution(52, distancia, matriz, 0, 0, 1, ver, 0)**	**Área**
Cuadrilátero	(16,30,40,20), (16,34,27,20)	16
Pentágono	(10,27,20,16,15), (11,28,26,19,16), (11,28,27,20,16), (15,35,27,20,16)	17
Hexágono	(11,28,26,19,16,15), (11,28,27,20,16,15)	18
Heptágono	(11,28,27,26,19,16,15)	18,5



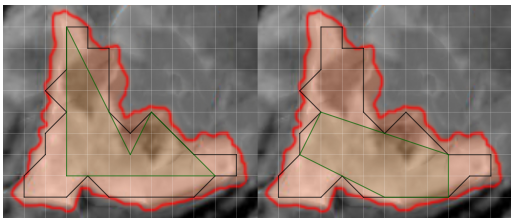
(a) Triángulo



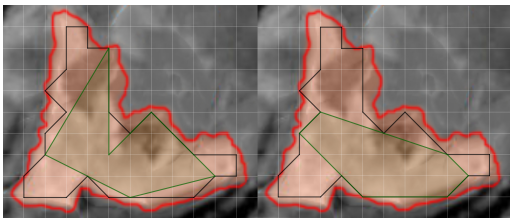
(b) Rectángulo



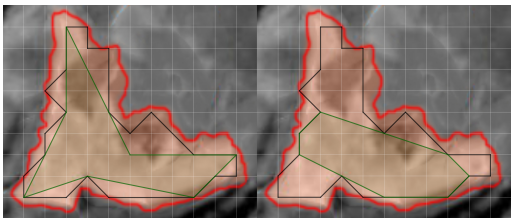
(c) Cuadrilátero simple, convexo



(d) Pentágono simple, convexo



(e) Hexágono simple, convexo



(f) Heptágono simple, convexo

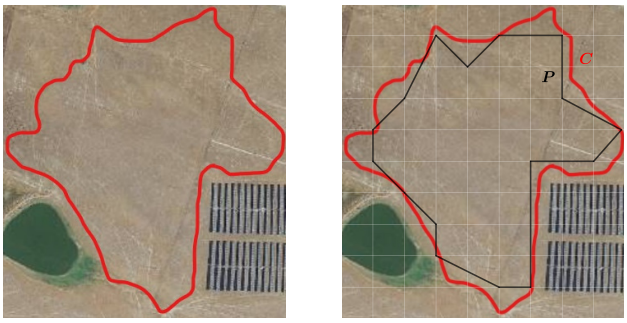
Figura 8.4: Tumor cerebral (I). Solución

8.1.2. Paneles solares

Las paneles solares no solo proporcionan una fuente de energía inagotable, sino que también permiten la generación de electricidad de forma gratuita sin producir gases de efecto invernadero ni otros contaminantes [14]. La Figura 8.5 presenta una imagen de un terreno parcialmente cubierto por paneles solares, en la que se destaca una nueva región de interés (ROI) con el objetivo de expandir esta instalación fotovoltaica utilizando la mayor superficie posible. Se está por tanto ante un problema de inclusión sin obstáculos arbitrarios.

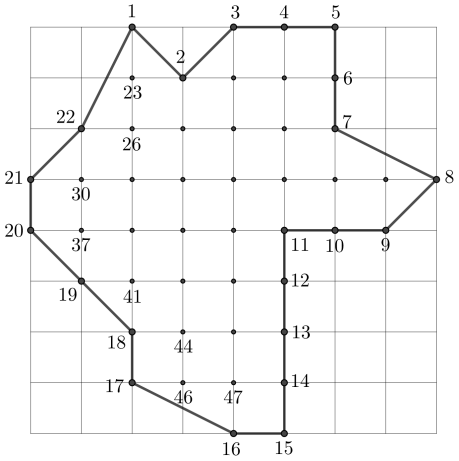
Figura 8.5: Paneles solares

La Figura 8.6a muestra el contorno cerrado C, y la Figura 8.6b el polígono reticulado P. Además, en la Figura 8.7 se definen las coordenadas del polígono reticulado junto a los conjuntos que hay que seleccionar para ejecutar los Algoritmos 7.1 y 7.2.



(a) Contorno cerrado C (b) Polígono reticulado P

Figura 8.6: Paneles solares. Extracción del polígono reticulado



$$\begin{bmatrix} Points = \{p_1, \ldots, p_{47}\} \\ Polygon = \{p_1, \ldots, p_{22}\} \\ PointsP = \emptyset \\ SegmentsP = \emptyset \\ HolesP = \emptyset \end{bmatrix}$$

Figura 8.7: Paneles solares. Definición del polígono reticulado

Las soluciones resultantes de la ejecución de los algoritmos se presentan en tablas, desde la Tabla 8.4 hasta la Tabla 8.6. Además, se proporcionan de manera gráfica a través de la Figura 8.8 para una mejor comprensión.

Tabla 8.4: Paneles solares. Solución: Polígono simple de k lados de mayor área

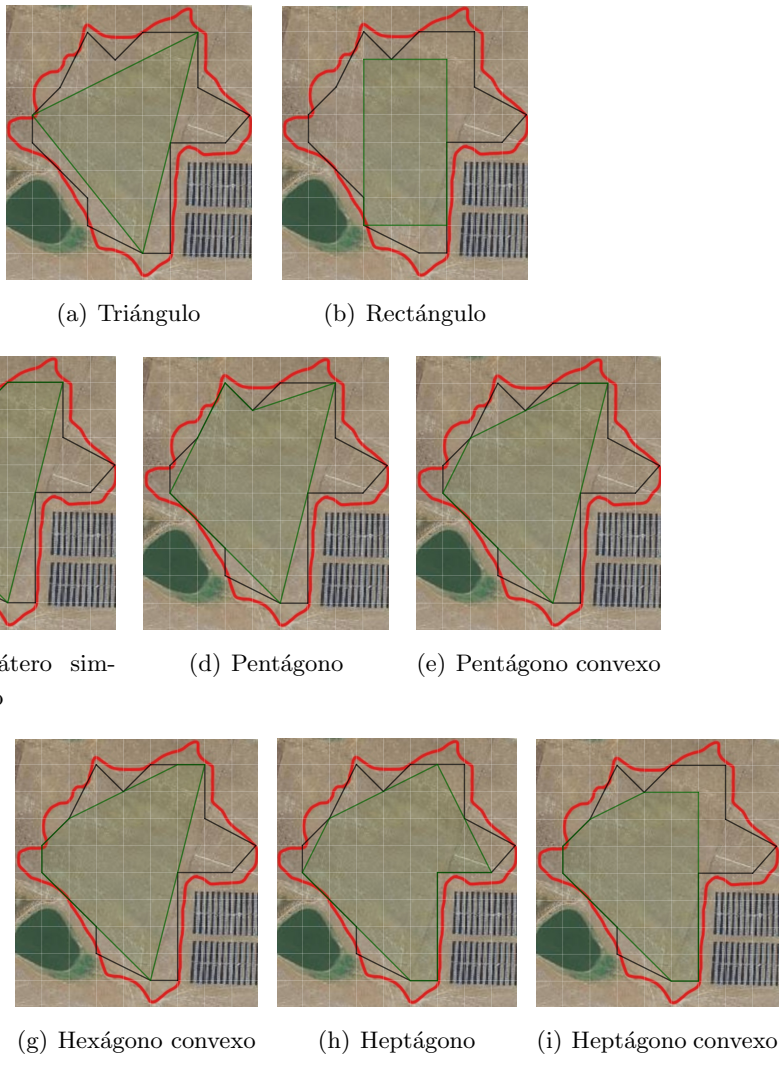
***k*-gon**	**Solution(47, distancia, matriz, 0, 0, 0, ver, 0)**	**Área**
Triángulo	(5,21,16)	21
Cuadrilátero	(3,20,16,5), (5,22,20,16)	24
Pentágono	(1,20,16,5,2)	26
Hexágono	(1,20,16,5,3,2), (4,22,20,15,11,9), (4,22,20,16,11,9)	27
Heptágono	(4,22,20,16,15,11,9)	29

Tabla 8.5: Paneles solares. Solución: Rectángulo de mayor área

Rectangle(47, matriz, 0, 0, ver, 0)	**Área**
(14,17,23,25)	18

Tabla 8.6: Paneles solares. Solución: Polígono convexo de k lados de mayor área

***k*-gon**	**Solution(47, distancia, matriz, 0, 0, 1, ver, 0)**	**Área**
Cuadrilátero	(3,20,16,5), (5,22,20,16)	24
Pentágono	(4,22,20,16,5), (4,22,20,16,15)	25
Hexágono	(4,22,21,20,16,5), (4,22,21,20,16,15)	25,5
Heptágono	(2,22,21,20,16,15,25), (4,22,20,46,47,6,5), (4,22,21,20,46,47,6) (15,29,24,23,21,20,16)	24,5



(a) Triángulo (b) Rectángulo

(c) Cuadrilátero simple, convexo (d) Pentágono (e) Pentágono convexo

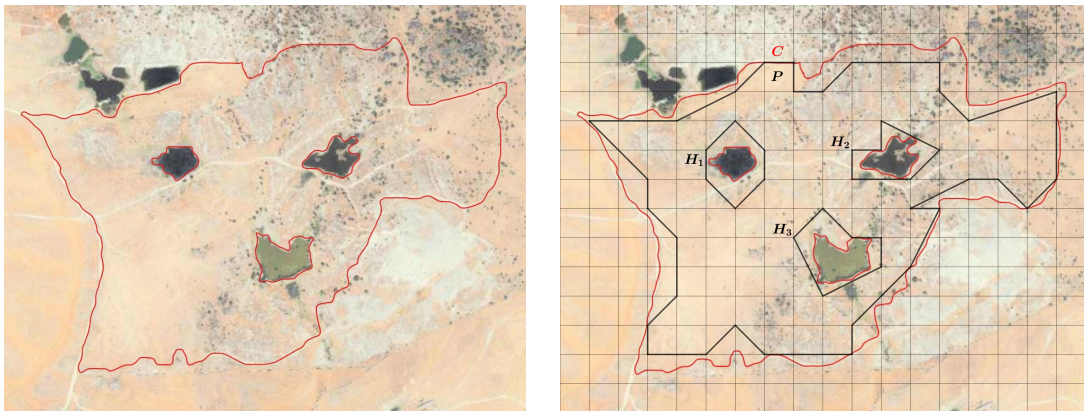
(f) Hexágono (g) Hexágono convexo (h) Heptágono (i) Heptágono convexo

Figura 8.8: Paneles solares. Solución

8.1.3. Parcela

De la aplicación Visor SigPac v4.12 [116] se han obtenido las imágenes de una parcela, como se muestra en la Figura 8.9. Con el propósito de calcular la edificación de cualquier forma poligonal con la mayor superficie posible, se ha delimitado el contorno cerrado C (Figura 8.10a). A continuación, se ha generado el polígono reticulado P (Figura 8.10b) realizando previamente una partición regular sobre dicho contorno. En esta situación, el objetivo es resolver un problema de inclusión con obstáculos arbitrarios.

Figura 8.9: Parcela



(a) Contorno cerrado C

(b) Polígono reticulado P

Figura 8.10: Parcela. Extracción del polígono reticulado

En la Figura 8.11 se define el polígono reticulado P sobre la anterior partición regular y los conjuntos elegidos para aplicar los Algoritmos 7.1 y 7.2.

$$\begin{bmatrix} Points = \{p_1, \ldots, p_{110}\} \\ Polygon = \{p_1, \ldots, p_{42}\} \\ PointsP = \emptyset \\ SegmentsP = \emptyset \\ HolesP = \{H_1, H_2, H_3\} \end{bmatrix}$$

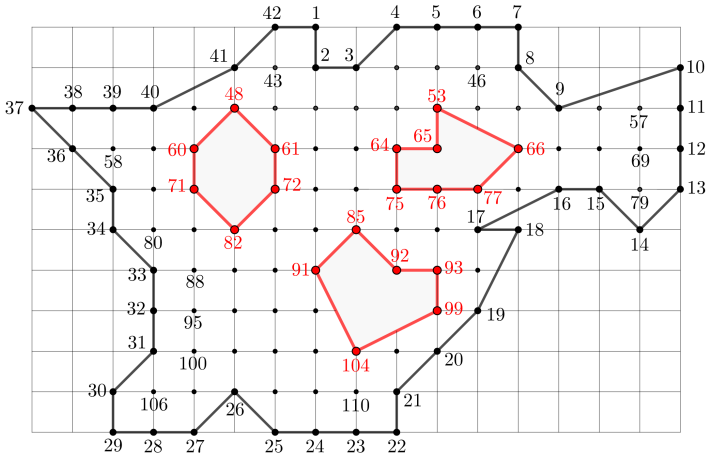


Figura 8.11: Parcela. Definición del polígono reticulado

Los resultados obtenidos al ejecutar los algoritmos anteriores se encuentran detallados en tres tablas, desde la Tabla 8.7 hasta la Tabla 8.9, mientras que las Figuras 8.12 y 8.13 muestran de forma gráfica dichas soluciones.

Tabla 8.7: Parcela. Solución: Polígono simple de k lados de mayor área

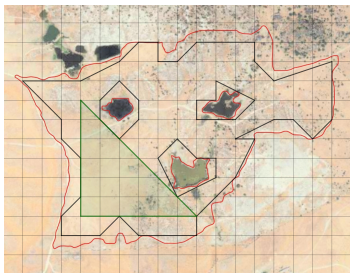
***k*-gon**	**Solution(110, distancia, matriz, 0, 0, 0, ver, 0)**	**Área**
Triángulo	(21,106,59)	18
Cuadrilátero	(6,82,39,27)	21,5
Pentágono	(4,31,22,91,6), (6,49,90,39,27)	26
Hexágono	(3,29,26,22,91,6), (4,31,24,22,91,6)	28

Tabla 8.8: Parcela. Solución: Rectángulo de mayor área

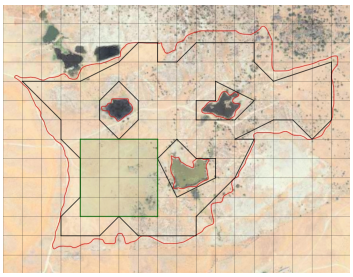
Rectangle(110, matriz, 0, 0, ver, 0)	**Área**
(80,106,109,84)	18

Tabla 8.9: Parcela. Solución: Polígono convexo de k lados de mayor área

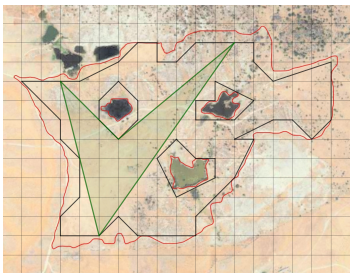
k-gon	Solution(110, distancia, matriz, 0, 0, 1, ver, 0)	Área
Cuadrilátero	(3,29,26,6)	21
Pentágono	(4,31,29,26,6)	22,5
Hexágono	(4,31,29,27,26,6)	23,5



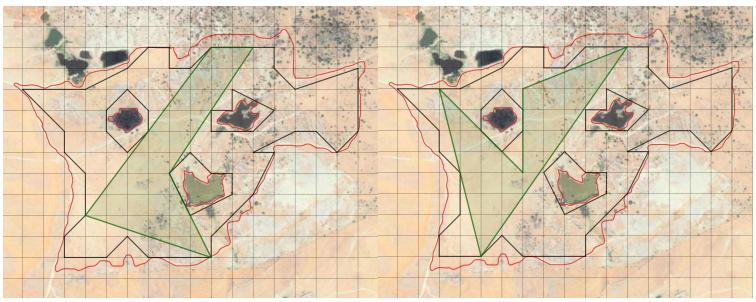
(a) Triángulo



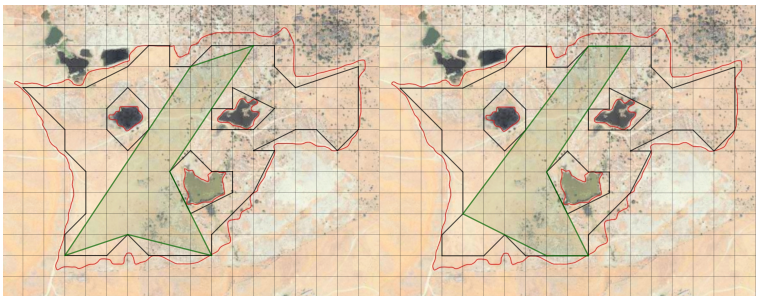
(b) Rectángulo



(c) Cuadrilátero

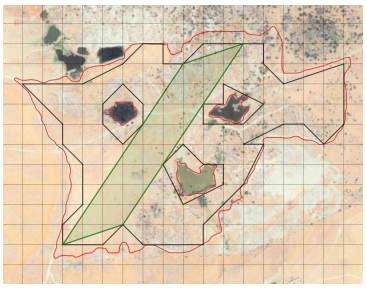


(d) Pentágono

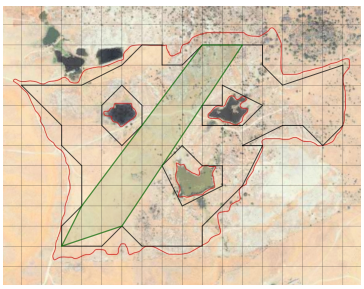


(e) Hexágono

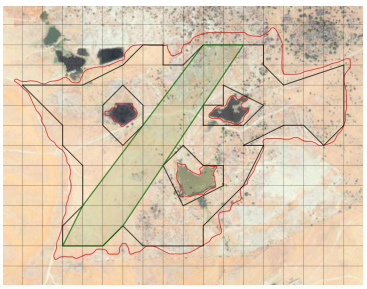
Figura 8.12: Parcela. Solución: Polígono simple de k lados y rectángulo de mayor área



(a) Cuadrilátero



(b) Pentágono



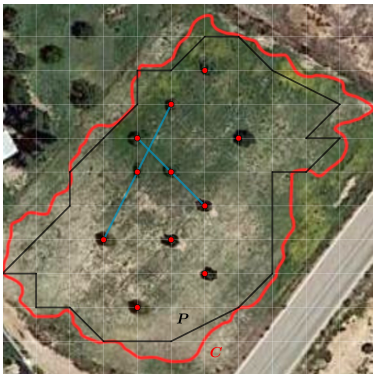
(c) Hexágono

Figura 8.13: Parcela. Solución: Polígono convexo de k lados de mayor área

8.1.4. Horticultura

La inclusión del análisis de polígonos en el ámbito de la horticultura se revela como una herramienta sumamente beneficiosa para los agricultores [124]. Otorga la capacidad de estudiar con precisión las formas y dimensiones de sus parcelas de tierra, al mismo tiempo que facilita una planificación y diseño más eficaz para la disposición óptima de sus cultivos. En este caso, se ha tomado una parcela agrícola (Figura 8.14a) y se ha construido el polígono reticulado P dentro del contorno cerrado C (Figura 8.14b). Finalmente, se han definido y seleccionado las coordenadas del polígono y de los conjuntos para aplicar los Algoritmos 7.1 y 7.2 (Figura 8.15). El objetivo es calcular el polígono de cualquier número de lados y con el área máxima para optimizar la cantidad de tierra disponible para el cultivo, aumentando así su producción y rendimiento. En consecuencia, el problema planteado consiste en un problema de inclusión con obstáculos arbitrarios.

(a) Contorno cerrado C



(b) Polígono reticulado P

Figura 8.14: Horticultura. Extracción del polígono reticulado

$$\begin{bmatrix} Points = \{p_1, \ldots, p_{64}\} \\ Polygon = \{p_1, \ldots, p_{26}\} \\ PointsP = \{p_{27}, p_{37}, p_{52}, p_{59}, p_{62}\} \\ SegmentsP = \{S_1, S_2\} \\ HolesP = \emptyset \\ S_1 = \{p_{34}, p_{41}, p_{47}\} \\ S_2 = \{p_{29}, p_{40}, p_{50}\} \end{bmatrix}$$

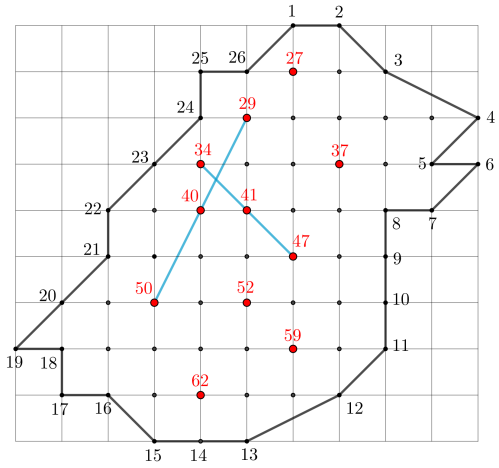


Figura 8.15: Horticultura. Definición del polígono reticulado

Desde la Tabla 8.10 hasta la Tabla 8.12 se encuentra un desglose detallado de las soluciones obtenidas, complementado por las Figuras 8.16 y 8.17 que proporcionan una representación visual.

Tabla 8.10: Horticultura. Solución: Polígono simple de k lados de mayor área

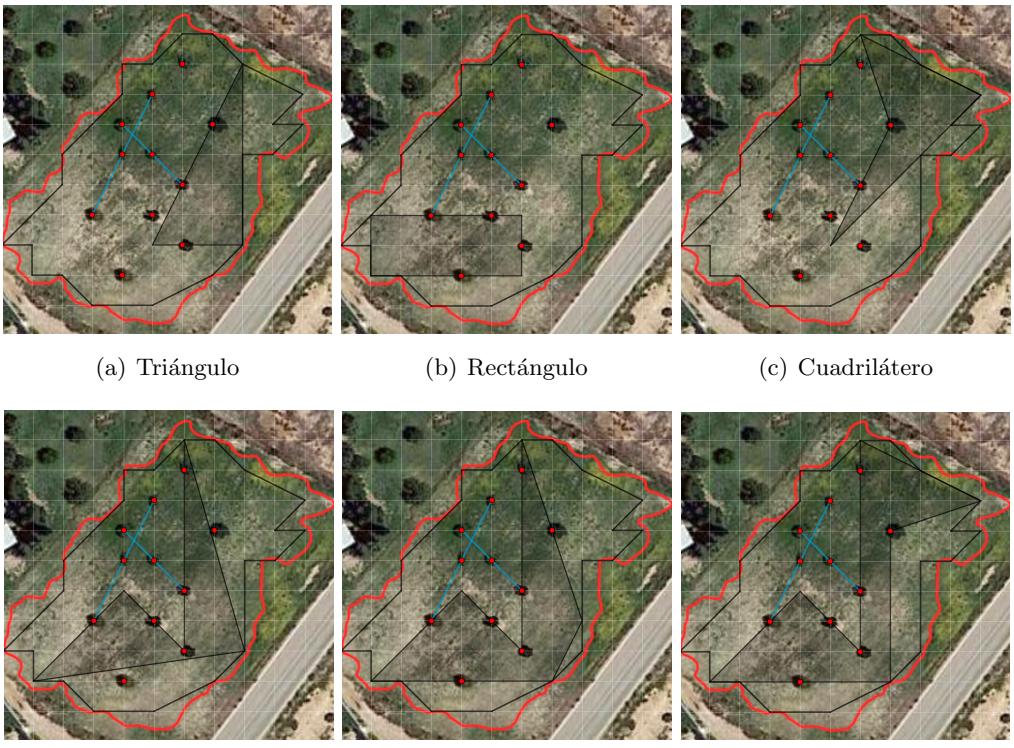
***k*-gon**	**Solution(64, distancia, matriz, 0, 0, 0, ver, 0)**	**Área**
Triángulo	(3,58,11), (12,45,17)	9
Cuadrilátero	(1,37,58,4), (1,53,19,11), (3,14,59,11), (3,47,57,11)	10
	(3,53,19,11), (10,62,17,20), (11,38,52,19), (12,52,20,17)	
	(12,52,41,17), (17,63,54,20), (17,64,53,20)	
Pentágono	(1,59,45,17,11)	14
Hexágono	(1,59,45,17,12,37,4)	17,5
Heptágono	(1,59,45,17,12,37,4)	20

Tabla 8.11: Horticultura. Solución: Rectángulo de mayor área

Rectangle(64, matriz, 0, 0, ver, 0)	**Área**
(17,20,53,64)	10

Tabla 8.12: Horticultura. Solución: Polígono convexo de k lados de mayor área

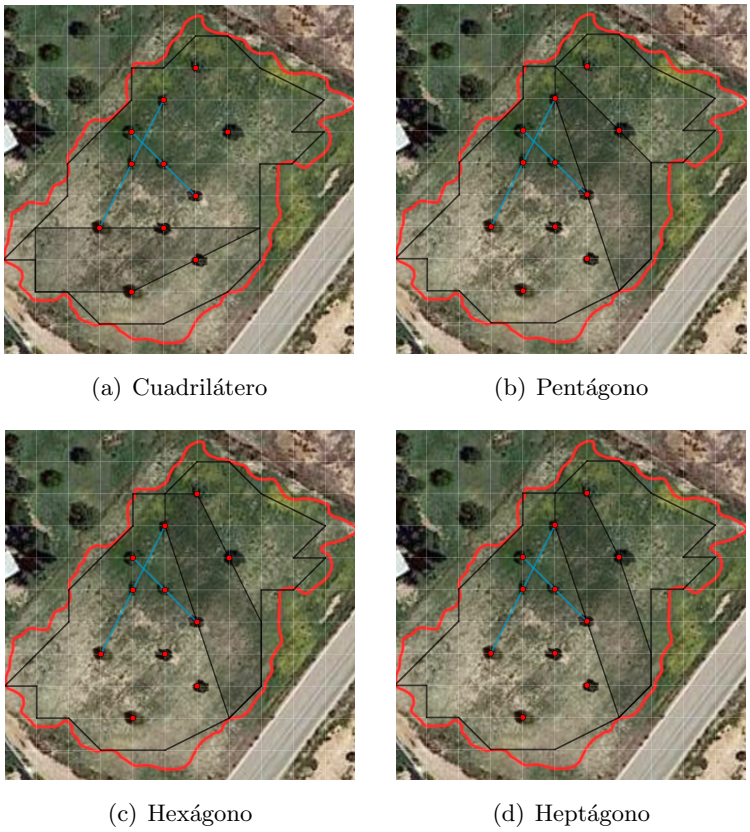
***k*-gon**	**Solution(64, distancia, matriz, 0, 0, 1, ver, 0)**	**Área**
Cuadrilátero	(10,20,17,62), (12,52,20,17), (17,63,54,20), (17,64,53,20)	10
Pentágono	(8,26,29,12,11), (9,27,29,12,11)	10
Hexágono	(9,27,26,29,12,11)	10,5
Heptágono	(10,37,27,26,29,12,11)	10



(a) Triángulo (b) Rectángulo (c) Cuadrilátero

(d) Pentágono (e) Hexágono (f) Heptágono

Figura 8.16: Horticultura. Solución: Polígono simple de k lados y rectángulo de mayor área



(a) Cuadrilátero (b) Pentágono

(c) Hexágono (d) Heptágono

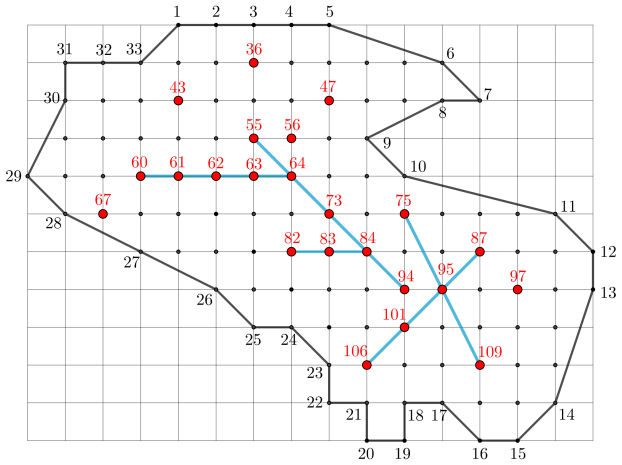
Figura 8.17: Horticultura. Solución: Polígono convexo de k lados de mayor área

8.1.5. Centro comercial

Construir un edificio de cualquier forma poligonal en un terreno, puede ser un desafío interesante para los arquitectos [85]. Aunque esta forma puede hacer que la estructura sea única y atractiva, también puede presentar complicaciones durante todas las etapas de diseño y construcción. En este contexto, se ha extraído la imagen de una nueva área residencial en desarrollo (Figura 8.18a) y se quiere construir un centro comercial con la mayor superficie posible. Para alcanzar este objetivo, se calcula el polígono reticulado P (Figura 8.18b) y posteriormente, se definen las coordenadas para aplicar los Algoritmos 7.1 y 7.2 (Figura 8.19). El problema que ahora se presenta es por tanto un problema de inclusión con obstáculos arbitrarios.

(a) Contorno cerrado C (b) Polígono reticulado P

Figura 8.18: Centro comercial. Extracción del polígono reticulado



$$
\begin{aligned}
&Points = \{p_1, \ldots, p_{113}\} \\
&Polygon = \{p_1, \ldots, p_{33}\} \\
&PointsP = \{p_{36}, p_{43}, p_{47}, p_{67}, p_{97}\} \\
&SegmentsP = \{S_1, S_2, S_3, S_4, S_5\} \\
&HolesP = \emptyset \\
&S_1 = \{p_{60}, p_{61}, p_{62}, p_{63}, p_{64}\} \\
&S_2 = \{p_{55}, p_{64}, p_{73}, p_{84}, p_{94}\} \\
&S_3 = \{p_{82}, p_{83}, p_{84}\} \\
&S_4 = \{p_{75}, p_{95}, p_{109}\} \\
&S_5 = \{p_{87}, p_{95}, p_{101}, p_{106}\}
\end{aligned}
$$

Figura 8.19: Centro comercial. Definición del polígono reticulado

Las Tablas 8.13, 8.14 y 8.15, junto con la Figura 8.20, presentan una variedad de soluciones generadas para el problema previamente planteado, al aplicar los algoritmos anteriores.

Tabla 8.13: Centro comercial. Solución: Polígono simple de k lados de mayor área

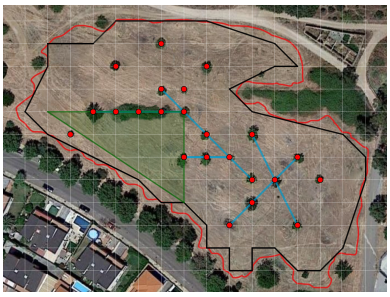
***k*-gon**	**Solution(113, distancia, matriz, 0, 0, 0, ver, 0)**	**Área**
Triángulo	(24,64,58), (29,99,63)	12
Cuadrilátero	(5,40,61,29)	14
Pentágono	(5,40,60,99,29), (5,40,60,100,29), (5,40,60,101,29)	18,5
Hexágono	(5,29,99,63,59,6)	25

Tabla 8.14: Centro comercial. Solución: Rectángulo de mayor área

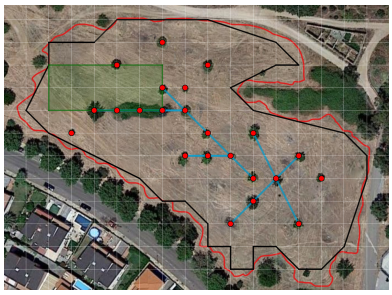
Rectangle(113, matriz, 0, 0, ver, 0)	**Área**
(6,8,30,31), (45,30,58,63)	10

Tabla 8.15: Centro comercial. Solución: Polígono convexo de k lados de mayor área

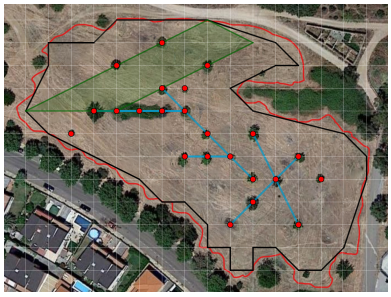
***k*-gon**	**Solution(113, distancia, matriz, 0, 0, 1, ver, 0)**	**Área**
Cuadrilátero	(5,29,61,40)	14
Pentágono	(26,106,101,61,58)	14,5
Hexágono	(26,106,101,93,50,58), (26,106,101,61,58,67) (26,106,101,93,50,67)	15



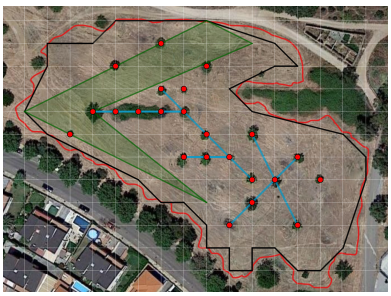
(a) Triángulo



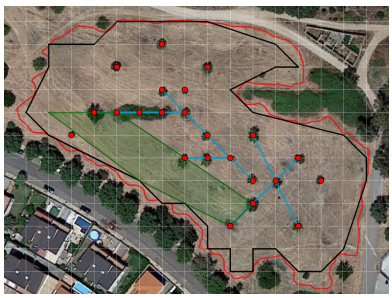
(b) Rectángulo



(c) Cuadrilátero simple, convexo

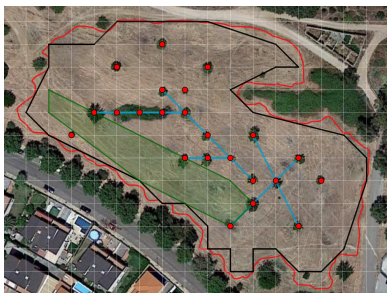


(d) Pentágono



(e) Pentágono convexo

(f) Hexágono



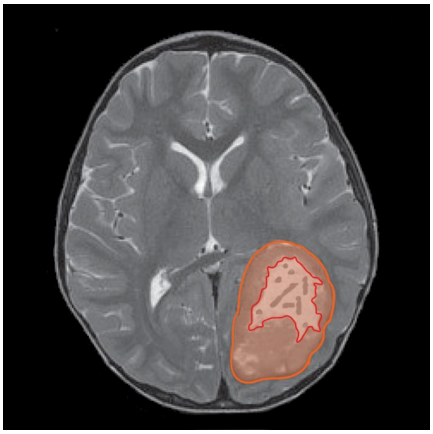
(g) Hexágono convexo

Figura 8.20: Centro comercial. Solución

8.1.6. Tumor cerebral (II)

En la Figura 8.21, se representa el contorno cerrado C (Figura 8.21a) y el polígono reticulado P (Figura 8.21b) de un tumor neuroectodérmico primitivo (TNEP) [66], tipo de tumor primario del sistema nervioso central (SNC) que afecta principalmente a niños, aunque también puede presentarse en adultos. El tratamiento recomendado consiste en llevar a cabo una extirpación quirúrgica completa, seguida de radioterapia. También se pueden observar conjuntos de células sanas en color oscuro que han ser evitadas durante la realización del tratamiento.

El objetivo principal consiste en calcular el k-gon de mayor área contenido en el tumor cerebral, para poder realizar posteriormente una intervención quirúrgica. Dada la naturaleza de este problema, se trata de un problema de inclusión con obstáculos arbitrarios.

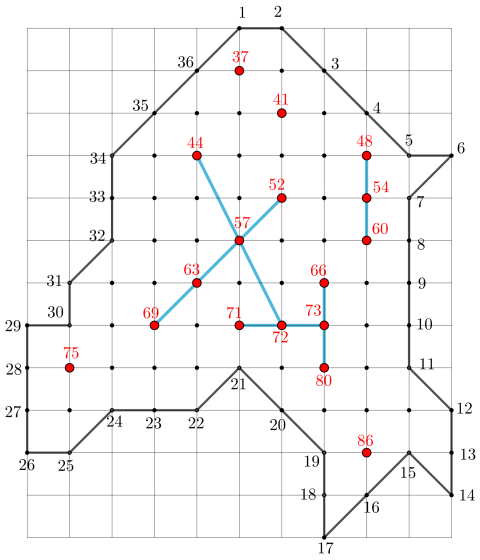


(a) Contorno cerrado C

(b) Polígono reticulado P

Figura 8.21: Tumor cerebral (II). Tumor neuroectodérmico primitivo (TNEP)

En la Figura 8.22 se proporcionan las coordenadas del polígono reticulado y los conjuntos necesarios para calcular la solución mediante los Algoritmos 7.1 y 7.2.



$$
\begin{bmatrix}
Points = \{p_1, \ldots, p_{86}\} \\
Polygon = \{p_1, \ldots, p_{36}\} \\
PointsP = \{p_{37}, p_{41}, p_{75}, p_{86}\} \\
SegmentsP = \{S_1, S_2, S_3, S_4, S_5\} \\
HolesP = \emptyset \\
S_1 = \{p_{44}, p_{57}, p_{72}\} \\
S_2 = \{p_{52}, p_{57}, p_{63}, p_{69}\} \\
S_3 = \{p_{48}, p_{54}, p_{60}\} \\
S_4 = \{p_{66}, p_{73}, p_{80}\} \\
S_5 = \{p_{71}, p_{72}, p_{73}\}
\end{bmatrix}
$$

Figura 8.22: Tumor cerebral (II). Definición del polígono reticulado

Las soluciones obtenidas se presentan en las Tablas 8.16, 8.17 y 8.18, acompañadas por las Figuras 8.23 y 8.24.

Tabla 8.16: Tumor cerebral (II). Solución: Polígono simple de k lados de mayor área

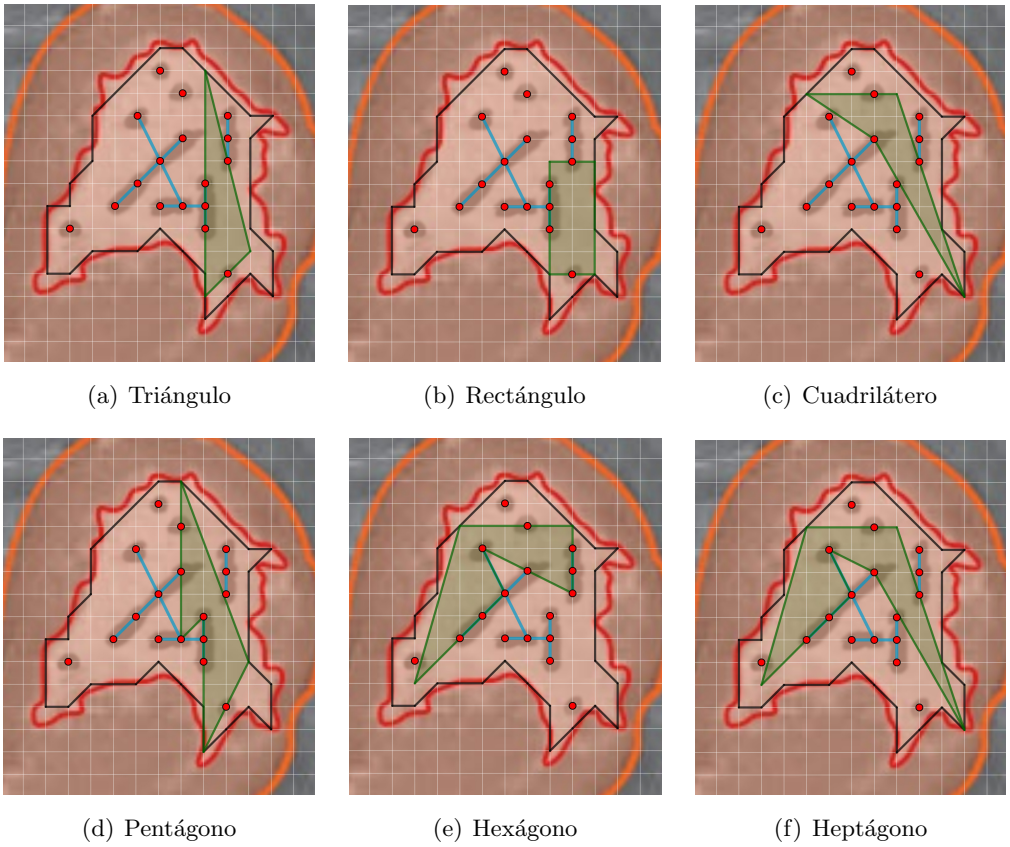
***k*-gon**	**Solution(86, distancia, matriz, 0, 0, 0, ver, 0)**	**Área**
Triángulo	(3,85,18), (11,42,17), (35,82,57)	10
Cuadrilátero	(14,52,35,42), (26,75,35,57)	11,5
Pentágono	(2,72,66,17,11), (11,42,39,59,17)	14,5
Hexágono	(4,60,44,57,82,35)	19
Heptágono	(14,52,44,57,82,35,42)	22,5

Tabla 8.17: Tumor cerebral (II). Solución: Rectángulo de mayor área

Rectangle(86, matriz, 0, 0, ver, 0)	**Área**
(8,15,19,59)	10

Tabla 8.18: Tumor cerebral (II). Solución: Polígono convexo de k lados de mayor área

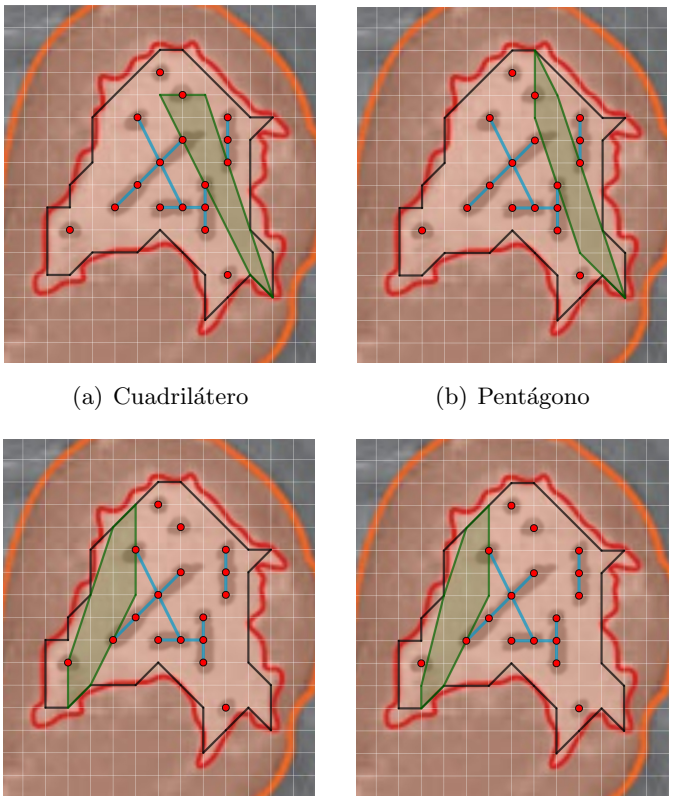
k-gon	Solution(86, distancia, matriz, 0, 0, 1, ver, 0)	Área
Cuadrilátero	(14,42,40,15), (35,75,82,57)	11
Pentágono	(2,46,84,14,42), (24,56,36,30,25), (35,75,82,57,44)	11,5
Hexágono	(24,56,36,35,75,25)	12
Heptágono	(24,56,36,35,32,82,25), (24,69,50,36,35,75,25)	11,5



(a) Triángulo (b) Rectángulo (c) Cuadrilátero

(d) Pentágono (e) Hexágono (f) Heptágono

Figura 8.23: Tumor cerebral (II). Solución: Polígono simple de k lados y rectángulo de mayor área



(a) Cuadrilátero (b) Pentágono

(c) Hexágono (d) Heptágono

Figura 8.24: Tumor cerebral (II). Solución: Polígono convexo de k lados de mayor área

8.2. Problemas de separabilidad, problemas de la envolvente

8.2.1. Invernadero

Un invernadero es una estructura cerrada, recubierta con materiales transparentes y diseñada para crear un microclima artificial que favorezca el cultivo de plantas durante todo el año [79]. Numerosas ventajas se obtienen de su construcción, como un mayor control de plagas, el aumento de la producción, la posibilidad de cultivar de forma continua durante todo el año y la creación de condiciones óptimas para la investigación. No obstante, la inversión inicial elevada para iniciar un invernadero, así como los altos costes de producción son desventajas económicas que han de tenerse en cuenta.

En lo que respecta a los árboles, no todos son apropiados para su cultivo en un invernadero. La selección de los árboles adecuados depende de varios factores, como el diseño del invernadero y las condiciones climáticas de la zona en la que se encuentra. En la Figura 8.25, se representa el contorno cerrado C de una plantación con diferentes árboles en una parcela agrícola. Una vez realizado el polígono reticulado P sobre dicho contorno (Figura 8.26), se clasifican los árboles en dos tipos: A y B. El primero es apropiado para el cultivo en invernaderos, mientras que el segundo no lo es. El objetivo principal consiste en calcular el invernadero de menor área de tal forma que contenga a los árboles del primer tipo y excluya a los del segundo tipo. La naturaleza del problema indica que se está ante un problema de separabilidad con obstáculos arbitrarios.

Figura 8.25: Invernadero. Contorno cerrado C

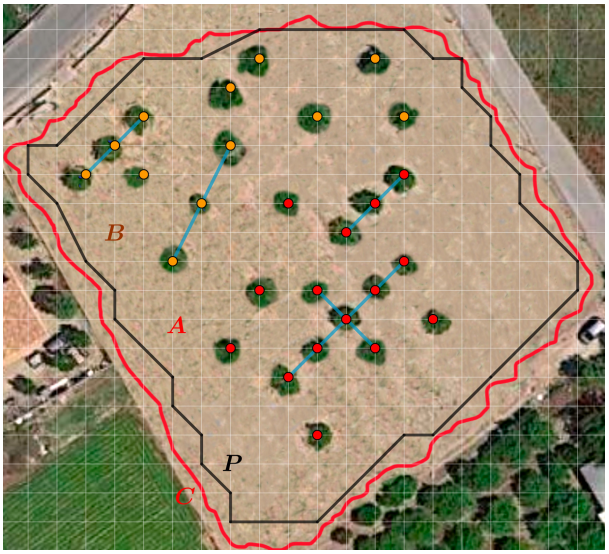


Figura 8.26: Invernadero. Polígono reticulado P

Para resolver el problema, en primer lugar se definen las coordenadas del polígono reticulado P (Figura 8.27) y luego se establecen los conjuntos necesarios para poder aplicar los algoritmos correspondientes.

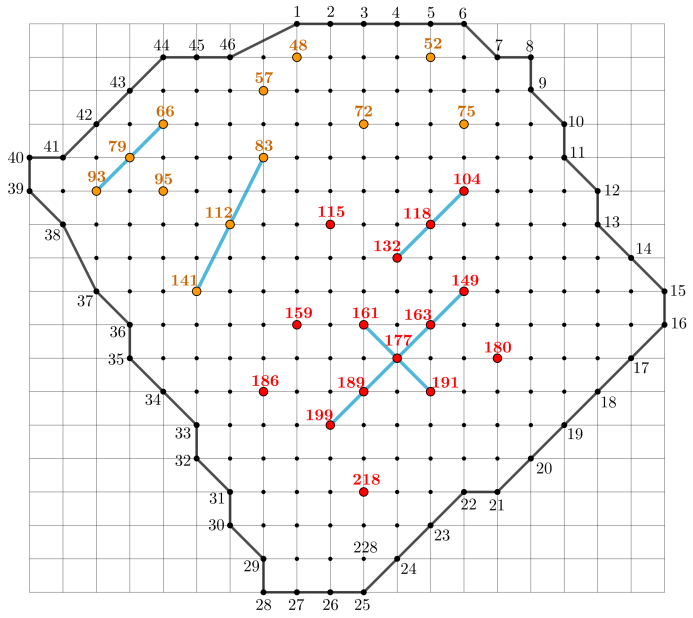


Figura 8.27: Invernadero. Definición del polígono reticulado

$$
\begin{cases}
Points = \{p_1, \ldots, p_{228}\} \\
Polygon = \{p_1, \ldots, p_{46}\} \\
PointsP = \{p_{48}, p_{52}, p_{57}, p_{72}, p_{75}, p_{95}, p_{115}, p_{159}, p_{180}, p_{186}, p_{218}\} \\
SegmentsP = \{S_1, S_2, S_3, S_4, S_5\} \\
HolesP = \emptyset \\
S_1 = \{p_{66}, p_{79}, p_{93}\} \\
S_2 = \{p_{83}, p_{112}, p_{141}\} \\
S_3 = \{p_{104}, p_{118}, p_{132}\} \\
S_4 = \{p_{161}, p_{177}, p_{191}\} \\
S_5 = \{p_{149}, p_{163}, p_{177}, p_{189}, p_{199}\} \\
O = \{ContP,\ ContS,\ ContH\},\ \text{donde:} \\
\quad ContP = \{p_{115}, p_{159}, p_{180}, p_{186}, p_{218}\} \\
\quad ContS = \{S_3, S_4, S_5\} \\
\quad ContH = \emptyset
\end{cases}
$$

Las soluciones obtenidas se detallan en las Tablas 8.19 y 8.20, mientras que para su comprobación gráfica, se incluye la Figura 8.28. Es importante destacar que no se ha encontrado ninguna solución para el triángulo de menor área.

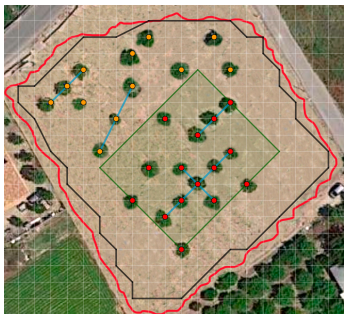
Tabla 8.19: Invernadero. Solución

***k*-gon**	**Solution(228, distancia, matriz, 0, 1, conv, ver, 1)**	**Área**
Triángulo	$\emptyset$	—
Cuadrilátero	(113,104,180,28), (73,172,218,152) (74,166,223,157), (74,171,218,166)	54
Pentágono	(104,180,218,186,115)	39
Hexágono Simple	(104,180,218,186,159,115)	38,5
Hexágono Convexo	(104,180,218,186,173,115)	40

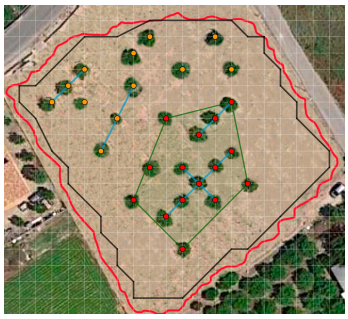
Tabla 8.20: Invernadero. Solución: Rectángulo de menor área

Rectangle(228, matriz, 0, 1, ver, 1)	**Área**
(156,73,152,218)	60

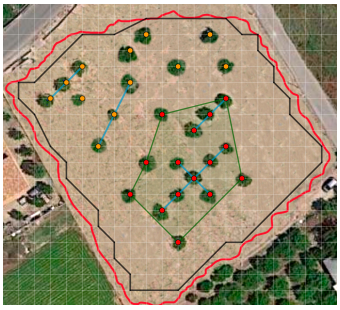
(a) Cuadrilátero



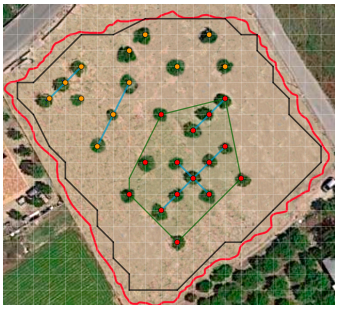
(b) Rectángulo



(c) Pentágono



(d) Hexágono simple



(e) Hexágono convexo

Figura 8.28: Invernadero. Solución

8.2.2. Parcelación

En el ámbito de la ganadería, la parcelación consiste en dividir una finca ganadera en parcelas más pequeñas con el objetivo de optimizar la gestión de los animales. Esta estrategia se lleva a cabo por diversas razones, como son para facilitar la rotación de pastos, manejo del ganado, aumentar la productividad y conservación del suelo y del agua.

En este contexto, en la Figura 8.29 se muestran dos tipos de animales representados por círculos rojos y azules, siendo la distribución que ahí aparece la típica para su ubicación habitual. Se ha determinado también el contorno cerrado C. Debido a una planificación que se quiere realizar sobre la finca, se quiere cercar el terreno para aislar por completo a los animales del primer tipo de los del segundo tipo, minimizando la longitud del perímetro vallado. Para lograrlo, en primer lugar, se determina el polígono reticulado P que se encuentra dentro de dicho contorno (Figura 8.30) y posteriormente se definen las coordenadas (Figura 8.31) y los conjuntos necesarios para calcular la solución. La naturaleza del problema indica que es un problema de separabilidad con obstáculos arbitrarios.

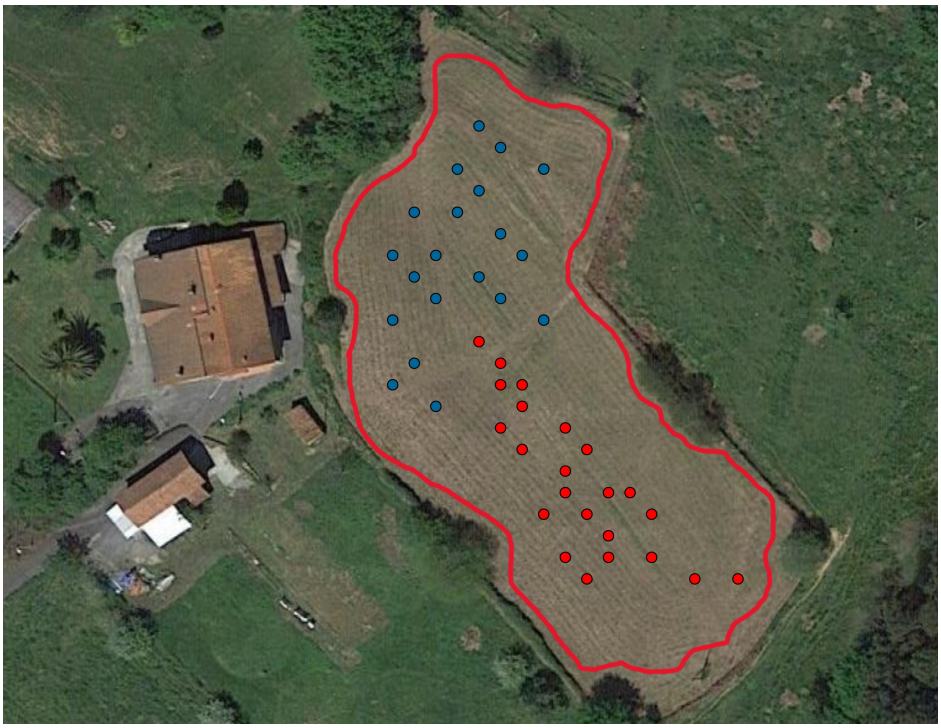


Figura 8.29: Parcelación. Contorno cerrado C

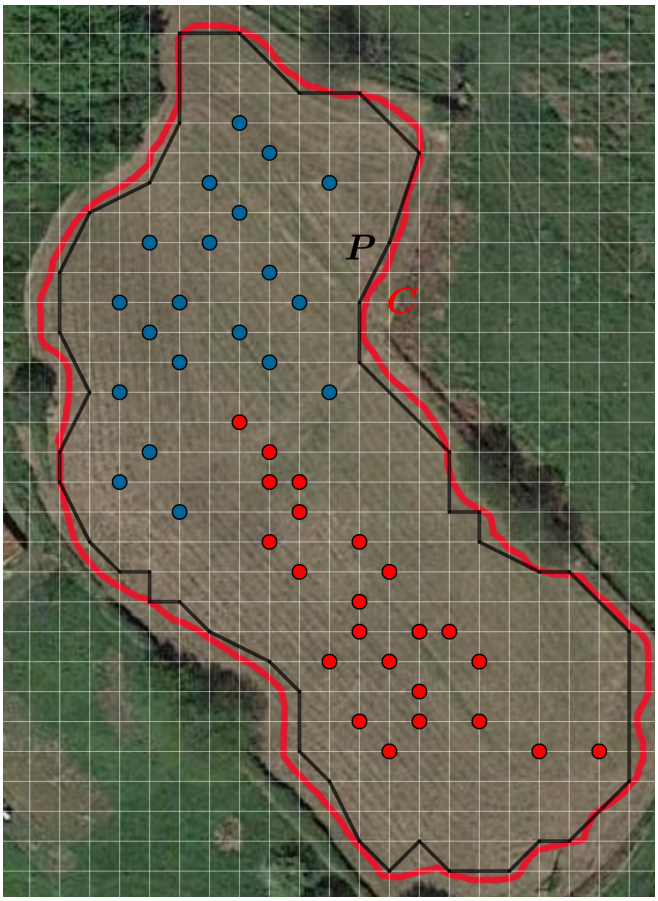


Figura 8.30: Parcelación. Polígono reticulado P

$$
\begin{aligned}
&Points = \{p_1, \ldots, p_{312}\}\\
&Polygon = \{p_1, \ldots, p_{60}\}\\
&PointsP = \{p_{67}, p_{75}, p_{81}, p_{85}, p_{92}, p_{100}, p_{102}, p_{114}, p_{119}, p_{121}, p_{125}, p_{129}, p_{132}, p_{139}, p_{142},\\
&\quad p_{145}, p_{152}, p_{159}, p_{167}, p_{171}, p_{178}, p_{183}, p_{184}, p_{192}, p_{196}, p_{206}, p_{209}, p_{217}, p_{220}, p_{230}, p_{242},\\
&\quad p_{244}, p_{245}, p_{252}, p_{254}, p_{257}, p_{265}, p_{273}, p_{275}, p_{277}, p_{284}, p_{289}, p_{291}\}\\
&SegmentsP = \emptyset\\
&HolesP = \emptyset\\
&O = \{ContP,\ ContS,\ ContH\},\ \text{donde:}\\
&\quad ContP = \{p_{159}, p_{171}, p_{183}, p_{184}, p_{196}, p_{206}, p_{209}, p_{217}, p_{220}, p_{230}, p_{242}, p_{244}, p_{245}, p_{252},\\
&\qquad p_{254}, p_{257}, p_{265}, p_{273}, p_{275}, p_{277}, p_{284}, p_{289}, p_{291}\}\\
&\quad ContS = \emptyset\\
&\quad ContH = \emptyset
\end{aligned}
$$

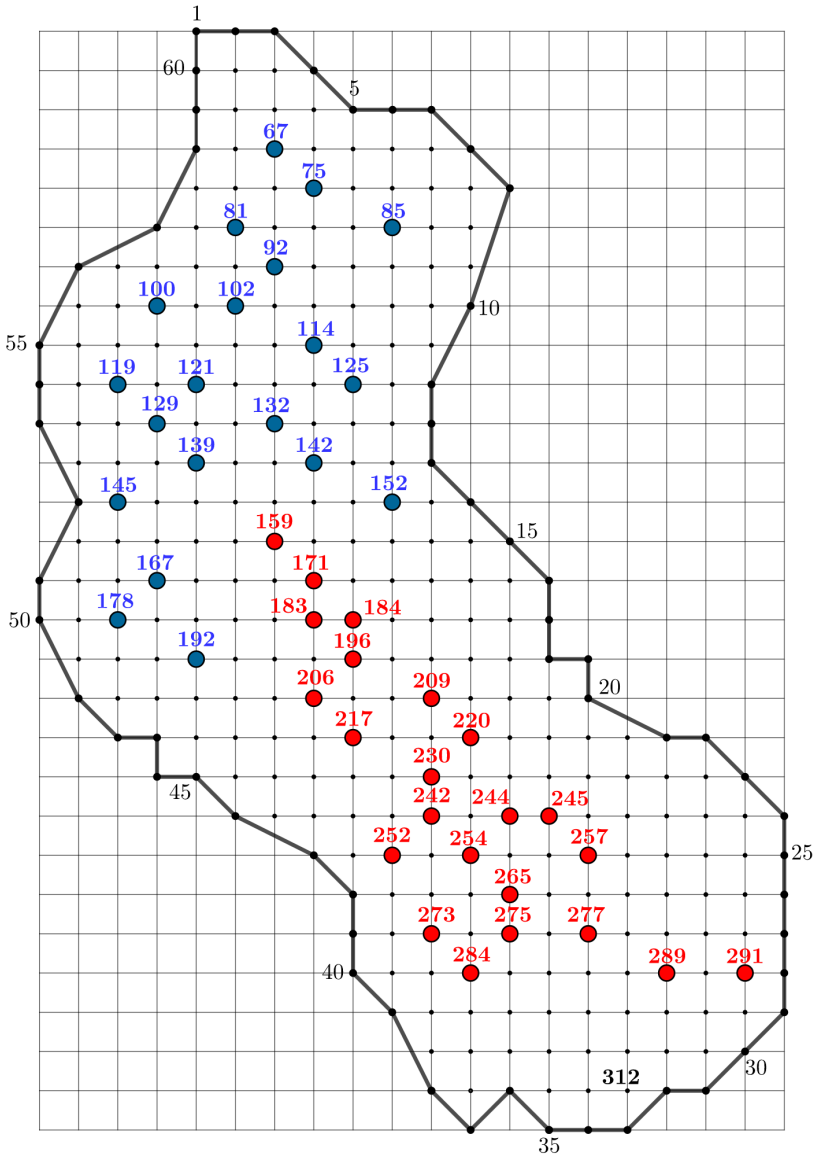
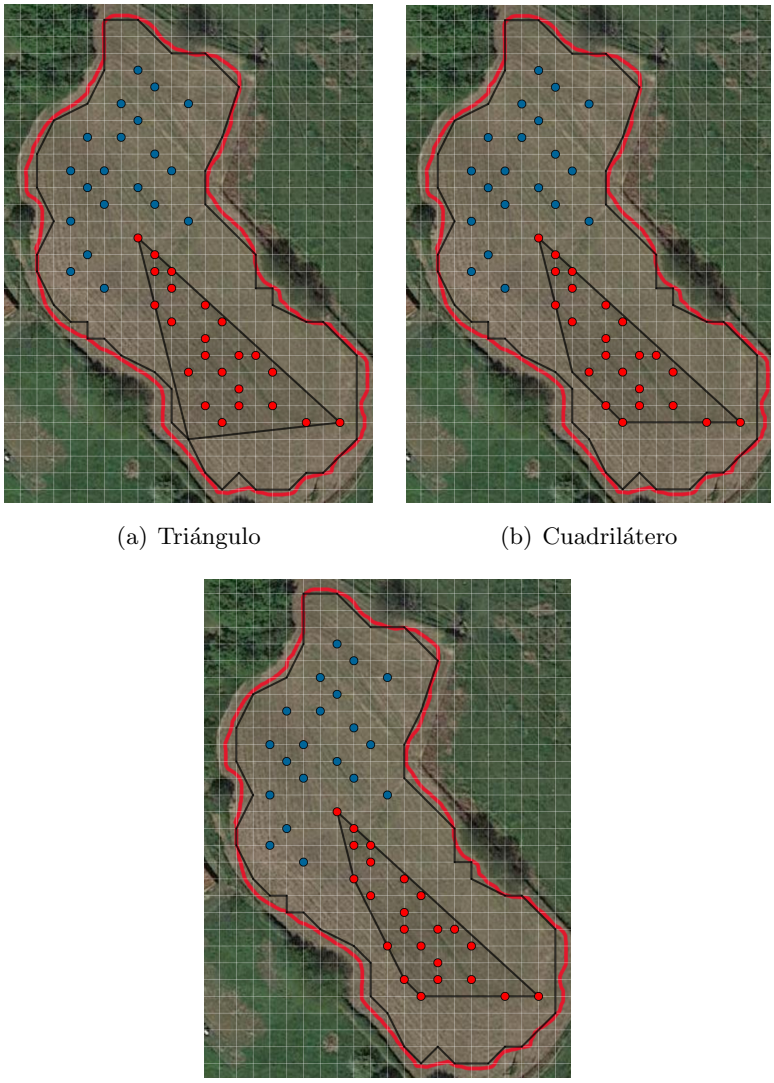


Figura 8.31: Parcelación. Definición del polígono reticulado

Las soluciones para polígonos, tanto simples como convexos, se presentan en la Tabla 8.21. Cabe destacar que no se ha hallado ninguna solución para el cálculo del rectángulo. Las representaciones geométricas de las soluciones previamente mencionadas se presentan en la Figura 8.32.

Tabla 8.21: Parcelación. Solución

***k*-gon**	**Solution(312, distancia, matriz, 1, 1, conv, ver, 1)**	**Perímetro**
Triángulo	(39,291,159)	18,85
Cuadrilátero	(159,291,284,251)	17,88
Pentágono	(159,291,284,273,206)	17,76



(a) Triángulo

(b) Cuadrilátero

(c) Pentágono

Figura 8.32: Parcelación. Solución

8.2.3. Tumor cerebral (III)

En la Figura 8.33, se puede observar el contorno cerrado C de una región del cerebro, donde se destaca en color gris oscuro un tumor. Debido el riesgo que representa, los médicos han decidido eliminarlo a través de una intervención quirúrgica que minimice al máximo el daño ocasionado a las células sanas circundantes. El objetivo principal es calcular el polígono de k lados de menor área que contenga al tumor. El planteamiento del problema indica que se está ante un problema de la envolvente.

Para lograr el objetivo anterior, en primer lugar se genera el polígono reticulado P contenido dentro del anterior contorno (Figura 8.34), y a continuación, se definen las coordenadas del polígono (Figura 8.35) y de los conjuntos necesarios para calcular la solución.

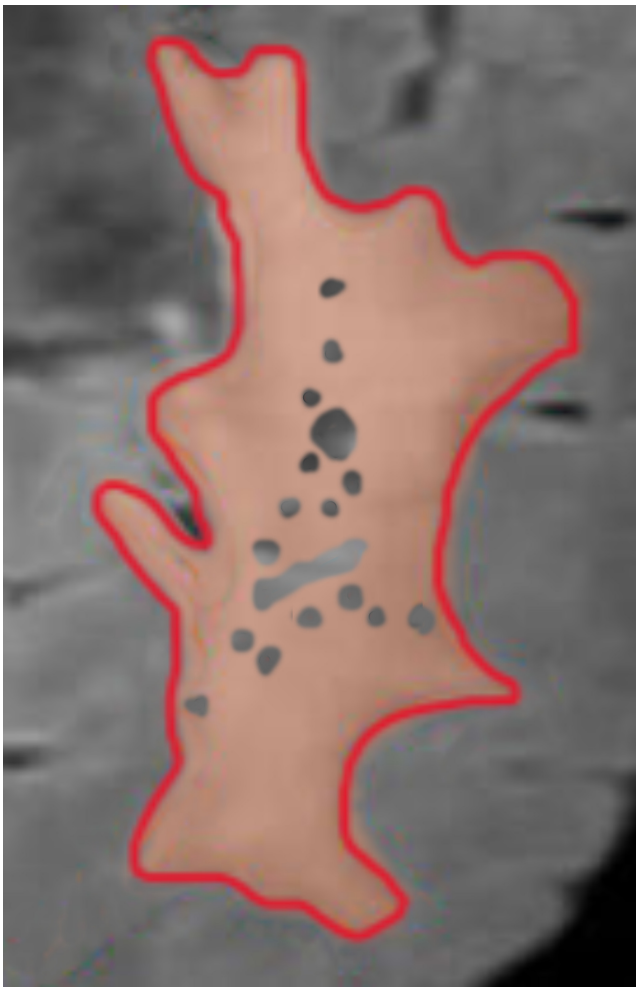


Figura 8.33: Tumor cerebral (III). Contorno cerrado C

Figura 8.34: Tumor cerebral (III). Polígono reticulado P

$$
\begin{cases}
Points = \{p_1, \ldots, p_{449}\} \\
Polygon = \{p_1, \ldots, p_{89}\} \\
PointsP = \{p_{146}, p_{187}, p_{216}, p_{254}, p_{267}, p_{275}, p_{277}, p_{296}, p_{324}, p_{333}, p_{336}, p_{338}, p_{341}, p_{354}, p_{377}\} \\
SegmentsP = S_1 = \{p_{300}, p_{311}, p_{320}\} \\
HolesP = H_1 = \{p_{218}, p_{231}, p_{244}, p_{256}, p_{255}, p_{230}\} \\
O = \{ContP,\ ContS,\ ContH\},\ \text{donde:} \\
\quad ContP = PointsP \\
\quad ContS = S_1 \\
\quad ContH = H_1
\end{cases}
$$

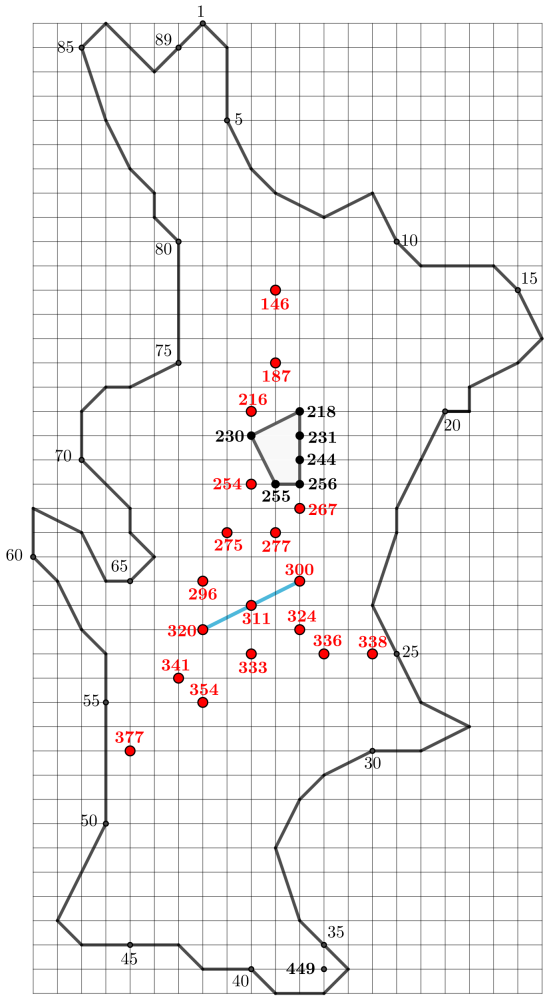
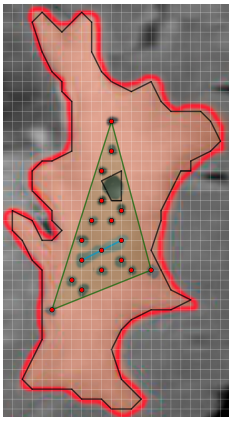


Figura 8.35: Tumor cerebral (III). Definición del polígono reticulado

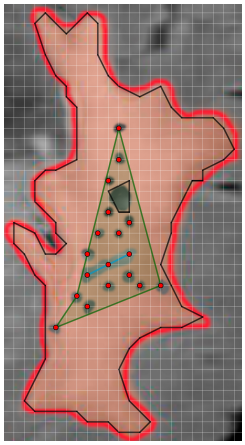
La Tabla 8.22 y la Figura 8.36 presentan las soluciones al utilizar los conjuntos anteriores. En el caso del rectángulo, no se ha obtenido ninguna solución.

Tabla 8.22: Tumor cerebral (III). Solución

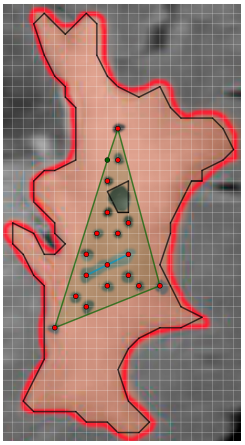
***k*-gon**	**Solution(449, distancia, matriz, 0, 1, conv, ver, 1)**	**Área**
Triángulo	(146,338,377)	83
Cuadrilátero	Simple	(146,341,377,338)	73
	Convexo	(146,338,377,186), (146,377,338,291)	83,5
Pentágono	Simple	(146,341,377,334,338)	65
	Convexo	(146,291,338,377,186)	84



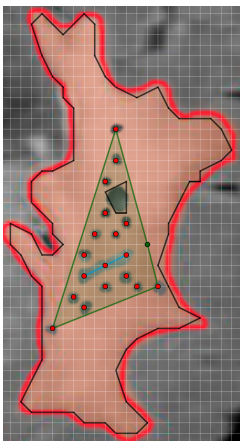
(a) Triángulo



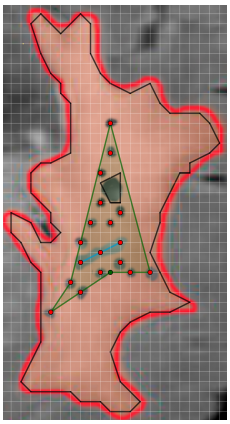
(b) Cuadrilátero simple



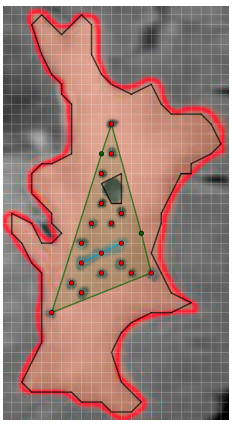
(c) Cuadrilátero convexo



(d) Cuadrilátero convexo



(e) Pentágono simple



(f) Pentágono convexo

Figura 8.36: Tumor cerebral (III). Solución

Capítulo 9

Paralelepípedo de mayor volumen y cualquier orientación contenido en un Volumen de Interés (VOI)

En este capítulo se presenta la resolución del último problema planteado en la Sección 1.1, relacionado con los Volúmenes de Interés (VOIs).

Industria de la piedra natural. Se cuenta con un bloque de granito de dimensiones conocidas y se pretende utilizarlo para fines decorativos. Si se busca obtener el máximo rendimiento del material, ¿cómo se podría calcular el paralelepípedo de mayor volumen que puede ser tallado a partir de este bloque?

Para respaldar el resultado obtenido, se hace referencia al artículo publicado por Molano et al. [122]. De esta forma, se establece una conexión entre el resultado presentado en este capítulo y el conocimiento ya establecido en la literatura. Además, se logra completar el último de los objetivos parciales que se establecieron en la Sección 1.6 dentro de esta Tesis Doctoral. La culminación de este proceso no solo refuerza el valor de este capítulo, sino que también establece un precedente para futuros trabajos que se centren en problemas similares relacionados con los Volúmenes de Interés (VOIs).

[**122**] R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *Finding the largest volume parallelepipedon of arbitrary orientation in a solid*, IEEE Access, 9 (2021), pp. 103600-103609.

Objetivo 6. Calcular el paralelepípedo de mayor volumen y cualquier orientación que está contenido en un Volumen de Interés (VOI).

9.1. Poliedro reticulado

9.1.1. Definición

Sea $K \subset S \subset \mathbb{R}^3$ el convexo de mayor volumen contenido en el VOI S [25, 93] y sea $O_{min} = [a,b] \times [c,d] \times [e,f]$, $a,b,c,d,e,f \in \mathbb{Z}$ el ortoedro de menor volumen que contiene a K [18]. Se llama *partición regular* $\Pi = \Pi_x \times \Pi_y \times \Pi_z$ de orden $r \times s \times t$, a tres colecciones ordenadas de $r+1$, $s+1$, $t+1$ puntos igualmente espaciados que verifican:

$$\begin{aligned} \Pi_x &= \{a = x_0 < x_1 < \ldots < x_r = b\} \\ \Pi_y &= \{c = y_0 < y_1 < \ldots < y_s = d\} \\ \Pi_z &= \{e = z_0 < z_1 < \ldots < z_t = f\} \end{aligned}$$

Sea $G_L = \{(x_i, y_j, z_k) : 0 \leq i \leq r,\ 0 \leq j \leq s,\ 0 \leq k \leq t\}$, la cuadrícula cúbica formada por los puntos de la partición Π. Se define *tamaño de partición*, $L = |x_{i+1} - x_i| = |y_{j+1} - y_j| = |z_{k+1} - z_k|$, a la longitud del lado de cada cubo formado por la cuadrícula cúbica. Además, se dice que la partición $\dot{\Pi}$ es más fina que la partición Π, si todos los puntos de Π pertenecen a $\dot{\Pi}$, y se denota como $\Pi \preceq \dot{\Pi}$.

Sea P el poliedro de mayor volumen cuyos puntos pertenecen a la cuadrícula G_L para una partición regular Π. Si las aristas del poliedro no intersectan excepto en sus vértices, se dice que P es un *poliedro reticulado* (Figura 9.1).

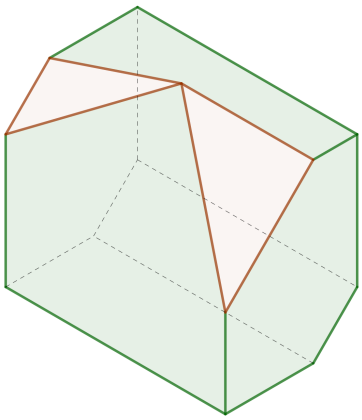


Figura 9.1: Poliedro reticulado P

Sea ∂P la familia de puntos frontera de P pertenecientes a la cuadrícula cúbica G_L, y su complementario en P, $\imath P$, los puntos del interior con coordenadas en G_L. Entonces, $P = \partial P \cup \imath P$ donde $\#(\partial P) = n$, $\#(\imath P) = o$, $\#(P) = N = n + o \simeq \lambda n, \lambda \in \mathbb{N}$.

9.1.2. Volumen de un poliedro reticulado

El teorema de Pick [31, 142] ofrece una expresión elegante para calcular el área de un polígono reticulado (Q). Este teorema se fundamenta en la relación entre el número de puntos que residen en la frontera del polígono (∂Q) y aquellos puntos que se encuentran en su interior ($\imath Q$), ambos pertenecientes a una cuadrícula con tamaño de partición L.

$$A(Q) \quad = \quad \left(\#(\imath Q) + \frac{\#(\partial Q)}{2} - 1 \right) \cdot L^2$$

La extensión del concepto utilizado en el Teorema de Pick al espacio tridimensional no es posible. Reeve [151, 152] demostró que no se puede establecer una relación similar entre el volumen de un poliedro y el número de puntos con coordenadas enteras en su interior y frontera, demostrando que las generalizaciones de conceptos geométricos simples en dimensiones más altas pueden volverse mucho más complejas y a menudo no tienen soluciones tan directas como en dimensiones más bajas.

Para resolver el problema, Reeve introduce un retículo secundario conocido como *retículo racional*,

$$\mathbb{Z}_n^3 = \{x \in \mathbb{R}^3 : nx \in \mathbb{Z}^3\},\ n \geq 1$$

donde $\mathbb{Z}_1^3 = \mathbb{Z}^3$.

Este retículo racional se convierte en una fuente adicional de información que resulta suficiente para la determinación del volumen de los poliedros reticulados. En este sentido, Kołodziejczyk y Reay [101], derivaron la siguiente fórmula general:

Teorema 9.1.1. *Si P es un poliedro reticulado en $\mathbb{R}^N$, entonces su volumen $V(P)$ se expresa como:*

$$V(P) = \frac{1}{(N+1)!} \sum_{k=1}^{N} (-1)^{N-k} \binom{N-1}{k-1} (B_k + 2I_k)$$

donde $B_k = \#(\mathbb{Z}_k^3 \cap \partial P)$ y $I_k = \#(\mathbb{Z}_k^3 \cap \imath P)$

Si se considera el poliedro reticulado P en $\mathbb{R}^3$, entonces la fórmula se puede simplificar de la siguiente forma:

$$V(P) = \frac{1}{4!} \sum_{k=1}^{3} (-1)^{3-k} \binom{2}{k-1} (B_k + 2I_k) = \frac{1}{4!} \ (B_1 + 2I_1 - 2(B_2 + 2I_2) + B_3 + 2I_3)$$

9.2. Cálculo del paralelepípedo de mayor volumen contenido en un poliedro reticulado

9.2.1. Paralelogramos contenidos en un poliedro reticulado (Algoritmo 9.1)

El Algoritmo 9.1, con coste computacional $O(n^3)$, calcula todos los paralelogramos que están contenidos dentro del poliedro reticulado P. Si $LP =< p_i, p_j, p_k, p_s >$ es uno de los paralelogramos (Figura 9.2), entonces $\overrightarrow{p_i p_j} = \overrightarrow{p_k p_s}$, es decir, $p_s = p_j - p_i + p_k$.

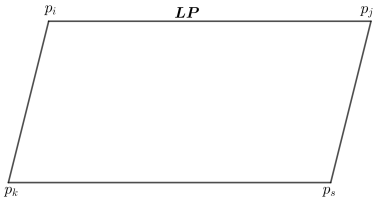


Figura 9.2: Paralelogramo $LP =< p_i, p_j, p_k, p_s >$

El algoritmo recorre todos los puntos del poliedro reticulado P, con el objetivo de buscar conjuntos de puntos $< p_i, p_j, p_k, p_s >$ que formen un paralelogramo (líneas 5 y 6). Una vez encontrados, el algoritmo calcula el área correspondiente mediante el módulo del producto vectorial de los vectores $\overrightarrow{p_i p_j}$ y $\overrightarrow{p_i p_k}$ (línea 7), para luego incluir esta solución en el conjunto global de resultados (líneas 8 y 9).

Algoritmo 9.1 Paralelogramos(P)

Input: P: poliedro reticulado, $\#(P) = N$

Output: Paralelogramos

```
1  paralelogramos ← ∅
2  for i ← 1 to N-3 do
3      for j ← i + 1 to N-2 do
4          for k ← j + 1 to N-1 do
5              p_s ← p_j − p_i + p_k
6              if p_s ∈ P then
7                  A ← |p_i p_j ⃗ × p_i p_k ⃗| // Producto vectorial
8                  LP ← (p_i, p_j, p_k, p_s, A)
9                  Insert(paralelogramos, LP)
10 return paralelogramos
```

9.2.2. Paralelepípedo de mayor volumen contenido en un poliedro reticulado (Algoritmo 9.2)

Sea $\{A_1, \ldots, A_u\}$ el conjunto de las áreas de los paralelogramos calculados mediante el Algoritmo 9.1, ordenadas de forma decreciente, y definidos de la siguiente forma:

$$\begin{aligned} A_1 &= \{LP_1^1, \ldots, LP_{n_1}^1\} = \{< p_{i_1}^1, p_{j_1}^1, p_{k_1}^1, p_{s_1}^1 >, \ldots, < p_{i_{n_1}}^1, p_{j_{n_1}}^1, p_{k_{n_1}}^1, p_{s_{n_1}}^1 >\} \\ &\vdots \\ A_u &= \{LP_1^u, \ldots, LP_{n_u}^u\} = \{< p_{i_1}^u, p_{j_1}^u, p_{k_1}^u, p_{s_1}^u >, \ldots, < p_{i_{n_u}}^u, p_{j_{n_u}}^u, p_{k_{n_u}}^u, p_{s_{n_u}}^u >\} \end{aligned}$$

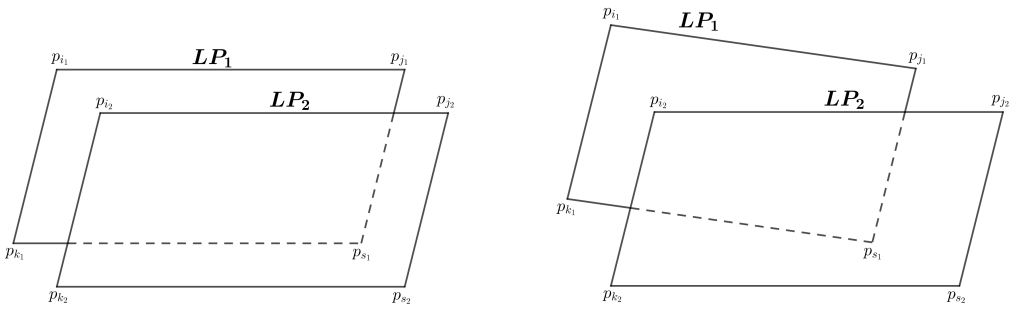
donde $area(LP_t^m) = \overline{A_m}$, $m \in \{1, \ldots, u\}$, $t \in \{1, \ldots, n_m\}$ y $\overline{A_p} > \overline{A_q}$ si $p < q$, $\forall p, q \in \{1, \ldots, u\}$.

Si se denota $\mathcal{LP}$ como el conjunto de todos los paralelogramos contenidos en P, entonces:

$$\begin{aligned} \mathcal{LP} &= \{A_1, \ldots, A_u\} = \left\{LP_1^1, \ldots, LP_{n_1}^1, LP_1^2, \ldots, LP_{n_2}^2, \ldots, LP_1^u, \ldots, LP_{n_u}^u\right\} = \\ &= \{LP_1, \ldots, LP_{n_1}, LP_{n_1+1}, \ldots, LP_{n_1+n_2}, \ldots, LP_{n_1+n_2+\ldots+n_u} = LP_M\} \end{aligned}$$

Definicion 9.2.1. *Sean $LP_1, LP_2 \in \mathcal{LP}$ y $\pi_1 \equiv A_1x + B_1y + C_1z + D_1 = 0$, $\pi_2 \equiv A_2x + B_2y + C_2z + D_2 = 0$ los planos definidos por $LP_1 =< p_{i_1}, p_{j_1}, p_{k_1}, p_{s_1} >$ y $LP_2 =< p_{i_2}, p_{j_2}, p_{k_2}, p_{s_2} >$, respectivamente. Entonces, LP_1 y LP_2 son **equipolentes** (Figura 9.3) si:*

$$\left[\begin{array}{l} \pi_1 \text{ y } \pi_2 \text{ son paralelos, } \dfrac{A_1}{A_2} = \dfrac{B_1}{B_2} = \dfrac{C_1}{C_2} \neq \dfrac{D_1}{D_2} \\ \overrightarrow{p_{i_1}p_{j_1}} = \overrightarrow{p_{i_2}p_{j_2}} \text{ y } \overrightarrow{p_{i_1}p_{k_1}} = \overrightarrow{p_{i_2}p_{k_2}} \end{array}\right.$$



(a) Equipolentes (b) No equipolentes

Figura 9.3: Paralelogramos equipolentes, no equipolentes

*Además, LP_1 y LP_2 son **fundamentales** y se denota $< LP_1 LP_2 >$, si existe un paralelepípedo $LPP \subset P$ a partir de los paralelogramos anteriores, es decir, $LPP =< LP_1 LP_2 > \subset P$, con volumen:*

$$Vol(LPP) = |\overrightarrow{p_{i_1}p_{i_2}} \cdot (\overrightarrow{p_{i_1}p_{j_1}} \times \overrightarrow{p_{i_1}p_{k_1}})|$$

definido como el módulo del producto mixto o triple producto escalar de los vectores $\overrightarrow{p_{i_1}p_{i_2}}$, $\overrightarrow{p_{i_1}p_{j_1}}$ y $\overrightarrow{p_{i_1}p_{k_1}}$.

En la Figura 9.4, se presentan tres paralelogramos fundamentales que generan el paralelepípedo $LPP =< LP_1 LP_3 >$.

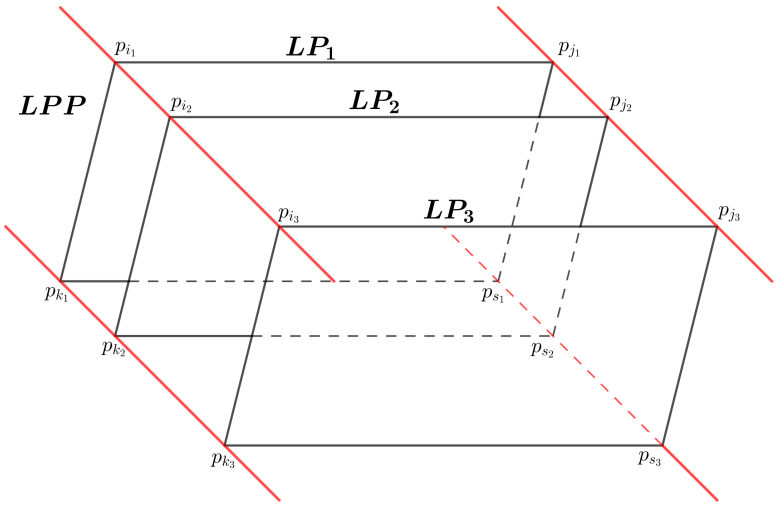


Figura 9.4: Paralelogramos fundamentales

El Algoritmo 9.2 utiliza tres funciones. En primer lugar, la función *Fundamentals(LP_1, LP_2)* que verifica si dos paralelogramos son fundamentales, y devuelve *true* en caso afirmativo. Luego, la función *Update(volume, LPP, $\overline{LPP}$)* que realiza la actualización de las soluciones, guardando siempre el paralelepípedo con mayor volumen. Por último, la función $LPP_{max}(LPP_a, LPP_b)$, con $LPP_a =< LP_a^1 LP_a^2 >$ y $LPP_b =< LP_b^1 LP_b^2 >$, que devuelve el mayor paralelepípedo a partir de los paralelogramos anteriores. El coste computacional de cada una de esta funciones es constante.

El algoritmo recorre el conjunto de los paralelogramos previamente calculados en el Algoritmo 9.1 (líneas 3, 4, y 10), ordenados de manera decreciente según su área. A medida que avanza, el algoritmo iterativamente calcula y actualiza la solución (líneas 9 y 17), determinando el paralelepípedo de mayor volumen que está contenido dentro del poliedro reticulado P. El coste computacional es $O(n^3)$.

Algoritmo 9.2 Paralelepipedos(paralelogramos)

Input: $\{A_1, \ldots, A_u\} = \left\{LP_1^1, \ldots, LP_{n_1}^1, LP_1^2, \ldots, LP_{n_2}^2, \ldots, LP_1^u, \ldots, LP_{n_u}^u\right\} = \{LP_1, \ldots, LP_{n_1}, LP_{n_1+1}, \ldots, LP_{n_1+n_2}, \ldots, LP_{n_1+n_2+\ldots+n_u} = LP_M\}, M \simeq kn, k \in \mathbb{N}, \overline{A_m} = area(LP_m)$

Output: Paralelepípedos de mayor volumen

```
1  max ← 0
2  paralelepipedos ← ∅
3  for a ← 1 to M-2 do
4      for b ← a + 1 to M-1 do
5          if (Ā_a = Ā_b and Fundamentals(LP_a, LP_b)) then
6              aux ← ∅
7              LPP_{a,b} ←< LP_a LP_b >
8              Insert(aux, LPP_{a,b})
9              Update(max, LPP_{a,b}, paralelepipedos)
10             for c ← b + 1 to M do
11                 if (Ā_b = Ā_c and Fundamentals(LP_b, LP_c)) then
12                     LPP_{b,c} ←< LP_b LP_c >
13                     Insert(aux, LPP_{b,c})
14                     LPP ← LPP_max(aux)
15                     aux ← ∅
16                     Insert(aux, LPP)
17                     Update(max, LPP, paralelepipedos)
18 return paralelepipedos
```

9.3. Cálculo del paralelepípedo de mayor volumen contenido en un VOI

Definicion 9.3.1. *Sea K el convexo de mayor volumen contenido en el VOI S y Π una partición regular definida sobre el ortoedro de menor volumen O_{min} que contiene a K, con tamaño de partición L.*

*Se llama **volumen inferior** $\underline{V}(K,\Pi)$ y **volumen superior** $\overline{V}(K,\Pi)$, al poliedro reticulado $\underline{P}$ de mayor volumen contenido en K y al poliedro reticulado $\overline{P}$ de menor volumen que contiene a K, respectivamente, y ambos construidos por puntos de G_L. Por el Teorema 9.1.1,*

$$\begin{aligned}\underline{V}(K,\Pi) &= \frac{1}{4!}\sum_{k=1}^{3}(-1)^{3-k}\binom{2}{k-1}(\underline{B_k}+2\underline{I_k})\\ \overline{V}(K,\Pi) &= \frac{1}{4!}\sum_{k=1}^{3}(-1)^{3-k}\binom{2}{k-1}(\overline{B_k}+2\overline{I_k})\end{aligned}$$

con $\underline{B_k}=\#(\mathbb{Z}_k^3\cap\partial\underline{P})$, $\underline{I_k}=\#(\mathbb{Z}_k^3\cap\imath\underline{P})$, $\overline{B_k}=\#(\mathbb{Z}_k^3\cap\partial\overline{P})$ y $\overline{I_k}=\#(\mathbb{Z}_k^3\cap\imath\overline{P})$, $k=1,2,3$.

Además, sean $\underline{LPP}$ y $\overline{LPP}$ los paralelepípedos de mayor volumen contenidos en $\underline{P}$ y $\overline{P}$ con volúmenes $\underline{V_{LPP}}(K,\Pi)$ y $\overline{V_{LPP}}(K,\Pi)$, respectivamente, y calculados a partir del Algoritmo 9.2.

Proposicion 9.3.2. *Si K es convexo, entonces $\underline{V}(K)=\overline{V}(K)$, donde:*

$$\begin{aligned}\underline{V}(K) &= \sup\{\underline{V}(K,\Pi):\Pi \textit{ partición regular}\}\\ \overline{V}(K) &= inf\{\overline{V}(K,\Pi):\Pi \textit{ partición regular}\}\end{aligned}$$

Demostración. Sea Π una partición regular más fina que $\dot{\Pi}$ y $\ddot{\Pi}$. Por tanto, $\underline{V}(K,\dot{\Pi})\leq\underline{V}(K,\Pi)$ y $\overline{V}(K,\ddot{\Pi})\geq\overline{V}(K,\Pi)$, luego $\underline{V}(K,\dot{\Pi})\leq\underline{V}(K,\Pi)\leq\overline{V}(K,\Pi)\leq\overline{V}(K,\ddot{\Pi})$ lo que implica $\underline{V}(K)\leq\overline{V}(K,\Pi)$ para toda partición regular Π. Además, como $\overline{V}(K)\leq\overline{V}(K,\Pi)$, se obtiene $\underline{V}(K)\leq\overline{V}(K)$. Finalmente, como K es convexo, $\underline{V}(K)=\overline{V}(K)$.

El valor común $\underline{V}(K)=\overline{V}(K)$ se llama **volumen de K** y se denota por $V(K)$. □

El Teorema 9.3.3 proporciona una demostración de cómo calcular el paralelepípedo de mayor volumen contenido en un VOI. Esto se logra realizando particiones regulares cada vez más finas y aplicando el Algoritmo 9.2 sobre el ortoedro de menor volumen que contiene al convexo de mayor volumen contenido en el VOI S.

Teorema 9.3.3. *Sea K convexo. Entonces, existe una sucesión de particiones regulares $\{\Pi_n\}_{n\in\mathbb{N}}$ con $\Pi_i \preceq \Pi_{i+1}$ para todo i, tal que*

$$\lim_{n\to\infty} \underline{V_{LPP}}(K,\Pi_n) = V_{LPP}(K)$$

donde $V_{LPP}(K)$ es el volumen del mayor paralelepípedo contenido en K.

Demostración. Por la Proposición 9.3.2, $\underline{V}(K) = \overline{V}(K)$, luego se pueden construir dos particiones regulares $\dot{\Pi}$, $\ddot{\Pi}$ tal que $|\overline{V}(K,\ddot{\Pi}) - \underline{V}(K,\dot{\Pi})| < \varepsilon$, para todo $\varepsilon > 0$. Sea Π una partición regular más fina que $\dot{\Pi}$ y $\ddot{\Pi}$. Entonces, $\overline{V}(K,\Pi) \leq \overline{V}(K,\ddot{\Pi})$ y $\underline{V}(K,\Pi) \geq \underline{V}(K,\dot{\Pi})$. Por tanto, $|\overline{V}(K,\Pi) - \underline{V}(K,\Pi)| \leq |\overline{V}(K,\ddot{\Pi}) - \underline{V}(K,\dot{\Pi})| < \varepsilon$.

Como $\underline{V_{LPP}}(K,\Pi) \leq \underline{V}(K,\Pi)$ y $\overline{V_{LPP}}(K,\Pi) \leq \overline{V}(K,\Pi)$, entonces $|\overline{V_{LPP}}(K,\Pi) - \underline{V_{LPP}}(K,\Pi)| \leq |\overline{V}(K,\Pi) - \underline{V}(K,\Pi)| < \varepsilon$. Así, existe una sucesión de particiones regulares $\{\Pi_n\}_{n\in\mathbb{N}}$ con $\Pi_i \preceq \Pi_{i+1}$ para todo i tal que $\lim_{n\to\infty}(\overline{V_{LPP}}(K,\Pi_n) - \underline{V_{LPP}}(K,\Pi_n)) = 0$, de manera que $\lim_{n\to\infty} \underline{V_{LPP}}(K,\Pi_n) = V_{LPP}(K)$. □

Capítulo 10

Descripción de los algoritmos utilizados

10.1. Matriz de adyacencia

3.1. Orientation(P, Q, R) $O(1)$

Orientación de puntos en el plano.

3.2. Alignedp(P, Q, R) $O(1)$

Punto dentro de un segmento.

3.3. AlignedPol(P, poly) $O(n, m)$

Punto contenido en algún lado de un polígono.

3.4. PointIn(P, poly) $O(n, m)$

Punto dentro de un polígono.

3.5. Intersection(l_1, l_2) $O(1)$

Intersección de segmentos.

3.6. Intersectionp(l_1, l_2) $O(1)$

Punto de intersección de dos segmentos.

3.7. IntersectionPoly(l, s, poly, num) $O(n, m^2)$

Intersección segmento–lado.

3.8. IntersectionPol(l) $O(n^2)$

Intersección segmento–polígono.

3.9. Matriz(Points) $O(n^5)$

Matriz de adyacencia de un polígono reticulado con obstáculos arbitrarios.

10.2. Problemas de inclusión, $Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$

4.1. PolygonsIncI(point1, point2, distancia, matriz) $O(n^3k)$

Cálculo entre dos puntos del polígono simple de k lados contenido en un polígono reticulado con obstáculos arbitrarios.

4.2. IntersectionLad(poly) $O(k^2)$

Intersección de los lados de un polígono.

4.3. Alignedc(polyk) $O(k)$

Puntos consecutivos alineados.

4.4. InterPointsP(poly) $O(k^2r)$

Posición puntos–polígono.

4.5. InterSegmentsP(poly) $O(k^2st)$

Posición segmentos–polígono.

4.6. InterHolesP(poly, hole) $O(k^2m)$

Posición agujeros–polígono.

4.7. Function(poly, fun) $O(k)$

Área o perímetro de un polígono reticulado.

4.8. UpdateIncI(pol, fun) $O(n^2k)$

Actualización de soluciones.

4.9. Duplicates(pol) $O(n^4k \log k)$

Eliminación de soluciones duplicadas.

4.10. SolutionIncI(N, distancia, matriz, fun) $O(n^5k)$

Cálculo del polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

4.11. Rect(poly) $O(1)$

Comprobar si un polígono es un rectángulo.

4.12. RectangleIncI(N, matriz, fun) $O(n^5)$

Cálculo del rectángulo de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

4.13. Convex(poly) $O(k)$

Comprobar si un polígono es convexo.

4.14. PolygonsConvexIncI(point1, point2, distancia, matriz) $O(n^3k)$

Cálculo entre dos puntos del polígono convexo de k lados contenido en un polígono reticulado con obstáculos arbitrarios.

4.15. SolutionConvexIncI(N, distancia, matriz, fun) $O(n^5k)$

Cálculo del polígono convexo de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

4.16. PolygonsIncR(point1, point2, first, iter, lados, distancia, matriz, fun) $O(n^k)$

Cálculo entre dos puntos del polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

4.17. UpdateIncR(poly, fun, distancia) $O(k)$

Actualización de soluciones.

4.18. SolutionIncR(N, distancia, matriz, fun) $O(n^k)$

Cálculo del polígono simple de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

4.19. StartIncR(point1, point2, distancia, matriz, fun) $O(n^k)$

Inicio del proceso recursivo.

4.20. RectangleIncR(N, matriz, fun) $O(n^6)$

Cálculo del rectángulo de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

4.21. SolutionConvexIncR(N, distancia, matriz, fun) $O(n^k)$

Cálculo del polígono convexo de k lados de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

4.22. SolutionInc(N, distancia, matriz, fun, conv, ver) $O(n^5k, n^k)$

Cálculo del polígono simple o convexo de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

4.23. RectangleInc(N, matriz, fun, ver) $O(n^5, n^6)$

Cálculo del rectángulo de mayor área o perímetro contenido en un polígono reticulado con obstáculos arbitrarios.

10.3. Problemas de separabilidad, $Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu)$

5.1. PolygonsSepI(point1, point2, distancia, matriz) $O(n^3k)$

Cálculo entre dos puntos del polígono simple de k lados que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.2. DifferenceP(set, cont) $O(r)$

Diferencia de conjuntos.

5.3. InterPointsP2(poly, diff) $O(kr)$

Posición puntos–polígono.

5.4. DifferenceSH(set, cont) $O(t^2, h^2)$

Diferencia de conjuntos.

5.5. InterSH(poly, cont, diff) $O(kst, kmh)$

Posición segmentos–polígono, agujeros–polígono.

5.6. SolutionSepI(N, distancia, matriz, fun, upd) $O(n^5k)$

Cálculo del polígono simple de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.7. UpdateSepI(pol, fun, upd) $O(n^2k)$

Actualización de soluciones.

5.8. RectangleSepI(N, matriz, fun, upd) $O(n^5)$

Cálculo del rectángulo de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.9. SolutionConvexSepI(N, distancia, matriz, fun, upd) $O(n^5k)$

Cálculo del polígono convexo de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.10. PolygonsSepR(point1, point2, first, iter, lados, distancia, matriz, fun, upd) $O(n^k)$

Cálculo entre dos puntos del polígono simple de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.11. UpdateSepR(poly, fun, distancia, upd) $O(k)$

Actualización de soluciones.

5.12. SolutionSepR(N, distancia, matriz, fun, upd) $O(n^k)$

Cálculo del polígono simple de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.13. StartSepR(point1, point2, distancia, matriz, fun, upd) $O(n^k)$

Inicio del proceso recursivo.

5.14. RectangleSepR(N, matriz, fun, upd) $O(n^6)$

Cálculo del rectángulo de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.15. SolutionConvexSepR(N, distancia, matriz, fun, upd) $O(n^k)$

Cálculo del polígono convexo de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.16. SolutionSep(N, distancia, matriz, fun, upd, conv, ver) $O(n^5k, n^k)$

Cálculo del polígono simple o convexo de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

5.17. RectangleSep(N, matriz, fun, upd, ver) $O(n^5, n^6)$

Cálculo del rectángulo de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

10.4. Unificación de problemas: Inclusión, Separabilidad y Envolvente

7.1. Solution(N, distancia, matriz, fun, upd, conv, ver, prob) $O(n^5k, n^k)$

Cálculo del polígono simple o convexo de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

7.2. Rectangle(N, matriz, fun, upd, ver, prob) $O(n^5, n^6)$

Cálculo del rectángulo de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado con obstáculos arbitrarios.

10.5. Paralelepípedo de mayor volumen

8.1. Paralelogramos(P) $O(n^3)$

Paralelogramos contenidos en un poliedro reticulado.

8.2. Paralelepipedos(paralelogramos) $O(n^3)$

Paralelepípedos de mayor volumen contenido en un poliedro reticulado.

Capítulo 11

Conclusiones y trabajos futuros

11.1. Conclusiones

La conclusión principal de esta Tesis Doctoral se deduce del Algoritmo 7.1: cálculo del polígono simple o convexo de k lados de mayor o menor área o perímetro que contiene a un conjunto O y está contenido en un polígono reticulado o Región de Interés (ROI) con obstáculos arbitrarios, como puntos, segmentos y agujeros. Si es necesario calcular el área o el perímetro de un rectángulo en lugar de un polígono simple o convexo, se dispone del Algoritmo 7.2 para llevar a cabo esta tarea.

Algoritmo 7.1. Solution(N, distancia, matriz, fun, upd, conv, ver, prob)
Algoritmo 7.2. Rectangle(N, matriz, fun, upd, ver, prob)

El Algoritmo 7.1 consta de ocho parámetros: N, indica el número del puntos del polígono reticulado; k = distancia +1, define el número de lados del polígono a calcular; *matriz*, que es la matriz de adyacencia; *fun*, indica si calcular el área (*fun = 0*) o el perímetro (*fun = 1*); *upd*, permite seleccionar el valor máximo (*upd = 0*) o mínimo (*upd = 1*); *conv*, ofrece la opción de obtener un polígono simple (*conv = 0*) o convexo (*conv = 1*); *ver*, permite elegir entre una versión iterativa (*ver = 0*) o recursiva (*ver = 1*) del algoritmo; y finalmente, *prob*, determina si resolver un problema de inclusión (*prob = 0*) o un problema de separabilidad o envolvente (*prob = 1*).

La presencia de estos ocho parámetros y la habilidad para calcular polígonos con cualquier número de lados permiten que tanto el Algoritmo 7.1 como el Algoritmo 7.2, ofrezcan una amplia variedad de posibilidades para adaptarse a diferentes necesidades. Ya no es necesario implementar algoritmos para calcular, por ejemplo, un pentágono convexo de mayor perímetro contenido en un polígono reticulado o un rectángulo de

menor área que contenga a una ROI. A través de estos únicos algoritmos, es posible obtener soluciones para cualquier escenario, ya sea maximizando o minimizando el área o el perímetro, generando un polígono simple o convexo, utilizando un algoritmo iterativo o recursivo, o incluso, resolviendo problemas de inclusión, separabilidad o envolvente.

Además, se ha resuelto un problema relacionado con Volúmenes de Interés (VOIs), lo que amplía aún más el alcance de la investigación y su interés en la aplicación de soluciones prácticas.

Esta conclusión principal está estrechamente relacionada con el objetivo principal establecido en la Sección 1.6 y desempeña un papel fundamental al contribuir al logro de los seis objetivos parciales descritos en esa misma sección. De esta manera, el Capítulo 4 Problemas de inclusión, $Inc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$, permite conseguir los objetivos parciales 1 y 2; el Capítulo 5 Problemas de separabilidad, $Sep(\mathcal{P}_{sim}, \mathcal{P}_k, O, \mu)$, el objetivo parcial 3; el Capítulo 6 Problemas de la envolvente, $Enc(\mathcal{P}_{sim}, \mathcal{P}_k, \mu)$, el objetivo parcial 4; y por último, el Capítulo 9 Paralelepípedo de mayor volumen y cualquier orientación contenido en un Volumen de Interés (VOI), el objetivo parcial 6. El objetivo parcial 5, orientado hacia el desarrollo del pseudocódigo y el código fuente en los lenguajes de programación Python y Java, ha sido considerado de gran importancia para obtener las soluciones del Capítulo 8 Resultados de la aplicación práctica.

11.2. Trabajos futuros

En esta Tesis Doctoral, se han identificado dos direcciones en las que se podría dar continuidad la investigación presentada.

En primer lugar, sería respecto al coste computacional. Actualmente, los Algoritmos 7.1 y 7.2 presentan variaciones significativas en términos de coste, dependiendo de la versión desarrollada, ya sea la iterativa o la recursiva. En el caso de la primera versión, los costes son $O(n^5k)$ y $O(n^5)$, respectivamente. Mientras que para la segunda versión, los costes ascienden a $O(n^k)$ y $O(n^6)$, respectivamente. Una dirección de investigación podría centrarse en el desarrollo de un nuevo algoritmo capaz de reducir estos costes computacionales, lo que tendría un impacto significativo en la optimización geométrica.

En segundo lugar, otra área de interés consistiría en desarrollar una interfaz gráfica diseñada para representar los puntos del polígono reticulado y, posteriormente, aplicar los algoritmos 7.1 y 7.2 para calcular la solución deseada. Esto no solo simplificaría la comprensión de los resultados por parte de los usuarios, sino que también haría que la herramienta fuera más práctica en aplicaciones del mundo real. De este modo, se contribuiría a la resolución de problemas relacionados con la geometría computacional.

Bibliografía

[1] A. Acharyya, M. De, S. C. Nandy, and S. Pandit, *Variations of largest rectangle recognition amidst a bichromatic point set*, Discrete Applied Mathematics, 286 (2020), pp. 35–50.

[2] A. Aggarwal, J.-S. Chang, and C. K. Yap, *Minimum area circumscribing polygons*, The Visual Computer, 1 (1985), pp. 112–117.

[3] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, *Geometric applications of a matrix searching algorithm*, in Proceedings of the second annual symposium on Computational geometry, 1986, pp. 285–292.

[4] A. Aggarwal and J. Park, *Notes on searching in multidimensional monotone arrays*, in 29th Annual Symposium on Foundations of Computer Science, 1988, pp. 497–512.

[5] A. Aggarwal and S. Suri, *Fast algorithms for computing the largest empty rectangle*, in Proceedings of the third annual symposium on Computational geometry, 1987, pp. 278–290.

[6] S. Alqazzaz, X. Sun, X. Yang, and L. Nokes, *Automated brain tumor segmentation on multi-modal mr image using segnet*, Computational Visual Media, 5 (2019), pp. 209–219.

[7] H. Alt, J. Blömer, and H. Wagener, *Approximation of convex polygons*, in Automata, Languages and Programming: 17th International Colloquium Warwick University, England, July 16–20, 1990 Proceedings 17, Springer, 1990, pp. 703–716.

[8] H. Alt, D. Hsu, and J. Snoeyink, *Computing the largest inscribed isothetic rectangle*, in CCCG, 1995, pp. 67–72.

[9] Ş. Andrei and C. Masalagiu, *About the collatz conjecture*, Acta Informatica, 35 (1998), pp. 167–179.

[10] K. Appel and W. Haken, *The solution of the four-color-map problem*, Scientific American, 237 (1977), pp. 108–121.

[11] D. S. Arnon and J. P. Gieselmann, *A linear time algorithm for the minimum area rectangle enclosing a convex polygon*, (1983).

[12] R. Ashraf, S. Afzal, A. U. Rehman, S. Gul, J. Baber, M. Bakhtyar, I. Mehmood, O.-Y. Song, and M. Maqsood, *Region-of-interest based transfer learning assisted framework for skin cancer detection*, IEEE Access, 8 (2020), pp. 147858–147871.

[13] J. Augustine, S. Das, A. Maheshwari, S. Nandy, S. Roy, and S. Sarvattomananda, *Querying for the largest empty geometric object in a desired location*, arXiv preprint arXiv:1004.0558, (2010).

[14] A. Awasthi, A. K. Shukla, M. M. SR, C. Dondariya, K. Shukla, D. Porwal, and G. Richhariya, *Review on sun tracking technology in solar pv system*, Energy Reports, 6 (2020), pp. 392–405.

[15] J. Backer and J. M. Keil, *The bichromatic square and rectangle problems*, tech. report, Citeseer, 2009.

[16] S. W. Bae and S. D. Yoon, *Empty squares in arbitrary orientation among points*, Algorithmica, (2022), pp. 1–46.

[17] S. Bandyapadhyay and A. Banik, *Polynomial time algorithms for bichromatic problems*, in Algorithms and Discrete Applied Mathematics: Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings 3, Springer, 2017, pp. 12–23.

[18] G. Barequet and S. Har-Peled, *Efficiently approximating the minimum-volume bounding box of a point set in three dimensions*, Journal of Algorithms, 38 (2001), pp. 91–109.

[19] H. Bast and S. Hert, *The area partitioning problem*, (2000).

[20] J. L. Bentley and M. I. Shamos, *Divide-and-conquer in multidimensional space*, in Proceedings of the eighth annual ACM symposium on Theory of computing, 1976, pp. 220–230.

[21] A. A. Berryman, *The orgins and evolution of predator-prey theory*, Ecology, 73 (1992), pp. 1530–1535.

[22] V. Bhateja, M. Misra, and S. Urooj, *Human visual system based unsharp masking for enhancement of mammographic images*, Journal of Computational Science, 21 (2017), pp. 387–393.

[23] B. Bhattacharya and A. Mukhopadhyay, *On the minimum perimeter triangle enclosing a convex polygon*, in Discrete and Computational Geometry: Japanese Conference, JCDCG 2002, Tokyo, Japan, December 6-9, 2002. Revised Papers, Springer, 2003, pp. 84–96.

[24] R. P. Boland and J. Urrutia, *Finding the largest axis aligned rectangle in a polygon in o (n log n) time.*, in CCCG, 2001, pp. 41–44.

[25] G. Borgefors and R. Strand, *An approximation of the maximal inscribed convex set of a digital object*, in Image Analysis and Processing–ICIAP 2005: 13th International Conference, Cagliari, Italy, September 6-8, 2005. Proceedings 13, Springer, 2005, pp. 438–445.

[26] M. Born, *Einstein's theory of relativity*, Courier Corporation, 1962.

[27] M. Boshra and H. Zhang, *Localizing a polyhedral object in a robot hand by integrating visual and tactile data*, Pattern Recognition, 33 (2000), pp. 483–501.

[28] J. E. Boyce, D. P. Dobkin, R. L. Drysdale III, and L. J. Guibas, *Finding extremal polygons*, in Proceedings of the fourteenth annual ACM symposium on Theory of computing, 1982, pp. 282–289.

[29] B. Braden, *The surveyor's area formula*, The College Mathematics Journal, 17 (1986), pp. 326–337.

[30] P. Brass, *Combinatorial geometry problems in pattern recognition*, Discrete and Computational Geometry, 28 (2002), pp. 495–510.

[31] M. Bruckheimer and A. Arcavi, *Farey series and pick's area theorem*, The mathematical intelligencer, 17 (1995), pp. 64–67.

[32] N. Buch, S. A. Velastin, and J. Orwell, *A review of computer vision techniques for the analysis of urban traffic*, IEEE Transactions on intelligent transportation systems, 12 (2011), pp. 920–939.

[33] S. Cabello, J. Cibulka, J. Kynčl, M. Saumell, and P. Valtr, *Peeling potatoes near-optimally in near-linear time*, in Proceedings of the thirtieth annual symposium on Computational geometry, 2014, pp. 224–231.

[34] P. B. Callahan, *Optimal parallel all-nearest-neighbors using the well-separated pair decomposition*, in Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science, IEEE, 1993, pp. 332–340.

[35] J. A. Carlson, A. Jaffe, and A. Wiles, *The millennium prize problems*, American Mathematical Soc., 2006.

[36] J.-S. Chang and C.-K. Yap, *A polynomial solution for potato-peeling and other polygon inclusion and enclosure problems*, in 25th Annual Symposium onFoundations of Computer Science, 1984., IEEE, 1984, pp. 408–416.

[37] J.-S. Chang and C.-K. Yap, *A polynomial solution for the potato-peeling problem*, Discrete & Computational Geometry, 1 (1986), pp. 155–182.

[38] L. Chapman and J. E. Thornes, *The use of geographical information systems in climatology and meteorology*, Progress in physical geography, 27 (2003), pp. 313–330.

[39] J. Chaudhuri, S. C. Nandy, and S. Das, *Largest empty rectangle among a point set*, Journal of algorithms, 46 (2003), pp. 54–78.

[40] B. Chazelle, *An optimal algorithm for intersecting three-dimensional convex polyhedra*, SIAM Journal on Computing, 21 (1992), pp. 671–696.

[41] B. Chazelle, R. Drysdale, and D. Lee, *Computing the largest empty rectangle*, SIAM Journal on Computing, 15 (1986), pp. 300–315.

[42] C. K. Cheung, W. Shi, and X. Zhou, *A probability-based uncertainty model for point-in-polygon analysis in gis*, GeoInformatica, 8 (2004), pp. 71–98.

[43] M.-T. Chien and H. Nakazato, *Circumscribed sphere of a convex polyhedron*, Applied mathematics letters, 18 (2005), pp. 1199–1203.

[44] Y. Choi, S. Lee, and H.-K. Ahn, *Maximum-area and maximum-perimeter rectangles in polygons*, Computational Geometry, 94 (2021), p. 101710.

[45] D. A. Chowdhry, A. Hussain, M. Z. U. Rehman, F. Ahmad, A. Ahmad, and M. Pervaiz, *Smart security system for sensitive area using face recognition*, in 2013 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (CSUDET), IEEE, 2013, pp. 11–14.

[46] J. Cohen and T. Hickey, *Two algorithms for determining volumes of convex polyhedra*, Journal of the ACM (JACM), 26 (1979), pp. 401–414.

[47] P. N. Cook, S. Batnitzky, K. R. Lee, E. Levine, H. I. Price, D. F. Preston, L. T. Cook, S. L. Fritz, W. Anderson, and S. J. Dwyer III, *Three-dimensional reconstruction from serial sections for medical applications*, in Three-Dimensional Machine Perception, vol. 283, SPIE, 1981, pp. 98–105.

[48] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms. third*, New York, (2009).

[49] G. Cramer, *Introduction à l'analyse des lignes courbes algébriques*, chez les frères Cramer et C. Philibert, 1750.

[50] K. Daniels, V. Milenkovic, and D. Roth, *Finding the largest area axis-parallel rectangle in a polygon*, Computational Geometry, 7 (1997), pp. 125–148.

[51] A. Day, *The implementation of an algorithm to find the convex hull of a set of three-dimensional points*, ACM Transactions on Graphics (TOG), 9 (1990), pp. 105–132.

[52] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational geometry algorithms and applications*, Spinger, 2008.

[53] L. De Floriani, P. Magillo, and E. Puppo, *Applications of computational geometry to geographic information systems.*, Handbook of computational geometry, 7 (2000), pp. 333–388.

[54] N. A. A. De Pano, *Polygon approximation with optimized polygonal enclosures: applications and algorithms*, Johns Hopkins University, 1987.

[55] R. Descartes, *La géométrie*, BoD-Books on Demand, 2022.

[56] D. P. Dobkin and L. Snyder, *On a general method for maximizing and minimizing among certain geometric problems*, in 20th Annual Symposium on Foundations of Computer Science (sfcs 1979), IEEE, 1979, pp. 9–17.

[57] C. Doukas and I. Maglogiannis, *Region of interest coding techniques for medical image compression*, IEEE Engineering in medicine and Biology Magazine, 26 (2007), pp. 29–35.

[58] H. Edelsbrunner, *A new approach to rectangle intersections part ii*, International Journal of Computer Mathematics, 13 (1983), pp. 221–229.

[59] A. Edwards, *Gh hardy (1908) and hardy–weinberg equilibrium*, Genetics, 179 (2008), pp. 1143–1150.

[60] A. W. Edwards, *Natural selection and the sex ratio: Fisher's sources*, The American Naturalist, 151 (1998), pp. 564–569.

[61] G. Farin, J. Hoschek, and M.-S. Kim, *Handbook of computer aided geometric design*, Elsevier, 2002.

[62] A. A. Farley, *Mathematical programming models for cutting-stock problems in the clothing industry*, Journal of the Operational Research Society, 39 (1988), pp. 41–53.

[63] P. Fischer and K.-U. Höffgen, *Computing a maximum axis-aligned rectangle in a convex polygon*, Information Processing Letters, 51 (1994), pp. 189–193.

[64] H. Freeman and R. Shapira, *Determining the minimum-area encasing rectangle for an arbitrary closed curve*, Communications of the ACM, 18 (1975), pp. 409–413.

[65] D. Friboulet, I. E. Magnin, and D. Revel, *Assessment of a model for overall left ventricular three-dimensional motion from mri data*, The International Journal of Cardiac Imaging, 8 (1992), pp. 175–190.

[66] N. Friedrichs, R. Vorreuther, C. Poremba, K.-L. Schafer, A. Böcking, R. Buettner, and H. Zhou, *Primitive neuroectodermal tumor (pnet) in the differential diagnosis of malignant kidney tumors*, Pathology-Research and Practice, 198 (2002), pp. 563–569.

[67] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik, *Recognizing objects in range data using regional point descriptors*, in Computer Vision-ECCV 2004: 8th European Conference on Computer Vision, Prague, Czech Republic, May 11-14, 2004. Proceedings, Part III 8, Springer, 2004, pp. 224–237.

[68] E. Galceran and M. Carreras, *A survey on coverage path planning for robotics*, Robotics and Autonomous systems, 61 (2013), pp. 1258–1276.

[69] J. Gauci, O. Falzon, C. Formosa, A. Gatt, C. Ellul, S. Mizzi, A. Mizzi, C. Sturgeon Delia, K. Cassar, N. Chockalingam, et al., *Automated region extraction from thermal images for peripheral vascular disease monitoring*, Journal of healthcare engineering, 2018 (2018).

[70] C. F. Gauss, *Disquisitiones generales circa superficies curvas*, vol. 1, Typis Dieterichianis, 1828.

[71] S. K. Ghosh and R. Shyamasundar, *A linear time algorithm for obtaining the convex hull of a simple polygon*, Pattern recognition, 16 (1983), pp. 587–592.

[72] C. Glymour, K. Ford, and P. Hayes, *The prehistory of android epistemology*, Artificial Intelligence: Critical Concepts, 1 (2000), p. 113.

[73] J. E. Goodman, *On the largest convex polygon contained in a non-convex n-gon, or how to peel a potato*, Geometriae Dedicata, 11 (1981), pp. 99–106.

[74] R. L. Graham, *An efficient algorithm for determining the convex hull of a finite planar set*, Info. Proc. Lett., 1 (1972), pp. 132–133.

[75] R. L. Graham and F. F. Yao, *Finding the convex hull of a simple polygon*, Journal of Algorithms, 4 (1983), pp. 324–331.

[76] C. Graumann, B. Fuerst, C. Hennersperger, F. Bork, and N. Navab, *Robotic ultrasound trajectory planning for volume of interest coverage*, in 2016 IEEE international conference on robotics and automation (ICRA), IEEE, 2016, pp. 736–741.

[77] T. C. Hales, *A proof of the kepler conjecture*, Annals of mathematics, (2005), pp. 1065–1185.

[78] O. Hall-Holt, M. J. Katz, P. Kumar, J. S. Mitchell, and A. Sityon, *Finding large sticks and potatoes in polygons*, in Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, Citeseer, 2006, pp. 474–483.

[79] J. J. Hanan, *Greenhouses: Advanced technology for protected horticulture*, CRC press, 2017.

[80] T. L. Heath et al., *The thirteen books of Euclid's Elements*, Courier Corporation, 1956.

[81] S. Hert and V. Lumelsky, *Polygon area decomposition for multiple-robot workspace division*, International Journal of Computational Geometry and Applications, 8 (1998), pp. 437–466.

[82] D. Hilbert, *Mathematical problems*, in Mathematics, Chapman and Hall/CRC, 2019, pp. 273–278.

[83] A. L. Hodgkin and A. F. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve*, The Journal of physiology, 117 (1952), p. 500.

[84] A. Inzartsev, A. Pavin, A. Kleschev, V. Gribova, and G. Eliseenko, *Application of artificial intelligence techniques for fault diagnostics of autonomous underwater vehicles*, in OCEANS 2016 MTS/IEEE Monterey, IEEE, 2016, pp. 1–6.

[85] M. Izadi and P. Saeedi, *Three-dimensional polygonal building model estimation from single satellite images*, IEEE Transactions on Geoscience and Remote Sensing, 50 (2011), pp. 2254–2272.

[86] R. A. Jarvis, *On the identification of the convex hull of a finite set of points in the plane*, Information processing letters, 2 (1973), pp. 18–21.

[87] K. Jin, *Maximal parallelograms in convex polygons*, arXiv preprint arXiv:1512.03897, 376 (2015).

[88] K. Jin, *Maximal area triangles in a convex polygon*, arXiv preprint arXiv:1707.04071, (2017).

[89] K. Jin and K. Matulef, *Finding the maximum area parallelogram in a convex polygon.*, in CCCG, Citeseer, 2011.

[90] Y. Kallus, *A linear-time algorithm for the maximum-area inscribed triangle in a convex polygon*, arXiv preprint arXiv:1706.03049, (2017).

[91] I. Kant, *Gedanken von der wahren Schätzung der lebendigen Kräfte und Beurtheilung der Beweise derer sich Herr von Leibnitz und anderer Mechaniker in dieser Streitsache bedienet haben, etc*, 1746.

[92] H. Kaplan and M. Sharir, *Finding the maximal empty rectangle containing a query point*, arXiv preprint arXiv:1106.3628, (2011).

[93] N. Karmakar and A. Biswas, *Construction of an approximate 3d orthogonal convex skull*, in Computational Topology in Image Context: 6th International Workshop, CTIC 2016, Marseille, France, June 15-17, 2016, Proceedings 6, Springer, 2016, pp. 180–192.

[94] V. Keikha, M. Löffler, A. Mohades, J. Urhausen, and I. van der Hoog, *Maximum-area quadrilateral in a convex polygon, revisited*, arXiv preprint arXiv:1708.00681, (2017).

[95] V. Keikha, M. Löffler, A. Mohades, J. Urhausen, and I. van der Hoog, *Maximum-area triangle in a convex polygon, revisited*, arXiv preprint arXiv:1705.11035, (2017).

[96] W. O. Kermack and A. G. McKendrick, *A contribution to the mathematical theory of epidemics*, Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character, 115 (1927), pp. 700–721.

[97] J. Kim, W. Cai, D. Feng, and H. Wu, *A new way for multidimensional medical data management: volume of interest (voi)-based retrieval of medical images with visual and functional features*, IEEE Transactions on Information Technology in Biomedicine, 10 (2006), pp. 598–607.

[98] D. G. Kirkpatrick and R. Seidel, *The ultimate planar convex hull algorithm?*, SIAM journal on computing, 15 (1986), pp. 287–299.

[99] V. Klee and M. C. Laskowski, *Finding the smallest triangles containing a given convex polygon*, Journal of Algorithms, 6 (1985), pp. 359–375.

[100] C. Knauer, L. Schlipf, J. M. Schmidt, and H. R. Tiwary, *Largest inscribed rectangles in convex polygons*, Journal of discrete algorithms, 13 (2012), pp. 78–85.

[101] K. Kołodziejczyk and J. Reay, *Polynomials and spatial pick-type theorems*, Expositiones Mathematicae, 26 (2008), pp. 41–53.

[102] D.-T. Lee, *On finding the convex hull of a simple polygon*, International journal of computer & information sciences, 12 (1983), pp. 87–98.

[103] D. T. Lee and F. P. Preparata, *Computational geometry - a survey*, IEEE Transactions on Computers, 33 (1984), pp. 1072–1101.

[104] S. Lee, T. Eom, and H.-K. Ahn, *Largest triangles in a polygon*, Computational Geometry, 98 (2021), p. 101792.

[105] Z. Li, J. D. Wegner, and A. Lucchi, *Topological map extraction from overhead images*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1715–1724.

[106] F. Lindemann, *Über die zahl π*, in Pi: A Source Book, 1882.

[107] Y. Liu and M. Nediak, *Planar case of the maximum box and related problems.*, in CCCG, vol. 3, 2003, pp. 11–13.

[108] R. Martin and P. Stephenson, *Putting objects into boxes*, Computer-aided design, 20 (1988), pp. 506–514.

[109] J. Maynard, *The twin prime conjecture*, Japanese Journal of Mathematics, 14 (2019), pp. 175–206.

[110] D. McCallum and D. Avis, *A linear algorithm for finding the convex hull of a simple polygon*, Information Processing Letters, 9 (1979), pp. 201–206.

[111] S. McCartney, *ENIAC: The triumphs and tragedies of the world's first computer*, Walker & Company, 1999.

[112] E. A. Melissaratos and D. L. Souvaine, *On solving geometric optimization problems using shortest paths*, in Proceedings of the sixth annual symposium on Computational geometry, 1990, pp. 350–359.

[113] E. A. Melissaratos and D. L. Souvaine, *Shortest paths help solve geometric optimization problems in planar regions*, SIAM Journal on Computing, 21 (1992), pp. 601–638.

[114] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, et al., *The multimodal brain tumor image segmentation benchmark (brats)*, IEEE transactions on medical imaging, 34 (2014), pp. 1993–2024.

[115] C. Micheloni, E. Salvador, F. Bigaran, and G. L. Foresti, *An integrated surveillance system for outdoor security*, in IEEE Conference on Advanced Video and Signal Based Surveillance, 2005., IEEE, 2005, pp. 480–485.

[116] Ministerio de Agricultura Pesca y Alimentación, *Visor sigpac v4.12*, `https://sigpac.mapa.gob.es/fega/visor`.

[117] J. S. Mitchell and V. Polishchuk, *Minimum-perimeter enclosures*, Information Processing Letters, 107 (2008), pp. 120–124.

[118] R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *An algorithm to compute any simple k-gon of a maximum area or perimeter inscribed in a region of interest*, SIAM Journal on Imaging Sciences, 15 (2022), pp. 1808–1832.

[119] R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *An efficient method for obtaining the maximum k-gon in an arbitrary polygon with obstacles*, Pattern Recognition (en revisión), (2023).

[120] R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *Obtaining the user-defined polygons inside a closed contour with holes*, The Visual Computer (aceptado), (2023).

[121] R. Molano, M. Ávila, J. C. Sancho, P. G. Rodríguez, and A. Caro, *Post-processing of closed contours to obtain inscribed k-sided polygons*, in International Conference on Computing, Intelligence and Data Analytics, 2023.

[122] R. Molano, D. Caballero, P. G. Rodríguez, M. D. M. Ávila, J. P. Torres, M. L. Durán, J. C. Sancho, and A. Caro, *Finding the largest volume parallelepipedon of arbitrary orientation in a solid*, IEEE Access, 9 (2021), pp. 103600–103609.

[123] R. Molano, P. G. Rodríguez, A. Caro, and M. L. Durán, *Finding the largest area rectangle of arbitrary orientation in a closed contour*, Applied Mathematics and Computation, 218 (2012), pp. 9866–9874.

[124] G. Moreda, M. Muñoz, M. Ruiz-Altisent, and A. Perdigones, *Shape determination of horticultural produce using two-dimensional computer vision–a review*, Journal of Food Engineering, 108 (2012), pp. 245–261.

[125] A. Mukhopadhyay and S. Rao, *Computing a largest empty arbitrary oriented rectangle: theory and implementation*, in Computational Science and Its Applications—ICCSA 2003: International Conference Montreal, Canada, May 18–21, 2003 Proceedings, Part III 3, Springer, 2003, pp. 797–806.

[126] N. Mutoh, *The polyhedra of maximal volume inscribed in the unit sphere and of minimal volume circumscribed about the unit sphere*, in Discrete and Computational Geometry: Japanese Conference, JCDCG 2002, Tokyo, Japan, December 6-9, 2002. Revised Papers, Springer, 2003, pp. 204–214.

[127] A. Naamad, D. Lee, and W.-L. Hsu, *On the maximum empty rectangle problem*, Discrete Applied Mathematics, 8 (1984), pp. 267–277.

[128] S. C. Nandy, A. Sinha, and B. B. Bhattacharya, *Location of the largest empty rectangle among arbitrary obstacles*, in International Conference on Foundations of Software Technology and Theoretical Computer Science, Springer, 1994, pp. 159–170.

[129] R. Naravaneni and K. Jamil, *Rapid detection of food-borne pathogens by using molecular techniques*, Journal of medical microbiology, 54 (2005), pp. 51–54.

[130] S. Olakanmi, C. Karunakaran, and D. Jayas, *Applications of x-ray micro-computed tomography and small-angle x-ray scattering techniques in food systems: A concise review*, Journal of Food Engineering, (2022), p. 111355.

[131] G. Oriolo, G. Ulivi, and M. Venditelli, *Real-time map building and navigation for autonomous robots in unknown environments*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 28 (1998), pp. 316–333.

[132] M. Orlowski, *A convex hull algorithm for planar simple polygons*, Pattern recognition, 18 (1985), pp. 361–366.

[133] M. Orlowski, *A new algorithm for the largest empty rectangle problem*, Algorithmica, 5 (1990), pp. 65–73.

[134] J. O'Rourke, *Finding minimal enclosing boxes*, International journal of computer & information sciences, 14 (1985), pp. 183–199.

[135] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin, *An optimal algorithm for finding minimal enclosing triangles*, Journal of Algorithms, 7 (1986), pp. 258–269.

[136] J. O'Rourke, C.-B. Chien, T. Olson, and D. Naddor, *A new linear algorithm for intersecting convex polygons*, Computer graphics and image processing, 19 (1982), pp. 384–391.

[137] D. O'sullivan and D. Unwin, *Geographic information analysis*, John Wiley & Sons, 2003.

[138] N. Panigrahi, *Computing in geographic information systems*, CRC Press, 2014.

[139] R. Penrose, *La nueva mente del emperador*, DEBOLS! LLO, 2015.

[140] T. Pérez-Palacios, T. Antequera, R. Molano, P. G. Rodríguez, and R. Palacios, *Sensory traits prediction in dry-cured hams from fresh product via mri and lipid composition*, Journal of food engineering, 101 (2010), pp. 152–157.

[141] J. F. Peters, *Foundations of computer vision: computational geometry, visual image structures and object shape detection*, vol. 124, Springer, 2017.

[142] G. Pick, *Geometrisches zur zahlenlehre*, Sitzenber. Lotos (Prague), 19 (1899), pp. 311–319.

[143] R. Platania, S. Shams, S. Yang, J. Zhang, K. Lee, and S.-J. Park, *Automated breast cancer diagnosis using deep learning and region of interest detection (bc-droid)*, in Proceedings of the 8th ACM international conference on bioinformatics, computational biology, and health informatics, 2017, pp. 536–543.

[144] F. P. Preparata and S. J. Hong, *Convex hulls of finite sets of points in two and three dimensions*, Communications of the ACM, 20 (1977), pp. 87–93.

[145] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction, 1985*, New York, (1985).

[146] I. A. Qasmieh, H. Alquran, and A. M. Alqudah, *Occluded iris classification and segmentation using self-customized artificial intelligence models and iterative randomized hough transform.*, International Journal of Electrical and Computer Engineering (2088-8708), 11 (2021).

[147] R. Qin and C. Duan, *The principle and applications of bernoulli equation*, in Journal of Physics: Conference Series, vol. 916, IOP Publishing, 2017, p. 012038.

[148] J. Radon, *1.1 über die bestimmung von funktionen durch ihre integralwerte längs gewisser mannigfaltigkeiten*, Classic papers in modern diagnostic radiology, 5 (2005), p. 124.

[149] N. S. Rao, S. S. Iyengar, B. J. Oommen, and R. L. Kashyap, *On terrain acquisition by a point robot amidst polyhedral obstacles*, IEEE Journal on Robotics and Automation, 4 (1988), pp. 450–455.

[150] M. Reed, *Methods of modern mathematical physics: Functional analysis*, Elsevier, 2012.

[151] J. Reeve, *A further note on the volume of lattice polyhedra*, Journal of the London Mathematical Society, 1 (1959), pp. 57–62.

[152] J. E. Reeve, *On the volume of lattice polyhedra*, Proceedings of the London Mathematical Society, 3 (1957), pp. 378–395.

[153] R. Rojas, *Konrad zuse's legacy: the architecture of the z1 and z3*, IEEE Annals of the History of Computing, 19 (1997), pp. 5–16.

[154] J. I. Rojas-Sola, G. del Río-Cidoncha, A. Fernández-de la Puente Sarriá, and V. Galiano-Delgado, *Blaise pascal's mechanical calculator: Geometric modelling and virtual reconstruction*, Machines, 9 (2021), p. 136.

[155] J. Rokne, S. Wang, and X. Wu, *The largest volume inscribed rooted tetrahedron in a convex polyhedron*, (1992).

[156] G. Rote, *The largest contained quadrilateral and the smallest enclosing parallelogram of a convex polygon*, arXiv preprint arXiv:1905.11203, (2019).

[157] F. Rottensteiner and C. Briese, *A new method for building extraction in urban areas from high-resolution lidar data*, in International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences, vol. 34, Citeseer, 2002, pp. 295–301.

[158] V. Sanchez, R. Abugharbieh, and P. Nasiopoulos, *3-d scalable medical image compression with optimized volume of interest coding*, IEEE Transactions on Medical Imaging, 29 (2010), pp. 1808–1820.

[159] D. Schmalstieg and T. Hollerer, *Augmented reality: principles and practice*, Addison-Wesley Professional, 2016.

[160] E. Schömer, J. Sellen, M. Teichmann, and C. Yap, *Smallest enclosing cylinders*, in Proceedings of the twelfth annual symposium on computational geometry, 1996, pp. 413–414.

[161] M. Schuster, *The largest empty circle problem*, in Proceedings of the Class of 2008 Senior Conference, Computer Science Department, Swarthmore College, Citeseer, 2008, pp. 28–37.

[162] J. T. Schwartz and C.-K. Yap, *Advances in robotics: Algorithmic and geometric aspects of robotics. volume 1*, (1986).

[163] C. Schwarz, J. Teich, A. Vainshtein, E. Welzl, and B. L. Evans, *Minimal enclosing parallelogram with application*, in Proceedings of the eleventh annual symposium on Computational geometry, 1995, pp. 434–435.

[164] C. Schwarz, J. Teich, E. Welzl, and B. Evans, *On finding a minimal enclosing parallelogram*, International Computer Science Institute, Berkeley, CA, Tech. Rep. tr-94-036, (1994).

[165] M. I. Shamos, *Computational geometry*, Yale University, 1978.

[166] M. I. Shamos and D. Hoey, *Geometric intersection problems*, in 17th Annual Symposium on Foundations of Computer Science (sfcs 1976), IEEE, 1976, pp. 208–215.

[167] C. E. Shannon, *A mathematical theory of communication*, The Bell system technical journal, 27 (1948), pp. 379–423.

[168] C. E. Shannon, *Communication in the presence of noise*, Proceedings of the IRE, 37 (1949), pp. 10–21.

[169] F. Sheikhi and S. Alipour, *On triangluar separation of bichromatic point sets in polygonal environment*, arXiv preprint arXiv:1809.00116, (2018).

[170] F. Sheikhi and S. Alipour, *On triangluar separation of bichromatic point sets in polygonal environment*, arXiv preprint arXiv:1809.00116, (2018).

[171] P. F. Sheron, K. Sridhar, S. Baskar, and P. M. Shakeel, *Projection-dependent input processing for 3d object recognition in human robot interaction systems*, Image and Vision Computing, 106 (2021), p. 104089.

[172] S. Y. Shin and T. C. Woo, *Finding the convex hull of a simple polygon in linear time*, Pattern recognition, 19 (1986), pp. 453–458.

[173] E. Sides, *Geological modelling of mineral deposits for prediction in mining*, Geologische Rundschau, 86 (1997), pp. 342–353.

[174] G. Springer, P. Hannah, R. J. Stonier, S. Smith, and P. Wolfs, *Simple strategies for collision-avoidance in robot soccer*, Robotics and autonomous systems, 21 (1997), pp. 191–196.

[175] I. Stewart, *Galois theory*, Chapman and Hall/CRC, 1990.

[176] C. Sun, W. Zhan, J. She, and Y. Zhang, *Object detection from the video taken by drone via convolutional neural networks*, Mathematical Problems in Engineering, 2020 (2020), pp. 1–10.

[177] D.-W. Sun, *Computer vision technology for food quality evaluation*, Academic Press, 2016.

[178] X. Sun, W. Zhao, R. V. Maretto, and C. Persello, *Building polygon extraction from aerial images and digital surface models with a frame field learning framework*, Remote Sensing, 13 (2021), p. 4700.

[179] J. Tabak, *Geometry: The Language of Space and Form (The History of Mathematics)*, Facts On File, Inc., 2004.

[180] G. T. Toussaint, *Computational geometric problems in pattern recognition*, in Pattern Recognition Theory and Applications: Proceedings of the NATO Advanced Study Institute held at St. Anne's College, Oxford, March 29–April 10, 1981, Springer, 1982, pp. 73–91.

[181] G. T. Toussaint, *Solving geometric problems with the rotating calipers*, in Proc. IEEE Melecon, vol. 83, 1983, p. A10.

[182] G. T. Toussaint, *Computational geometry and computer vision*, Contemporary Mathematics, 119 (1991), pp. 213–224.

[183] J. Traub, T. Sielhorst, S.-M. Heining, and N. Navab, *Advanced display and visualization concepts for image guided surgery*, Journal of Display Technology, 4 (2008), pp. 483–490.

[184] A. Tsalatsanis, K. Valavanis, and A. Yalcin, *Vision based target tracking and collision avoidance for mobile robots*, Journal of Intelligent and Robotic Systems, 48 (2007), pp. 285–304.

[185] A. M. Turing et al., *On computable numbers, with an application to the entscheidungsproblem*, J. of Math, 58 (1936), p. 5.

[186] J. Vahrenhold, *Line-segment intersection made in-place*, Computational Geometry, 38 (2007), pp. 213–230.

[187] I. van der Hoog, V. Keikha, M. Löffler, A. Mohades, and J. Urhausen, *Maximum-area triangle in a convex polygon, revisited*, Information Processing Letters, 161 (2020), p. 105943.

[188] D. E. Varberg, *Pick's theorem revisited*, The American Mathematical Monthly, 92 (1985), pp. 584–587.

[189] F. Vivien and N. Wicker, *Minimal enclosing parallelepiped in 3d*, Computational Geometry, 29 (2004), pp. 177–190.

[190] J. Von Neumann, *Mathematical foundations of quantum mechanics: New edition*, vol. 53, Princeton university press, 2018.

[191] Y. Wang, *The Goldbach Conjecture*, vol. 4, World scientific, 2002.

[192] P. L. Wantzel, *Recherches sur les moyens de reconnaître si un problème de géométrie peut se résoudre avec la règle et le compas*, éditeur inconnu, 1837.

[193] Q. Wei, J. Sun, X. Tan, X. Yao, and Y. Ren, *The simple grid polygon exploration problem*, Journal of Combinatorial Optimization, 41 (2021), pp. 625–639.

[194] E. Welzl, *Smallest enclosing disks (balls and ellipsoids)*, in New Results and New Trends in Computer Science: Graz, Austria, June 20–21, 1991 Proceedings, Springer, 2005, pp. 359–370.

[195] J. Wiemels, M. Wrensch, and E. B. Claus, *Epidemiology and etiology of meningioma*, Journal of neuro-oncology, 99 (2010), pp. 307–314.

[196] A. Wiles, *Modular elliptic curves and fermat's last theorem*, Annals of mathematics, 141 (1995), pp. 443–551.

[197] M. R. Williams, *The origins, uses, and fate of the edvac*, IEEE Annals of the History of Computing, 15 (1993), pp. 22–38.

[198] O. Wirjadi, *Survey of 3d image segmentation methods*, (2007).

[199] P. J. Withers, C. Bouman, S. Carmignato, V. Cnudde, D. Grimaldi, C. K. Hagen, E. Maire, M. Manley, A. Du Plessis, and S. R. Stock, *X-ray computed tomography*, Nature Reviews Methods Primers, 1 (2021), p. 18.

[200] W.-J. Yan and Y.-H. Chen, *Measuring dynamic micro-expressions via feature extraction methods*, Journal of Computational Science, 25 (2018), pp. 318–326.

[201] I. E. Zevgolis, A. V. Deliveris, and N. C. Koukouzas, *Probabilistic design optimization and simplified geotechnical risk analysis for large open pit excavations*, Computers and Geotechnics, 103 (2018), pp. 153–164.

[202] J. Zhang and S. You, *Speeding up large-scale point-in-polygon test based spatial join on gpus*, in Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, 2012, pp. 23–32.

[203] W. Zhao, C. Persello, and A. Stein, *Building outline delineation: From aerial images to polygons with an improved end-to-end learning framework*, ISPRS Journal of Photogrammetry and Remote Sensing, 175 (2021), pp. 119–131.

[204] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, *Object detection in 20 years: A survey*, Proceedings of the IEEE, (2023).

[205] D. Zwillinger, *CRC standard mathematical tables and formulas*, CRC press, 2018.