# Modelos de optimización en el diseño artístico



**Aitor Hernández González**

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Directores del trabajo: Herminia I. Calvete Fernández
y Jose Ángel Iranzo Sanz

10 de septiembre de 2021

# Prólogo

La optimización es el área de las matemáticas que estudia problemas en los que se trata de determinar la mejor manera posible de hacer algo, por ejemplo, diseñar un sistema, establecer un proceso de producción, etc., entre un conjunto de alternativas, de acuerdo con un criterio establecido. La optimización matemática es ampliamente usada en diferentes áreas como pueden ser la economía, la ingeniería, las telecomunicaciones, las ciencias naturales, la logística o las ciencias de la salud. Se pueden encontrar numerosos ejemplos de este tipo de aplicaciones en la vida cotidiana. Otra área, no tan habitual o evidente como las anteriores, en la que es posible aplicar los métodos estudiados en optimización es el diseño artístico.

El objetivo de esta memoria es mostrar cómo se pueden aplicar métodos de optimización matemática para la construcción de elementos artísticos visuales, concretamente para la construcción de mosaicos a partir de imágenes digitales. Los modelos formulados y los métodos utilizados se enmarcan en el área de la optimización lineal entera. La memoria está compuesta por tres capítulos cuyo contenido se resume a continuación.

En el capítulo 1 se introduce brevemente lo que es la técnica artística del mosaico y de sus orígenes. Además, se dan unas ideas sobre tratamiento de imágenes digitales, representación de imágenes digitales como matrices de números, modelos de la teoría del color y transformación de imágenes a escala de grises.

En el capítulo 2 se presenta el modelo general de optimización lineal entera, que es un modelo de optimización lineal que incluye restricciones de integridad sobre todas o algunas de las variables. En este contexto es importante el concepto de *relajación lineal* de un problema de optimización lineal entera, que es el problema de optimización lineal que surge al eliminar del modelo todas las restricciones de integridad sobre las variables. Este concepto es clave para el desarrollo de métodos de resolución de problemas de optimización lineal entera. En este capítulo se estudia con detalle el método de *ramificación y acotación*, un método clásico de resolución de problemas de optimización lineal entera que se basa en la estrategia conocida como *divide y vencerás*. Se presentan los principales resultados que justifican el uso de esta estrategia y se exponen algunos criterios de decisión que se usan en la práctica en las diferentes etapas del método. Para cerrar el capítulo se introduce el concepto de *desigualdades válidas*. Las desigualdades válidas son restricciones adicionales que se añaden al problema para mejorar la formulación de los modelos de optimización lineal entera y perfeccionar el desarrollo de, por ejemplo, el método de de ramificación y acotación.

El capítulo 3 se centra en la formulación de modelos de optimización lineal entera para la construcción de mosaicos a partir de imágenes digitales en escala de grises con un conjunto de teselas determinado. Los modelos varían en función de las características de las teselas utilizadas y de las restricciones que tenga sentido imponer con esas teselas. Para mostrar el funcionamiento de los modelos formulados, se hace uso de una imagen del autorretrato del pintor zaragozano Francisco de Goya y Lucientes a partir de la cual se construyen mosaicos con los diferentes modelos. Se presentan fundamentalmente tres tipos de modelos.

En primer lugar, se formulan tres modelos para la construcción de mosaicos con un conjunto de teselas en escala de grises con motivos estampados tales que pueden ser ordenadas de más oscura a más clara. Este conjunto de teselas ha sido creado para este trabajo por la diseñadora zaragozana Marta Longas Clemente con la temática “Artistas contemporáneos de los siglos XIX y XX” y consiste en una serie de doce retratos de artistas destacados de esta época (seis hombres y seis mujeres). El primer modelo

no impone restricciones sobre las teselas, el segundo modelo obliga a utilizar conjuntos completos de teselas y el tercer modelo impide que se formen agrupamientos de teselas iguales, es decir, ninguna tesela está en contacto con otra tesela igual.

En segundo lugar, se formula un modelo para la construcción de mosaicos con un conjunto de teselas en escala de grises que también se pueden ordenar de más oscura a más clara y que siguen patrones, es decir, que las teselas encajan entre sí en el sentido de que una tesela solo admite como teselas adyacentes un subconjunto de teselas del conjunto completo.

En la tercera parte de este capítulo se han considerado teselas no homogéneas, con una estructura más compleja que no han sido tratadas previamente en la literatura. Estas teselas están basadas en las piezas del famoso juego de lógica Tetris®. Cada una de las piezas está coloreada de un tono de gris diferente, siendo los tonos de gris utilizados equidistantes en la escala de grises. Se admiten también todas las posibles rotaciones en el plano de las piezas. El hecho de que deban encajar piezas que no solo difieren en su color sino también en su forma precisa un modelo de optimización lineal entera más elaborado que los anteriores y que es una aportación novedosa de esta memoria al diseño artístico de mosaicos.

Para la resolución de todos los modelos que aparecen a lo largo de esta memoria se ha utilizado la versión 20.1.0 del paquete de software IBM ILOG CPLEX Optimization Studio. Por otro lado, para la construcción de mosaicos a partir de los modelos se han programado varias funciones con el lenguaje de programación Python que ejecutan todo el proceso.

Al final de la memoria se incluyen cuatro anexos. En el primero, se muestra el código Python que se ha programado para la construcción de los mosaicos. En el segundo, se presenta un ejemplo de aplicación del método de ramificación y acotación para resolver un problema de optimización lineal entera. En el tercero, se muestra el código CPLEX implementado para la resolución de algunos de los modelos que se formulan en el capítulo 3. Y por último, en el cuarto anexo se incluyen otros mosaicos que, por razones de espacio, no han podido mostrarse en la parte principal de la memoria.

# Summary

Optimization is the branch of mathematics that studies problems which deal with determining the best posible way to do something, for instance, design a system, establish a production process, etc., among a set of alternatives according to some establish criteria. Mathematical optimization is widely used in different areas such as economy, engineering, telecommunications, natural sciences, logistics or health sciences. It is easy to find a lot of situations in daily life in which these applications are present. Another area not so usual but not less important in which is possible to apply methods and techniques studied in optimization is artistic design.

The main purpose of this report is to show how methods of mathematical optimization can be applied to create visual artistic elements, in particular to create mosaics from digital pictures. The models that are formulated and the methods that are used are part of the subject named integer linear optimization. This report consists in three chapters which are summarized below.

The first chapter begins with a brief introduction about the artistic technique of mosacking and its roots. Moreover, several ideas about digital pictures treatment, digital pictures representation like matrices, color theory models and transformation of pictures into grayscale pictures are presented.

In the second chapter, the general model of integer linear optimization is introduced. It is a linear optimization model which includes integer constraints over every variable or some of them. In this context it is important the concept of *linear relaxation* of an integer linear optimization problem which is a linear problem that comes up when every integer constraint over the variables of the model is removed. This concept is key to the development of methods for solving integer linear optimization problems. In this chapter it is studied in detail the *branch and bound* method, a classic method for solving integer linear optimization problems which relys on the strategy known as *divide and conquer*. Fundamental results that justify the use of this strategy and some decision criteria used in practice in different steps of the method are presented. In order to wind up the chapter, it is introduced the concept of *valid inequalities*. Valid inequalities are additional constraints which are added to the problem in order to improve the formulation of integer linear optimization models and enhance the performance of methods like the branch and bound method.

The third chapter focus on the formulation of integer linear optimization models to create mosaics from digital pictures using a set of tiles. Models vary according to the characteristics of the tiles and the constraints that make sense to impose with the tiles. For showing the performance of the formulated models, several mosaics are created from a picture of the self-portrait of the artist Francisco de Goya y Lucientes from Zaragoza. Three kind of models are mainly presented.

First of all, three models are formulated to create mosaics with a grayscale set of tiles with stamped motifs such that they can be ordered from the darkest to the brightest. This set of tiles has been created by the designer Marta Longas Clemente from Zaragoza. It is about "Contemporary artists from the XIX and XX centuries" and consists in a series of twelve protraits of artist of that period (six males and six females). The first model does not impose any constraint over the tiles, the second one forces to use entire sets of tiles and the third one avoids clumpiness of tiles of the same type, in other words, none of the tiles is in contact with other tile of the same type.

Secondly, a model is formulated to create mosaics with a grayscale set of tiles which can be also ordered from the darkest one to the brightest one and form patterns, meaning that a tile only admits as neighbour tiles a subset of tiles of the entire set.

In the third part of this chapter it has been considered a set of non-homogeneous tiles with a more

complex structure which has not appeared previously in literature. This set of tiles relys on the pieces of the popular logics game Tetris®. Every piece is colored with a gray tone different from the others and all the gray tones used are equidistant in the gray scale. Every posible rotation of the pieces in the plane is allowed. The fact that tiles may fit and not only differ from color but from shape requires a more elaborate integer linear optimization model than previous ones. This model is an innovative contribution of this report to the mosacking artistic design.

In order to solve all the models appearing along this report it has been used the version 20.1.0 of the software package IBM ILOG CPLEX Optimization Studio. Furthermore, in order to create mosaics from models, several functions has been programmed in Python.

At the end of this report, four appendices has been included. In the fisrt one, it is shown the Pyhton code which has been programmed to create mosaics. In the second one, it is presentes an instance of an application of the branch and bound method to solve an integer linear optimization problem. In the third one, it is shown the CPLEX code implemented to solve some of the models stated in the third chapter. Lastly, in the fourth appendix they are included other mosaics which have not been able to show in the main part of this report because of lack of space.

# Índice general

**Prólogo** III

**Summary** V

**1. Mosaicos** 1
1.1. Introducción . . . 1
1.2. El sistema *RGB* . . . 1
1.3. Representación de imágenes digitales en escala de grises . . . 3
1.4. Tratamiento de imágenes digitales con ordenador . . . 3

**2. Optimización lineal entera** 5
2.1. Optimización . . . 5
2.2. Optimización lineal entera . . . 6
2.3. Relajación lineal . . . 7
2.4. Método de ramificación y acotación . . . 8
2.4.1. Criterios de selección de subproblemas . . . 9
2.4.2. Criterios de ramificación . . . 11
2.4.3. Método de ramificación y acotación incompleto . . . 11
2.5. Desigualdades válidas . . . 11

**3. Modelos de optimización lineal entera para la construcción de mosaicos** 13
3.1. Introducción . . . 13
3.2. Función objetivo . . . 13
3.3. Construción de mosaicos con teselas de motivos estampados . . . 14
3.3.1. Modelo sin restricciones sobre las teselas . . . 15
3.3.2. Modelo con conjuntos completos de teselas . . . 15
3.3.3. Modelo sin agrupamientos de teselas iguales . . . 16
3.4. Construción de mosaicos con teselas que siguen patrones . . . 17
3.5. Construcción de mosaicos con piezas de Tetris® . . . 20

**Bibliografía** 25

**Anexos** 27

**A. Código Python para la creación de mosaicos** 29

**B. Ejemplo de aplicación del método de ramificación y acotación** 31

**C. Ejemplos de códigos CPLEX para la resolución de modelos** 35

**D. Otros mosaicos** 43

# Capítulo 1

# Mosaicos

## 1.1. Introducción

Los mosaicos nacen en la antigüedad como un tipo de creación artística que consistía en la decoración de superficies utilizando pequeñas piezas de vidrio o cerámica, piedras, guijarros, etc., de distintos colores que eran fijados con algún tipo de material aglomerante como podía ser barro, yeso, argamasa o cemento. Los mosaicos se diseñaban, en general, para representar imágenes, retratos o motivos geométricos [10, 11].

Los primeros mosaicos de los que se tiene constancia aparecen en Mesopotamia y desde ahí se extienden a civilizaciones posteriores como la Antigua Grecia y la Antigua Roma. Con estas dos civilizaciones puede afirmarse que comienza la expansión del uso de los mosaicos. Los romanos realizaban mosaicos con pequeñas piezas cúbicas a las que llamaron teselas que en ocasiones se coloreaban con el objetivo de tener una mayor gama cromática en el proceso de creación. Las teselas empezaron a sustituir a las piezas irregulares ya que al poder colocarlas de manera ordenada y sin dejar huecos se conseguían diseños más precisos y claros.

Los mosaicos siguieron desarrollándose durante el Imperio Bizantino y fueron utilizados ampliamente por culturas tan diversas como la cristiana, la islámica o los pueblos precolombinos durante la Edad Media para decorar centros de culto o para crear piezas ornamentales. Tuvieron también importancia durante el período Renacentista, época en la que se rescató el arte griego y romano clásico, y resurgieron de nuevo con el Modernismo y artistas como Antonio Gaudí que los incluyó en sus edificios y esculturas.

En la figura 1.1 se muestra el mosaico de Orfeo, que se encuentra expuesto en el Museo de Zaragoza, datado entre los inicios del siglo II y los inicios del III d. C. y que perteneció a la sala principal de una antigua domus de la colonia de Caesaraugusta.

El objetivo de esta memoria es presentar cómo se puede utilizar la optimización para diseñar mosaicos a partir de imágenes en escala de grises creados con distintos tipos de teselas. Estas ideas se basan en el trabajo del matemático Robert Bosch [1]. En la figura 1.2 se observa una fotografía junto a un mosaico creado a partir de ella con un conjunto de cinco teselas distintas llamadas *teselas de Truchet* descritas en [1].

Antes de estudiar los modelos de optimización, en este capítulo se dan además algunas ideas sobre escalas de medición de color, representación de imágenes en escala de grises y tratamiento de imágenes digitales.

## 1.2. El sistema *RGB*

El sistema *RGB* es un modelo de la teoría del color que se fundamenta en la síntesis aditiva del color. Este sistema explica la obtención de colores como combinación de tres colores denominados primarios que son rojo, verde y azul. Así, un color cualquiera $C$ del espectro queda determinado por una tupla de tres componentes $(r,g,b)$ donde $r$, $g$ y $b$ son valores que miden la intensidad o el grado en el que intervienen los colores primarios rojo, verde y azul, respectivamente, en la formación del color $C$. Se

Figura 1.1: Mosaico de Orfeo. Foto: José Garrido. Museo de Zaragoza [7].

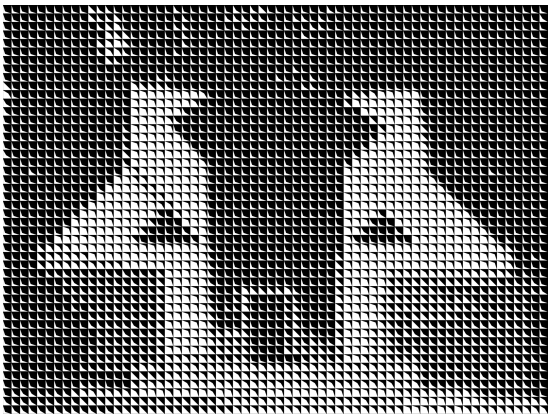


Figura 1.2: Imagen junto a un mosaico creado a partir de ella. En la imagen de la izquierda se muestra el monumento “Las Pajaritas” situado en el parque Miguel Servet de la ciudad de Huesca. Fuente: [9].

dice que el sistema *RGB* tiene por tanto tres canales, el canal *R* para el color rojo, el canal *G* para el color verde y el canal *B* para el color azul.

Para poder asignar los valores $r$, $g$ y $b$ a un color $C$ es necesario disponer de una escala de medición. Generalmente, se utiliza una escala de valores enteros entre el 0 y el 255 donde 0 es el mínimo valor de intensidad y 255 el máximo valor. Por ejemplo, si tenemos el color $C$ caracterizado por la tupla $(0,0,255)$, $r = 0$ y $g = 0$ indican que los colores rojo y verde no intervienen en la formación del color $C$ y $b = 255$ indica que el color azul interviene en la formación del color $C$ con su máximo valor de intensidad, es decir, el color $C$ es el azul. Se utiliza esta escala de medición porque normalmente la intensidad de los colores primarios se codifica con un byte, lo que da lugar a 256 valores posibles de intensidad para cada color primario. Otra escala de medición que se suele emplear, y que en ocasiones es más cómoda, es el intervalo $[0,1]$ donde 0 es el mínimo valor de intensidad y 1 el máximo valor.

El sistema *RGB* es utilizado en muchos de los dispositivos que forman imágenes empleando la emisión directa de luz como pueden ser televisores, ordenadores, teléfonos móviles, cámaras digitales, etc. Las pantallas de estos dispositivos están formadas por píxeles. Un píxel es la menor unidad homogénea de color que forma parte de una imagen digital. Por tanto, cada píxel de una pantalla o de una imagen digital tendrá asociada una tupla $(r,g,b)$ que se corresponderá con el color que tenga dicho píxel. De esta manera, se puede representar una imagen digital como un conjunto de tres matrices de tamaño $h \times w$, con $h$ el número de píxeles de alto de la imagen, $w$ el número de píxeles de ancho de la imagen y donde cada matriz almacena la información de cada uno de los tres canales del sistema *RGB*.

En esta memoria siempre se van a utilizar imágenes en escala de grises para la creación de mosaicos. En el sistema *RGB* la escala de grises queda determinada por las tuplas que tienen sus tres componentes iguales. Por ejemplo $(0,0,0)$ representa el color negro, $(255,255,255)$ representa el color blanco y $(128,128,128)$ representa un color gris equidistante del negro y del blanco. Por tanto, para imágenes en escala de grises solo es necesario almacenar un valor de intensidad por píxel en lugar de tres ya que las componentes de la tupla $(r,g,b)$ asociada al píxel son iguales. Cuando se trabaja con imágenes en escala de grises es común emplear el intervalo $[0,1]$ como escala de medida para los valores $r$, $g$ y $b$.

## 1.3. Representación de imágenes digitales en escala de grises

Para convertir imágenes a color en imágenes en escala de grises se asocia a cada píxel el valor de intensidad que se obtiene al hacer una media ponderada de los valores de intensidad asociados al píxel en la imagen a color. Si en la imagen a color un píxel tiene asociada una tupla $(r,g,b)$, en la imagen en escala de grises ese mismo píxel tendrá asociado como valor de intensidad en el intervalo $[0,1]$ el valor

$$\frac{0{,}299 \cdot r + 0{,}587 \cdot g + 0{,}114 \cdot b}{255} \tag{1.1}$$

La razón de utilizar esta ponderación de los canales del sistema *RGB*, aunque existen otras, reside en motivos biológicos del ojo humano ya que los receptores de luz de los ojos no son igual de sensibles a los tres colores primarios.

Por tanto, si se tiene almacenada una imagen digital a color como un conjunto de tres matrices (una por cada uno de los canales del sistema *RGB*), para transformar esa imagen a escala de grises, basta hacer una media ponderada de las tres matrices con los coeficientes indicados en (1.1). La matriz final será la representación de la imagen en escala de grises.

La figura 1.3 muestra a a la izquierda una imagen a color de la Catedral de León y a la derecha su transformación en escala de grises.

## 1.4. Tratamiento de imágenes digitales con ordenador

Para el tratamiento de imágenes digitales y la creación de mosaicos se ha decidido utilizar el lenguaje de programación Python. Los principales motivos de esta elección son la versatilidad del lenguaje y


Figura 1.3: Imagen de la Catedral de León a color junto a su transformación en escala de grises.

su legibilidad. Se han utilizado principalmente tres librerías: *scikit-image*, para la lectura de imágenes; *numpy*, para el tratamiento de dichas imágenes; y *matplotlib* para representaciones gráficas.

Para la lectura de imágenes se emplea la función *imread* del módulo *io* de la librería *scikit-image*. Esta función permite convertir una imagen en un array tridimensional de *numpy* donde se almacenan las tuplas $(r,g,b)$ asociadas a los píxeles de la imagen. El array tridimensional es en esencia un conjunto de tres matrices donde se almacena la información de los tres canales del sistema *RGB*.

Para transformar una imagen a color en una imagen en escala de grises se utiliza una función propia que hace la ponderación de la fórmula (1.1) con las tres matrices que contienen la información de la imagen. El código de esta función se muestra en el Anexo A.

Por último, para la creación de mosaicos se han programado una serie de funciones que extraen de las imágenes los datos necesarios para resolver los modelos de optimización y construyen los mosaicos a partir de los soluciones óptimas obtenidas. El código de estas funciones también se muestra en el Anexo A.

# Capítulo 2

# Optimización lineal entera

## 2.1. Optimización

Un problema de optimización de máximo se formula como

$$\begin{aligned} &\max && f(X) && (2.1a)\\ &\text{sujeto a} && && \\ & && X \in S && (2.1b) \end{aligned}$$

donde $X$ es el vector $n \times 1$ de *variables de decisión*, $f$ es la *función objetivo* a optimizar y $S \subset \mathbb{R}^n$ es la *región factible* del problema, es decir, el conjunto de vectores $X$ que satisfacen todas las restricciones del problema. Los vectores de la región factible $S$ se denominan *soluciones factibles*. En caso de que la región factible sea el conjunto vacío, se dice que el problema es *no factible*.

Dependiendo de las características de la función objetivo y de la región factible se tendrá un tipo de problema de optimización u otro. En particular, si la función objetivo $f$ es una función lineal y la región factible $S$ viene definida por un conjunto de restricciones también lineales se tiene un problema de optimización lineal. Este es probablemente uno de los problemas de optimización con más aplicaciones y que ha sido estudiado más en profundidad por sus buenas propiedades teóricas que se traducen en métodos muy eficientes de resolución. Un problema de optimización lineal se puede formular sin pérdida de generalidad como

$$\begin{aligned} &\max && cX && (2.2a)\\ &\text{sujeto a} && && \\ & && AX \leqslant b && (2.2b)\\ & && X \geqslant 0_n && (2.2c) \end{aligned}$$

donde $A$ es una matriz $m \times n$, $b$ es un vector $m \times 1$, $c$ es un vector $1 \times n$, $0_n$ es un vector $n \times 1$ de ceros y $X \geqslant 0_n$ indica que todas las componentes del vector $X$ son no negativas. Cuando las restricciones (2.2b) son igualdades se dice que el problema está en forma estándar.

La región factible del problema (2.2) es un poliedro. Basándose en este hecho, el siguiente teorema, cuya demostración puede verse en [2], identifica dónde se encuentra una solución óptima del problema, si la hay.

**Teorema 2.1.1.** *Sea un problema de optimización lineal en forma estándar tal que el rango de A es m y en el que la región factible es no vacía. Sean $X_1, \dots X_k$ los puntos extremos y $d_1, \dots, d_h$ las direcciones extremas de la región factible. Una condición necesaria y suficiente para la existencia de una solución óptima finita del problema es*

$$cd_i \leqslant 0, \quad i = 1, \dots, h.$$

*Si se cumple esta condición, entonces existe un punto extremo que es solución óptima del problema.*

Teniendo en cuenta este teorema, se han podido desarrollar diferentes algoritmos de resolución de problemas de optimización lineal. Probablemente el algoritmo más conocido y utilizado sea el *algoritmo simplex* desarrollado por George Dantzig en 1947. Este algoritmo recorre puntos extremos de la región factible de manera que el valor de la función objetivo no empeora en las sucesivas iteraciones hasta que se alcanza un punto extremo óptimo o se encuentra una dirección extrema para la que no se verifica la condición del teorema 2.1.1.

## 2.2. Optimización lineal entera

Cuando en el problema (2.2) todas las variables o parte de ellas están restringidas a tomar valores enteros se tiene un problema de optimización lineal entera. Si todas las variables del problema son enteras se tiene un *problema lineal entero puro* (*PLEP*). Si todas las variables del problema han de tomar valores 0 o 1 se tiene un *problema lineal entero binario* (*PLEB*). Finalmente, si solo se requiere que algunas de las variables tomen valores enteros se tiene un *problema lineal entero mixto* (*PLEM*).

Un PLEM se formula sin pérdida de generalidad como

$$\begin{aligned} \max \quad & cX + hY && \text{(2.3a)} \\ \text{sujeto a} \quad & \\ & AX + GY \leqslant b && \text{(2.3b)} \\ & X \geqslant 0_n && \text{(2.3c)} \\ & Y \in \mathbb{Z}_+^p && \text{(2.3d)} \end{aligned}$$

donde $A$ y $G$ son matrices $m \times n$ y $m \times p$ respectivamente, $b$ es un vector $m \times 1$, $c$ es un vector $1 \times n$, $h$ es un vector $1 \times p$, $X$ es el vector $n \times 1$ de variables de decisión continuas e $Y$ es el vector $p \times 1$ de variables de decisión enteras. Se denota con $Z$ al valor óptimo de la función objetivo del problema (2.3).

La región factible del problema (2.3)

$$S = \left\{(X,Y) \in \mathbb{R}_+^n \times \mathbb{Z}_+^p \mid AX + GY \leqslant b\right\}$$

ya no es un poliedro. En consecuencia, los problemas de optimización entera son, en general, más complicados de resolver y requieren técnicas más sofisticadas de resolución que los problemas de optimización lineal ya que, al contrario que para estos últimos, no se dispone de un teorema semejante al teorema 2.1.1 que de una idea sobre cómo deberían recorrerse las soluciones factibles para llegar a la solución óptima.

Hay dos aspectos que hacen más complicada la resolución de este tipo de problemas en comparación con la resolución de problemas de optimización lineal. El primer aspecto a considerar es la formulación del problema. Supóngase, por ejemplo, que se tiene un *PLEP* cuya región factible es

$$S = \{(1,2),(2,1),(2,2),(2,3),(3,1),(3,2),(4,1),(4,2)\}$$

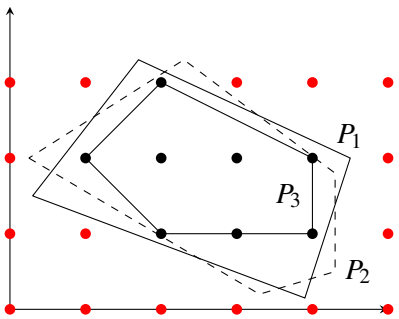



Figura 2.1: Tres formulaciones diferentes para el conjunto $S$.

Los poliedros $P_1$, $P_2$ y $P_3$ representados en la figura 2.1 contienen solo las soluciones enteras de $S$ pero no son igualmente adecuados para resolver el *PLEP*. En este caso, el poliedro $P_3$ puede considerarse

el poliedro ideal ya que al resolver el *PLEP* considerado sustituyendo la región de factibilidad por el poliedro $P_3$, su solución será un punto extremo. Como todos los puntos extremos son enteros, se habrá resuelto el *PLEP* resolviendo un problema de optimización lineal. No obstante no hay una caracterización simple de las inecuaciones que permiten definir el poliedro $P_3$ en un caso general [4].

El segundo aspecto a considerar es el procedimiento elegido para la resolución del problema. Si todas las variables del problema son enteras, puede pensarse en resolver el *PLEP* haciendo una *enumeración explícita* de todas las soluciones factibles del problema (en caso de que el número de soluciones sea finito), evaluando su función objetivo y seleccionando la solución que proporciona el mejor valor de la función objetivo. Aunque el número de soluciones sea finito, al crecer el número de variables involucradas en el problema, el número de soluciones factibles aumenta rápidamente y en la práctica resulta imposible examinarlas todas, por lo que el procedimiento no es viable. En la literatura se han desarrollado procedimientos generales de los que en esta memoria se estudiará el *método de ramificación y acotación*. Además, se han desarrollado procedimientos específicos para problemas particulares. Las referencias [3, 4, 5] dan una idea general de los procedimientos desarrollados.

## 2.3. Relajación lineal

Los métodos de resolución de problemas de optimización lineal entera están basados, en general, en la resolución de problemas de optimización lineal relacionados [3, 4]. Una de las formas más habituales de obtener un problema de optimización lineal a partir de un problema de optimización lineal entera es lo que se conoce como *relajación lineal* que se define a continuación.

**Definición 2.3.1.** Dado el *PLEM* formulado en (2.3), se llama *relajación lineal* o *problema lineal relajado* (*PLR*) al problema de optimización lineal que se obtiene al eliminar las restricciones de integridad sobre las variables.

$$\begin{aligned} &\max && cX + hY && (2.4a)\\ &\text{sujeto a} && && \\ &&& AX + GY \leqslant b && (2.4b)\\ &&& X \geqslant 0_n && (2.4c)\\ &&& Y \geqslant 0_p && (2.4d) \end{aligned}$$

La región factible del problema (2.4) es

$$S_R = \left\{(X,Y) \in \mathbb{R}^n_+ \times \mathbb{R}^p_+ \mid AX + GY \leqslant b\right\}$$

Se denota por $Z^R$ el valor óptimo de la función objetivo del *PLR*.

Nótese que $S^R$ es un poliedro que contiene a $S$ ya que se han añadido a $S$ los puntos de $\mathbb{R}^n_+ \times \mathbb{R}^p_+$ que satisfacen las restricciones lineales $AX + GY \leqslant b$ que tienen alguna componente de $Y$ no entera.

**Proposición 2.3.2.** *Dado el PLEM formulado en (2.3) y su PLR formulado en (2.4). Se verifica que*

1. $Z \leqslant Z^R$.
2. *Si* $(X^*,Y^*)$ *es una solución óptima del PLR y* $(X^*,Y^*) \in S$*, entonces* $(X^*,Y^*)$ *es una solución óptima del PLEM.*
3. *Si el PLR es no factible, entonces el PLEM es no factible.*

*Demostración.*

1. Sea $(X^*,Y^*)$ una solución óptima del *PLEM*. Entonces, $(X^*,Y^*) \in S \subset S^R$ y $Z = cX^* + hY^*$. Como $(X^*,Y^*) \in S^R$, $cX^* + hY^*$ es una cota inferior de $Z^R$, luego $Z \leqslant Z^R$.

2. Como $(X^*,Y^*)$ es una solución óptima del *PLR*, $Z^R = cX^* + hY^*$. Como $(X^*,Y^*) \in S$, $cX^* + hY^*$ es una cota inferior de $Z$, es decir $Z^R \leqslant Z$. Como $Z \leqslant Z^R$ por el apartado anterior, $Z = Z^R$ y así $(X^*,Y^*)$ es una solución óptima del *PLEM*.

3. Si el *PLR* es no factible, $S^R = \emptyset$ y como $S \subset S^R$ también $S = \emptyset$, luego el *PLEM* es no factible.

□

## 2.4. Método de ramificación y acotación

La estrategia que sigue el método de ramificación y acotación para resolver problemas de optimización lineal entera es la que se conoce coloquialmente como *divide y vencerás*, que consiste en dividir un problema en subproblemas más pequeños cuyas soluciones sean más fáciles de obtener y a partir de ellas llegar a la solución del problema original. La idea de esta estrategia queda reflejada en la siguiente proposición.

**Proposición 2.4.1.** *Dado el PLEM formulado en (2.3). Sea $\{S_i\}_{i=1}^k$ una descomposición de la región factible S. Para $i = 1,\ldots,k$, sea el subproblema $PLEM_i$ formulado como:*

$$\begin{aligned} &\max && cX + hY && (2.5a)\\ &\text{sujeto a} && && \\ & && (X,Y) \in S_i && (2.5b) \end{aligned}$$

*Sea $Z_i$ el valor óptimo de la función objetivo del subproblema $PLEM_i$. Entonces $Z = \max\limits_{i=1,\ldots,k} Z_i$.*

*Demostración.* Para todo $i = 1,\ldots,k$ se tiene que $Z_i \leqslant Z$ ya que $S_i \subseteq S$ y se está maximizando la función objetivo. Sea $(X^*,Y^*)$ una solución óptima del *PLEM*, como $(X^*,Y^*) \in S$, existirá al menos un $i \in \{1,\ldots,k\}$ tal que $(X^*,Y^*) \in S_i$, por lo que $Z_i = Z$ para ese $i$. Tomando máximos a ambos lados de la igualdad se tiene el resultado. □

El problema lineal relajado del $PLEM_i$ formulado en (2.5) se denota $PLR_i$.

En general, para representar el método de ramificación y acotación se utiliza un árbol de enumeración. Cada hoja o nodo del árbol se corresponde con uno de los subproblemas en los que se ha dividido el problema original. El nodo raíz, que se corresponde con el *PLEM* original se coloca en la parte superior y el árbol de enumeración se va construyendo de manera descendente.

El método de ramificación y acotación resuelve sucesivamente subproblemas del problema original, contruyendo el arbol de enumeración y podando las ramas de éste que no presenten interés debido a que ninguna de las soluciones factibles que contienen puede ser la solución óptima del *PLEM*.

**Definición 2.4.2.** En las condiciones de la proposición 2.4.1, se dice que la rama asociada al conjunto $S_i$ se poda o que el subproblema $PLEM_i$ se cierra si no es necesario examinar el conjunto $S_i$ para encontrar la solución óptima del *PLEM*. Esto equivale a que se verifique alguna de las siguientes condiciones:

1. Se conoce una solución óptima del subroblema $PLEM_i$.

2. $Z_i \leqslant \widetilde{Z}$ siendo $\widetilde{Z}$ el valor de la función objetivo del *PLEM* en una solución factible $\left(\widetilde{X},\widetilde{Y}\right)$ conocida de éste.

3. El subproblema $PLEM_i$ es no factible.

**Proposición 2.4.3 (*Criterios de poda*).** *En las condiciones de la proposición 2.4.1, la rama asociada al conjunto $S_i$ se poda, o equivalentemente el problema $PLEM_i$ se cierra, si se verifica alguna de las siguientes condiciones:*

1. ***Poda por optimalidad**. Existe una solución óptima del $PLR_i$ que pertenece a $S_i$.*

2. ***Poda por acotación***. *Existe una solución óptima del* $PLR_i$ *y* $Z_i^R \leqslant \widetilde{Z}$ *siendo* $\widetilde{Z}$ *el valor de la función objetivo del PLEM en una solución factible* $\left(\widetilde{X},\widetilde{Y}\right)$ *conocida de éste.*

3. ***Poda por infactibilidad***. *El* $PLR_i$ *es no factible.*

*Demostración.*

1. Como existe una solución óptima del $PLR_i$ que pertenece a $S_i$, dicha solución también es una solución óptima del $PLEM_i$ por el apartado 1 de la proposición 2.3.2. Por tanto, la rama asociada al conjunto $S_i$ se poda según la definición 2.4.2.

2. Aplicando el apartado 2 de la proposición 2.3.2, se tiene la siguiente cadena de desigualdades, $Z_i \leqslant Z_i^R \leqslant \widetilde{Z} \leqslant Z$. Por tanto, $Z_i \leqslant Z$ y la rama asociada al conjunto $S_i$ se poda según la definición 2.4.2.

3. Si $PLR_i$ es no factible, entonces $PLEM_i$ también es no factible por el apartado 3 de la proposición 2.3.2. Por tanto, la rama asociada al conjunto $S_i$ se poda según la definición 2.4.2.

□

El algoritmo 1 describe las características generales de un algoritmo de ramificación y acotación. Entre los aspectos que hay que fijar está la forma en la que se selecciona el subproblema que se estudia en cada iteración y la forma en la que se descomponen los subproblemas. A continuación se describen algunos criterios para abordar estos problemas. A modo de ejemplo, en el anexo B se presenta la resolución de un *PLEB* aplicando diferentes criterios. Puede observarse que en un caso la solución óptima se obtiene tras explorar 13 subproblemas. Sin embargo, en el otro caso, en el que se ha cambiado la forma en la que se selecciona el subproblema a estudiar, basta con explorar 9 subproblemas.

### 2.4.1. Criterios de selección de subproblemas

La selección del siguiente subproblema a tratar de entre los subproblemas no explorados de la lista $\mathscr{L}$ se puede hacer atendiendo a diferentes criterios. Algunos de los mas utilizados en la práctica son:

1. Selección arbitraria de cualquiera de los subproblemas no explorados de la lista $\mathscr{L}$.

2. Criterio *LIFO (Last In First Out)*. Se selecciona el subproblema no explorado que ha entrado más tarde en la lista.

3. Criterio *FIFO (First In First Out)*. Se selecciona el subproblema no explorado que ha entrado más pronto en la lista.

4. Criterio *Best-Bound search*. Se selecciona el subproblema no explorado que tenga mayor cota superior.

   Con este criterio se consigue minimizar el número de subproblemas a explorar ya que nunca se ramificará a partir de un subproblema cuya cota superior sea menor que el valor óptimo del problema original.

En la práctica se suele utilizar una combinación de varios criterios. Lo más común es comenzar con una estrategia *Depth-First Search* atendiendo al criterio *LIFO* que consiste en descender lo más rápido posible en el árbol de enumeración de subproblemas con el fin de encontrar una solución factible que nos permita podar por acotación muchos de los subproblemas abiertos. Una vez encontrada una solución factible que permita realizar podas por acotación, se sigue con una estrategia *Best-Bound search* con el fin de realizar podas por optimalidad.

**Algoritmo 1:** Algoritmo de ramificación y acotación.

**Inicialización:**
$\overline{Z} \leftarrow +\infty$.
$\mathscr{L} = \{PLEM\}$.
$\underline{Z} = -\infty$.
$(X^*, Y^*) = \emptyset$.

**Paso 1: Finalización.**
  **si** $\mathscr{L} = \emptyset$ **entonces**
    **si** $(X^*, Y^*) = \emptyset$ **entonces**
      El *PLEM* es no factible.
    **en otro caso**
      $(X^*, Y^*)$ es una solución óptima del *PLEM*.

**Paso 2: Selección de subproblemas.**
  Se selecciona un subproblema $PLEM_i$ de la lista $\mathscr{L}$ con cota superior $\overline{Z_i}$.
  $\mathscr{L} \leftarrow \mathscr{L} \setminus \{PLEM_i\}$.
  **si** $\overline{Z_i} \leqslant \underline{Z}$ **entonces**
    Se poda el conjunto $S_i$ por acotación.
    **Ir al Paso 1.**

**Paso 3: Resolución de la relajación del subproblema seleccionado.**
  Se resuelve el $PLR_i$ y se obtiene, en caso de existir, una solución óptima $\left(X_i^R, Y_i^R\right)$.
  $\overline{Z_i} \leftarrow Z_i^R$.

**Paso 4: Aplicación de los criterios de poda.**
  **si** $PLR_i$ *es no factible* **entonces**
    $PLEM_i$ es también no factible.
    Se poda el conjunto $S_i$ por infactibilidad.
    **Ir al Paso 1.**
  **si no, si** $PLR_i$ *es factible* **y** $Z_i^R \leqslant \underline{Z}$ **entonces**
    Se poda el conjunto $S_i$ por acotación.
    **Ir al Paso 1.**
  **si no, si** $PLR_i$ *es factible* **y** $\left(X_i^R, Y_i^R\right) \in S_i$ **entonces**
    Se poda el conjunto $S_i$ por optimalidad.
    $\underline{Z} \leftarrow Z_i^R$.
    $(X^*, Y^*) \leftarrow \left(X_i^R, Y_i^R\right)$.
    **Ir al Paso 1.**
  **en otro caso**
    **Ir al Paso 5.**

**Paso 5: Ramificación.**
  **si** $PLR_i$ *es factible* **y** $\left(X_i^R, Y_i^R\right) \notin S_i$ **y** $Z_i^R > \underline{Z}$ **entonces**
    Descomponer el conjunto $S_i$ en k subconjuntos $\left\{S_{ij}\right\}_{j=1}^{k}$.
    Añadir los subproblemas $\left\{PLEM_{ij}\right\}_{j=1}^{k}$ a $\mathscr{L}$ con $\overline{Z_{ij}} = Z_i^R$ para todo $j = 1, \ldots, k$.
    **Ir al Paso 1.**

### 2.4.2. Criterios de ramificación

En caso de que sea necesario hacer una ramificación hay que descomponer la región factible $S$ del problema que se esté considerando en $k$ subregiones $\{S_i\}_{i=1}^{k}$ y estudiar por separado los subproblemas generados por cada uno de los $k$ elementos de la descomposición.

Existen varias formas de descomponer la región factible $S$. Una de las más utilizadas es elegir una variable $y_j$ del problema que debe tomar valor entero pero toma valor no entero en la solución óptima actual y dividir $S$ en dos subregiones $S_1$ y $S_2$. Sea $y_j^*$ el valor óptimo no entero de la variable $y_j$ en la solución actual del problema considerado,

$$S_1 = S \cap \left\{X \in S \mid y_j \leqslant \lfloor y_j^* \rfloor\right\}$$
$$S_2 = S \cap \left\{X \in S \mid y_j \geqslant \lceil y_j^* \rceil\right\}$$

Notemos que esto elimina del estudio la solución $(X^*, Y^*)$ y los valores de $y_j$ tales que $\lfloor y_j^* \rfloor < y_j < \lceil y_j^* \rceil$. Sin embargo, no se elimina ninguna solución factible en la que $y_j$ sea entera.

En general, pueden existir varias variables $y_j$ que deban tomar valor entero pero tomen valor no entero en la solución actual y por tanto se debe tener un criterio con el que elegir a partir de qué variable se va a hacer la ramificación. Algunos criterios que se usan en la práctica son:

1. Elección arbitraria de cualquiera de las variables de la solución actual que debe tomar valor entero y que toma valor no entero.

2. Elección de la variable más fraccionaria. Se escoge la variable

$$\arg\max_{j \in C} \min\left\{y_j^* - \lfloor y_j^* \rfloor, \lceil y_j^* \rceil - y_j^*\right\}$$

   donde $C = \left\{j \in \{1, \ldots, n\} : y_j^* \notin \mathbb{Z}\right\}$.

3. Elección de una variable siguiendo ciertos criterios de prioridad sobre las variables establecidos antes de comenzar el proceso de resolución del problema.

### 2.4.3. Método de ramificación y acotación incompleto

En ocasiones, cuando se aplica el método de ramificación y acotación para resolver problemas de optimización lineal entera el árbol de enumeración es muy grande debido a las dimensiones de los problemas y esto conlleva que resolverlos hasta alcanzar la solución óptima requiera un gran tiempo de cálculo. En estos casos el árbol de enumeración se trunca y se considera como solución óptima la mejor solución factible encontrada hasta el momento.

Para evaluar la calidad de la solución obtenida se utiliza una magnitud llamada *"duality gap"* definida como:

$$\frac{Best\overline{Z} - \widetilde{Z}}{\widetilde{Z}} \cdot 100 \tag{2.6}$$

donde $\widetilde{Z}$ es el valor de la función objetivo en la mejor solución factible encontrada hasta el momento y $Best\overline{Z}$ es la mejor de las cotas superiores obtenidas.

Los criterios de selección de subproblemas y de selección de variables para ramificar cobran mucha importancia en estos casos ya que de ellos dependerá la calidad de la solución obtenida.

## 2.5. Desigualdades válidas

El algoritmo de ramificación y acotación es capaz de resolver cualquier *PLEM* sea cual sea su formulación. Sin embargo, cuando el número de variables enteras del problema crece, el número de nodos del árbol de enumeración aumenta exponencialemente, lo que conlleva un mayor tiempo de ejecución del algoritmo.

El número de nodos del árbol de enumeración del método de ramificación y acotación depende en gran medida de la formulación utilizada. Es crucial por tanto encontrar una buena formulación del problema.

Cualquier *PLEM* puede resolverse como un problema de optimización lineal usando como formulación del problema aquella en la que la región de factibilidad es la envolvente convexa de la región factible del *PLEM*. El uso de esta estrategia presenta principalmente dos problemas:

1. La descripción explícita de la envolvente convexa de $S$ mediante un conjunto de desigualdades lineales no se conoce en general y encontrarla puede suponer un problema tan complejo como resolver el *PLEM*.

2. Para la mayoría de *PLEM*, la descripción explícita de la envolvente convexa de su región factible involucra un gran número de desigualdades lineales lo que en ocasiones provoca que no sea posible resolver el *PLR* o que sea muy costoso en cuanto a tiempo de cálculo.

Debido a estos dos problemas no siempre es posible utilizar la estrategia de encontrar la formulación ideal del problema. Pero lo que sí es posible es hacer una reformulación parcial del problema añadiendo lo que se conoce como *desigualdades válidas* con el propósito de obtener una formulación lo más cercana posible a la envolvente convexa de la región factible del *PLEM* que se esté considerando.

**Definición 2.5.1.** Dado el *PLEM* formulado en (2.3), una *desigualdad válida* para $S$ es una desigualdad de la forma $\alpha^\top X + \beta^\top Y \leqslant \gamma$, con $\alpha \in \mathbb{R}^n$, $\beta \in \mathbb{R}^p$ y $\gamma \in \mathbb{R}$, que satisfacen todos los puntos de $S$.

El objetivo de introducir desigualdades válidas es eliminar secciones de la región factible del *PLR* para hacer más fácil la resolución del *PLEM*. Existen diferentes procedimientos para construir desigualdades válidas. Uno de los más conocidos es el procedimiento de *Chvátal-Gomory*. Aplicando este procedimiento un número finito de veces se pueden construir todas las desigualdades válidas para un conjunto arbitrario [4].

Si se hace una reformulación de un *PLEM* a partir de una familia de desigualdades válidas antes de aplicar el método de ramificación y acotación, el rendimiento de este método mejora en gran medida ya que, si las desigualdades válidas elegidas son adecuadas, las cotas obtenidas a lo largo del algoritmo son mejores y por tanto el número de nodos del árbol de enumeración disminuye. En la práctica, esta estrategia resulta eficiente si el número de desigualdades válidas es moderado ya que si se añaden muchas desigualdades válidas el tamaño del problema crece tanto que los algoritmos empiezan a presentar problemas en cuanto a tiempo de ejecución.

# Capítulo 3

# Modelos de optimización lineal entera para la construcción de mosaicos

## 3.1. Introducción

El objetivo de este capítulo es presentar la formulación de algunos modelos de optimización lineal entera para construir mosaicos con distintos tipos de teselas así como los resultados obtenidos a partir de ellos. En primer lugar se estudia la forma de la función objetivo, que es común a todos los modelos planteados. Los apartados siguientes muestran las restricciones que pueden tener interés según el tipo de tesela. Hay que notar que cuantas más restricciones se añadan al problema, en general, peor será el valor de la función objetivo y por tanto cabe esperarse un "menor parecido" entre la imagen original y el mosaico. Así mismo, cuantas más variables se incluyan, en general, "mayor parecido" habrá con la imagen original. Para la resolución de los modelos de optimización formulados en esta memoria se ha utilizado la versión 20.1.0 de IBM ILOG CPLEX Optimization Studio con las configuraciones que vienen por defecto. Los modelos propuestos en las secciones 3.3 y 3.4 han sido resueltos en un ordenador PC Intel Core i7-4500U con 1.8 GHz, 8.00 GB de RAM y Windows 10 64-bits como sistema operativo. y los modelos propuestos en la sección 3.5 se han resuelto en un PC Intel Core i7-6700 con 3.4 GHz, 32.0 GB de RAM y Windows 10 64-bits como sistema operativo. El código CPLEX programado para resolver algunos de los modelos formulados puede verse en el anexo C.

## 3.2. Función objetivo

Dada una imagen en escala de grises, se quiere crear un mosaico a partir de ella utilizando un conjunto finito de $t$ teselas cuadradas diferentes. Para ello es necesario dividir la imagen en $n \times m$ bloques de la misma forma y tamaño que las teselas y sustituir cada uno de los bloques por una copia de una tesela determinada. El problema que se plantea es elegir, para cada uno de los bloques en los que se ha dividido la imagen, qué tesela sustituye cada bloque.

Para modelar este problema de decisión se definen para cada $i = 1,\ldots,n$, $j = 1,\ldots,m$ y $c = 1,\ldots,t$ las variables binarias $x_{cij}$ de la siguiente manera:

$$x_{cij} = \begin{cases} 1 & \text{si se coloca una copia de la tesela } c \text{ en el bloque } (i,j)\,. \\ 0 & \text{en caso contrario.} \end{cases} \tag{3.1}$$

A continuación, hay que "medir" la calidad del mosaico, para lo que se necesita una magnitud que evalúe si una tesela es adecuada para sustituir un bloque o no. Esta magnitud es lo que de aquí en adelante se llama brillo del bloque o brillo de la tesela. Como la imagen y las teselas con las que se trabaja están en escala de grises, cada píxel de la imagen o de las teselas tiene asociado un único valor de intensidad. El brillo del bloque $(i,j)$, denotado por $\beta_{ij}$,se calcula como la media aritmética de los valores de intensidad

de los píxeles que pertenezcan a ese bloque. Análogamente, el brillo de la tesela $c$, denotado por $\beta_c$, se calcula como la media aritmética de los valores de intensidad de los píxeles que forman dicha tesela.

El objetivo es que el mosaico generado se parezca lo más posible a la imagen, es decir, que el brillo de cada bloque difiera lo menos posible del brillo de la tesela por la que se va a sustituir. En esta memoria se utilizan como medida de la desviación entre brillos el cuadrado de la diferencia entre el brillo de la tesela y el brillo del bloque [1] y el valor absoluto de esta diferencia. Por tanto, las funciones objetivo a optimizar son:

$$\sum_{c=1}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m}(\beta_c-\beta_{ij})^2 x_{cij} \tag{3.2}$$

$$\sum_{c=1}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m}|\beta_c-\beta_{ij}| x_{cij} \tag{3.3}$$

En las siguientes secciones se van a presentar varios modelos que permiten la construcción de diferentes mosaicos que varían en función de las características de las teselas y de las restricciones impuestas en cada modelo. La imagen con la que se van a construir los mosaicos corresponde al autorretrato del pintor zaragozano Francisco de Goya y Lucientes que se muestra en la figura 3.1. Otras imágenes y sus correspondientes mosaicos pueden verse en el anexo D. En este anexo se incluyen también los mosaicos obtenidos con los modelos equivalentes a los desarrollados en los apartados siguientes con la función (3.2) usando la función objetivo (3.3) que, por razones de espacio, no es posible presentar en este capítulo.

Figura 3.1: Autorretrato del pintor Francisco de Goya y Lucientes perteneciente a su colección de grabados llamada *Caprichos* publicada en 1799. Fuente: [8]. Dimensiones de la imagen: $4608 \times 3456$ píxeles.

## 3.3. Construción de mosaicos con teselas de motivos estampados

Para la construcción de los mosaicos en esta sección, se dispone de teselas en escala de grises decoradas con algún motivo estampado, que se pueden ordenar según su brillo en una escala del 0 al 1 donde el 0 se corresponde con el color negro y el 1 se corresponde con el color blanco.

El conjunto de teselas que se va a utilizar ha sido creado por la diseñadora zaragozana Marta Longas Clemente y su temática es “Artistas contemporáneos de los siglos XIX y XX”. Consiste en una serie de doce retratos de artistas destacados de esa época (seis hombres y seis mujeres). En la figura 3.2 se muestra este conjunto de teselas ordenadas según su brillo, de más oscura a más clara.

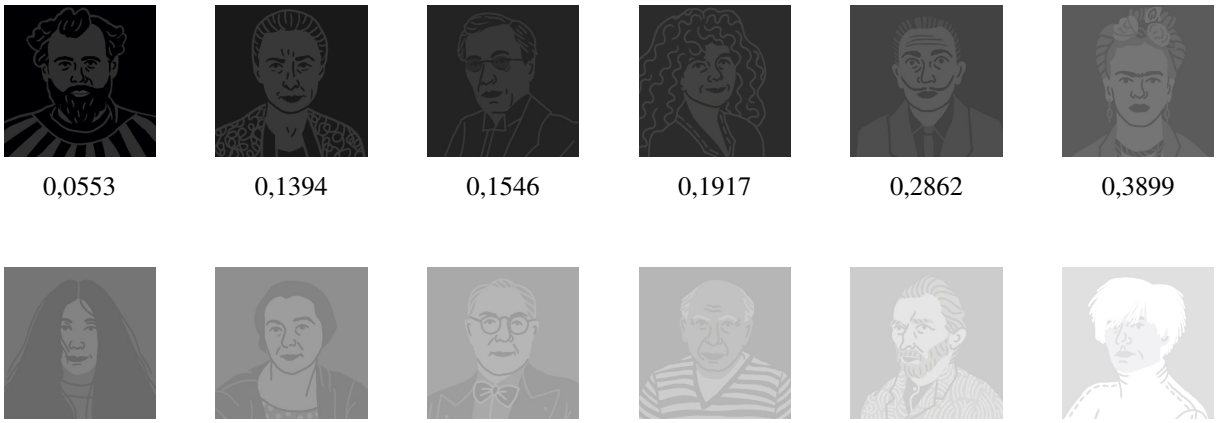



Figura 3.2: De izquierda a derecha y de arriba hacia abajo aparecen los artistas contemporáneos Gustav Klimt, Georgia O'Keeffe, Wassily Kandinsky, Barbara Kruger, Salvador Dalí, Frida Kahlo, Yoko Ono, Sonia Delaunay, Henri Matisse, Pablo Picasso, Vincent van Gogh y Andy Warhol. El número debajo de cada tesela indica su valor de brillo.

### 3.3.1. Modelo sin restricciones sobre las teselas

Si no se incluyen restricciones sobre las teselas que se pueden utilizar, se construye un mosaico en el que cada bloque es sustituido por la tesela cuyo brillo se asemeje más al valor de brillo del bloque. La única restricción que hay que imponer de forma natural es la que asegura que cada bloque es sustituido por una única tesela.

La formulación del modelo es:

$$\text{min} \quad \sum_{c=1}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m}(\beta_c-\beta_{ij})^2 x_{cij} \tag{3.4a}$$

$$\text{sujeto a} \quad \sum_{c=1}^{t} x_{cij} = 1 \qquad i=1,\dots,n \qquad j=1,\dots,m \tag{3.4b}$$

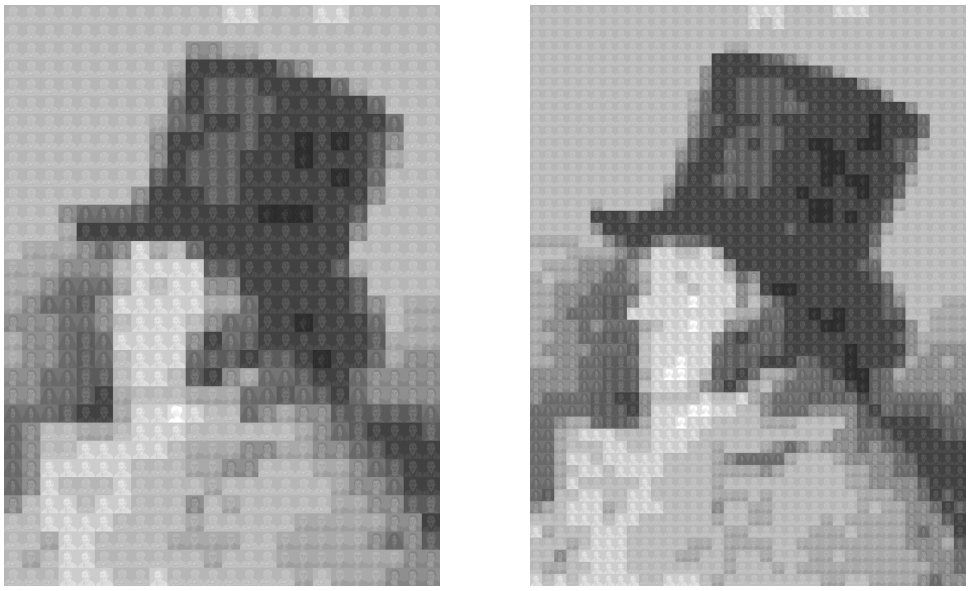
$$x_{cij} \in \{0,1\} \qquad c=1,\dots,t \qquad i=1,\dots,n \qquad j=1,\dots,m \tag{3.4c}$$

La expresión (3.4a) muestra la función objetivo. Las restricciones (3.4b) garantizan que cada bloque es sustituido por una tesela. Las restricciones (3.4c) exigen que las variables sean binarias. En la figura 3.3 se muestran dos ejemplos de mosaicos construidos a partir de este modelo, así como las características de las teselas, el tiempo que ha costado obtener una solución óptima del modelo y el valor de la función objetivo en dicha solución. Cuando las teselas empleadas para la construcción del mosaico tienen unas dimensiones de $144 \times 144$ píxeles, el modelo tiene 9216 variables y el mosaico tiene unas dimensiones de $32 \times 24$ teselas. Cuando las teselas tienen unas dimensiones de $96 \times 96$ píxeles, el modelo tiene 20736 variables y el mosaico tiene unas dimensiones de $48 \times 36$ teselas.

### 3.3.2. Modelo con conjuntos completos de teselas

Con el modelo (3.4) puede ocurrir, por ejemplo, que si se tienen 12 teselas diferentes para construir el mosaico numeradas del 1 al 12 se utilicen muchas copias de la tesela número 2 y muy pocas de la número 5 si en la imagen hay muchos bloques cuyo brillo se asemeje al brillo de la tesela número 2 y hay pocos bloques cuyo brillo se asemeje al brillo de la tesela número 5. Si para la construcción del mosaico se quiere conseguir que se utilicen conjuntos de teselas completos, es decir que cada tesela sea usada un mismo número de veces $N \in \mathbb{N}$ hay que añadir al conjunto de restricciones del modelo (3.4) las restricciones

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{cij} = N \quad c=1,\dots,t$$



(a) Teselas de tamaño $144 \times 144$ píxeles. Resolver el modelo ha costado 53 centésimas de segundo y el valor de la función objetivo ha sido 0,583.

(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 1 segundo y 87 centésimas de segundo y el valor de la función objetivo ha sido 1,287.

Figura 3.3: Mosaicos construidos a partir del modelo (3.4).

En este caso, para que el modelo sea factible, es necesario que el número de bloques en que se divide la imagen sea múltiplo del número de teselas diferentes disponibles para la construcción del mosaico.

La formulación del modelo es:

$$\text{min} \quad \sum_{c=1}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m}(\beta_c-\beta_{ij})^2 x_{cij} \tag{3.5a}$$

$$\text{sujeto a} \quad \sum_{c=1}^{t} x_{cij} = 1 \qquad i=1,\dots,n \qquad j=1,\dots,m \tag{3.5b}$$

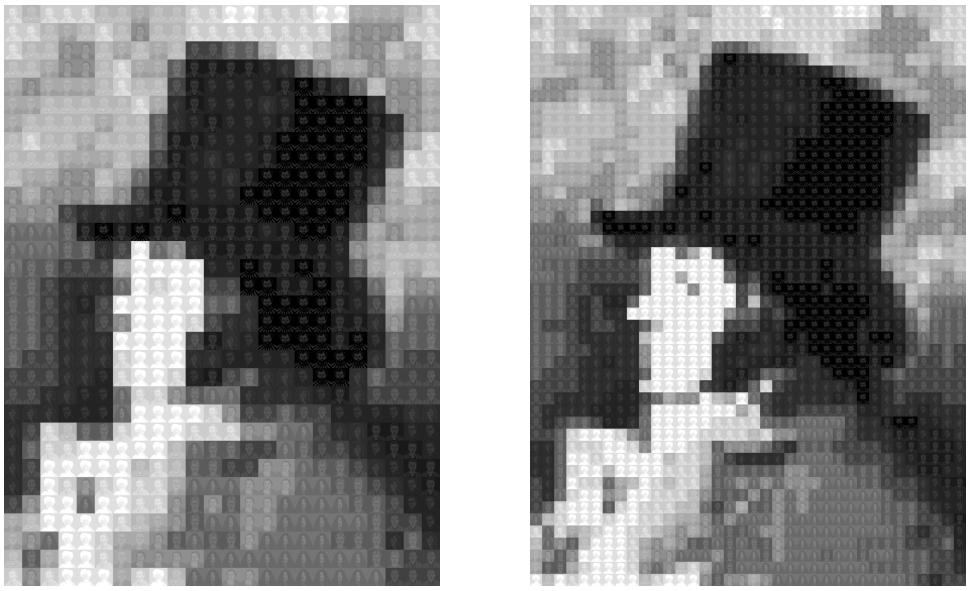
$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{cij} = N \qquad c=1,\dots,t \tag{3.5c}$$

$$x_{cij} \in \{0,1\} \qquad c=1,\dots,t \qquad i=1,\dots,n \qquad j=1,\dots,m \tag{3.5d}$$

La función objetivo (3.5a) y las restricciones (3.5b) y (3.5d) han sido explicadas en el modelo (3.4). Las restricciones (3.5c) garantizan el uso de conjuntos completos de teselas. En la figura 3.4 se muestran dos ejemplos de mosaicos construidos a partir de este modelo, así como sus características.

### 3.3.3. Modelo sin agrupamientos de teselas iguales

El modelo (3.5) permite que en determinadas zonas del mosaico se concentren muchas teselas del mismo tipo, si los bloques de esa zona tienen valores de brillo muy similares entre sí. Para evitar este fenómeno, hay que imponer restricciones al modelo que aseguren que ninguna tesela del mosaico esté en contacto con otra tesela del mismo tipo.



(a) Teselas de tamaño $144 \times 144$ píxeles. Resolver el modelo ha costado 1 segundo y 13 centésimas de segundo y el valor de la función objetivo has sido 26,931.

(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 1 segundo y 91 centésimas de segundo y el valor de la función objetivo ha sido 59,613.

Figura 3.4: Mosaicos construidos a partir del modelo (3.5).

La formulación del modelo es:

$$\min \quad \sum_{c=1}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m}(\beta_c-\beta_{ij})^2 x_{cij} \tag{3.6a}$$

$$\text{sujeto a} \quad \sum_{c=1}^{t} x_{cij} = 1 \qquad i=1,\dots,n \quad j=1,\dots,m \tag{3.6b}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{cij} = N \qquad c=1,\dots,t \tag{3.6c}$$

$$x_{cij} + x_{ci+1j} \leq 1 \qquad c=1,\dots,t \quad i=1,\dots,n-1 \quad j=1,\dots,m \tag{3.6d}$$

$$x_{cij} + x_{cij+1} \leq 1 \qquad c=1,\dots,t \quad i=1,\dots,n \quad j=1,\dots,m-1 \tag{3.6e}$$

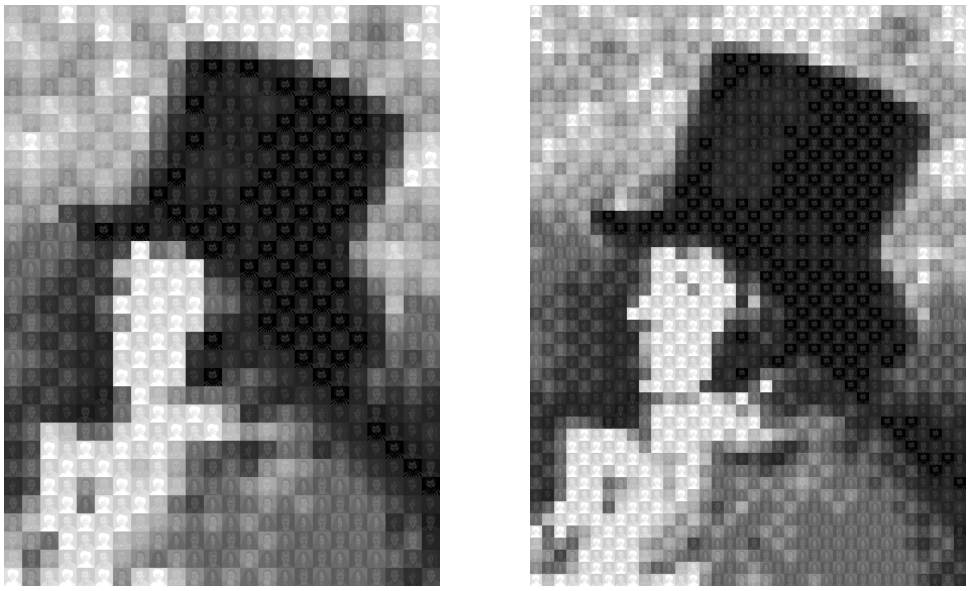
$$x_{cij} \in \{0,1\} \qquad c=1,\dots,t \quad i=1,\dots,n \quad j=1,\dots,m \tag{3.6f}$$

La función objetivo (3.6a) y las restricciones (3.6b), (3.6c) y (3.6f) han sido explicadas en el modelo (3.5). Las restricciones (3.6d) y (3.6e) garantizan que ninguna tesela está en contacto con otra igual verticalmente u horizontalmente, respectivamente. En la figura 3.5 se muestran dos ejemplos de mosaicos construidos a partir de este modelo, así como sus características.

## 3.4. Construción de mosaicos con teselas que siguen patrones

En esta sección, para la construcción de los mosaicos se dispone de un conjunto de teselas en escala de grises decoradas con motivos geométricos de tal manera que puedan encajar unas con otras en el sentido de que cada tesela solo admite como teselas adyacentes a un subconjunto del conjunto total de teselas.

Dadas las dos teselas cuadradas que se muestran en la figura 3.6, en las que dos cuartos de octógono dividen el cuadrado en tres zonas, es posible crear conjuntos de teselas. Así, utilizando $n$ tonos equidistantes en la escala de grises (incluyendo el blanco y el negro) para colorear las tres zonas en las que están



(a) Teselas de tamaño $144 \times 144$ píxeles. Resolver el modelo ha costado 1 segundo y 53 centésimas de segundo y el valor de la función objetivo ha sido 27,839.

(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 3 segundos y 73 centésimas de segundo y el valor de la función objetivo ha sido 61,809.

Figura 3.5: Mosaicos construidos a partir del modelo (3.6).

divididas las teselas, con la condición de que dos zonas adyacentes no tengan el mismo color, es posible crear $t = 2n(n-1)^2$ teselas, que se numerarán de 1 a $t$ para identificarlas más fácilmente.

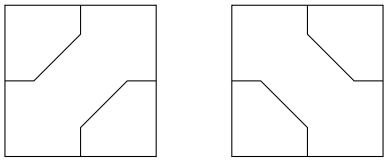


Figura 3.6: Teselas.

Si se denota por $N, G_1, \ldots, G_{n-2}, B$ los $n$ colores disponibles para colorear las teselas donde $N$ denota el negro, $B$ denota el blanco y $G_1, \ldots, G_{n-2}$ denotan los $n-2$ tonos de gris intermedios, se calcula el valor de brillo $\beta_c$ de una tesela cualquiera $c$ según la fórmula:

$$\beta_c = \frac{1}{A_T}\left(0 \cdot A_N + \frac{1}{n-1} \cdot A_{G_1} + \ldots + \frac{n-2}{n-1} \cdot A_{G_{n-2}} + 1 \cdot A_B\right)$$

entendiéndose que $A_T$ es el área total de la tesela y $A_C$ es el área de la tesela coloreada con el color $C$ donde $C \in \{N, G_1, \ldots, G_{n-2}, B\}$.

Cada uno de los lados, superior, inferior, derecho e izquierdo, de una tesela tiene asociado un patrón de colores que se codifica por una 2-tupla cuyas componentes son los dos tonos de color de las zonas de la tesela que tocan dicho lado siempre vistos de izquierda a derecha o de arriba a abajo. Como se dispone de $n$ colores y las zonas adyacentes de una tesela nunca tienen el mismo color, el número de patrones posibles para los lados de las teselas son las variaciones sin repetición de $n$ elementos tomados de dos en dos, es decir $n(n-1)$. Para poder identificar fácilmente los patrones, estos se numerarán desde 1 hasta $s = n(n-1)$.

Para modelar el hecho de que las teselas encajen unas con otras es necesario imponer que si se coloca una tesela $c$ en un determinado bloque $(i, j)$, el patrón de su lado superior ha de coincidir con el patrón del lado inferior de la tesela colocada en el bloque inmediatamente superior, que el patrón de su lado inferior ha de coincidir con el patrón del lado superior de la tesela colocada en el bloque inmediatamente

inferior, que el patrón de su lado derecho ha de coincidir con el patrón del lado izquierdo de la tesela colocada en el bloque inmediatamente a la derecha y que el patrón de su lado izquierdo ha de coincidir con el patrón del lado derecho de la tesela colocada en el bloque inmediatamente a la izquierda.

Se definen, para cada tesela $c = 1,\ldots,t$ y para cada patrón $p = 1,\ldots,s$, los parámetros $l_{cp}, r_{cp}, t_{cp}, b_{cp}$ como sigue:

$$l_{cp} = \begin{cases} 1 & \text{si la tesela } c \text{ tiene el patrón } p \text{ en su lado izquierdo.} \\ 0 & \text{en caso contrario.} \end{cases}$$

$$r_{cp} = \begin{cases} 1 & \text{si la tesela } c \text{ tiene el patrón } p \text{ en su lado derecho.} \\ 0 & \text{en caso contrario.} \end{cases}$$

$$t_{cp} = \begin{cases} 1 & \text{si la tesela } c \text{ tiene el patrón } p \text{ en su lado superior.} \\ 0 & \text{en caso contrario.} \end{cases}$$

$$b_{cp} = \begin{cases} 1 & \text{si la tesela } c \text{ tiene el patrón } p \text{ en su lado inferior.} \\ 0 & \text{en caso contrario.} \end{cases}$$

La formulación del modelo es:

$$\min \quad \sum_{c=1}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m}(\beta_c - \beta_{ij})^2 x_{cij} \tag{3.7a}$$

$$\text{sujeto a} \quad \sum_{c=1}^{t} x_{cij} = 1 \qquad i = 1,\ldots,n \qquad j = 1,\ldots,m \tag{3.7b}$$

$$\sum_{c=1}^{t} r_{cp}\cdot x_{cij} = \sum_{c=1}^{t} l_{cp}\cdot x_{cij+1} \qquad i = 1,\ldots,n \qquad j = 1,\ldots,m-1 \qquad p = 1,\ldots,s \tag{3.7c}$$

$$\sum_{c=1}^{t} b_{cp}\cdot x_{cij} = \sum_{c=1}^{t} t_{cp}\cdot x_{ci+1j} \qquad i = 1,\ldots,n-1 \qquad j = 1,\ldots,m \qquad p = 1,\ldots,s \tag{3.7d}$$

$$x_{cij} \in \{0,1\} \qquad c = 1,\ldots,t \qquad i = 1,\ldots,n \qquad j = 1,\ldots,m \tag{3.7e}$$

La función objetivo (3.7a) y las restricciones (3.7b) y (3.7e) han sido ya explicadas. Las restricciones (3.7c) garantizan que cada tesela encaje horizontalmente con al tesela adyacente a su izquierda y a su derecha. En efecto, cada lado de la igualdad puede tomar valor 0 o 1. Toma valor 0 si todas las variables $x_{cij}$ toman valor 0 y toma valor 1 si una de las variables $x_{cij}$ toma valor 1. Necesariamente solo una de las variables tomará valor 1 ya que si más de una variable tomara valor 1 se estaría incumpliendo la restricción (3.7b).

Si el lado izquierdo de la igualdad toma valor 1 significa que el patrón del lado derecho de la tesela colocada en el bloque $(i, j)$ es el patrón $p$ y si toma valor 0 significa que ese patrón no es $p$. De igual manera, si el lado derecho de la igualdad toma valor 1 significa que el patrón del lado izquierdo de la tesela colocada en el bloque $(i, j+1)$ es el patrón $p$ y si toma valor 0 significa que ese patrón no es $p$. Por tanto, al imponer esta restricción se obliga a que los patrones de los lados derecho e izquierdo de las teselas colocadas en los bloques $(i, j)$ e $(i, j+1)$, respectivamente, coincidan.

Análogamente, las restricciones (3.7d) garantizan que cada tesela encaje verticalmente con la tesela adyacente superior e inferior.

Si se utilizan los colores negro, blanco y un gris equidistante del negro y del blanco para colorear las teselas de la figura 3.6 se obtiene un conjunto de 24 teselas que se muestran en la figura 3.7.

En la figura 3.8 se muestran dos ejemplos de mosaicos construidos a partir del modelo (3.7) con las teselas de la figura 3.7, así como sus características.

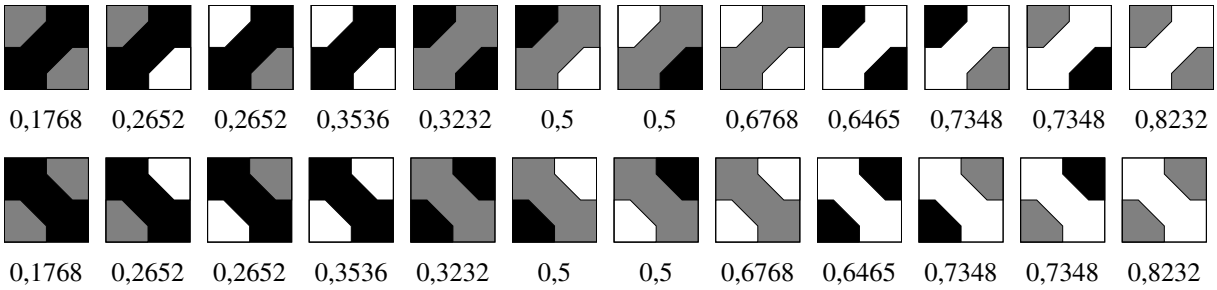
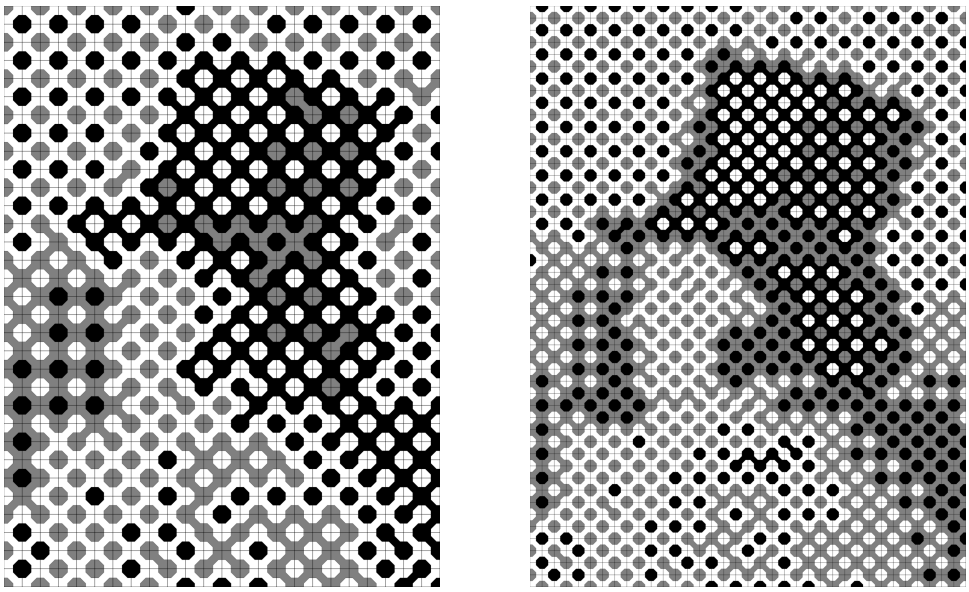



Figura 3.7: Teselas de la figura 3.6 coloreadas con tres colores. El número debajo de cada tesela indica su valor de brillo.



(a) Teselas de tamaño $144 \times 144$ píxeles. Resolver el modelo ha costado 3 segundos y 48 centésimas de segundo y el valor de la función objetivo ha sido 2,139.

(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 44 segundos y 37 centésimas de segundo y el valor de la función objetivo ha sido 4,193.

Figura 3.8: Mosaicos construidos a partir del modelo (3.7).

## 3.5. Construcción de mosaicos con piezas de Tetris®

En esta sección se va a presentar un modelo para la construcción de mosaicos utilizando las piezas del famoso juego de lógica Tetris®. Cada pieza está coloreada con un tono de gris diferente, siendo todos los tonos de gris utilizados equidistantes entre sí en la escala de grises. Además, se consideran todas las posibles rotaciones de las piezas en el plano. Por tanto, se dispone para la construcción de los mosaicos de un conjunto de diecinueve piezas numeradas del 1 al 19 y coloreadas con siete tonalidades de gris numeradas del 1 al 7. Este conjunto de piezas se muestra en la figura 3.9. Para facilitar la formulación del modelo, cada pieza tiene asignado un punto de referencia que será el vértice superior izquierdo del menor rectángulo que encierra toda la pieza aunque este punto no pertenezca a la pieza en sí.

En este modelo se ha de cubrir la imagen digital con las diecinueve teselas, de manera que estas encajen sin superponerse y sin dejar huecos. Notar que, en este caso, cada tesela ocupa cuatro bloques de la imagen digital, según su forma.

En este modelo han de definirse variables binarias que asocian el bloque y el color, así como el punto de referencia de la pieza y el bloque.

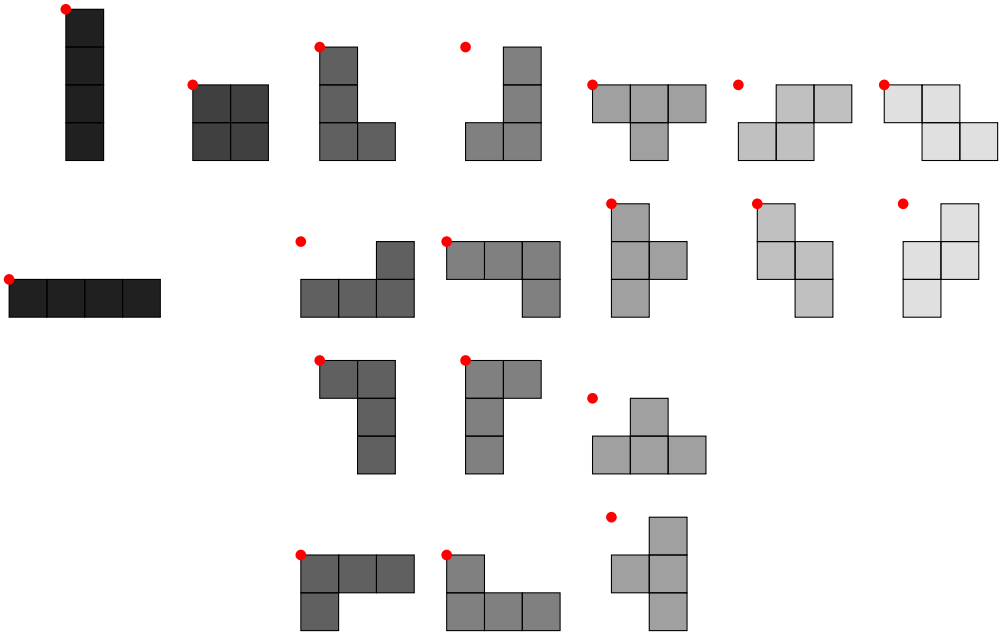


Figura 3.9: Piezas de Tetris® coloreadas, con su punto de referencia en rojo.

Sean las variables binarias $x_{c\,i\,j}$ con $i = 1,\ldots,n$, $j = 1,\ldots,m$ y $c = 1,\ldots,7$, definidas como:

$$x_{c\,i\,j} = \begin{cases} 1 & \text{si el el bloque } (i,j) \text{ se colorea con el tono de gris } c. \\ 0 & \text{en caso contrario.} \end{cases}$$

Sean las variables binarias $y_{k\,i\,j}$ con $i = -2,\ldots,n$, $j = -2,\ldots,m$ y $k = 1,\ldots,19$, definidas como:

$$y_{k\,i\,j} = \begin{cases} 1 & \text{si el punto de referencia de la pieza } k \text{ está en el bloque } (i,j)\,. \\ 0 & \text{en caso contrario.} \end{cases}$$

La formulación del modelo es:

$$\text{min} \quad \sum_{c=1}^{7}\sum_{i=1}^{n}\sum_{j=1}^{m}(\beta_c - \beta_{ij})^2 x_{c\,i\,j} \tag{3.8a}$$

$$\text{sujeto a} \quad \sum_{k=1}^{19}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{k\,i\,j} = \frac{n\cdot m}{4} \tag{3.8b}$$

$$\left.\begin{aligned} y_{1\,i\,j} &\leqslant x_{1\,i\,j} \\ y_{1\,i\,j} &\leqslant x_{1\,i+1\,j} \\ y_{1\,i\,j} &\leqslant x_{1\,i+2\,j} \\ y_{1\,i\,j} &\leqslant x_{1\,i+3\,j} \end{aligned}\right\} \qquad i = 1,\ldots,n-3 \quad j = 1,\ldots,m \tag{3.8c}$$

$$\ldots \tag{3.8d}$$

$$\sum_{c=1}^{7} x_{c\,i\,j} = 1 \qquad i = 1,\ldots,n \quad j = 1,\ldots,m \tag{3.8e}$$

$$x_{2\,i\,j} = y_{3\,i\,j} + y_{3\,i-1\,j} + y_{3\,i\,j-1} + y_{1\,i-1\,j-1} \quad i = 1,\ldots,n \quad j = 1,\ldots,m \tag{3.8f}$$

$$\ldots \tag{3.8g}$$

$$\left.\begin{array}{r} y_{1\,n-2\,j} = 0 \\ y_{1\,n-1\,j} = 0 \\ y_{1\,n\,j} = 0 \\ \vdots \end{array}\right\} \qquad j = 1,\ldots,m \tag{3.8h}$$

$$\left.\begin{array}{r} y_{k\,-2\,j} = 0 \\ y_{k\,-1\,j} = 0 \\ y_{k\,0\,j} = 0 \end{array}\right\} \qquad k = 1,\ldots,19 \quad j = 1,\ldots,m \tag{3.8i}$$

$$\left.\begin{array}{r} y_{k\,i\,-2} = 0 \\ y_{k\,i\,-1} = 0 \\ y_{k\,i\,0} = 0 \end{array}\right\} \qquad k = 1,\ldots,19 \quad i = 1,\ldots,n \tag{3.8j}$$

$$x_{cij} \in \{0,1\} \qquad c = 1,\ldots,7 \quad i = 1,\ldots,n \quad j = 1,\ldots,m \tag{3.8k}$$

$$y_{kij} \in \{0,1\} \qquad k = 1,\ldots,19 \quad i = -2,\ldots,n \quad j = 1,\ldots,m \tag{3.8l}$$

Como todas las piezas ocupan cuatro bloques, el número de piezas que hay que utilizar es la cuarta parte de la cantidad de bloques en los que está dividida la imagen a partir de la cual se construye el mosaico. Es decir hay que colocar exactamente $\frac{n \cdot m}{4}$ puntos de referencia. Para garantizar esto, se introduce la restricción (3.8b).

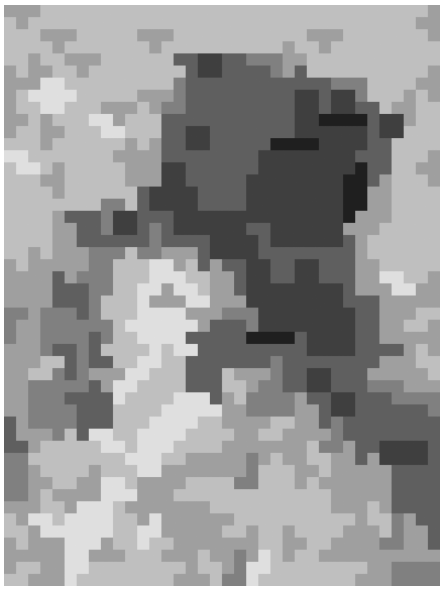
Colocar el punto de referencia de una pieza determinada en un bloque $(i, j)$ implica que dicha pieza ocupa cuatro bloques "cercanos" al punto de referencia. Los bloques que se ocupan varían en función de la pieza que se esté colocando. Las restricciones (3.8c) modelan el hecho de colocar el punto de referencia de la barra vertical en el bloque $(i, j)$, si se considera que la barra vertical es la pieza numerada con $k = 1$ y cuyo color es $c = 1$. Los puntos suspensivos (3.8d) indican que, análogamente, hay que imponer este tipo de restricciones para cada una de las diecinueve piezas.

Las restricciones (3.8e) garantizan que cada bloque se coloree con un único color. Sin embargo, imponiendo solo esto no se garantiza que piezas del mismo color no se superpongan. Para evitar esto, hay que imponer restricciones para que un bloque $(i, j)$ se coloree de un único color y solo sea ocupado por una pieza. Las restricciones (3.8f) garantizan este hecho para los bloques que se colorean con el tono de gris $c = 2$. Los puntos suspensivos (3.8g) indican que, análogamente, hay que imponer este tipo de restricciones para cada uno de los siete tonos que gris que se utilizan. Notar que, como este tipo de restricciones hay que imponerlas para cada bloque $(i, j)$ se necesitan unos bloques "auxiliares" en los bordes superior e izquierdo de la imagen donde en principio se podrían colocar puntos de referencia para que no haya conflicto con los índices y para que las restricciones (3.8f) puedan formularse de manera genérica. Como en estos bloques "auxiliares" no se pueden colocar puntos de referencia, hay que imponer restricciones que eviten esto. Las restricciones (3.8i) y (3.8j) garantizan este hecho.

Por otro lado, hay bloques cercanos a los bordes inferior y derecho de la imagen donde no se pueden colocar puntos de referencia ya que no encajarían las piezas dentro de los límites de la imagen. Por ejemplo, el punto de referencia de la barra vertical no se puede colocar en ningún bloque de las tres últimas filas de bloques. Las restricciones (3.8h) garantizan que esto no ocurra.

En la figura 3.10 se muestran dos ejemplos de mosaicos construidos a partir de este modelo, así como sus características. Cuando los bloques en los que se divide la imagen tienen unas dimensiones de $144 \times 144$ píxeles, el modelo tiene 23331 variables y 86428 restricciones y el mosaico tiene unas dimensiones de $32 \times 24$ bloques. Cuando los bloques en los que se divide la imagen tienen unas dimensiones de $96 \times 96$ píxeles, el modelo tiene 49887 variables y 192904 restricciones y el mosaico tiene unas dimensiones de $48 \times 36$ bloques. Debido a la complejidad del modelo, para la construcción de estos mosaicos no se ha utilizado la solución óptima sino que se ha utilizado la mejor solución factible encontrada después de 10 horas de procesamiento.

(a) Bloques de tamaño $144 \times 144$ píxeles. Se muestra la solución obtenida tras 10 horas de computación. El valor de la función objetivo ha sido 3,800.



(b) Bloques de tamaño $96 \times 96$ píxeles. Se muestra la solución obtenida tras 10 horas de computación. El valor de la función objetivo ha sido 12,637.

Figura 3.10: Mosaicos construidos a partir del modelo (3.8).

# Bibliografía


[1] R. BOSCH, *Opt Art*, Princeton University Press, 2019.

[2] M.S. BAZARAA, J.J. JARVIS AND H.D. SHERALI, *Linear Programming and Network Flows*, Wiley, Hoboken, NJ, 4th edition, 2010.

[3] Y. POCHET AND L.A. WOLSEY, *Production Planning by Mixed Integer Programming*, Springer, 2006.

[4] L.A. WOLSEY, *Integer Programming*, Wiley, 2nd edition, 2021.

[5] M. JÜNGER ET AL., *50 years of Integer Programming 1958-2008*, Springer, 2010.

[6] F.S HILLIER Y G.J. LIEBERMAN, *Introducción a la investigación de operaciones*, McGraw-Hill, 9ª edición, 2010.

[7] MUSEO DE ZARAGOZA, *Mosaico de Orfeo*, `http://www.museodezaragoza.es/colecciones/roma/`, disponible en `http://www.museodezaragoza.es/`.

[8] MUSEO DEL PRADO, *Autoretrato de Francisco de Goya y Lucientes*, `https://www.museodelprado.es/coleccion/obra-de-arte/autorretrato-francisco-goya-y-lucientes-pintor/16d0d5d4-ac78-4295-9d32-31d8c9646c13`, disponible en `https://www.museodelprado.es/`.

[9] PAULA VERSNICK, *Las Pajaritas*, `https://www.flickr.com/photos/orihouse/2610540224/`, disponible en `https://www.flickr.com/`.

[10] MY MODERN MET, *Conoce la antigua historia de los mosaicos y descubre cómo hacer uno*, `https://mymodernmet.com/es/como-hacer-mosaico/`.

[11] DOMUS SOPHIAE, *Mosaico*, `https://domussophiae.com/historia/`.

# Anexos

# Anexo A

# Código Python para la creación de mosaicos

En la figura A.1 se muestra el código de la función `RGB2grayscale`. Esta función transforma una imagen en formato *RGB* en una imagen en escala de grises según la ponderación de la formula (1.1). La función toma como argumento un array tridimensional de *numpy* donde se almacena la información de la imagen en formato *RGB* y devuelve un array bidimensional de *numpy* donde se almacena la información de la imagen en escala de grises.

```
def RGB2grayscale(image):
    grayscale_image = (0.299*image[:,:,0] + 0.587*image[:,:,1] +
                       0.114*image[:,:,2])
    return grayscale_image
```

Figura A.1: Función `RGB2grayscale`.

En la figura A.2 se muestra el código de la función `imagePreprocessing`. Esta función realiza el preprocesamiento necesario de la imagen para la construcción del mosaico a partir de ella. La función toma como argumentos la ruta de acceso al directorio donde se encuentra la imagen y el parámetro $k$, que es el tamaño medido en píxeles que tienen de lado los bloques cuadrados en los que se divide la imagen. Devuelve dos arrays bidimensionales de *numpy*; en el primero se almacena la información de la imagen transformada en escala de grises y en el segundo se almacena la información de la imagen en escala de grises dividida en bloques de tamaño $k \times k$ píxeles.

```
def imagePreprocessing(file,k):
    image=io.imread(file)/255.0
    h,w,d = image.shape
    grayscale_image = RGB2grayscale(image)
    blocks_image = np.zeros((h//k,w//k))
    for i in range(0,h//k):
        for j in range(0,w//k):
            blocks_image[i,j] = sum(sum(grayscale_image[i*k:(i+1)*k,
                                                        j*k:(j+1)*k]))/(k**2)
    return grayscale_image, blocks_image
```

Figura A.2: Función `imagePreprocessing`.

En la figura A.3 se muestra el código de la función `coefficientsArray`. Esta función calcula los valores $(\beta_c - \beta_{ij})^2$ y $|\beta_c - \beta_{ij}|$ necesarios para construir las funciones objetivo de las fórmulas (3.2) y (3.3) y los escribe en un fichero de texto. La función toma como argumentos el array bidimensional de *numpy* donde se almacena la información de la imagen en escala de grises dividida en bloques y un diccionario que almacena el valor del brillo de cada tesela del conjunto de teselas que se quiera utilizar para construir el mosaico. Devuelve valor `True` si el proceso se ha completado satisfactoriamente.

```
def coefficientsArray(blocks_image, brightness_dict):
    arrays_2d_list = []
    for c in brightness_dict.keys():
```

```
        arrays_2d_list.append((blocks_image -
                    brightness_dict[c]*np.ones(blocks_image.shape))**2)
#       arrays_2d_list.append(np.abs(blocks_image -
#                   brightness_dict[c]*np.ones(blocks_image.shape)))
    brightness_array = np.array(arrays_2d_list)
    f = open("coefficients_array.txt", "w")
    f.write(str(brightness_array.tolist()))
    f.close()

    return True
```

Figura A.3: Función `coefficientsArray`.

En la figura A.4 se muestra el código de la función `createMosaic`. Esta función construye un mosaico a partir de una imagen determinada utilizando para ello una solución óptima de un modelo concreto. Además, guarda el mosaico construido en un archivo .pdf con el nombre deseado. La función toma como argumentos los arrays bidimensionales de *numpy* donde se almacena la información de la imagen transformada en escala de grises y de su división en bloques, el conjunto de teselas con el que se va a construir el mosaico, la ruta de acceso al directorio donde se encuentra el archivo de texto con la solución del modelo considerado y el nombre que se quiere poner al archivo .pdf donde se guarda el mosaico.

```
def createMosaic(grayscale_image, blocks_image, tiles_set,
                 solution_file, mosaic_name):
    arrays_list = np.array_split(np.loadtxt(solution_file), len(tiles_set))
    mosaic = np.zeros(grayscale_image.shape)
    for i in range(0,blocks_image.shape[0]):
        for j in range(0,blocks_image.shape[1]):
            mosaic[i*k:i*k+k,j*k:j*k+k] = tiles_set[np.where(
                np.array(arrays_list)[:,i,j] == 1)[0][0]]
    plt.imshow(mosaic,vmin=0,vmax=1)
    plt.axis("off")
    plt.savefig(mosaic_name, bbox_inches="tight", pad_inches=0, dpi=600)

    return True
```

Figura A.4: Función `createMosaic`.

# Anexo B

# Ejemplo de aplicación del método de ramificación y acotación

En el capítulo 2 se ha comentado que al aplicar el método de ramificación y acotación para resolver problemas de optimización entera existen diferentes criterios que se pueden utilizar para seleccionar el siguiente subproblema a explorar o para elegir la variable a partir de la cual hacer una ramificación.

No existe ningún criterio óptimo y, dependiendo de la forma del problema, unos funcionan mejor que otros, aunque este es un aspecto que no se conoce a priori. Para ejemplificar este hecho se va a resolver el siguiente *PLEB* cuyo enunciado se propone en [6] utilizando el método de ramificación y acotación.

$$\begin{array}{lrcrcrcrcrcl} \max & x_1 & + & \frac{9}{5}x_2 & + & \frac{8}{5}x_3 & + & \frac{4}{5}x_4 & + & \frac{7}{5}x_5 & & \\ \text{sujeto a} & & & & & & & & & & & \\ & 6x_1 & + & 12x_2 & + & 10x_3 & + & 4x_4 & + & 8x_5 & \leqslant & 20 \\ & & & & & x_j \in \{0,1\}, & & j=1,\ldots,5 & & & & \end{array} \tag{B.1}$$

Dependiendo del criterio usado para la selección de subproblemas es necesario explorar más subproblemas o menos. En este caso no es necesario emplear criterios para elegir a partir de qué variable hacer la ramificación ya que cuando sea necesario hacer una ramificación solo existirá una posibilidad.

Como las variables del problema son binarias, cuando se ramifica a partir de una variable $x_j$, se crean dos subproblemas incluyendo las restricciones $x_j = 0$ o $x_j = 1$. Se añadirá siempre en primer lugar a la lista de subproblemas el subproblema generado a partir de la restricción $x_j = 0$.

Si se utiliza el criterio *LIFO* para la selección de subproblemas se obtiene el árbol de enumeración que se muestra en la figura B.1. Cada cuadro muestra el resultado de resolver el problema original con las restricciones que se indican en las ramas que llevan a dicho subproblema. En la cabecera del cuadro se indica el orden en el que se van resolviendo los subproblemas. Con este criterio es necesario explorar 13 subproblemas para resolver el problema entero binario. La solución óptima se muestra en el cuadro sombreado.

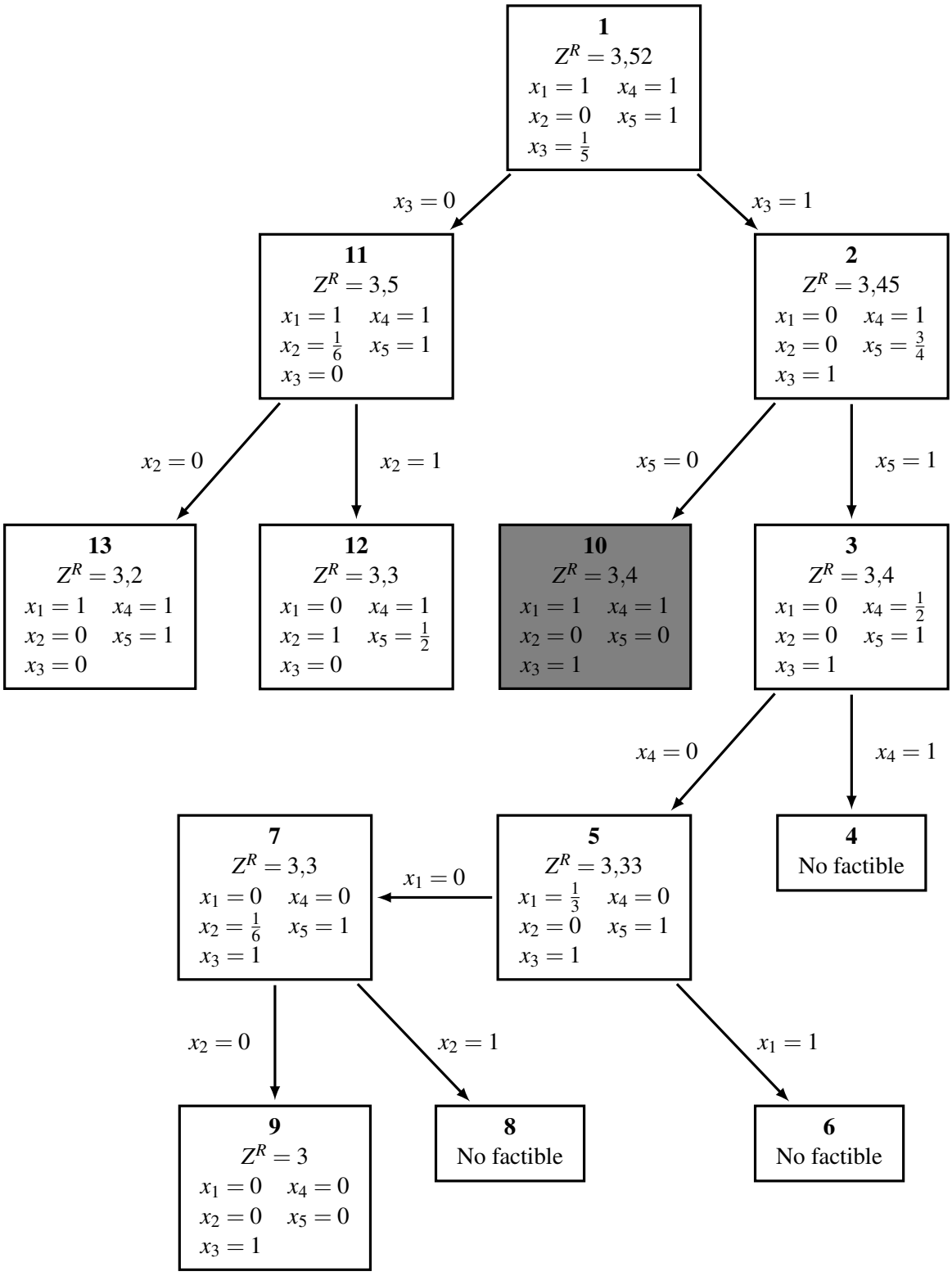



Figura B.1: Árbol de enumeración del problema (B.1) empleando el criterio *LIFO* como criterio de selección de subproblemas.

Si se utiliza el criterio *FIFO* para la selección de subproblemas se obtiene el árbol de enumeración que se muestra en la figura B.2. Con este criterio es necesario explorar 9 subproblemas para resolver el problema entero binario.

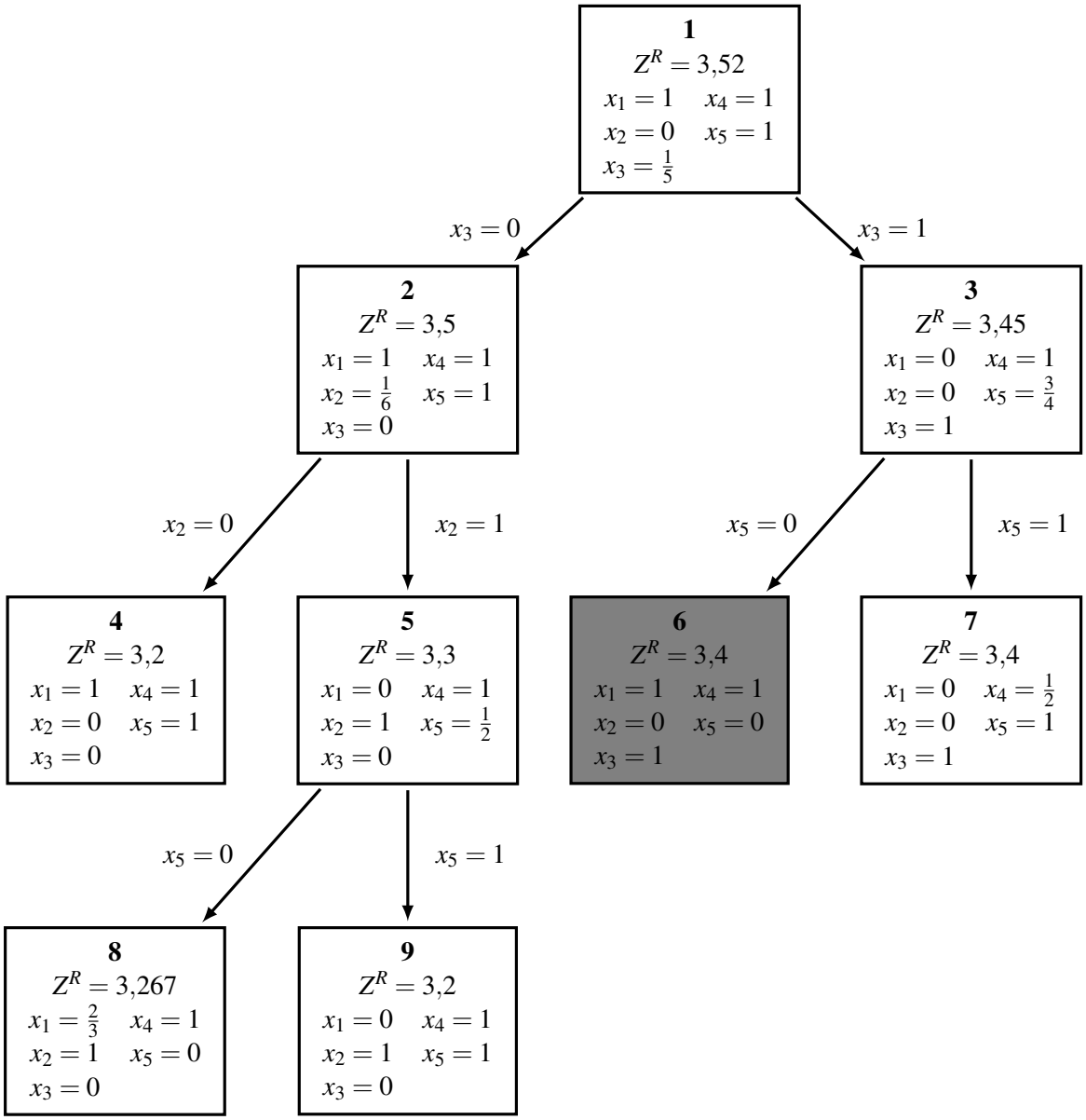



Figura B.2: Árbol de enumeración del problema (B.1) empleando el criterio *FIFO* como criterio de selección de subproblemas.

# Anexo C

# Ejemplos de códigos CPLEX para la resolución de modelos

Como se ha indicado en la sección 3.1, para resolver los problemas de optimización que se han formulado a lo largo de esta memoria se ha utilizado el paquete de software IBM ILOG CPLEX Optimization Studio versión 20.1.0 desarrollado por la compañía IBM, más concretamente se ha empleado el IDE llamado CPLEX Optimization Studio y un lenguaje de modelado llamado Optimization Programming Language (OPL). Para resolver un modelo hacen falta dos archivos, un archivo con extensión .mod donde se programa el modelo y un archivo .dat de donde se leen los datos necesarios para resolver el modelo, es decir los parámetros definidos o las matrices que contienen los coeficientes de las restricciones.

En la figura C.1 se muestra el código del archivo .mod del modelo (3.6). Al final de este fichero se incluye un pequeño fragmento de código que al ejecutarse escribe la solución del modelo proporcionada por CPLEX en un archivo de texto que se usa en el proceso de construcción del mosaico programado en Python.

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: Aitor
 * Creation Date: 5 jul. 2021 at 15:43:48
 *********************************************/

/* Datos */
int t = ...;
int n = ...;
int m = ...;
int N = ...;

range C = 1..t;
range I = 1..n;
range J = 1..m;
range I_ = 1..n-1;
range J_ = 1..m-1;

float d[C][I][J] = ...;

/* Variables */
dvar boolean x[C][I][J];

/* Función objetivo */
minimize sum(c in C, i in I, j in J)(d[c][i][j]*x[c][i][j]);

/* Restricciones */
subject to {
  forall(i in I, j in J){
    sum(c in C) x[c][i][j] == 1;
  }
```

```
  forall(c in C){
      sum(i in I, j in J) x[c][i][j] == N;
  }
  forall(c in C, i in I_, j in J){
    x[c][i][j] + x[c][i+1][j] <= 1;
  }
  forall(c in C, i in I, j in J_){
    x[c][i][j] + x[c][i][j+1] <= 1;
  }
}


/* Escritura de la solución encontrada en un archivo de texto (Array único)*/
execute{
  var archivoSalida = new IloOplOutputFile("solucion_modelo2.txt");
  for(var c in thisOplModel.C){
    for(var i in thisOplModel.I){
      for (var j in thisOplModel.J){
      archivoSalida.write(thisOplModel.x[c][i][j]," ");

    }
    archivoSalida.writeln();
  }
  archivoSalida.writeln();
  }
  archivoSalida.close();
}
```

Figura C.1: Código del fichero .mod correspondiente al modelo (3.6).

En la figura C.2 se muestra el formato del archivo .dat del modelo (3.6). No se incluye la matriz $d$ completa por ser muy extensa.

```
/*********************************************
 * OPL 20.1.0.0 Data
 * Author: Aitor
 * Creation Date: 5 jul. 2021 at 15:45:19
 *********************************************/

t = 12;
n = 32;
m = 24;
N = 64;

d = [[[0.37139212403900124, 0.3441536479659094, 0.3433364627962752...
  ...
  ... 0.35801481338977986, 0.3233425295602834, 0.28459234291095353]]];
```

Figura C.2: Código del fichero .dat correspondiente al modelo (3.6).

En la figura C.3 se muestra el código del archivo .mod del modelo (3.8). Al final de este fichero se incluye un pequeño fragmento de código que al ejecutarse escribe la solución del modelo proporcionada por CPLEX en un archivo de texto que se usa en el proceso de construcción del mosaico programado en Python. El formato del archivo .dat de este modelo es el mismo que el de la figura C.2.

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: Aitor
 * Creation Date: 4 sept. 2021 at 18:35:20
 *********************************************/

int n = ...;
int m = ...;

range C = 1..7;
range K = 1..19;
range I = 1..n;
range Ineg=-2..n;
range I1 = 1..n-1;
range I2 = 1..n-2;
range I3 = 1..n-3;
range J = 1..m;
range Jneg=-2..m;
range J1 = 1..m-1;
range J2 = 1..m-2;
range J3 = 1..m-3;

float d[C][I][J] = ...;

/* Variables */
dvar boolean x[C][I][J];
dvar boolean y[K][Ineg][Jneg];

/* Funcion objetivo */
minimize sum(c in C, i in I, j in J)(d[c][i][j]*x[c][i][j]);

/* Restricciones */
subject to {
  /* Esta restriccion impone que cada bloque solo se coloree de un unico color.
    */
  forall(i in I, j in J){
    sum(c in C) x[c][i][j] == 1;
  }

  /* Estas restricciones impiden que se coloquen puntos de referencia en bloques
     auxiliares.*/
  forall(k in K, j in J){
    y[k][0][j] == 0;
    y[k][-1][j] == 0;
    y[k][-2][j] == 0;
  }
  forall(k in K, i in I){
    y[k][i][0] == 0;
    y[k][i][-1] == 0;
    y[k][i][-2] == 0;
  }

  /* Estas restricciones impiden que se coloquen puntos de referencia en bloques
     cercanos a los bordes inferior y derecho.*/
  forall(j in J){
    y[1][n-2][j] == 0;
    y[1][n-1][j] == 0;
    y[1][n][j] == 0;

    y[3][n][j] == 0;

    y[4][n-1][j] == 0;
    y[4][n][j] == 0;
```

```
61
62      y[5][n][j] == 0;
63
64      y[6][n-1][j] == 0;
65      y[6][n][j] == 0;
66
67      y[7][n][j] == 0;
68
69      y[8][n-1][j] == 0;
70      y[8][n][j] == 0;
71
72      y[9][n][j] == 0;
73
74      y[10][n-1][j] == 0;
75      y[10][n][j] == 0;
76
77      y[11][n][j] == 0;
78
79      y[12][n][j] == 0;
80
81      y[13][n-1][j] == 0;
82      y[13][n][j] == 0;
83
84      y[14][n][j] == 0;
85
86      y[15][n-1][j] == 0;
87      y[15][n][j] == 0;
88
89      y[16][n][j] == 0;
90
91      y[17][n-1][j] == 0;
92      y[17][n][j] == 0;
93
94      y[18][n][j] == 0;
95
96      y[19][n-1][j] == 0;
97      y[19][n][j] == 0;
98
99  }
100
101 forall(i in I){
102     y[2][i][m-2] == 0;
103     y[2][i][m-1] == 0;
104     y[2][i][m] == 0;
105
106     y[3][i][m] == 0;
107
108     y[4][i][m] == 0;
109
110     y[5][i][m-1] == 0;
111     y[5][i][m] == 0;
112
113     y[6][i][m] == 0;
114
115     y[7][i][m-1] == 0;
116     y[7][i][m] == 0;
117
118     y[8][i][m] == 0;
119
120     y[9][i][m-1] == 0;
121     y[9][i][m] == 0;
122
123     y[10][i][m] == 0;
```

```
124
125     y[11][i][m-1] == 0;
126     y[11][i][m] == 0;
127
128     y[12][i][m-1] == 0;
129     y[12][i][m] == 0;
130
131     y[13][i][m] == 0;
132
133     y[14][i][m-1] == 0;
134     y[14][i][m] == 0;
135
136     y[15][i][m] == 0;
137
138     y[16][i][m-1] == 0;
139     y[16][i][m] == 0;
140
141     y[17][i][m] == 0;
142
143     y[18][i][m-1] == 0;
144     y[18][i][m] == 0;
145
146     y[19][i][m] == 0;
147   }
148
149   forall(i in I, j in J){
150     x[1][i][j] == y[1][i][j] + y[1][i-1][j] + y[1][i-2][j] + y[1][i-3][j] + y
      [2][i][j] + y[2][i][j-1] + y[2][i][j-2] + y[2][i][j-3];
151
152     x[2][i][j] == y[3][i][j] + y[3][i-1][j] + y[3][i][j-1] + y[3][i-1][j-1];
153
154     x[3][i][j] == y[4][i][j] + y[4][i-1][j] + y[4][i-2][j] + y[4][i-2][j-1] + y
      [5][i-1][j] + y[5][i-1][j-1] + y[5][i-1][j-2] + y[5][i][j-2] +  y[6][i][j] +
       y[6][i][j-1] + y[6][i-1][j-1] + y[6][i-2][j-1] + y[7][i][j] + y[7][i-1][j]
      + y[7][i][j-1] + y[7][i][j-2];
155
156     x[4][i][j] == y[8][i-2][j] + y[8][i-2][j-1] + y[8][i-1][j-1] + y[8][i][j-1]
      + y[9][i][j] + y[9][i][j-1] + y[9][i][j-2] + y[9][i-1][j-2] +  y[10][i][j] +
       y[10][i-1][j] + y[10][i-2][j] + y[10][i][j-1] + y[11][i][j] + y[11][i-1][j]
       + y[11][i-1][j-1] + y[11][i-1][j-2];
157
158     x[5][i][j] == y[12][i][j] + y[12][i][j-1] + y[12][i][j-2] + y[12][i-1][j-1]
      + y[13][i][j] + y[13][i-1][j] + y[13][i-2][j] + y[13][i-1][j-1] + y[14][i
      -1][j] + y[14][i][j-1] + y[14][i-1][j-1] + y[14][i-1][j-2] + y[15][i-1][j] +
       y[15][i][j-1] + y[15][i-1][j-1] + y[15][i-2][j-1];
159
160     x[6][i][j] == y[16][i-1][j] + y[16][i][j-1] + y[16][i-1][j-1] + y[16][i][j
      -2] + y[17][i][j] + y[17][i-1][j] + y[17][i-1][j-1] + y[17][i-2][j-1];
161
162     x[7][i][j] == y[18][i][j] + y[18][i][j-1] + y[18][i-1][j-1] + y[18][i-1][j
      -2] + y[19][i-1][j] + y[19][i][j-1] + y[19][i-1][j-1] + y[19][i-2][j];
163   }
164
165   /* Esta restriccion impone que se coloquen los puntos de referencia necesarios
      . */
166   sum(k in K, i in I, j in J) y[k][i][j] == (n*m)/4;
167
168   /*I*/
169   forall(i in I3, j in J){
170     y[1][i][j] <= x[1][i][j];
171     y[1][i][j] <= x[1][i+1][j];
172     y[1][i][j] <= x[1][i+2][j];
173     y[1][i][j] <= x[1][i+3][j];
```

```
174   }
175   forall(i in I, j in J3){
176     y[2][i][j] <= x[1][i][j];
177     y[2][i][j] <= x[1][i][j+1];
178     y[2][i][j] <= x[1][i][j+2];
179     y[2][i][j] <= x[1][i][j+3];
180   }
181
182   /*O*/
183   forall(i in I1, j in J1){
184     y[3][i][j] <= x[2][i][j];
185     y[3][i][j] <= x[2][i+1][j];
186     y[3][i][j] <= x[2][i][j+1];
187     y[3][i][j] <= x[2][i+1][j+1];
188   }
189
190   /*L*/
191   forall(i in I2, j in J1){
192     y[4][i][j] <= x[3][i][j];
193     y[4][i][j] <= x[3][i+1][j];
194     y[4][i][j] <= x[3][i+2][j];
195     y[4][i][j] <= x[3][i+2][j+1];
196   }
197   forall(i in I1, j in J2){
198       y[5][i][j] <= x[3][i][j+2];
199     y[5][i][j] <= x[3][i+1][j];
200     y[5][i][j] <= x[3][i+1][j+1];
201     y[5][i][j] <= x[3][i+1][j+2];
202   }
203   forall(i in I2, j in J1){
204     y[6][i][j] <= x[3][i][j];
205     y[6][i][j] <= x[3][i][j+1];
206     y[6][i][j] <= x[3][i+1][j+1];
207     y[6][i][j] <= x[3][i+2][j+1];
208   }
209   forall(i in I1, j in J2){
210     y[7][i][j] <= x[3][i][j];
211     y[7][i][j] <= x[3][i][j+1];
212     y[7][i][j] <= x[3][i][j+2];
213     y[7][i][j] <= x[3][i+1][j];
214   }
215
216   /*J*/
217   forall(i in I2, j in J1){
218     y[8][i][j] <= x[4][i+2][j];
219     y[8][i][j] <= x[4][i][j+1];
220     y[8][i][j] <= x[4][i+1][j+1];
221     y[8][i][j] <= x[4][i+2][j+1];
222   }
223   forall(i in I1, j in J2){
224     y[9][i][j] <= x[4][i][j];
225     y[9][i][j] <= x[4][i][j+1];
226     y[9][i][j] <= x[4][i][j+2];
227     y[9][i][j] <= x[4][i+1][j+2];
228   }
229   forall(i in I2, j in J1){
230     y[10][i][j] <= x[4][i][j];
231     y[10][i][j] <= x[4][i+1][j];
232     y[10][i][j] <= x[4][i+2][j];
233     y[10][i][j] <= x[4][i][j+1];
234   }
235   forall(i in I1, j in J2){
236     y[11][i][j] <= x[4][i][j+2];
```

```
237     y[11][i][j] <= x[4][i+1][j];
238     y[11][i][j] <= x[4][i+1][j+1];
239     y[11][i][j] <= x[4][i+1][j+2];
240   }
241
242   /*T*/
243   forall(i in I1, j in J2){
244     y[12][i][j] <= x[5][i][j];
245     y[12][i][j] <= x[5][i][j+1];
246     y[12][i][j] <= x[5][i][j+2];
247     y[12][i][j] <= x[5][i+1][j+1];
248   }
249   forall(i in I2, j in J1){
250     y[13][i][j] <= x[5][i][j];
251     y[13][i][j] <= x[5][i+1][j];
252     y[13][i][j] <= x[5][i+2][j];
253     y[13][i][j] <= x[5][i+1][j+1];
254   }
255   forall(i in I1, j in J2){
256       y[14][i][j] <= x[5][i][j+1];
257     y[14][i][j] <= x[5][i+1][j];
258     y[14][i][j] <= x[5][i+1][j+1];
259     y[14][i][j] <= x[5][i+1][j+2];
260   }
261   forall(i in I2, j in J1){
262     y[15][i][j] <= x[5][i+1][j];
263     y[15][i][j] <= x[5][i][j+1];
264     y[15][i][j] <= x[5][i+1][j+1];
265     y[15][i][j] <= x[5][i+2][j+1];
266   }
267
268   /*S*/
269   forall(i in I1, j in J2){
270     y[16][i][j] <= x[6][i+1][j];
271     y[16][i][j] <= x[6][i][j+1];
272     y[16][i][j] <= x[6][i+1][j+1];
273     y[16][i][j] <= x[6][i][j+2];
274   }
275   forall(i in I2, j in J1){
276     y[17][i][j] <= x[6][i][j];
277     y[17][i][j] <= x[6][i+1][j];
278     y[17][i][j] <= x[6][i+1][j+1];
279     y[17][i][j] <= x[6][i+2][j+1];
280   }
281
282   /*Z*/
283   forall(i in I1, j in J2){
284     y[18][i][j] <= x[7][i][j];
285     y[18][i][j] <= x[7][i][j+1];
286     y[18][i][j] <= x[7][i+1][j+1];
287     y[18][i][j] <= x[7][i+1][j+2];
288   }
289   forall(i in I2, j in J1){
290     y[19][i][j] <= x[7][i+1][j];
291     y[19][i][j] <= x[7][i+2][j];
292     y[19][i][j] <= x[7][i][j+1];
293     y[19][i][j] <= x[7][i+1][j+1];
294   }
295 }
296
297 /* Escritura de la solucion encontrada en un archivo de texto */
298 execute{
299   var archivoSalida = new IloOplOutputFile("solucion_modelo_tetrisX.txt");
```

```
300   for(var c in thisOplModel.C){
301     for(var i in thisOplModel.I){
302       for (var j in thisOplModel.J){
303       archivoSalida.write(thisOplModel.x[c][i][j]," ");
304
305     }
306     archivoSalida.writeln();
307   }
308   archivoSalida.writeln();
309   }
310   archivoSalida.close();
311
312   var archivoSalida2 = new IloOplOutputFile("solucion_modelo_tetrisY.txt");
313   for(var k in thisOplModel.K){
314     for(var i in thisOplModel.Ineg){
315       for (var j in thisOplModel.Jneg){
316       archivoSalida2.write(thisOplModel.y[k][i][j]," ");
317
318     }
319     archivoSalida2.writeln();
320   }
321   archivoSalida2.writeln();
322   }
323   archivoSalida2.close();
324
325 }
```

Figura C.3: Código del fichero .mod correspondiente al modelo (3.8).

# Anexo D

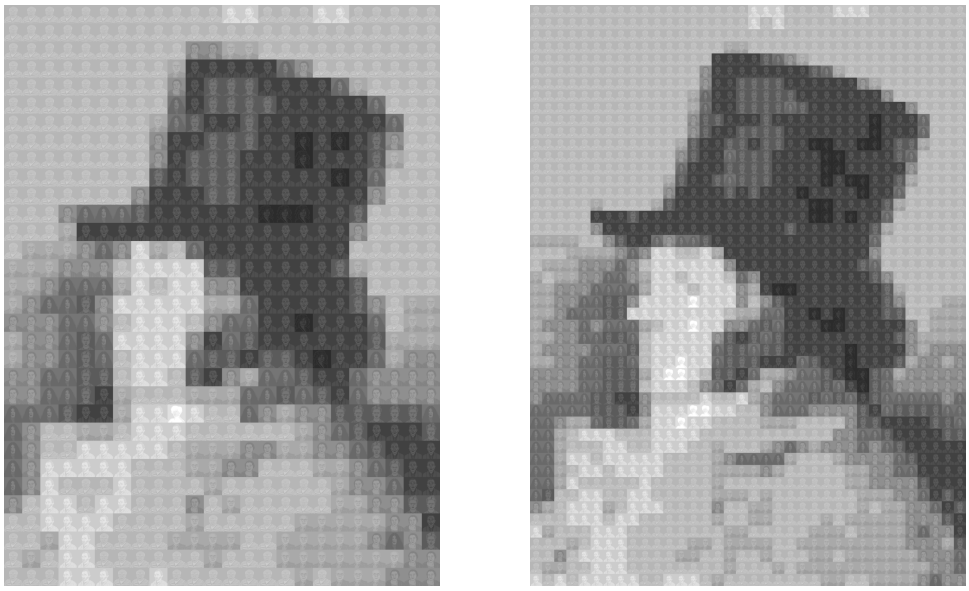
# Otros mosaicos

En este anexo se incluye una colección de mosaicos que por cuestiones de espacio no ha sido posible incluir en el cuerpo principal de la memoria pero que no eran esenciales para entender el desarrollo de la misma.

En las figuras D.1, D.2, D.3 y D.4 se muestran los mosaicos que se construyen a partir de los modelos (3.4), (3.5), (3.6) y (3.7) respectivamente cambiando la función objetivo por la de la fórmula (3.3).

En la figura D.5 se muestra una ampliación del mosaico que se encuentra en la parte derecha de la figura 3.5 para que se aprecien mejor los detalles de las teselas diseñadas para la construcción de este tipo de mosaicos.

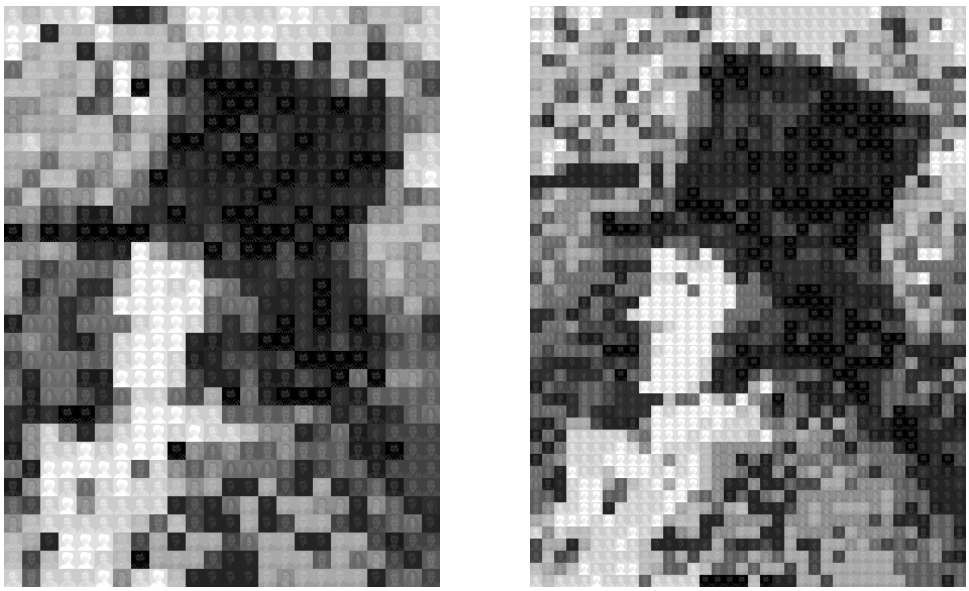
En la figura D.6 se muestran otros dos mosaicos construidos a partir de la imagen del monumento de "Las Pajaritas" que aparece a la izquierda de la figura 1.2.



(a) Teselas de tamaño $144 \times 144$ píxeles. Resolver el modelo ha costado 62 centésimas de segundo y el valor de la función objetivo ha sido 18,225.

(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 1 segundo y 61 centésimas de segundo y el valor de la función objetivo ha sido 40,066.

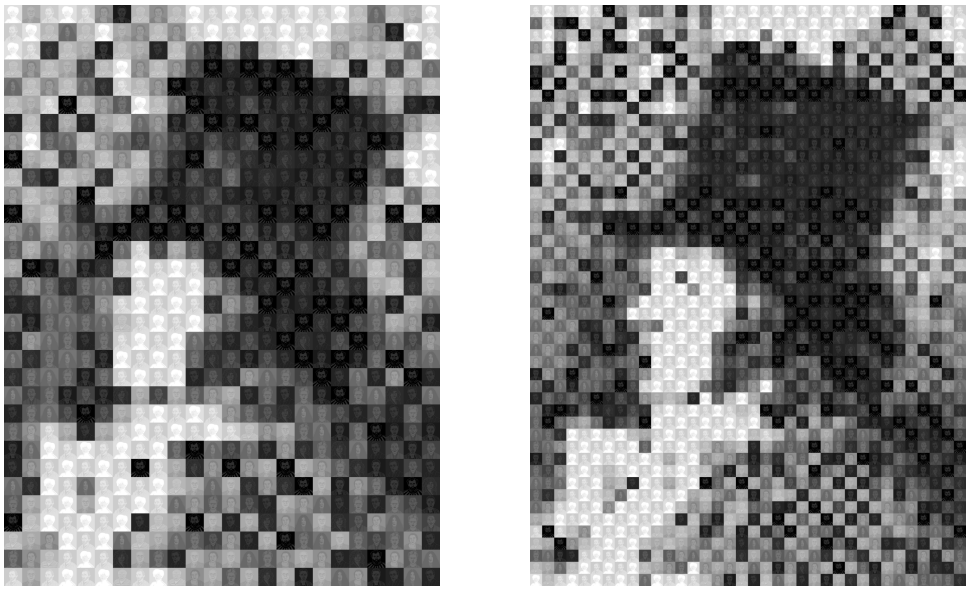
Figura D.1: Mosaicos construidos a partir del modelo (3.4) cambiando la función objetivo por la de la fórmula (3.3).



(a) Teselas de tamaño $144 \times 144$ píxeles. Resolver el modelo ha costado 77 centésimas de segundo y el valor de la función objetivo ha sido 129,181.

(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 1 segundo y 65 centésimas de segundo y el valor de la función objetivo ha sido 287,643.

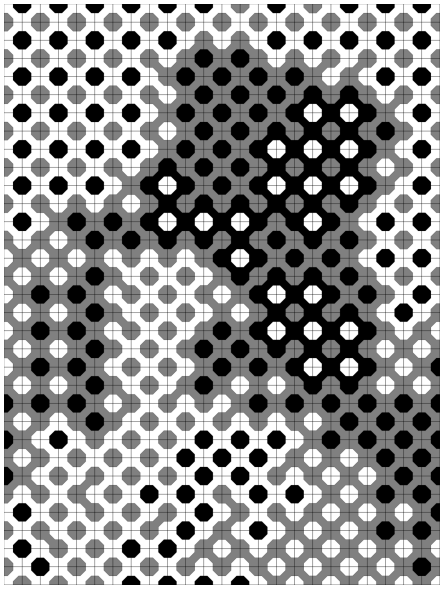
Figura D.2: Mosaicos construidos a partir del modelo (3.5) cambiando la función objetivo por la de la fórmula (3.3).



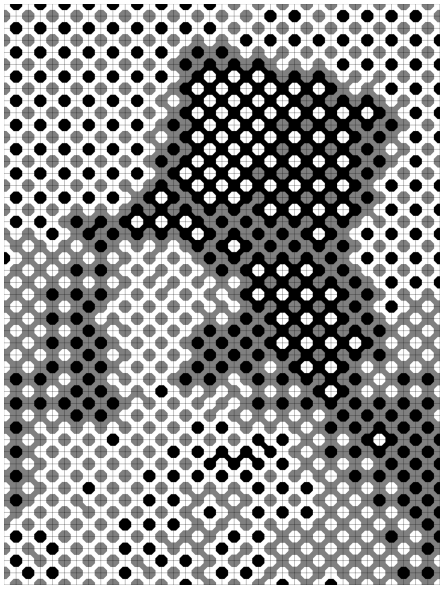
(a) Teselas de tamaño $144 \times 144$ píxeles. Resolver el modelo ha costado 1 segundo y 48 centésimas de segundo y el valor de la función objetivo ha sido 129,921.

(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 3 segundo y 28 centésimas de segundo y el valor de la función objetivo ha sido 290,031.

Figura D.3: Mosaicos construidos a partir del modelo (3.6) cambiando la función objetivo por la de la fórmula (3.3).



(a) Teselas de tamaño $144 \times 144$ píxeles. Resolver el modelo ha costado 4 segundos y 58 centésimas de segundo y el valor de la función objetivo ha sido 29,514.



(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 1 minuto, 4 segundos y 14 centésimas de segundo y el valor de la función objetivo ha sido 60,99.

Figura D.4: Mosaicos construidos a partir del modelo (3.7) cambiando la función objetivo por la de la fórmula (3.3).

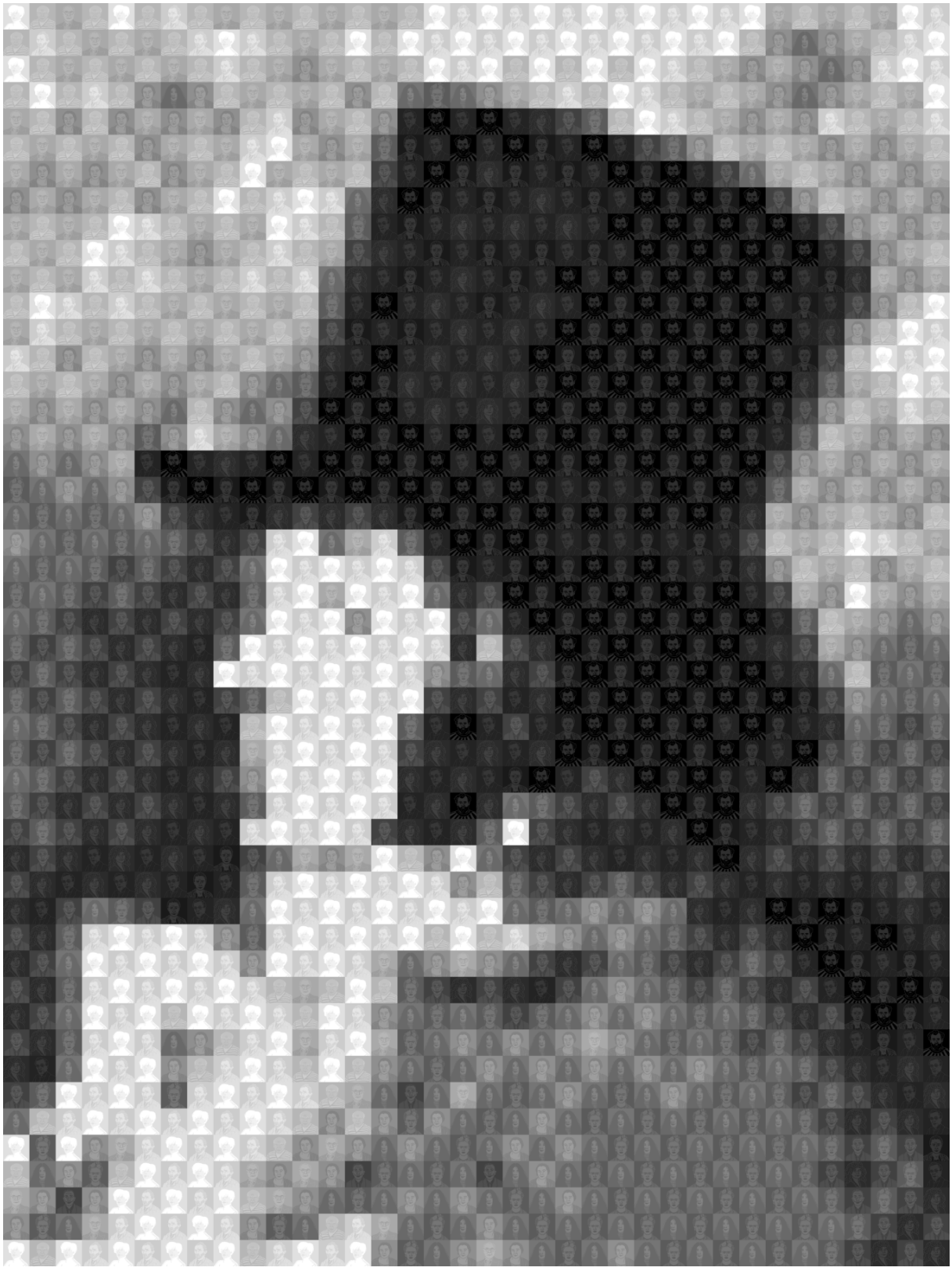
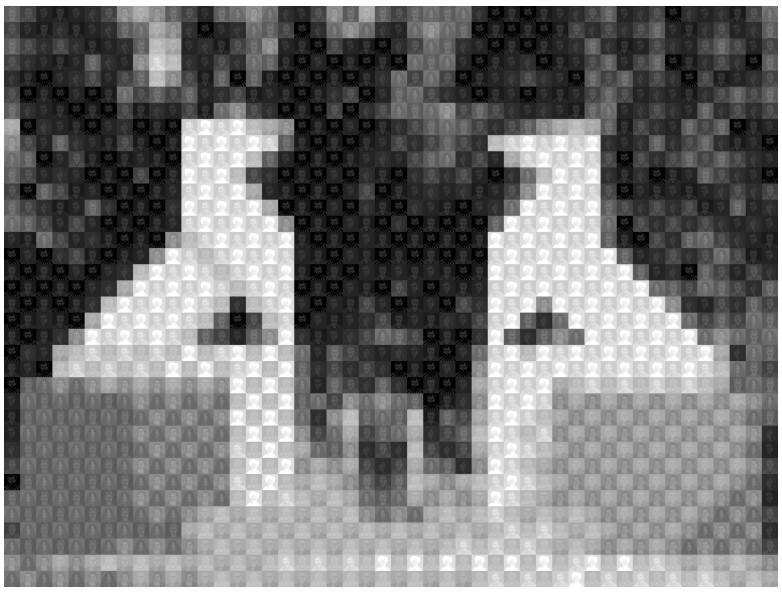
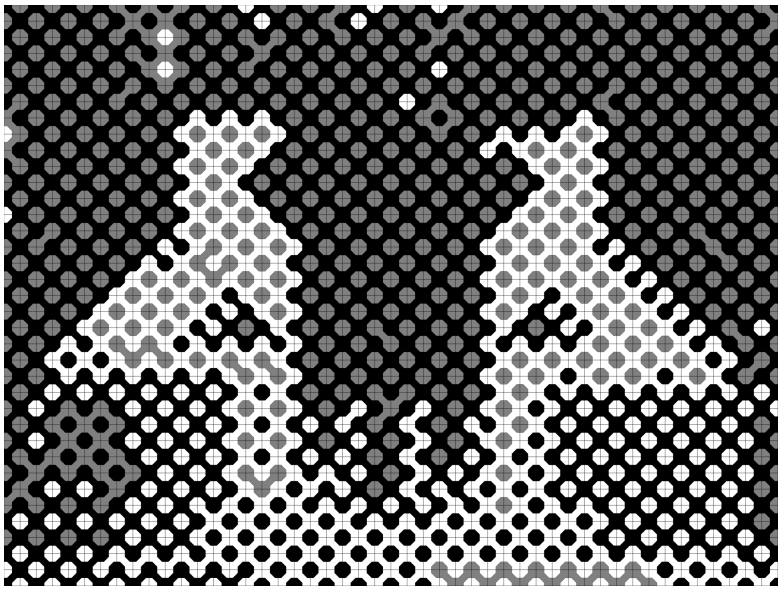


Figura D.5: Ampliación del mosaico que se encuentra en la parte derecha de la figura 3.5.



(a) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 2 segundos y 88 centésimas de segundo y el valor de la función objetivo ha sido 25,143.



(b) Teselas de tamaño $96 \times 96$ píxeles. Resolver el modelo ha costado 5 segundos y 92 centésimas de segundo y el valor de la función objetivo ha sido 14,193.

Figura D.6: Mosaicos construidos a partir de los modelos (3.6) (arriba) y (3.7) (abajo).