Sección de Matemáticas
Universidad de La Laguna

Ainoa González García

Optimización del área de un polígono dados sus vértices

Optimizing the area of a polygon given its vertices

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Marzo de 2023

Dirigido por
Hipólito Hernández Pérez

Hipólito Hernández Pérez
Matemáticas, Estadística e
Investigación Operativa
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

En primer lugar, quiero darle las gracias a Hipólito por haberme guiado a lo largo de todo este trabajo. Gracias por su paciencia, empeño y esfuerzo. Sin él, no hubiera sido posible.

Gracias a mi familia, en especial a mis padres y a mi hermano, por haber estado apoyándome durante todos estos años. Siempre dándome ánimos para seguir y nunca abandonar.

Gracias a mis amigos de siempre por haberme acompañado en otra etapa más de mi vida.

Y por último, gracias a esos compañeros de clase que, con el paso de los años, se han convertido en amigos.

Ainoa González García
La Laguna, 9 de marzo de 2023

Resumen · Abstract

Resumen

En este trabajo describimos dos formulaciones alternativas para resolver dos problemas diferentes que comparten las mismas características. Dado un conjunto S de n puntos en el plano, queremos encontrar un polígono simple cuyo conjunto de vértices sean exactamente los n puntos del conjunto S y cuya área sea máxima o mínima. Veremos como ambos problemas están estrechamente relacionados con el Problema del Viajante de Comercio. Realizamos varios experimentos para ver con qué modelo (y variante de éste) se obtienen los mejores resultados. Por último, se comparan los resultados obtenidos con los ya existentes en la literatura.

Palabras clave: *Polígono – Optimización – Problema del Viajante de Comercio – Geometría Computacional.*

Abstract

In this work we describe two alternative formulations to solve two different problems that share the same characteristics. Given a set S of n points in the plane, we want to find a simple polygon whose set of vertices are exactly the n points of the set S and whose area is maximum or minimum. We will see how both problems are closely related to the Traveling Salesman Problem. We ran several experiments to see which model (and its variant) gives the best results. Finally, the results obtained are compared with those already existing in the literature.

Keywords: *Polygon – Optimization – Traveling Salesman Problem – Computational Geometry.*

Contenido

Agradecimientos iii

Resumen/Abstract v

Introducción ix

1. Herramientas matemáticas 1
- 1.1. Definiciones y notaciones 1
- 1.2. Software 2

2. Problemas de polígonos 5
- 2.1. Problemas de optimización de polígonos 5
  - 2.1.1. El Problema del Viajante de Comercio 6
- 2.2. Modelos matemáticos 7
  - 2.2.1. Modelo 1 8
  - 2.2.2. Modelo 2 9
  - 2.2.3. Modelo 3 11

3. Resultados computacionales 13
- 3.1. Resultados preliminares 14
- 3.2. Resultados finales 17

A. Apéndice 23
- A.1. Implementación de los modelos 23
- A.2. Otros códigos 30

Bibliografía 31

Lista de símbolos y abreviaciones 33

Poster 35

Introducción

Uno de los problemas que trata la Geometría Computacional es la construcción de objetos geométricos dado un conjunto de puntos en el plano. Entre estos objetos se encuentran las triangulaciones de conjuntos de puntos, los diagramas de Voronoi, los polígonos construidos a partir de un conjunto de puntos, etc.

Este trabajo trata sobre la optimización de polígonos simples (i.e., polígonos sin agujeros ni aristas que se crucen) dado un conjunto de puntos que queremos que coincidan con los vértices de dicho polígono. Lo haremos optimizando el área, siendo ésta máxima o mínima, según convenga. Este tipo de problemas siempre han sido de interés en la Geometría Computacional pero han tenido especial relevancia tras el concurso *Computational Geometry: Solving Hard Optimization Problems*, celebrado en el año 2019, y que instaba a los participantes a resolver este tipo de problemas.

En el capítulo 1 daremos las definiciones y notaciones necesarias para poder comprender el resto de este escrito. Además, hablaremos del *software* que se ha utilizado para llevar a cabo esta serie de experimentos.

En el capítulo 2 comenzamos hablando, de manera breve, sobre una serie de problemas que son de especial interés en el ámbito de la Geometría Computacional y nos expenderemos un poco más comentando el más famoso de ellos. Posteriormente, para otros dos problemas, formularemos dos modelos diferentes para su resolución.

En la primera parte del capítulo 3 veremos una serie de resultados preliminares que se obtuvieron después de realizar varios experimentos añadiendo y quitando restricciones de refuerzo a cada uno de los dos modelos planteados generando así distintas variantes del mismo. Y en la segunda parte, se seleccionó la mejor combinación de restricciones que fortalecían cada uno de los modelos y

se compararon los resultados que obtuvo Ramos et al. en [3].

Finalizaremos este trabajo con las conclusiones y terminaremos esta memoria con un apéndice que contiene el código que se ha elaborado para la resolución de ambos modelos.

1

Herramientas matemáticas

El objetivo de este primer capítulo es proporcionar las herramientas teóricas necesarias para la comprensión del resto de este escrito. Para ello, introducimos algunas definiciones y notaciones. Además, también hablaremos del *software* que hemos utilizado.

1.1. Definiciones y notaciones

A continuación vamos definir algunos conceptos y sus respectivas notaciones.

Trabajaremos en $\mathbb{R}^2$ con la distancia euclídea.

Un *polígono* P está definido por un conjunto finito de segmentos tales que los extremos de los segmentos están conectados por exactamente otros dos segmentos y ningún subconjunto de ellos tiene la misma propiedad. Cada segmento del polígono se denomina *arista* y cada extremo de cada segmento se denomina *vértice*.

Un polígono P es *simple* si no hay un par de aristas no consecutivas que compartan un vértice. Como consecuencia, no hay dos aristas que se superpongan. Un polígono simple divide el plano en dos regiones disjuntas, el *interior* (acotada) y el *exterior* (no acotada) que están separadas por el polígono. Definimos la *frontera* de un polígono simple P como las aristas mencionadas anteriormente. En el resto de este escrito consideramos que P es un polígono simple formado por la unión de su frontera y su interior. Además, denotaremos por $a(P)$ al área del polígono P (considerando su interior).

Un polígono simple P es *convexo* si para todo p y q dos vértices de P, el segmento $\overline{pq}$ está contenido en P. A partir de aquí denotaremos los segmentos $\overline{pq}$ de la siguiente forma (p, q). Dado un conjunto de puntos S contenidos en el

plano, la *envolvente convexa* de S, denotada como $\mathcal{CH}(S)$, es el polígono convexo de menor área que contiene a S. Por convenio, decidimos que la envolvente convexa de un polígono P viene dada en sentido antihorario.
Para más conceptos geométricos remitimos al lector a Preparata y Shamos (ver [1]).

La *diagonal* de un polígono simple P es un segmento que conecta dos vértices de P y se encuentra en el interior de P. Una triangulación de P es una partición de P en triángulos por un conjunto máximo de diagonales. Se conoce que cada triangulación de un polígono con n vértices tiene $n-2$ triángulos [2].

Denotamos por S a un conjunto de puntos en el plano y por E_S al conjunto de aristas cuyos vértices están en S. Así, definimos el grafo no dirigido $G = (S, E_S)$. Además, $E_{\mathcal{CH}(S)}$ es el conjunto de aristas cuyos vértices están en la envolvente convexa del conjunto S.

Sea T un conjunto de triángulos. Diremos que dos triángulos t_1,$t_2 \in T$ se *cruzan* cuando una arista de t_1 tiene intersección con una de las aristas de t_2.

Por último, denotamos por $\Delta(S)$, y lo definimos como $\Delta(S) = \{t = (i, j, k) : i, j, k \in S, i \neq j \neq k\}$ al conjunto de triángulos cuyos vértices están en S.

1.2. Software

En primer lugar, vamos a explicar el *software* que se ha utilizado en la literatura con la que compararemos los resultados que hemos obtenido y a continuación explicaremos el que hemos utilizado para llevar a cabo los experimentos.

En lo que respecta al software usado por Ramos et al. en [3], los algoritmos fueron implementados en C++ y compilados con GCC 7.5.0. Además, todos los modelos de programación lineal entera que propusieron fueron resueltos con CPLEX, versión 12.9, mientras que para los cálculos geométricos se utilizó la librería CGAL [5], versión 4.13.

Por otra parte, el algoritmo que hemos creado fue implementado en la versión 3.8 de Python mientras que se han resuelto con dos solucionadores distintos, CBC y GUROBI, este último con la versión 9.5.1. Además, hemos utilizado la versión 5.3.1 de la librería CGAL.

- C++ es un lenguaje de programación desarrollado en los años 80 por Bjarne Stroustrup con la intención de extender al lenguaje de programación C los mecanismos necesarios para la manipulación de objetos.
- *The CGAL* (*The Computational Geometry Algorithms Library*) *Project* [6] es un biblioteca que da acceso a algoritmos geométricos eficientes mediante una librería en C++. CGAL se utiliza en áreas que requieren computación geométrica, como pueden ser sistemas de información geográfica, diseño asistido por computadora, biología molecular, imágenes médicas, gráficos por ordenador y robótica. Esta biblioteca ofrece estructuras de datos y algoritmos como triangulaciones, operaciones booleanas en polígonos y poliedros, procesamiento de conjuntos de puntos, algoritmos para la envolvente convexa, etc. The CGAL Project está en continuo desarrollo mediante una comunidad de desarrolladores en la que trabajan institutos de investigación, universidades y empresas. En dicha comunidad se incluye la CGAL Editorial Board, que es responsable de guiar el desarrollo de la librería, a los desarrolladores y a la comunidad de usuarios.
- CPLEX es el nombre con el que se conoce a IBM ILOG CPLEX Optimization Studio [7], un paquete de optimización matemática para problemas de programación lineal, programación de enteros combinada y programación cuadrática. Fue desarrollado por Robert Bixby y vendido comercialmente desde 1988 por CPLEX Optimization Inc. Éste fue adquirido por ILOG en 1997 y posteriormente ILOG fue adquirido por IBM en enero de 2009. En la actualidad, IBM continúa desarrollando CPLEX. Este software tiene una capa de modelado conocida como Concert Technology [8], la cual proporciona interfaces para los lenguajes de programación C++, C# y Java. Además, existe una interfaz para lenguaje Python basada en la interfaz de C.
- Python [9] es uno de los lenguajes de programación más utilizados ya que se caracteriza por la legibilidad de su código. Fue creado a principio de los años noventa por Guido V. Rossum aunque otras muchas personas han contribuido en su desarrollo. La implementación tradicional de Python se denomina CPython pero también existen otras como, por ejemplo, Jython (es Python ejecutándose en una máquina virtual de Java). También existen reempaquetados de CPython que por lo general incluyen más librerías o están especializados en una aplicación en particular como, por ejemplo, WinPython (es una distribución científica de Python para Windows).
- GUROBI [14] es un *software* especializado en optimización matemática que fue desarrollado por Gurobi Optimization, LLC. Dicha empresa fue fundada por Zonghao Gu, Edward Rothberg y Robert Bixby en el año 2008 y su nombre se compone por las tres primeras sílabas de los apellidos de sus fundadores. Es capaz de resolver problemas de programación lineal, programación cuadrática, programación con restricciones cuadráticas, programación lineal entera mixta, programación cuadrática con enteros mixtos y programación

entera mixta con restricciones cuadráticas. Existen interfaces orientadas a objetos para C++, Java, Microsoft .NET y Python, y una interfaz orientada a matrices para R, MATLAB, C y Julia.

- CBC (*COIN-OR Branch and Cut*) [13] es un *software* de código abierto destinado a la resolución de problemas de programación lineal y programación entera mixta escrito en C++. Podemos usar este solucionador en Python a través del *software* (también de código abierto) OR-Tools.

2

Problemas de polígonos

En este capítulo hablaremos de diferentes problemas relacionados con la optimización de polígonos según cual sea nuestro objetivo y qué queremos optimizar. Entre ellos, se encuentran los dos problemas sobre los que se basa este trabajo y uno de los problemas más conocidos de la Optimización Combinatoria: el Problema del Viajante de Comercio (*Traveling Salesman Problem* - *TSP*).

2.1. Problemas de optimización de polígonos

Dos de las estructuras fundamentales de la Geometría Computacional son los conjuntos de puntos en el plano y los polígonos. Debido a esto, es frecuente que se planteen distintos problemas para encontrar polígonos que utilicen los n puntos de un conjunto finito dado, S, como vértices y se esté interesado en minimizar o maximizar el área de dicho polígono o la longitud de su frontera. Además, también se hace una distinción entre polígonos en general (los cuales pueden tener huecos en su interior) y polígonos simples. Es así como obtenemos una familia de ocho problemas [12] que optimizan polígonos cuyos vértices están en un conjunto de puntos. En la Tabla 2.1 tenemos una descripción simple de los ocho problemas relacionados con la optimización de polígonos.

	Polígono general		Polígono simple	
	Área	Frontera	Área	Frontera
Minimizar	MinArea	MinFront	SMinArea	SMinFront
Maximizar	MaxArea	MaxFront	SMaxArea	SMaxFront

Tabla 2.1. Descripción general de los ocho problemas.

El miembro más conocido de esta familia es, sin duda alguna, el SMinFront, es decir, el TSP. Sin embargo, en este trabajo vamos a estudiar el SMinArea y el SMaxArea. Crearemos dos formulaciones diferentes con las

que podremos maximizar y/o minimizar el área de un polígono simple y compararemos los resultados que hemos obtenido con los que ha obtenido Ramos et al. en [3].
En la figura 2.1 podemos observar un ejemplo del SMaxArea cuando tenemos diez puntos en el plano mientras que en la figura 2.2 tenemos un ejemplo para el problema SMinArea con el mismo conjunto de puntos.

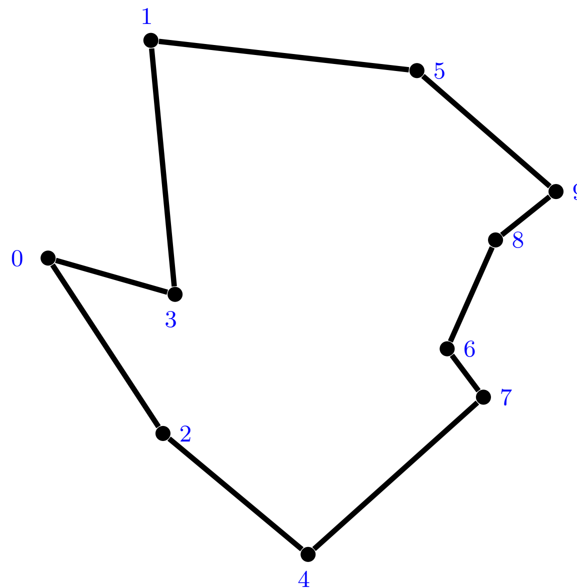

Figura 2.1. Ejemplo de un polígono simple de área máxima con 10 puntos.

2.1.1. El Problema del Viajante de Comercio

También conocido como el Problema del Vendedor Viajero o el Problema del Vendedor Ambulante, entre otras nomenclaturas, el Problema del Viajante de Comercio es uno de los más conocidos dentro de la Optimización Combinatoria. En dicho problema suponemos que un viajante tiene que visitar un número finito de ciudades, partiendo de una ciudad en particular y regresando a la misma ciudad de partida, de tal forma que la distancia recorrida sea la menor posible y pasando por cada ciudad una única vez a excepción de la ciudad de partida/llegada.

A pesar de que los orígenes de este problema no están del todo claros, es en el año 1857 cuando el matemático irlandés William R. Hamilton y el matemático británico Thomas Kirkman crearon el juego icosiano (*icosian game*) [11], el cual consiste en encontrar un camino hamiltoniano por las aristas de un dodecaedro pasando una única vez por cada vértice y que el vértice de partida coincida con

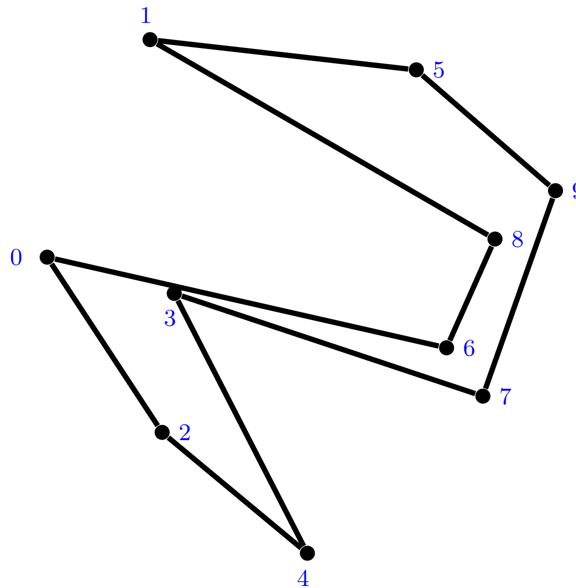

Figura 2.2. Ejemplo de un polígono simple de área mínima con 10 puntos.

el de llegada. No obstante, es en la década de 1930 cuando el matemático Karl Menger estudia este problema de manera formal por primera vez. Entre los años 1950 y 1960, George Dantzig, Delbert R. Fulkerson y Selmer M. Johnson [15] realizaron la primera formulación matemática, expresándolo como un problema de programación lineal de enteros y resolviéndolo mediante el método de ramificación y acotación.

A pesar de que, computacionalmente, es un problema complejo, se conocen gran cantidad de heurísticas y métodos exactos por lo que es posible resolver planteamientos concretos desde cien hasta miles de ciudades.

Tiene aplicaciones en la planificación de tareas, la logística y la fabricación de circuitos electrónicos, entre otros. También aparece, con algunas modificaciones, en la secuenciación del ADN. Además, según los diferentes ámbitos de aplicación del problema, éste puede tener restricciones adiciones como pueden ser las ventanas de tiempo (*TSP with Time Windows*) por lo que resulta ser aún más complejo.

2.2. Modelos matemáticos

En esta sección se reflejan tres formulaciones diferentes para los problemas SMinArea y SMaxArea. La primera de ellas fue enunciada por Ramos et al.

[3] y es con ésta con la que vamos a comparar los resultados que hemos obtenido, mientras que las dos restantes se han propuesto para la elaboración de este trabajo.

A continuación, describimos las variables de decisión con las que trabajaremos en estos tres modelos.

- La variable y_t, para cada $t \in \Delta(S)$, es igual a uno si el triángulo t está en la triangulación solución.
- La variable w_{ts} es igual a uno si t y s son dos triángulos adyacentes en la triangulación solución.
- La variable z_{ij}, para cada $i, j \in S$, es igual a uno si el segmento (i, j) es una diagonal de la triangulación solución.
- La variable x_{ij}, para cada $i, j \in S$, es igual a uno si el segmento (i, j) está en el perímetro del polígono.
- Por último, la variable u_i, para cada $i \in S \setminus \{0\}$, indica la posición menos uno del vértice i en la ruta.

En el resto de este escrito, consideraremos que las variables x_{ij} son variables orientadas.

Adicionalmente a la notación de $\Delta(S)$ como conjunto de triángulos que se pueden formar con los puntos de S, definimos los siguientes conjuntos. Para $i \in S$ denotamos por $\Delta(S, i)$ al conjunto de triángulos de $\Delta(S)$ que incluyen a i como uno de sus vértices. Para $t \in \Delta(S)$ denotamos por $\Delta(S, t)$ al conjunto de triángulos de $\Delta(S)$ adyacentes a t. Y, para $i, j \in S$ denotamos por $\Delta(S, i, j)$ al conjunto de triángulos de $\Delta(S)$ que tienen a i y j como vértices.

2.2.1. Modelo 1

El modelo que fue propuesto por Ramos et al. en [3] está basado en la correspondencia que existe entre una triangulación de un polígono y un árbol del grafo dual definido de la siguiente forma. Consideremos $G = (\Delta(S), E)$ cuyo conjunto de nodos $\Delta(S)$ se corresponde con cada posible triángulo de una triangulación de un polígono P, y cada elemento de E se corresponde con dos triángulos adyacentes.

La formulación de dicho modelo es la siguiente:

$$\text{mín} \,/\, \text{máx} \sum_{t \in \Delta(S)} a_t y_t \tag{2.1}$$

sujeto a:

$$\sum_{t \in \Delta(S)} y_t = n - 2 \tag{2.2}$$

$$\sum_{(t,s) \in E} w_{ts} = n - 3 \tag{2.3}$$

$$y_t + y_s \leq 1 \qquad \forall t, s \in \Delta(S) \text{ triángulos incompatibles} \tag{2.4}$$

$$\sum_{t \in \Delta(S,i)} y_t \geq 1 \qquad \forall i \in S \tag{2.5}$$

$$\sum_{s \in \Delta(S,t)} w_{ts} \leq \alpha y_t \qquad \forall t \in \Delta(S) \tag{2.6}$$

$$\sum_{t,s \in U} w_{ts} \leq |U| - 1 \qquad \forall U \subset \Delta(S) \tag{2.7}$$

$$y_t \in \{0, 1\} \qquad \forall t \in \Delta(S) \tag{2.8}$$

$$w_{ts} \in \{0, 1\} \qquad \forall t, s \in \Delta(S), t < s. \tag{2.9}$$

El número de vértices y aristas en una solución se imponen en las ecuaciones (2.2) y (2.3), respectivamente. No van a haber triángulos que se cruzan debido a la restricción (2.4). Que cada punto $i \in S$ tenga, al menos, un triángulo adyacente se garantiza en la restricción (2.5). La relación entre los vértices y las aristas incidentes en ellos viene dada por la restricción (2.6), donde $\alpha = min\{d^1, 3\}$, ya que cada vértice tiene grado, a lo sumo, tres. La restricción (2.7) garantiza que no existan subciclos en la solución. Por último, la integrabilidad de las variables se garantiza por las restricciones (2.8) y (2.9).

Se aplicaron diversas heurísticas para disminuir el tiempo de resolución pero la que mejor resultados obtuvo fue la que sustituyó la restricción (2.3) por la restricción (2.10), donde (2.10) es una combinación lineal de las restricciones (2.2) y (2.3).

$$\sum_{e \in E(\Delta(S,i))} w_e = \sum_{t \in \Delta(S,i)} y_t - 1 \quad \forall i \in S. \tag{2.10}$$

2.2.2. Modelo 2

La idea de este segundo modelo es que, dada una triangulación de un polígono simple P, hay exactamente un triángulo adyacente a cada lado del polígono y hay exactamente dos triángulos adyacentes a cada diagonal. Así surge esta formulación alternativa para resolver los problemas SMaxArea y SMinArea:

[1] El conjunto de vértices adyacentes a un vértice dado $i \in S$ lo denotamos por $N(i)$, y el grado de i es $d =| N(i) |$.

$$\text{mín} / \text{máx} \sum_{t \in \Delta(S)} a_t y_t \tag{2.11}$$

sujeto a:

$$\sum_{t \in \Delta(S)} y_t = n - 2 \tag{2.12}$$

$$y_t + y_s \leq 1 \qquad \forall t, s \in \Delta(S) \text{ triángulos incompatibles} \tag{2.13}$$

$$\sum_{t \in \Delta(S,i,j)} y_t = 2z_{ij} + x_{ij} + x_{ji} \qquad \forall i, j \in S \tag{2.14}$$

$$x_{ij} + x_{ji} + z_{ij} \leq 1 \qquad \forall i, j \in S \tag{2.15}$$

$$\sum_{j \in S, j \neq i} x_{ij} = \sum_{j \in S, j \neq i} x_{ji} = 1 \qquad \forall i \in S \tag{2.16}$$

$$u_j \geq u_i + x_{ij} - (n-2)(1 - x_{ij}) \qquad \forall i, j \in S, i, j \neq 0, i \neq j \tag{2.17}$$

$$u_j \geq u_i + 1 \qquad \text{si } (i,j) \text{ es un segmento de } \mathcal{CH}(S) \tag{2.18}$$

$$0 \leq u_i \leq n - 2 \qquad \forall i \in S \tag{2.19}$$

$$y_t \in \{0, 1\} \qquad \forall t \in \Delta(S) \tag{2.20}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in E_S \tag{2.21}$$

$$z_{ij} \in \{0, 1\} \qquad \forall (i,j) \in E_S \tag{2.22}$$

La restricción (2.12) dice que una triangulación contiene exactamente $n-2$ triángulos. Mediante las restricciones (2.13) nos aseguramos de que dos triángulos incompatibles no pueden estar en la misma solución. Las restricciones (2.14) y (2.15) son las que fuerzan a que un lado del polígono tiene exactamente un triángulo adyacente y que una diagonal de la triangulación del polígono tiene exactamente dos triángulos adyacentes. Las restricciones (2.16) y (2.17) son como las restricciones de grado y conectividad del TSP (ya que un polígono debe ser una solución del TSP). Un polígono simple debe respetar el orden en el que se recorren los vértices de la envolvente convexa, así podemos añadir las restricciones (2.18). Finalmente, el resto de restricciones corresponde a restricciones sobre las cotas e integrabilidad de las variables.

Una observación importante, si buscamos que la solución sea un polígono simple, es que no puede haber una arista que conecte dos vértices no consecutivos de la envolvente convexa, por lo tanto si $i, j \in \mathcal{CH}(S)$ e i y j son dos vértices no consecutivos en $\mathcal{CH}(S)$, entonces podemos fijar la variable x_{ij} a 0. Adicionalmente, una arista de la envolvente convexa no puede ser una diagonal, por lo que si $i, j \in \mathcal{CH}(S)$ e i y j son vértices consecutivos de $\mathcal{CH}(S)$, entonces la variable z_{ij} puede fijarse a 0.

La restricción (2.15) puede reforzarse mediante:

$$x_{ij} + x_{ji} + z_{ij} + x_{kl} + x_{lk} + z_{kl} \leq 1 \quad \text{si } (i,j),(k,l) \text{ son segmentos que se cruzan} \tag{2.23}$$

La restricción (2.17) puede reforzarse mediante:

$$u_j \geq u_i + x_{ij} - (n-2)(1-x_{ij}) + (n-3)x_{ji} \quad \forall i,j \in S, i,j \neq 0, i \neq j \tag{2.24}$$

La restricción (2.18) puede reforzarse mediante:

$$u_j \geq u_i + 2 - x_{ij} \qquad \text{si } (i,j) \text{ es un segmento de } \mathcal{CH}(S) \tag{2.25}$$

2.2.3. Modelo 3

La idea de este tercer modelo es tener en cuenta el área que se forma entre la envolvente convexa y el polígono. De esta forma vamos a triangular esta superficie mediante las variables y_t.

$$\text{mín / máx} \qquad a(\mathcal{CH}(S)) - \sum_{t \in \Delta(S)} a_t y_t \tag{2.26}$$

sujeto a:

$$\sum_{t \in \Delta(S)} y_t = n - |\mathcal{CH}(S)| \tag{2.27}$$

$$y_t + y_s \leq 1 \qquad \forall t, s \in \Delta(S) \text{ triángulos incompatibles} \tag{2.28}$$

$$\sum_{t \in \Delta(S,i,j)} y_t = 1 - x_{ij} - x_{ji} \qquad \forall (i,j) \in E_{\mathcal{CH}(S)} \tag{2.29}$$

$$\sum_{t \in \Delta(S,i,j)} y_t = 2z_{ij} + x_{ij} + x_{ji} \qquad \forall (i,j) \in E_S \setminus E_{\mathcal{CH}(S)} \tag{2.30}$$

$$x_{ij} + x_{ji} + z_{ij} \leq 1 \qquad \forall i,j \in S \tag{2.31}$$

$$\sum_{j \in S, j \neq i} x_{ij} = \sum_{j \in S, j \neq i} x_{ji} = 1 \qquad \forall i \in S \tag{2.32}$$

$$u_j \geq u_i + x_{ij} - (n-2)(1-x_{ij}) \qquad \forall i,j \in S, i,j \neq 0, i \neq j \tag{2.33}$$

$$u_j \geq u_i + 1 \qquad \text{si } (i,j) \text{ es un segmento de } \mathcal{CH}(S) \tag{2.34}$$

$$0 \leq u_i \leq n-2 \qquad \forall i \in S \tag{2.35}$$

$$y_t \in \{0,1\} \qquad \forall t \in \Delta(S) \tag{2.36}$$

$$x_e \in \{0,1\} \qquad \forall e \in E(S) \tag{2.37}$$

$$z_e \in \{0,1\} \qquad \forall e \in E(S) \tag{2.38}$$

La explicación de las restricciones de este modelo es similar a la del modelo anterior salvo en algunas restricciones. Ahora, el número de triángulos que debemos coger es $n - |\mathcal{CH}(S)|$ como se impone con (2.27). El número de triángulos adyacentes a una arista de la envolvente convexa es uno si la arista no está en el polígono y cero si la arista está en el polígono, tal y como se impone con (2.29). Por otro lado, si se trata de otra arista (no en la envolvente convexa), debe haber un triángulo adyacente si forma parte del polígono, dos triángulos adyacentes si es una diagonal de la superficie fuera del polígono, y cero triángulos adyacentes en otro caso. Esto se impone con las restricciones (2.30) y (2.31).

En este modelo, al igual que en el anterior, también podemos fijar la variable x_{ij} a 0 si $i, j \in \mathcal{CH}(\mathcal{S})$ son dos vértices no consecutivos de $\mathcal{CH}(\mathcal{S})$. Además, de manera análoga también podemos fijar la variable z_{ij} a 0 si $i, j \in \mathcal{CH}(\mathcal{S})$ son vértices consecutivos de $\mathcal{CH}(\mathcal{S})$.

La restricción (2.33) puede reforzarse mediante:

$$u_j \geq u_i + x_{ij} - (n-2)(1 - x_{ij}) + (n-3)x_{ji} \quad \forall i, j \in S, i, j \neq 0, i \neq j \tag{2.39}$$

La restricción (2.34) puede reforzarse mediante:

$$u_j \geq u_i + 2 - x_{ij} \quad \text{si } (i, j) \text{ es un segmento de } \mathcal{CH}(S) \tag{2.40}$$

Las restricciones de refuerzo (2.24) y (2.25) del Modelo 2 son iguales a las de refuerzo (2.39) y (2.40) del Modelo 3, respectivamente. Se han escrito de manera teórica en cada modelo para una mayor completitud teórica de la formulación, sin embargo, en la implementación del código se tratarán como dos restricciones en lugar de cuatro.
El código que se ha creado puede consultarse en el apéndice A.1.

3

Resultados computacionales

A modo de introducción en este tercer capítulo comenzaremos hablando sobre cómo hemos obtenido los datos con los que hemos trabajado y cuáles son las características de las máquinas que se han utilizado. Luego, presentaremos los resultados preliminares obtenidos al resolver los mismos problemas pero con solucionadores diferentes. Y por último, vamos a comparar nuestros mejores resultados con los ya existentes en la literatura.

Se crearon diez instancias (*instances*) de tamaño n, donde $n \in \{10, 15, 20, 25\}$, usando el mismo *software* generador[1] empleado para el concurso *CG:SHOP* en el año 2019, implementado en C++, es decir, se crearon cuarenta instancias en total. Asimismo, las coordenadas de los puntos en el plano fueron seleccionadas como números enteros pares para que el área de cualquier poligonalización sea también un número entero [10].

- CG:SHOP (*Computational Geometry: Solving Hard Optimization Problems*) [4] es una página web en la que una o varias personas proponen retos de optimización de Geometría Computacional. En particular, el desafío que se propuso en el año 2019 fue organizado por Erik Demaine, Sándor Fekete y Joseph S. B. Mitchell, y se basó en poligonalizaciones de área óptima: dado un conjunto S de n puntos en el plano, calcular una poligonalización simple de S que tenga área máxima o mínima entre todas las poligonalizaciones de S.

La Tabla 3.1 especifica las características de las máquinas que se han utilizado. La máquina M1 fue empleada por Ramos et al. en [3] y la máquina M2 es la que hemos utilizado para obtener los resultados que veremos en este capítulo.

[1] https://gitlab.ibr.cs.tu-bs.de/hip/hip-generators

Máquina	CPU	Clock	RAM
M1	Intel(R) Core(TM) i7-3770	3.9GHz	32GB
M2	Intel(R) Core(TM) i7-2600	3.4GHz	16GB

Tabla 3.1. Características de las máquinas utilizadas.

3.1. Resultados preliminares

En esta sección presentaremos, mediante tablas, los resultados que hemos obtenido tras realizar diferentes experimentos bajo ciertas características que explicaremos a medida que vayamos introduciendo cada tabla. Todos los tiempos que aparecen en las tablas están en segundos y se han redondeado a una cifra decimal.

En primer lugar, mostramos las Tablas 3.2 y 3.3. En ellas ejecutamos cinco instancias de tamaño quince para los problemas SMaxArea y SMinArea respectivamente. En cada modelo comparamos los tiempos de resolución de cada instancia con los solucionadores CBC y GUROBI. Para este experimento se han ejecutado los Modelos 2 y 3 sin restricciones que los fortalezcan.

		Modelo 2		Modelo 3	
Instancia	Área	CBC	GUROBI	CBC	GUROBI
n15s01	26248	157.3	1.9	11.2	0.3
n15s02	21592	200.7	1.4	19.7	0.4
n15s03	22368	229.3	2.0	38.5	0.6
n15s04	27324	100.0	0.8	9.1	0.5
n15s05	22918	495.3	1.3	31.0	0.5
Promedio		236.5	1.5	21.9	0.5

Tabla 3.2. Comparación de tiempos entre CBC y GUROBI para el SMaxArea.

		Modelo 2		Modelo 3	
Instancia	Área	CBC	GUROBI	CBC	GUROBI
n15s01	7668	25.3	0.9	180.4	1.0
n15s02	5512	573.3	2.1	1786.7	4.2
n15s03	6850	209.7	2.3	448.0	1.9
n15s04	7572	110.8	1.0	66.9	0.6
n15s05	3764	136.1	0.9	152.5	1.5
Promedio		211.0	1.4	526.9	1.8

Tabla 3.3. Comparación de tiempos entre CBC y GUROBI para el SMinArea.

Solamente se han ejecutado cinco instancias en cada tabla ya que han sido suficientes para comprobar que ambos modelos se resuelven de manera más rá-

pida con el *software* GUROBI. Se puede observar que dicha diferencia es mucho mayor en el Modelo 2 que en el Modelo 3 cuando maximizamos y sucede lo contrario cuando minimizamos, es decir, el tiempo de resolución es mayor en el Modelo 3.

Una vez demostrado que el solucionador GUROBI es notablemente más eficaz que el solucionador CBC vamos a estudiar qué combinación de restricciones de refuerzo proporciona el menor tiempo de resolución.
Antes de presentar los resultados preliminares, vamos a aclarar cómo denotaremos a estas combinaciones para el Modelo 2:

- V0. Modelo 2 sin restricciones que fortalezcan el modelo, es decir el modelo formado por (2.12)–(2.22).
- V1. Modelo 2 fortalecido cambiando (2.15) por (2.23).
- V2. Modelo 2 fortalecido cambiando (2.17) por (2.24).
- V3. Modelo 2 fortalecido cambiando (2.18) por (2.25).
- V4. Modelo 2 fijando algunas variables x_{ij} y z_{ij} a cero, tal y como se describe en la Sección 2.2.2.
- V5. Modelo 2 fortalecido todo lo posible, es decir, incluyendo los cambios de V1, V2, V3 y V4.

De manera análoga para el Modelo 3 se incluyen las variantes V0, V2, V3, V4 y V5. En este modelo no existe V1 debido a que no se consiguió implementar una restricción similar a la (2.23) del Modelo 2 para este Modelo 3.

Una vez hecha esta aclaración respecto a la notación, presentamos las Tablas 3.4 y 3.5. En la Tabla 3.4 se puede ver que, al resolver el problema SMaxArea para instancias de tamaño quince, la mejor combinación de restricciones, tanto en el Modelo 2 como en el 3, viene dada por la variante V4. Recordemos que en éstas algunas variables x_{ij} y z_{ij} han sido fijadas a cero. Tienen unos tiempos medios de resolución de 1.5 segundos para el Modelo 2 y 0.5 segundos para el Modelo 3. También se observa que, para este problema, la variante del Modelo 3 tiene un tiempo medio de resolución menor que la del Modelo 2.

En la Tabla 3.5 se han resuelto instancias de tamaño veinte y para el Modelo 2 se observa que los menores tiempos de resolución se corresponden con las variantes V2 y V4. Sin embargo, para el Modelo 3 los menores tiempos se corresponden con las variantes V3 y V4, mientras que la V2 pertenece a uno de los mayores.

Por tanto, para resolver el problema SMaxArea utilizaremos la variante V4 debido a que presenta una menor variabilidad en los datos.

		Modelo 2						Modelo 3				
Instancia	Área	V0	V1	V2	V3	V4	V5	V0	V2	V3	V4	V5
n15s01	26248	1.6	1.7	1.5	1.7	1.3	1.6	0.3	0.3	0.2	0.2	0.2
n15s02	21592	1.2	1.8	1.5	2.4	1.4	1.9	0.4	0.4	0.4	0.5	0.3
n15s03	22368	1.9	2.0	1.9	2.2	2.0	2.2	0.6	0.7	0.6	0.6	1.1
n15s04	27324	0.8	1.0	0.7	0.8	0.8	0.9	0.5	0.8	0.9	0.4	0.7
n15s05	22918	1.2	3.0	1.2	1.2	1.0	2.8	0.5	0.4	0.4	0.4	0.4
n15s06	17076	3.5	4.0	3.0	2.8	3.1	3.9	0.7	0.7	0.5	0.7	0.8
n15s07	19760	1.5	2.6	1.5	1.4	1.5	2.2	0.3	0.4	0.5	0.3	0.3
n15s08	18558	1.5	1.8	1.6	1.9	1.5	2.0	0.9	0.9	0.8	0.8	0.6
n15s09	21028	0.9	1.0	0.9	1.0	0.9	1.5	0.3	0.4	0.3	0.3	0.4
n15s10	23562	2.0	2.8	2.0	1.9	1.9	2.5	0.7	0.7	0.6	0.8	0.6
Promedio		1.6	2.2	1.6	1.7	1.5	2.1	0.5	0.6	0.5	0.5	0.5

Tabla 3.4. Comparación de tiempos para diferentes restricciones en los Modelos 2 y 3 para el SMaxArea con instancias de tamaño quince.

		Modelo 2						Modelo 3				
Instancia	Área	V0	V1	V2	V3	V4	V5	V0	V2	V3	V4	V5
n20s01	22042	94.9	119.0	103.6	101.7	84.5	94.7	9.8	9.0	6.7	5.6	7.9
n20s02	24328	127.1	107.3	99.7	117.1	85.9	79.9	4.6	3.1	3.7	5.0	3.1
n20s03	20498	60.2	127.6	65.3	73.0	78.5	117.0	6.5	5.5	18.4	6.8	5.6
n20s04	25420	40.2	42.3	29.1	36.3	40.6	54.0	4.8	7200.0	6.4	7.7	5.2
n20s05	24986	102.0	131.8	101.5	130.6	106.0	176.5	11.8	7.3	8.6	7.7	6.7
n20s06	28060	19.9	22.0	43.3	45.0	38.9	17.8	2.8	4.2	3.7	2.3	3.6
n20s07	26452	133.7	120.0	89.6	96.7	115.2	91.4	31.1	22.2	25.8	32.5	7200.0
n20s08	24674	70.2	39.8	66.1	67.6	54.7	43.5	4.5	5.4	3.7	3.5	2.8
n20s09	28156	103.5	114.0	77.7	96.3	86.5	144.5	5.5	4.1	3.9	5.8	5.3
n20s10	24102	16.1	13.2	17.6	12.6	17.9	19.5	2.3	1.9	2.5	1.7	2.0
Promedio		76.8	83.7	69.3	77.7	70.9	83.9	8.4	726.3	8.3	7.8	724.2

Tabla 3.5. Comparación de tiempos para diferentes restricciones en los Modelos 2 y 3 para el SMaxArea con instancias de tamaño veinte.

En la Tabla 3.6 se presentan los resultados del problema SMinArea para instancias de tamaño quince. Al igual que ocurre con el problema SMaxArea, la variante V4 proporcionan los menores tiempos de resolución. En este caso, la variante del Modelo 2 ha concluido con un tiempo medio de 1.2 segundos mientras que la del Modelo 3 lo hizo a los 1.6 segundos. Al contrario de lo que ocurre con el experimento anterior, en esta ocasión ha sido la variante del Modelo 2 la que ha proporcionado un tiempo medio menor.

Por otra parte, en la Tabla 3.7 observamos que para el Modelo 2 los menores tiempos de resolución se corresponden con las variantes V0 y V2, mientras que para el Modelo 3 se corresponden con las variantes V2 y V4. Debido a que la variante V0 representa en el Modelo 3 uno de los tiempos más altos y observando que la variante V4 nos brinda un tiempo medio de resolución similar al de la variante V2, por convenio se ha decidido resolver el problema SMinArea con la variante V4 para ambos modelos.

Instancia	Área	Modelo 2						Modelo 3				
		V0	V1	V2	V3	V4	V5	V0	V2	V3	V4	V5
n15s01	7668	1.0	1.4	1.0	0.7	0.6	1.2	1.0	1.1	1.4	0.9	1.0
n15s02	5512	2.3	3.4	2.2	2.2	2.2	3.2	4.5	4.4	4.7	4.4	4.2
n15s03	6850	2.2	2.2	2.0	2.2	1.7	2.7	2.1	1.9	2.1	1.7	1.4
n15s04	7572	0.9	1.0	1.1	0.9	1.0	1.1	0.6	0.6	0.6	0.5	0.6
n15s05	3764	0.9	1.6	0.9	1.4	1.1	1.4	1.5	1.5	1.6	0.9	1.6
n15s06	4622	1.1	1.5	1.1	1.9	0.8	1.6	1.0	1.0	1.0	1.0	1.1
n15s07	5092	1.0	1.0	1.0	1.1	1.1	1.2	1.1	1.1	1.0	1.0	1.0
n15s08	4202	1.4	1.9	1.3	1.6	1.4	2.5	1.8	1.8	1.8	2.1	1.7
n15s09	7942	1.1	1.1	1.0	1.0	0.9	1.1	0.8	0.8	0.7	0.8	0.7
n15s10	4244	1.4	2.5	1.8	1.4	1.4	2.8	3.2	3.2	3.2	2.8	3.0
Promedio		1.3	1.8	1.3	1.4	1.2	1.9	1.8	1.8	1.8	1.6	1.6

Tabla 3.6. Comparación de tiempos para diferentes restricciones en los Modelos 2 y 3 para el SMinArea con instancias de tamaño quince.

Instancia	Área	Modelo 2						Modelo 3				
		V0	V1	V2	V3	V4	V5	V0	V2	V3	V4	V5
n20s01	4266	27.6	37.8	23.7	30.4	39.0	38.9	118.4	74.5	81.5	74.0	1431.5
n20s02	5486	28.0	21.7	28.0	19.9	27.0	22.6	10.3	10.3	9.9	10.3	8.1
n20s03	5052	17.0	23.9	13.7	30.7	16.2	24.9	112.3	102.1	131.2	146.7	113.4
n20s04	5342	7.4	8.0	10.1	9.8	8.9	12.1	4.9	4.6	6.4	4.8	4.3
n20s05	5282	28.6	43.2	31.1	28.6	26.4	36.6	112.3	100.3	7200.0	102.9	115.7
n20s06	5846	5.8	7.0	5.1	4.7	4.6	8.3	5.2	5.4	4.7	4.8	6.3
n20s07	3738	10.4	13.9	9.5	11.1	10.2	10.9	9.3	9.4	9.3	9.6	9.3
n20s08	6006	18.1	18.6	16.2	15.6	13.3	14.8	71.5	84.2	65.8	59.0	21.6
n20s09	5958	16.3	30.5	22.3	25.9	25.5	18.7	115.8	96.5	142.6	79.6	87.5
n20s10	7338	4.9	6.6	4.8	4.3	4.2	7.1	4.4	4.4	4.3	3.4	3.4
Promedio		16.4	21.1	16.4	18.1	17.5	19.5	57.4	49.2	765.6	49.4	180.1

Tabla 3.7. Comparación de tiempos para diferentes restricciones en los Modelos 2 y 3 para el SMinArea con instancias de tamaño veinte.

3.2. Resultados finales

Por último, en esta sección vamos a comparar los resultados que obtuvo Ramos et al. en [3] con los que hemos obtenido tras realizar diversos experimentos.

En cada una de las tablas que presentaremos a continuación veremos los resultados de ejecutar las cuarenta instancias creadas. Además, Ramos et al. fijaron un tiempo máximo de ejecución para cada instancia de 10800 segundos mientras que para este trabajo se ha decidido fijarlo en 7200 segundos. Recordemos que, en la sección anterior, decidimos resolver los Modelos 2 y 3 con la variante V4 por lo que utilizaremos ésta para realizar dicha comparación.

En lo que respecta a las áreas de los polígonos, todas las que están reflejadas en las tablas son óptimas pues, tal y como veremos seguidamente, el Modelo 3 ha sido capaz de resolver todas las instancias mencionadas anteriormente con

optimalidad.

En la Tabla 3.8 presentamos los resultados finales que hemos obtenido para el problema SMaxArea. Respecto al Modelo 1, se observa que el tiempo medio de resolución en las instancias de tamaño 10 no presenta una variación destacable si lo comparamos con los otros dos modelos. Es a partir del tamaño 15 cuando comienzan las primeras diferencias. Con tamaño 20, hay algunas que llegan al límite de tiempo establecido mientras que la mayoría de instancias de tamaño 25 no es capaz de resolverlas con optimalidad. Por otra parte, el Modelo 2 es capaz de resolver todas las instancias de tamaño 10, 15 y 20 a pesar de contar con un tiempo máximo de resolución menor y solamente llega a dicho límite en tres instancias de tamaño 25. Si lo comparamos con el Modelo 3, ambos problemas presentan tiempos medios de resolución similares en los tamaños 10 y 15 pero es a partir del tamaño 20 cuando hay un incremento considerable entre ambos tiempos. Tal y como se puede observar, es el Modelo 3 el que da mejores resultados pues ha sido capaz de resolver todas las instancias de manera óptima, sin llegar al tiempo máximo establecido en ningún momento y con un tiempo medio visiblemente inferior, en comparación con los Modelos 1 y 2, en todas las instancias pero en especial en las de tamaño 20 y 25.

Sin embargo, en la Tabla 3.9 no podemos comparar los resultados que hemos obtenido con la literatura ya que Ramos et al. en [3] no ejecutó las instancias para el problema SMinArea.
Para este problema se puede observar que el modelo que presenta menor tiempo medio de resolución es el Modelo 2. A pesar de no ser así en las instancias de tamaño 10, la diferencia es apenas notable. Lo mismo ocurre con las instancias de tamaño 15, aunque en este caso ya es el Modelo 2 el que presenta un tiempo medio menor. Apreciamos que cuanto mayor es el número de puntos de cada instancia, mayor es la diferencia de tiempos entre los Modelos 2 y 3. Notamos que para el problema SMinArea todas las instancias se han resulto de manera óptima con ambos modelos.

Instancia	Área	Modelo 1	Modelo 2	Modelo 3
n10s01	17532	0.7	0.0	0.0
n10s02	12140	0.3	0.0	0.0
n10s03	18296	0.3	0.1	0.1
n10s04	16918	0.4	0.0	0.0
n10s05	18162	0.3	0.1	0.1
n10s06	9832	0.1	0.0	0.0
n10s07	15828	0.3	0.1	0.1
n10s08	17534	0.3	0.1	0.0
n10s09	13900	0.5	0.1	0.0
n10s10	23282	0.2	0.0	0.0
Promedio		0.4	0.1	0.0
n15s01	26248	20.8	1.3	0.2
n15s02	21592	12.0	1.4	0.5
n15s03	22368	10.6	1.9	0.7
n15s04	27324	4.6	0.7	0.4
n15s05	22918	5.3	1.0	0.4
n15s06	17076	17.7	3.0	0.7
n15s07	19760	30.1	1.5	0.3
n15s08	18558	7.5	1.5	0.8
n15s09	21028	11.3	0.9	0.3
n15s10	23562	9.8	1.9	0.7
Promedio		13.0	1.5	0.5
n20s01	22042	10800.0	81.3	5.3
n20s02	24328	10800.0	81.9	4.6
n20s03	20498	276.3	73.6	6.5
n20s04	25420	364.6	38.3	5.1
n20s05	24986	4785.1	89.5	7.1
n20s06	28060	346.6	37.0	2.2
n20s07	26452	134.8	110.0	30.8
n20s08	24674	584.1	51.3	3.4
n20s09	28156	10800.0	82.2	5.4
n20s10	24102	271.1	16.9	1.7
Promedio		3916.3	66.2	7.2
n25s01	26210	10800.0	395.8	7.5
n25s02	25412	10800.0	507.9	12.8
n25s03	26408	10800.0	7200.0	12.9
n25s04	16882	10800.0	1291.0	71.8
n25s05	28350	10800.0	7200.0	49.9
n25s06	26544	6301.2	144.0	12.9
n25s07	26458	10800.0	7200.0	73.6
n25s08	27630	4505.1	294.4	22.7
n25s09	27576	10800.0	2249.3	61.0
n25s10	25050	5076.9	157.1	10.9
Promedio		9148.3	2663.9	33.6

Tabla 3.8. Comparación de tiempos entre los Modelos 1, 2 y 3 para el problema SMaxArea.

Instancia	Área	Modelo 2	Modelo 3
n10s01	9096	0.0	0.0
n10s02	2736	0.0	0.0
n10s03	5766	0.1	0.1
n10s04	6882	0.0	0.0
n10s05	2832	0.1	0.1
n10s06	4766	0.0	0.0
n10s07	3276	0.1	0.1
n10s08	5608	0.0	0.0
n10s09	4958	0.1	0.0
n10s10	8018	0.0	0.0
Promedio		0.1	0.0
n15s01	7668	0.7	0.9
n15s02	5512	2.1	4.1
n15s03	6850	1.8	1.8
n15s04	7572	1.0	0.5
n15s05	3764	1.1	0.9
n15s06	4622	0.8	1.1
n15s07	5092	1.0	0.9
n15s08	4202	1.3	2.1
n15s09	7942	0.9	0.8
n15s10	4244	1.4	2.9
Promedio		1.2	1.6
n20s01	4266	37.6	77.3
n20s02	5486	27.5	10.7
n20s03	5052	16.1	148.2
n20s04	5342	9.0	4.8
n20s05	5282	26.5	86.2
n20s06	5846	4.5	5.0
n20s07	3738	10.4	9.5
n20s08	6006	13.4	59.6
n20s09	5958	26.3	80.1
n20s10	7338	4.2	3.3
Promedio		17.6	48.5
n25s01	6112	77.0	312.3
n25s02	4306	80.1	294.5
n25s03	5116	62.6	395.5
n25s04	3466	286.2	1158.3
n25s05	4056	100.8	5641.0
n25s06	4454	52.9	34.9
n25s07	6902	531.1	1558.9
n25s08	5608	284.7	860.9
n25s09	4592	252.1	2258.3
n25s10	5344	243.1	523.3
Promedio		197.1	1303.8

Tabla 3.9. Comparación de tiempos entre los Modelos 2 y 3 para el problema SMinArea.

Conclusiones

Esta memoria describe el problema que consiste en construir un polígono a partir de un conjunto de puntos en el plano de forma que se optimice (maximice o minimice) el área del polígono. También describe tres modelos matemáticos para estos dos problemas denominados SMaxArea y SMinArea y muestra numerosos resultados computacionales.

De los tres modelos, dos de ellos se han propuesto como novedad en esta memoria, llamados Modelo 2 y Modelo 3. Mediante diversos experimentos realizados se han comparado estos dos modelos y, adicionalmente, se han comparado con un tercer modelo ya existente en la literatura, denominado Modelo 1.

De estos resultados computacionales se deduce que los Modelos 2 y 3 son mejores que el Modelo 1, mejorando ampliamente los tiempos computacionales. Asimismo se puede concluir que el Modelo 3 es mejor que el Modelo 2 para el problema SMaxArea, pero que el Modelo 2 es mejor que el Modelo 3 para el problema SMinArea.

Por último, se ha comprobado que todas las instancias fueron resueltas de forma óptima para el problema SMaxArea con el Modelo 3 y para el problema SMinArea con los Modelos 2 y 3.

A

Apéndice

En esta sección vamos a mostrar el código que se ha creado para la realización de este trabajo.

A.1. Implementación de los modelos

En las secciones 2.2.2 y 2.2.3 se encuentran las formulaciones de los Modelos 2 y 3 respectivamente. Sin embargo, a la hora de implementarlos no hemos creado dos códigos por separado sino que lo hemos hecho en un único código.

En primer lugar, importamos las librerías que vamos a utilizar.

```
1 import random
2 import numpy as np
3 import math
4 import matplotlib.pyplot as plt
5 from ortools.linear_solver import pywraplp
6 from scipy.spatial import ConvexHull, convex_hull_plot_2d
```

Listing A.1. Celda de código 1

Después definimos un épsilon y creamos una función que lea todos los datos que contiene cada uno de los ficheros. Para ello, se ha creado una lista en la que se almacena cada línea del fichero excepto las dos primeras ya que no proporcionan datos que sean necesarios para el correcto funcionamiento del código. Posteriormente, se declaran todas las variables como globales para poder usarlas fuera de la función. Luego, guardamos en vectores toda la información que nos proporciona cada fichero y se declaran más variables que nos facilitarán la implementación de las restricciones.

```
1 EPS = 0.001 # epsilon
2
3 def lectura_fichero(nombre_fichero):
4   lista=[]
5   lista_limpia=[]
6   f = open(nombre_fichero, 'r')
```

```
7   lines = f.readlines() # Lee linea a linea
8   for i in lines:
9     lista.append(i)
10  for k in range(2,len(lista)):
11    limpiar=lista[k].strip('\n').split()
12    lista_limpia.append(limpiar)
13
14  global nP   # Numero de puntos
15  global nCH # Numero de puntos en la envolvente convexa
16  global nT   # Numero de triangulos
17  global nSC # Numero de segmentos que se cruzan
18  global nTI # Numero de triangulos incompatibles
19  global area_CH # Area de la envolvente convexa
20  global S    # Conjunto de puntos
21  global S0   # Conjunto de puntos sin el punto 0
22  global T    # Conjunto de triángulos
23  global coord
24  global CH
25  global triangulos
26  global areas
27  global SC
28  global triangulos_incompatibles
29  global triangulos_adyacentes
30  global estaEnCH
31
32  nP = int(lista_limpia[0][0])
33  nCH = int(lista_limpia[0][1])
34  nT = int(lista_limpia[0][2])
35  nSC = int(lista_limpia[0][3])
36  nTI = int(lista_limpia[0][4])
37  area_CH =  float(lista_limpia[0][5])
38
39  # Coordenadas de los puntos
40  coord = np.zeros((nP,2))
41  first_line = 2
42  for i in range(nP) :
43    coord[i][0] = float(lista_limpia[first_line + i][1])
44    coord[i][1] = float(lista_limpia[first_line + i][2])
45
46  # Puntos que forman la envolvente convexa
47  CH = np.zeros(nCH, dtype = int)
48  first_line = 2+nP+1
49  for i in range(nCH) :
50    CH[i] = int(lista_limpia[first_line + i][1])
51
52  # Coordenadas de los triangulos y sus respectivas areas
53  triangulos = np.zeros((nT,3), dtype=int)
54  areas =  np.zeros(nT)
55  first_line = 2+nP+1+nCH+1
56  triangulos_adyacentes=[]
57  for i in range(nP) :
58    triangulos_adyacentes.append([])
59    for j in range(nP) :
60      if j>i :
61        triangulos_adyacentes[i].append([])
```

```
62   for t in range(nT) :
63     i = int(lista_limpia[first_line + t][1])
64     j = int(lista_limpia[first_line + t][2])
65     k = int(lista_limpia[first_line + t][3])
66     triangulos[t][0] = i
67     triangulos[t][1] = j
68     triangulos[t][2] = k
69     areas[t] = float(lista_limpia[first_line + t][4])
70     triangulos_adyacentes[i][j-i-1].append(t)
71     triangulos_adyacentes[i][k-i-1].append(t)
72     triangulos_adyacentes[j][k-j-1].append(t)
73 
74   # Puntos de los segmentos que se cruzan
75   SC = np.zeros((nSC,4), dtype=int)
76   first_line = 2+nP+1+nCH+1+nT+1
77   for i in range(nSC) :
78     SC[i][0] = int(lista_limpia[first_line + i][2])
79     SC[i][1] = int(lista_limpia[first_line + i][1])
80     SC[i][2] = int(lista_limpia[first_line + i][4])
81     SC[i][3] = int(lista_limpia[first_line + i][3])
82 
83   # Vector que nos dice si un punto esta o no en la CH
84   estaEnCH = np.zeros(nP, dtype=int)
85   for i in range(nCH) :
86     estaEnCH[CH[i]] = 1
87 
88 
89   # Triangulos incompatibles
90   triangulos_incompatibles = np.zeros((nTI,2), dtype=int)
91   first_line = 2+nP+1+nCH+1+nT+1+nSC+1
92   for i in range(nTI) :
93     triangulos_incompatibles[i][0] = int(lista_limpia[first_line + i][1])
94     triangulos_incompatibles[i][1] = int(lista_limpia[first_line + i][2])
95 
96   S = range(nP)
97   S0 = range(1,nP)
98   T = range(nT)
```

Listing A.2. Celda de código 2

Se ha creado una función que dibuja la solución en caso de que ésta sea óptima o factible.

```
1 def dibuja(selected):
2   plt.figure(figsize=(8,8))
3   plt.xlim(0,200)
4   plt.ylim(0,200)
5   plt.xlabel("Coordenada X", fontsize='16')
6   plt.ylabel("Coordenada Y", fontsize='16')
7 
8   for i in S :
9     plt.plot(coord[i][0], coord[i][1],'ro')
10    plt.annotate('%i'%(i), (coord[i][0],coord[i][1]+2),fontsize='12',color=
      'black')
11  for (i,j) in selected:
12    plt.plot([coord[i][0],coord[j][0]], [coord[i][1],coord[j][1]], 'y-')
```

```
13
14 plt.show()
```

Listing A.3. Celda de código 3

Creamos una función a la que le pasaremos los siguientes parámetros:

- MAXIMIZE o MINIMIZE según queramos maximizar o minimizar la función objetivo.
- CBC o GUROBI dependiendo de qué solver queramos utilizar.
- 2 o 3 según qué modelo vamos a utilizar para resolver las instancias.
- TRUE o FALSE dependiendo de si queremos reforzar el Modelo 2 con la restricción (2.23).
- TRUE o FALSE dependiendo de si queremos reforzar los Modelos 2 o 3 con la restricción (2.24).
- TRUE o FALSE dependiendo de si queremos reforzar los Modelos 2 o 3 con la restricción (2.25).
- TRUE o FALSE dependiendo de si queremos reforzar los Modelos 2 o 3 fijando algunas variables a 0.

Definimos las variables, implementamos la función objetivo y las distintas restricciones e imprimimos por pantalla si la solución a la que se ha llegado es óptima o factible, el área y el tiempo empleado en la resolución de cada instancia.

```
1  def modelo(objetive, lp_solver, model, seg_cruzan_fort, rest_u_fort,
     pos_u_fort, fijar_var_0):
2    selected = {}
3    if lp_solver == 'GUROBI' :
4      solver = pywraplp.Solver('Modelo', pywraplp.Solver.
     GUROBI_MIXED_INTEGER_PROGRAMMING)
5    else :
6      solver = pywraplp.Solver('Modelo', pywraplp.Solver.
     CBC_MIXED_INTEGER_PROGRAMMING)
7
8    solver.SetTimeLimit(7200000)
9
10   # Variables de decision
11   y = { (i) : solver.BoolVar('y[%i]' % (i)) for i in T }
12   z = { (i,j) : solver.BoolVar('z[%i,%i]' % (i,j)) for i in S for j in S if
      i<j }
13   x = { (i,j) : solver.BoolVar('x[%i,%i]' % (i,j)) for i in S for j in S if
      j!=i }
14   u = { i : solver.NumVar(0.0, nP-2, 'u[%i]' % i) for i in S0 }
15
16   # Funcion objetivo
17   if objetive == 'Maximize' :
18     if model == 2 :
19       solver.Maximize(solver.Sum( areas[t]*y[t] for t in T ))
20     else :
21       solver.Maximize(area_CH - solver.Sum( areas[t]*y[t] for t in T ))
22   else :
23     if model == 2 :
```

```
24      solver.Minimize(solver.Sum( areas[t]*y[t] for t in T )) # Maximize o Minimize
25    else :
26      solver.Minimize(area_CH - solver.Sum( areas[t]*y[t] for t in T ))
27
28  # Restricciones
29  # Restriccion 1
30  if model == 2 :
31    solver.Add( solver.Sum( y[t] for t in T) == nP - 2 )
32  else :
33    solver.Add( solver.Sum( y[t] for t in T) == nP - nCH )
34  # Restriccion 2
35  [ solver.Add( y[triangulos_incompatibles[i,0]] + y[triangulos_incompatibles[i,1]] <= 1 ) for i in range(nTI)]
36  # Restriccion 3
37  if model == 2 :
38    [ solver.Add( solver.Sum( y[t] for t in triangulos_adyacentes[i][j-i-1])  == 2*z[i,j] + x[i,j] + x[j,i] ) for i in S for j in S if i < j ]
39  else :
40    [ solver.Add( solver.Sum( y[t] for t in triangulos_adyacentes[CH[i]][CH[i+1]-CH[i]-1])  == 1 - x[CH[i],CH[i+1]] - x[CH[i+1],CH[i]] ) for i in range(0,nCH-1) if CH[i] < CH[i+1] ]
41    [ solver.Add( solver.Sum( y[t] for t in triangulos_adyacentes[CH[i+1]][CH[i]-CH[i+1]-1])  == 1 - x[CH[i],CH[i+1]] - x[CH[i+1],CH[i]] ) for i in range(0,nCH-1) if CH[i] > CH[i+1] ]
42    if CH[nCH-1] < CH[0] :
43      solver.Add( solver.Sum( y[t] for t in triangulos_adyacentes[CH[nCH-1]][CH[0]-CH[nCH-1]-1])  == 1 - x[CH[nCH-1],CH[0]] - x[CH[0],CH[nCH-1]] )
44    else :
45      solver.Add( solver.Sum( y[t] for t in triangulos_adyacentes[CH[0]][CH[nCH-1]-CH[0]-1])  == 1 - x[CH[nCH-1],CH[0]] - x[CH[0],CH[nCH-1]] )
46  # Restriccion 4
47  if model == 2 :
48    if seg_cruzan_fort == 0 :
49      [ solver.Add( x[i,j] + x[j,i] + z[i,j] <= 1 ) for i in S for j in S if i<j ]
50    else :
51      [ solver.Add( x[SC[i][0],SC[i][1]] + x[SC[i][1],SC[i][0]] + z[SC[i][0],SC[i][1]] + x[SC[i][2],SC[i][3]] + x[SC[i][3],SC[i][2]] + z[SC[i][2],SC[i][3]] <= 1 )
52          for i in range(nSC) ]
53  else :
54    [ solver.Add( solver.Sum( y[t] for t in triangulos_adyacentes[i][j-i-1])  == 2*z[i,j] + x[i,j] + x[j,i] )
55        for i in S for j in S if i < j if (estaEnCH[i] == 0 or estaEnCH[j] == 0) ]
56  # Restriccion 5
57  [ solver.Add( solver.Sum( x[i,j] for j in S if j!=i) == 1 )  for i in S ]
58  [ solver.Add( solver.Sum( x[j,i] for j in S if j!=i) == 1 )  for i in S ]
59  # Restriccion 6
60  if rest_u_fort == 0 :
61    [ solver.Add( u[j] >= u[i] + x[i,j] - (nP-2)*(1-x[i,j]) ) for i in S0 for j in S0 if j!=i ]
62  else :
```

```
63      [ solver.Add( u[j] >= u[i] + x[i,j] - (nP-2)*(1-x[i,j]) + (nP-3)*x[j,i]
         ) for i in S0 for j in S0 if j!=i ]
64    # Restriccion 7
65    if pos_u_fort == 0 :
66      [ solver.Add( u[CH[i+1]] >= u[CH[i]] + 1 ) for i in range(1,nCH-1) ]
67    else :
68      [ solver.Add( u[CH[i+1]] >= u[CH[i]] + 2 - x[CH[i],CH[i+1]]) for i in
        range(1,nCH-1) ]
69    # Restricciones de refuerzo
70    if fijar_var_0 != 0 :
71      [ solver.Add( x[CH[i],CH[j]] == 0 ) for i in range(nCH) for j in range(
        nCH) if (i-j > 1 and (i!=nCH-1 or j!=0 )) or (j-i > 1 and (i!=0 or j!=
        nCH-1 ))]
72      [ solver.Add( z[CH[i],CH[j]] == 0 ) for i in range(nCH) for j in range(
        nCH) if CH[i] < CH[j]]
73
74    # Resolvemos el problema
75    global status
76    status = solver.Solve()
77    global tiempo
78    global area_total
79    tiempo = solver.WallTime()/1000
80    area_total = 0
81    if status == pywraplp.Solver.OPTIMAL :
82      area_total = solver.Objective().Value()
83      print('La solucion optima tiene un area de ', area_total, ' y el
        problema se ha resuelto en ', tiempo, ' segundos')
84      selected = [(i,j) for i in S for j in S if i!=j if x[i,j].
        solution_value() > EPS]
85      dibuja(selected)
86    elif status == pywraplp.Solver.FEASIBLE :
87      area_total = solver.Objective().Value()
88      print('La solucion factible tiene un area de ', area_total, ' y el
        problema se ha resuelto en ', tiempo, ' segundos')
89      selected = [(i,j) for i in S for j in S if i!=j if x[i,j].
        solution_value() > EPS]
90      dibuja(selected)
91    else:
92      print('Este problema no tiene solucion factible')
```

Listing A.4. Celda de código 4

Por último, tenemos el programa principal, en el cual se generan todas las tablas que aparecen en el capítulo 3 y se crea un archivo de texto en el que se graban los resultados obtenidos tras la ejecución de todos los ficheros.

```
1 fichero_resultados = "results.txt"
2
3 # Programa principal
4 f = open(fichero_resultados, 'a')
5 f.write("Objetivo\tSolver\tModelo\tNombre\tn\tseg_cruzan_fort\trest_u_fort\
    tpos_u_fort\tfijar_var_0\tstatus\tTiempo\tArea\n")
6 f.close()
7
8 # Tablas 3.2 y 3.3
```

```
9  opciones_objetivo = ['Minimize'] # 'Maximize' o 'Minimize'
10 solucionadores = ['GUROBI']   # 'CBC' o 'GUROBI'
11 modelos = [2,3]
12 # tamanios = [10,15,20]
13 tamanios = [15]
14 # semillas = ['01','02','03','04','05','06','07','08','09','10']
15 semillas = ['01','02','03','04','05']
16 opciones = [  # seg_cruzan_fort, rest_u_fort, pos_u_fort, fijar_var_0
17     [0, 0, 0, 0]
18     ]
19
20 # Tablas 3.4 y 3.5
21 # opciones_objetivo = ['Maximize', 'Minimize'] # 'Maximize' o 'Minimize'
22 # solucionadores = ['GUROBI']   # 'CBC' o 'GUROBI'
23 # modelos = [2,3]
24 ## tamanios = [10,15,20]
25 # tamanios = [15]
26 # semillas = ['01','02','03','04','05','06','07','08','09','10']
27 # opciones = [  # seg_cruzan_fort, rest_u_fort, pos_u_fort, fijar_var_0
28 #     [0, 0, 0, 0],
29 #     [1, 0, 0, 0],
30 #     [0, 1, 0, 0],
31 #     [0, 0, 1, 0],
32 #     [0, 0, 0, 1],
33 #     [1, 1, 1, 1]
34 #     ]
35
36 # Tablas 3.6 y 3.7
37 # opciones_objetivo = ['Minimize'] # 'Maximize' o 'Minimize'
38 # solucionadores = ['GUROBI']   # 'CBC' o 'GUROBI'
39 # modelos = [2,3]
40 # tamanios = [10,15,20,25]
41 # semillas = ['01','02','03','04','05','06','07','08','09','10']
42 # opciones = [  # seg_cruzan_fort, rest_u_fort, pos_u_fort, fijar_var_0
43     [0, 0, 0, 1]
44     ]
45
46
47 for obj in opciones_objetivo :
48   for sol in solucionadores :
49     for tam in tamanios :
50       for s in semillas :
51         nombre_fichero = 'n' + str(tam) + 's' + s + '.pre'
52         lectura_fichero(nombre_fichero)
53         print('---------',nombre_fichero, '--------- ')
54         for m in modelos :
55           for opt in opciones :
56             print('Solving ' + obj + ' ' + sol + ' model ' + str(m) + ' ' +
       str(opt))
57             modelo(obj, sol, m, opt[0], opt[1], opt[2], opt[3])
58             f = open(fichero_resultados, 'a')
59             f.write(obj)
60             f.write("\t")
61             f.write(sol)
62             f.write("\t")
```

```
63              f.write(str(m))
64              f.write("\t")
65              f.write(nombre_fichero)
66              f.write("\t")
67              f.write(str(nP))
68              f.write("\t")
69              f.write(str(opt[0]))
70              f.write("\t")
71              f.write(str(opt[1]))
72              f.write("\t")
73              f.write(str(opt[2]))
74              f.write("\t")
75              f.write(str(opt[3]))
76              f.write("\t")
77              f.write(str(status))
78              f.write("\t")
79              f.write(str(tiempo))
80              f.write("\t")
81              f.write(str(area_total))
82              f.write("\n")
83              f.close()
```

Listing A.5. Celda de código 5

A.2. Otros códigos

También se creó un código para ver representada en el plano la envolvente convexa de un conjunto de puntos.

```
1  def dibuja_envolvente(selected):
2    plt.figure(figsize=(8,8))
3    plt.xlim(0,200)
4    plt.ylim(0,200)
5    plt.xlabel("Coordenada X", fontsize='16')
6    plt.ylabel("Coordenada Y", fontsize='16')
7    plt.title('Envolvente convexa \n', fontsize='18')
8
9    hull = ConvexHull(coord)
10   plt.plot(coord[:,0], coord[:,1], 'ro')
11   for simplex in hull.simplices:
12     plt.plot(coord[simplex, 0], coord[simplex, 1], 'y-')
13   for i in S :
14     plt.annotate('%i'%(i), (coord[i][0],coord[i][1]+2),fontsize='12',color=
       'black')
15   plt.show()
16 dibuja_envolvente({})
```

Listing A.6. Código para representar la envolvente convexa

Bibliografía

[1] Preparata, F. P., y Shamos, M. I., (1985). *Computational Geometry*. Nueva York: Springer-Verlag.

[2] Berg, M., Kreveld, M., Overmars, M., y Schwarzkopf, O., (1997). *Computational Geometry*. Heidelberg: Springer.

[3] Ramos, N., de Rezende, P. J., y de Souza, C. C.. Optimal area polygonization problems: Exact solutions through geometric duality. *Computers & Operations Research*, volume 145, article 105842. https://www.sciencedirect.com/science/article/abs/pii/S0305054822001204.

[4] CG:SHOP. [Consulta: 6 de septiembre de 2022]. Recuperado de https://cgshop.ibr.cs.tu-bs.de/.

[5] The CGAL Project, CGAL User and Manual, 4,13 ed., CGAL Editorial Board, 2018. [Consulta: 6 de septiembre de 2022]. https://doc.cgal.org/4.13/Manual/packages.html.

[6] The CGAL Project. [Consulta: 6 de septiembre de 2022]. Recuperado de https://www.cgal.org/index.html.

[7] IBM ILOG CPLEX Optimization Studio. [Consulta: 6 de septiembre de 2022]. Recuperado de https://www.ibm.com/products/ilog-cplex-optimization-studio.

[8] Concert Technology. [Consulta: 6 de septiembre de 2022]. Recuperado de https://www.ibm.com/support/pages/introduction-concert-technology.

[9] Python. [Consulta: 6 de septiembre de 2022]. Recuperado de https://www.python.org/.

[10] O'Rourke, J., (1998). *Computational Geometry in C*. Nueva York: Cambridge University Press.

[11] Icosian Game. Wikipedia. [Consulta: 7 de septiembre de 2022]. Recuperado de https://es.wikipedia.org/wiki/Juego_Icosian.

[12] Fekete, S.P., Friedrichs, S., Hemmer, M., Papenberg, M., Schmidt, A., y Troegel, J. (2015). Area- and Boundary-Optimal Polygonalization of Planar Point Sets. EuroCG 2015.

[13] CBC User's Guide. [Consulta: 8 de septiembre de 2022]. Recuperado de https://coin-or.github.io/Cbc/.

[14] Gurobi. [Consulta: 8 de septiembre de 2022]. Recuperado de https://www.gurobi.com/.

[15] Dantzig, G., Fulkerson R. y Johnson, R. (1954). "Solution of a Large-Scale Traveling-Salesman Problem". Journal of the Operations Research Society of America. Vol. 2, (pp. 393-410).

Lista de símbolos y abreviaciones

$\mathbb{R}^{2}$	Espacio vectorial
$\boldsymbol{P}$	Polígono simple
$\boldsymbol{a(P)}$	Área del polígono simple $\boldsymbol{P}$
$\overline{\boldsymbol{pq}}$	Segmento que une los vértices $\boldsymbol{p}$ y $\boldsymbol{q}$
$\boldsymbol{(p,q)}$	Segmento que une los vértices $\boldsymbol{p}$ y $\boldsymbol{q}$
$\boldsymbol{\mathcal{CH}(S)}$	Envolvente convexa de un conjunto de puntos $\boldsymbol{S}$
$\boldsymbol{E_S}$	Conjunto de aristas cuyos vértices están en un conjunto de puntos $\boldsymbol{S}$
$\boldsymbol{E_{\mathcal{CH}(S)}}$	Conjunto de aristas cuyos vértices están en la envolvente convexa $\boldsymbol{\mathcal{CH}(S)}$
$\boldsymbol{G=(S,E_S)}$	Grafo no dirigido cuyo conjunto de puntos es $\boldsymbol{S}$ y tiene a $\boldsymbol{E_S}$ como conjunto de aristas
$\boldsymbol{T}$	Conjunto de triángulos
$\boldsymbol{\Delta(S)}$	Conjunto de triángulos cuyos vértices están en $\boldsymbol{S}$
$\boldsymbol{\Delta(S,i)}$	Conjunto de triángulos cuyos vértices están en $\boldsymbol{S}$ y tienen a $\boldsymbol{i}$ como uno de sus vértices
$\boldsymbol{\Delta(S,i,j)}$	Conjunto de triángulos cuyos vértices están en $\boldsymbol{S}$ y tienen a $\boldsymbol{i}$ y a $\boldsymbol{j}$ como dos de sus vértices

Optimizing the area of a polygon given its vertex

Ainoa González García
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0100895810@ull.edu.es

Abstract

In this work we will present two alternative formulations to solve two different problems that share the same characteristics. Given a set S of n points in the plane, we want to find a simple polygon whose set of vertices are exactly the n points of the set S and whose area is maximum or minimum. The results obtained are compared with those already existing in the literature.

1. Introduction

Given a finite set S of n points in the plane, we want to find a polygon whose vertex set is S. We can distinguish two types of polygons: polygons in general (they can have holes inside) and simple polygons. Also, we may be interested in maximize and/or minimize its area or the length of its boundary. This is how we create a family of eight problems [2] that optimize polygons whose vertices are in a given set of points.

	General polygon		Simple polygon	
	Area	Boundary	Area	Boundary
Minimize	MinArea	MinBound	SMinArea	SMinBound
Maximize	MaxArea	MaxBound	SMaxArea	SMaxBound

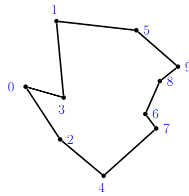

Figure 1: Optimal solution for the SMaxArea problem with 10 points.

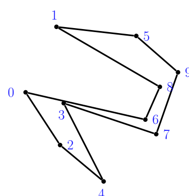

Figure 2: Optimal solution for the SMinArea problem with 10 points.

The best known member of this family is the SMinBound, that is, the Traveling Salesmsn Problem. However, in this work we are going to study the SMaxArea and the SMinArea problems.

2. Models

The first model that was proposed by Ramos et al. in [1] is based on the correspondence between a triangulation of a polygon and a tree of the dual graph defined as follows. Consider $G = (\Delta(S), E)$ whose set of nodes $\Delta(S)$ corresponds to each possible triangle of a triangulation of a polygon P, and each element of E corresponds with two adjacent triangles.

The concept of this second model is that, given a triangulation of a simple polygon P, there is exactly one triangle adjacent to each side of the polygon and there are exactly two triangles adjacent to each diagonal.

The idea of this third model is to take into account the area that is formed between the convex hull and the polygon. In this way we will triangulate this surface using the variables defined in the model.

This is how these alternative formulations emerged to solve the problems SMaxArea and SMinArea.

3. Computational results

We carried out several experiments to determine which solvers were the most efficient and which variants of Models 2 and 3 provided the shortest mean time. It was concluded that both models were more efficient with the V4 variant when using the GUROBI solver.
Here we compare our best work with the literature [1] for the SMaxArea:

	SMaxArea		
# Points	Model 1	Model 2	Model 3
10	0.4	0.1	0.0
15	13.0	1.5	0.5
20	3916.3	66.2	7.2
25	9148.3	2663.9	33.6

Clearly Model 3 provides the lowest average resolution time for the SMaxArea problem. However, we can not compare our results with the literature for the SMinArea problem because no instances were run for that problem. Here are the results that we obtained:

	SMinArea	
# Points	Model 2	Model 3
10	0.1	0.0
15	1.2	1.6
20	17.6	48.5
25	197.1	1303.8

For the SMinArea problem Model 2 provides the best results.

References

[1] Ramos, N., de Rezende, P. J., y de Souza, C. C.. Optimal area polygonization problems: Exact solutions through geometric duality. Computers & Operations Research, volume 145, article 105842. https://www.sciencedirect.com/science/article/abs/pii/S0305054822001204.

[2] Fekete, S.P., Friedrichs, S., Hemmer, M., Papenberg, M., Schmidt, A., y Troegel, J. (2015). Area- and Boundary-Optimal Polygonalization of Planar Point Sets. EuroCG 2015.

Sección de Matemáticas
Universidad de La Laguna