

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



**Foto-mosaicos y optimización para el  
arte (Opt-Art)**

TESIS

QUE PARA OBTENER EL TÍTULO DE  
**LICENCIADO EN MATEMÁTICAS APLICADAS**

PRESENTA

**JULIO CÉSAR ESPINOSA LEÓN**

ASESOR

**DR. EDGAR POSSANI ESPINOSA**

CIUDAD DE MÉXICO

2023

«Con fundamento en los artículos 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada **“Diseño, Construcción y Validación de un Modelo de Riesgo Crediticio”**, otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la Biblioteca Raúl Baillères Jr., la autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una contraprestación.»

**JULIO CÉSAR ESPINOSA LEÓN**

---

FECHA

---

FIRMA

*A mis padres,  
por su apoyo, dedicación, paciencia y amor.*

# Agradecimientos

Quiero agradecer a todas las personas que me acompañaron durante este camino

Al Dr. Edgar Possani, por sumarse al trabajo y apoyarme con tanta paciencia.

# Índice general

<b>1. Introducción</b>	<b>1</b>
<b>2. La programación lineal</b>	<b>3</b>
<b>3. TSP: Agente viajero</b>	<b>7</b>
3.1. Problema de Agente viajero, descripción general . . . . .	7
3.2. Arte con el agente viajero, o arte con línea continua . . .	9
<b>4. Fotomosaicos</b>	<b>12</b>
4.1. Fotomosaicos en escalas de grises . . . . .	12
4.2. Fotomosaicos en la triada RGB, o espacios de color . . .	14
4.2.1. Programa lineal . . . . .	14
4.2.2. Idea del color promedio . . . . .	16
<b>5. Implementación</b>	<b>20</b>
5.1. Partición de la matriz en submatrices . . . . .	20
5.2. Arte con el agente viajero . . . . .	24
5.2.1. Arte con el agente viajero, caso Twitter . . . . .	24
5.2.2. Arte con el agente viajero, complejidad y exploración de instancias más complejas . . . . .	28
5.3. Fotomosaicos . . . . .	31

5.3.1. Fotomosaicos en blanco y negro . . . . .	31
5.3.2. Fotomosaicos a color . . . . .	43
<b>Bibliografía</b>	<b>43</b>

# Capítulo 1

## Introducción

Históricamente, la optimización matemática ha sido una disciplina altamente recurrida para la solución de problemas en varias áreas del conocimiento y la práctica (economía, informática, industria, logística, etc.). Sin embargo, fuera de la amplia esfera de problemas técnicos o puramente científicos abordables por la optimización hay una serie de planteamientos resolvibles que concilian la optimización, particularmente del tipo lineal, con problemas de carácter estético más vinculados con ramas como el arte o la composición fotográfica. La presente tesina abordará el problema de reproducir o replicar con buena similaridad una imagen mediante el empleo, ya sea de trazos continuos a lo largo de un cuadro, o mediante imágenes diversas que, posicionadas estratégicamente, lograrán construir un producto que, a ojo humano, tendrá una forma fácilmente asociable con la imagen objetivo. Siendo ambos problemas de optimización lineal entera: el primero, una aplicación del problema del agente viajero (TSP por sus siglas en inglés); y el segundo, un problema lineal con variables de decisión en tres dimensiones (siendo éstas asociadas a renglón,

columna y tipo de imagen).

# Capítulo 2

## La programación lineal

La programación lineal es un área de la programación matemática y la optimización dedicado a minimizar o maximizar una función lineal sobre múltiples variables (multivariada) de tal manera que dichas variables se expresen mediante un sistema de restricciones también lineales en forma de igualdades o desigualdades. La expresión estándar de un problema lineal es la siguiente

$$\min_{x_i} \sum_{i=1}^n c_i x_i \quad (2.1a)$$

$$\text{s.a. } Ax \leq b \quad (2.1b)$$

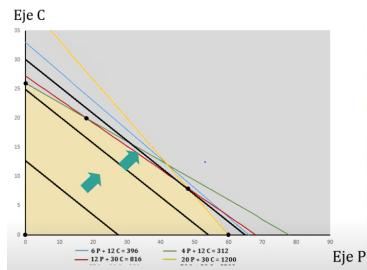
$$x \geq 0 \quad (2.1c)$$

Las restricciones mencionadas, expresadas matricialmente arriba, generan un espacio geométrico cerrado, más específicamente un poliedro convexo, y de contornos lineales, al cual se les denomina región de factibilidad.

La disciplina y teoría de la programación lineal se manifiesta en problemas que lo que buscan es maximizar un beneficio o minimizar un costo bajo circunstancias que restringen la usabilidad de recursos. Dicho beneficio o costo depende de las cantidades de dichos recursos. Algunos de sus primeros usos se remontan a la Segunda Guerra Mundial en la búsqueda de planificar gastos y recursos para reducir costos del enemigo y aumentar las pérdidas del enemigo. Otros usos, empleados a lo largo del siglo XXI consisten en problemas de flujos de redes en informática y de mercancías en comercio; así como economía y gestiones de inventario, portafolios financieros y suministro de alimentos.

Los programas lineales fácilmente se pueden garantizar que son resolvibles en la medida de que la región factible sea no vacía y acotada en la dirección del gradiente de la función objetivo. Gráficamente la solución se puede representar como el contacto de un poliedro convexo con las curvas de valor de una función objetivo cóncava. Sin embargo, al no haber convexidad estricta de la región factible ni concavidad estricta de la función, no se puede garantizar la unicidad de las soluciones; es más, para programas lineales no discretos puede haber hasta una infinidad de soluciones. Lo que sí se puede asegurar es que cuando menos un vértice de la región factible es una solución óptima. La siguiente gráfica da un ejemplo de un programa lineal en el espacio 2-dimensional donde las curvas de valor van desplazándose hasta tocar el vértice óptimo. Esta solución gráfica tiene una noción que extiende su alcance y validez a más dimensiones y una amplia variedad de restricciones lineales.

Las variaciones de esta clase de problemas extendidas a más dimensiones pueden resolverse mediante el método Simplex, que incorpora variables de holgura para convertir las desigualdades en igualdades y ejecuta operaciones lineales simples sobre la función



**(a) Ejemplo bi-dimensional de la resolución gráfica un problema lineal**

objetivo y las restricciones para encontrar el óptimo de un problema lineal. Este método irá conduciendo a quien lo resuelve de vértice en vértice hacia mejores soluciones cada vez hasta dar con el(s) óptimo(s).

Cuando se añade la integralidad de las variables a un programa lineal, el problema de abordar las propiedades geométricas del problema no entero (o relajado) directamente para resolución se vuelve más complejo, dado que pueden haber casos donde, si las soluciones del problema no íntegro son truncadas a enteros sin salirnos del área factible, se está aún lejos del óptimo entero, por lo que hay explorar distintas zonas de la región factible mediante algoritmos como el de ramificación y acotamiento.

El algoritmo de ramificación y acotamiento parte inicialmente de la resolución del problema lineal sin restricciones de enterez (también llamado problema relajado) y, dada esta primera solución, o primer óptimo, va incorporando dos restricciones sobre alguna variable no entera (no importa mucho cuál): una restricción con el valor de esta variable por encima del 'techo' del primer óptimo y otra por debajo del 'piso' del mismo, eligiéndose de ahí aquella restricción con la mejor

solución; en caso de que esta última solución sea entera, se detiene la ejecución de los pasos dando con un óptimo, de otra forma, se procede con la elección de una nueva variable no entera a ramificar. Dada la solución más reciente el algoritmo reproduce y evalua las ramificaciones de distintas variables fraccionales hasta dar con una solución con todas sus variables enteras.

# Capítulo 3

## TSP: Agente viajero

### 3.1. Problema de Agente viajero, descripción general

La teoría de grafos estudia las propiedades y relaciones de un grafo que es un par  $(V, E)$  con  $V$  un conjunto de vértices y  $E$  conjunto de aristas que los une. Siendo la arista modelada, ya sea como un par dirigido, donde el orden de los vértices importa (digrafo), o bien, no dirigido. Existen problemas de optimización planteados sobre esquemas de grafos que pueden ser resueltos por la vía de la programación lineal, siendo el problema del agente viajero (*Traveling Salesman Problem*, TSP) uno de los más interesantes y populares. Este problema se esboza partiendo de un individuo (agente) que desea visitar una cantidad  $n_*$  de ciudades partiendo de una ciudad origen y finalizando el recorrido en dicha ciudad con el objetivo de minimizar la distancia total recorrida. Existe un único conjunto de parámetros necesario para definir el problema: la distancia que hay entre cada par de ciudades existente; para ello defínase  $c_{i,j}$  como el valor cuantitativo

de dicha la distancia que hay entre las ciudades (o vértices)  $(i, j)$  para cada par  $(i, j) \in \{1, \dots, n_*\}^2$ . Explicado este problema, podemos desarrollar una intuitiva formulación matemática mediante la programación lineal entera siguiente.

$$\min_{x_{i,j}} \sum_{i=1}^{n_*} \sum_{j=1}^{n_*} c_{i,j} x_{i,j} \quad (3.1a)$$

$$\text{s.a. } \sum_{i=1}^{n_*} x_{i,j} = 1 \quad \forall j \in \{1, \dots, n_*\} \quad (3.1b)$$

$$\sum_{j=1}^{n_*} x_{i,j} = 1 \quad \forall i \in \{1, \dots, n_*\} \quad (3.1c)$$

$$\sum_{i \in S} \sum_{j \in S} x_{i,j} \leq |S| - 1 \quad \forall S \subset \{1, \dots, n_*\} \quad (3.1d)$$

Donde  $S$  es el conjunto potencia generado con  $\{1, \dots, n_*\}$ . El problema lineal esbozado arriba se conforma por componentes. Para entenderlo, defínase antes el conjunto de variables de decisión como  $\{x_{i,j} | (i, j) \in \{1, \dots, n_*\}^2\}$  con dos asignaciones de valor posibles: 0 y 1. 1 representa que el agente realizó un recorrido directo partiendo de la ciudad  $i$  hacia la ciudad  $j$  con una distancia correspondiente de  $c_{i,j}$ ; 0, por el contrario, representa el caso opuesto: no se viaja de  $i$  a  $j$ . En términos técnicos, hablamos de que el agente viajero ocupó (o no) el dígrafo dirigido  $(i, j)$ .

Así, la primera expresión ec. (3.1a) es la función objetivo: el recorrido total para arribar a las  $n_*$  ciudades. (3.1b) exige que de cada ciudad se parte hacia una única ciudad; (3.1c), análogo a (3.1b), requiere que a cada ciudad se arribe desde un único punto de partida. La factibilidad del *tour* generado por el agente se termina de construir en el conjunto de ecuaciones (3.1d), ya que sin éste podrían haber

*tours* donde a cada ciudad se llega una única vez y cada ciudad parte hacia un único destino, pero dos o más subconjuntos resultado de la partición de  $\{1, \dots, n_*\}$  generan cada uno un recorrido o *tour* cerrado. Supongamos que se generó una solución óptima con las restricciones (3.1b) y (3.1c) cumplidas, pero que constara de tres *subtours*:  $T_1, T_2, T_3$  con  $T_1 \cup T_2 \cup T_3 = \{1, \dots, n_*\}$ . Poniéndonos imaginativos, esto querría decir que el agente terminó de recorrer  $T_1$ , al terminar pasó a recorrer  $T_2$  sin haber algún digrafo que enlazara de algún punto de  $T_1$  a un punto de  $T_2$  (*¿entonces se teletransportó?*), finalmente siguió haciendo lo propio de  $T_2$  a  $T_3$ . Claramente esto realmente no tiene sentido; además, se viola el supuesto de que el agente parte de un punto y termina en el mismo tras recorrer todas las ciudades. La restricción (3.1d) elimina la posibilidad de lo que llamaremos *subtours*. Sin embargo, al tratarse de una restricción que explora a cada uno de los  $2^{n_*}$  posibles subconjuntos generables con  $\{1, \dots, n_*\}$ , es una restricción con una elevada complejidad computacional asociada que debe simplificarse con adecuadas restricciones alternativas (véanse restricciones de Miller, Tucker & Zemlin, 1960). Si a eso le sumamos que el programa lineal tiene como conjunto factible los  $n_*!$  posibles ordenamientos de  $\{1, \dots, n_*\}$ , podemos entender que se diga que los programas que lo resuelven trabajan bajo los llamados “algoritmos de optimización combinatoria”.

### 3.2. Arte con el agente viajero, o arte con línea continua

En la introducción se hace referencia a dos maneras de replicar imágenes con optimización lineal que se desarrollarán en el presente estudio. La primer versión es vía el TSP: parte del supuesto de que se

cuenta con una imagen en blanco y negro con una gama de *greyscales* (o escala de grises). Asociamos una mayor *greyscale* a una mayor luminosidad o blancura. Así pues, procedemos de la siguiente manera

- Particionamos la imagen en grupos de píxeles. Cada grupo constará de rectángulos con  $k$  píxeles a lo largo por  $h$  píxeles a lo ancho. Derivamos de la imagen  $m$  submatrices-renglón de grupos o rectángulos y  $n$  submatrices-columna de éstos, de tal manera que la matriz grande cuenta con dimensiones  $\mu' \times \nu'$  con  $\mu' = k m$  y  $\nu' = h n$ ; asimismo, obtenemos una escala de grises para cada pixel en el rango discreto de 0 a 255.
- Obtenemos la *greyscale* promedio de cada rectángulo y fijamos un parámetro que ponderará el nivel de detalle que deseamos para la imagen a reproducir:  $\gamma \in [4, 9] \cap \mathbb{N}$ . De tal manera, para el rectángulo  $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$  la *greyscale* del pixel promedio será de  $\psi_{ij}$ , con valores entre 0 y 255. De tal forma constrúyase

$$g_{ij} = \lfloor \gamma - \gamma \psi_{ij} / 255 \rfloor \quad (3.2)$$

Esta será una escala de oscuridad del rectángulo.

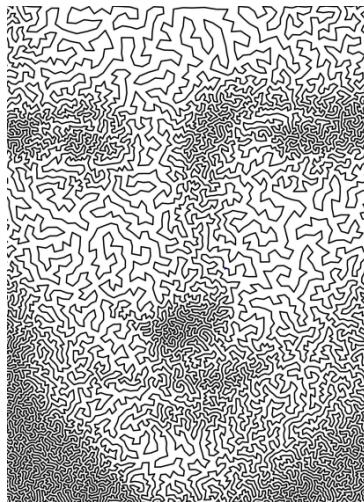
- Dividimos la imagen en  $m \times n$  rectángulos, en cada uno situamos uniformemente  $g_{ij}$  ciudades. El conjunto total de ciudades construidas será de cardinalidad  $\tau$ , definida como

<sup>0</sup>En adelante se emplearan indistintamente 'greyscales' o 'escala de grises'

<sup>0</sup>Este rango de [4, 9] no tiene otra justificación más allá de la visual, i.e. se define a la luz de los resultados usando esos números; por el contrario valores superiores a 9 podrían generar imágenes sobre-saturadas de puntos

$$\tau = \sum_{i=1}^m \sum_{j=1}^n g_{ij} \quad (3.3)$$

Computamos las distancias (o “costos”) entre las  $\tau$  distintas coordenadas en una matriz  $C = [c]_{ij}$  de  $\tau \times \tau$ ; resolvemos el TSP sobre estas distancias. Siendo que ahora el planteamiento se esboza sobre  $\tau$  ciudadesk podemos entender que ahora la  $n_*$  introducida en la sección anterior tiene un valor en el problema del arte con línea continua  $n_* = \tau$ .



**(a) Rostro de la Monalisa  
reproducido por Robert  
Bosch usando el problema  
del agente viajero**

# Capítulo 4

## Fotomosaicos

### 4.1. Fotomosaicos en escalas de grises

En el área del *marketing* han pululado técnicas de *advertising* que consisten en reproducir con elevada similaridad una imagen de interés conformada por un conjunto de imágenes o grupos de píxeles. Similar al ejercicio del TSP aplicado al cuadro de la Monalisa podríamos asignar una intensidad lumínica a cada celda del mismo (o cualquier otro) rectángulo, reescalando el parámetro  $\psi_{ij}$  al intervalo  $[0, 1]$  mediante  $\beta_{ij} = \psi_{ij}/255$ . Así, de manera similar al punto (1), supóngase que contamos con un conjunto  $F = \{1, \dots, cf\}$  de rectángulos (con  $cardinalidad(F) = cf$ ), cada rectángulo representando a, digamos, una fotografía. Cada imagen o figura de tipo  $f \in F$  cuenta con una intensidad lumínica con valor  $b_f$ . Además, supóngase que se cuenta con al menos un rectángulo con la fotografía  $f$  impresa, pero podemos tener más:  $u_f$  es la cantidad de rectángulos de tipo  $f$  disponible para su uso. De tal manera contamos con un total de  $\sum u_f = U \geq m \cdot n$  fichas disponibles. La formulación con

programación lineal del problema de los fotomosaicos es la siguiente.

$$\min_{x_{fij}} \sum_{f \in F} \sum_{i=1}^n \sum_{j=1}^m (b_f - \beta_{ij})^2 x_{fij} \quad (4.1a)$$

$$\text{s.a. } \sum_{i=1}^n \sum_{j=1}^m x_{fij} \leq u_f \quad \forall f \in F \quad (4.1b)$$

$$\sum_{f \in F} x_{fij} = 1 \quad \forall (i, j) \in \{1, \dots, m\} \times \{1, \dots, n\} \quad (4.1c)$$

La expresión ec. (4.1a) minimiza la suma de diferencias cuadráticas—o bien, de los cuadrados de las distancias euclídeas en  $\mathbb{R}^1$ —entre las luminosidades del área subyacente sobre la cual se coloca la ficha  $f$  en la posición  $(i, j)$  (Possani). La función de costos  $c_{i,j}$  del problema viajero ahora tiene un análogo tridimensional:  $c_{fij} = (b_f - \beta_{ij})^2$ . (4.1b) garantiza que se no se excedan las dotaciones disponibles de cada figura. La restricción (4.1c) se asegura que haya exactamente una foto en la posición  $(i, j)$ . Cabe destacar que, si bien este problema es de programación lineal entera (los valores factibles son solo 0 y 1), el mismo se puede plantear como un problema con variables de decisión continuas, dando lugar a un resultado óptimo entero y factible. Esto es una noticia buena en la medida de que añadir la propiedad de “entero” o “enterez” implica en la solución restricciones asociadas con una complejidad computacional y de planteamiento no siempre tan triviales. En otras palabras, la restricción de enterez dificulta el cálculo de soluciones.

## 4.2. Fotomosaicos en la triada RGB, o espacios de color

### 4.2.1. Programa lineal

Ahora, vale la pena analizar el cómo se ha de construir un fotomosaico usando el principio de optimización lineal planteado al inicio del capítulo, pero ahora en lugar de optimizar la asignación de *grayscale*, hacerlo mediante la asignación de imágenes a color. Para ello, vale la pena cuestionarse cómo se representa una imagen a color. En los niveles primarios de educación es común mostrar empíricamente (con acuarelas, digamos) cómo los colores se producen mezclando rojo, amarillo y azul, a esto se le llama en el modelo RYB, también conocido como un modelo de tipo sustractivo. Un modelo sustractivo parte del principio de que “el color de un objeto depende de las partes del espectro electromagnético que son reflejadas por él, o dicho de otro modo, de las partes del espectro que no absorbe”, el modelo RYB de esta clase es el más antiguo registrado. La manera en que el ojo humano interpreta un objeto específico depende no necesariamente de la luz que el objeto en sí emane, sino de las longitudes de onda que deja de absorber. Por el contrario, en la representación de un objeto por una pantalla se depende de la emisión de la luz por la misma, de ahí que se ha de emplear un modelo opuesto, i.e., el de la síntesis aditiva del color. Una ventaja que tiene la misma es que el color digital de un pixel procesado por un dispositivo puede ser representado como una adición los valores numéricos de cada componente que se van incorporando en un espacio 3-dimensional. El modelo aditivo consensuadamente más dominante en la informática y la fotografía es el que parte de los colores primarios rojo, verde y azul (RGB). Cabe

mencionar que, así como en las escalas de grises, en las imágenes a color cada componente puede representarse convencionalmente en escalas continuas entre 0 y 1 o discretas de 256 valores. Una extensión del principio de la solución de fotomosaicos sería minimizar la suma de los cuadrados de la distancias euclídeanas, pero esta vez ya no en  $\mathbb{R}^1$ , sino en  $\mathbb{R}^3$ . Para ello, la fórmula enunciada en (4.1a) puede modificarse incorporando tres componentes: la primera dimensión,  $r$  (que antes existía, sin embargo se obviaba la indexación porque el pixel se representaba en una única dimensión), la segunda,  $g$  y, finalmente,  $b$ .

Dicho esto, podemos ocupar los conceptos definidos inicialmente en este capítulo, pasando así de  $\beta_{ij}$  y  $b_f$  a  $\beta_{ij}^A$  y  $b_f^A$  donde el superíndice  $A$  refiere a “aditivo”,  $\beta_{ij}^A = (\beta_{ijr}, \beta_{ijg}, \beta_{ijb})$  es el color del pixel “promedio” (si se puede decir así, y en efecto se puede comprobar que semejante pixel existe) y  $b_f^A = (b_{fr}, b_{fg}, b_{fb})$  es el color de la ficha  $f$ . Habiendo definido esto, generamos el problema lineal siguiente

$$\min_{x_{fij}} \sum_{f \in F} \sum_{i=1}^n \sum_{j=1}^m [(b_{fr} - \beta_{ijr})^2 + (b_{fg} - \beta_{ijg})^2 + (b_{fb} - \beta_{ijb})^2] x_{fij} \quad (4.2a)$$

$$\text{s.a. } \sum_{i=1}^n \sum_{j=1}^m x_{fij} \leq u_f \quad \forall f \in F \quad (4.2b)$$

$$\sum_{f \in F} x_{fij} = 1 \quad \forall (i, j) \in \{1, \dots, m\} \times \{1, \dots, n\} \quad (4.2c)$$

Podemos simplificar el coeficiente  $(b_{fr} - \beta_{ijr})^2 + (b_{fg} - \beta_{ijg})^2 + (b_{fb} - \beta_{ijb})^2$  como  $d^2(b_f^A, \beta_{ij}^A)$  con  $d$  la norma euclídea en  $\mathbb{R}^3$ . Vale la pena observar que, si bien la fórmula se ve más compleja debido a la manera en que se desarrolla el

coeficiente de costo, en realidad el problema matemáticamente visto es en esencia idéntico con dos salvedades: primera, que el cálculo de los costos es más largo; segunda, que encontrar fichas que cubran de manera razonablemente similar el total del espectro de colores que hay en la imagen grande a representar ya no es tan sencillo, debido a que se cuenta con tres dimensiones. Así, por ejemplo, si antes se deseaba tener 10 imágenes con escalas de grises promedio homogéneamente distribuídas sobre la escala  $[0, 1]$  ahora se necesitarían  $10^3 = 1000$  imágenes con escalas distintas para cumplir con una tarea análoga, por lo tanto la cardinalidad del conjunto de fichas  $F$  debe de aumentar considerablemente si se desea generar una representación legible. Dadas estas salvedades, el problema matemático es esencialmente idéntico y solo debe su diferencia a la implementación computacional inherente al mencionado aumento en la cardinalidad del conjunto de fichas a colocar.

#### 4.2.2. Idea del color promedio

Hace algunos párrafos se aseguraba anticipadamente que sí existe el concepto de un color dominante en una imagen. Pues bien, desde la década de 1970 existe una teoría en torno a la cuantificación del color que busca comprimir un conjunto de valores a un único valor y que dió luz a nociones que hoy se manejan ampliamente, tales como el de la existencia de la popular extensión de archivo digital de nombre *JPEG*. Hay dos conceptos centrales en torno a la cuantificación del color. El primero es que éste se usa para reducir costos computacionales, en especial en lo referente a dispositivos con una capacidad limitada, al reducir optimalmente la información desplegada por un archivo de imagen (tal es el caso del *JPEG*). El segundo, es que permite a uno entender las variedades de color para categorizar el color de un



(a) Imagen original



(b) 2 tipos de píxeles



(c) 3 tipos de píxeles



(d) 10 tipos de píxeles

**Figura 4.1. Simplificación de imágenes basado en selección de píxeles-centroides del modelo k-means**

producto o bien desplegar una paleta optimizada derivada de una imagen.

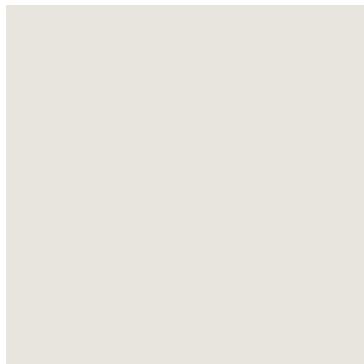
Para el primer caso, destaca el algoritmo de *kmeans* que permite usar exactamente la misma dimensión (*renglon × columna*) de píxeles pero donde la totalidad de valores de los mismos en el hiperplano *RGB* sea limitada y arbitrariamente asignada. Así, si asigno  $k = 2$  se tiene un tipo de imagen; un refinamiento ocurriría si se usa  $k = 3$ ; y mucho mayor sería este si se pasa a  $k = 10$ . Dada la  $k$ , el algoritmo va a definir esta cantidad de centroides para agrupar la totalidad de píxeles. Se desplegará cada centroide reemplazando al pixel original dando lugar a una imagen con la misma dimensión pero menor variedad en sus colores. El caso trivial es cuando  $k$  equivale al total de distintos valores de píxeles, pues entonces tendríamos como resultante la misma imagen que la original. A continuación se desarrolla esto, en un ejemplo con tres despliegues cada uno con una cantidad de  $k$ 's distinta.

La imagen original cuenta con un despliegue de  $517 \times 517$  pixeles. Esto da a un total de  $517^2 = 267,289$  unidades de información. Sin embargo, solo existen 58,168 valores de pixeles, lo cual implica que uno o más de éstos se repiten. De estos valores, solo 22 tienen más de 500 pixeles cada uno; es decir, tienen más del 0.2% del total de la información. Lo relevante se pone de manifiesto al observar que de estos 22 valores, todos tienen valores de  $r$ ,  $g$  y  $b$  superiores a cero. Lo cual quiere decir que son diferentes escalas del blanco ¿por qué entonces no agruparlos en un mismo cluster, junto quizás con otros valores en una vecindad cercana? Así pues, los ejercicios arriba expuestos simplifican esta diversidad de pixeles a solo 2, 3 o 10, según sea el caso.

Ahora, la asignación de la  $k$  depende de la clase de problema a abordar. En general para *kmeans* (en aplicaciones más allá de la cuantificación del color) existen varios algoritmos que permiten encontrar una  $k$  que satisfaga algún criterio de información. Como la participación del algoritmo *kmeans* en este escrito es meramente auxiliar y complementario para ilustrar el problema de optimización en la asignación de fichas de color, se desea simplificar su uso. Para el caso específico de interés, a más homogénea es una imagen, más sencillo resulta anticipar que una  $k$  baja será suficiente para desplegar la imagen con aceptable precisión. El lector puede imaginarse una imagen con fondo verde (un jardín) o blanco (retratos para identificación), como es común en la fotografía. La presencia del fondo y su contraste con el objeto central (considéresele “ruido”) dará lugar a una pérdida de información si asignamos un  $k$  muy pequeña. Por lo tanto, para efectos de simplicidad, en el capítulo de implementación será importante que la ficha a asignar sea una imagen con colores homogéneos, para así elegir el valor del centroide con más población de pixeles asociada y llamarle el “pixel promedio” de la ficha. Tantas

imágenes con atributos similares al de la flor blanca (simples, uni-color, con un fondo nulo o irrelevante para minimizar el ruido) se pueden imaginar como colores de flores hayan. En el capítulo de la implementación esta inventiva se llevará a cabo análogamente. Bajo esta idea existirá un “pixel promedio” en cada una de las  $mn$  celdas que posicionadas contiguamente conforman la imagen objetivo.

En relación a la imagen de la flor blanca, desglosemos el color dominante derivado de  $k = 2$ , este color ya se conoció anteriormente en la figura 4.1b, pues es aquél visualmente más abundante de ésta.



(a) **Color promedio de flores blancas derivadas de**

$$k = 2$$

# Capítulo 5

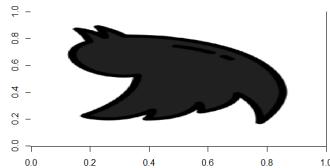
## Implementación

### 5.1. Partición de la matriz en submatrices

Una imagen en escala de grises se entiende como un despliegue visual en el que cada unidad básica mostrada en la pantalla —mejor conocida como pixel— se ve representada por una cifra que cuantifica la cantidad de luz (o información de intensidad lumínica) con que cuenta. Representando la mínima unidad al negro y, la máxima, al blanco. La escala tiende a volverse discreta, ya que contar con un continuo estándar de 0 a 1 (o 0 % a 100 %) representaría un reto computacional y de almacenaje considerable. Algunas imágenes en escala de grises tempranas cuantificaban hasta 16 valores, lo cual implicaba un almacenaje de 4 bits por pixel; en la medida de que la rama de la fotografía digital evolucionó, aumentó hasta 256 intensidades, implicando un peso de 8 bits por pixel. Si bien actualmente hay adicionales variaciones con mayor o menor número de valores (también discretos), los cuales se eligen en función de la aplicación de la imagen, un estándar recurrentemente usado es el de

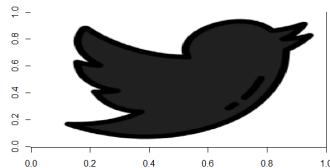
256 escalas. Por eso, y en aras de seguir convenciones, se usará esta escala discreta, o bien, dependiendo del caso, la escala continua de 0 a 1. Para pasar de la escala  $[0, 1]$  a la escala  $\{0, \dots, 255\}$  hacemos  $e_{255} = \text{round}(255e_1)$ ; para pasar en sentido inverso se usará  $e_1 = \frac{1}{255}e_{255}$ .

En este apartado se describirá cómo se procede para formular y computar los dos enfoques para reproducir imágenes: el de TSP y el de los fotomosaicos, explicados brevemente en el capítulo 3. Para iniciar los ejercicios decidí usar una imagen de la red social Twitter, la cual se anexa a continuación. La librería de R encargada de procesar y graficar las imágenes de tipo png tiene como nombre “png”. La función *readPNG()* se encarga de leer el archivo-imagen objetivo y sintetizarlo en un objeto numérico estructurado en un arreglo 3-dimensional con dimensiones para sus pixeles de renglón ( $\mu$ )  $\times$  columna ( $\nu$ )  $\times$  3. De los elementos extrapolados del primer componente de la dimensión tres, *readPNG()[, , 1]*, se extrae una matriz  $\mu \times \nu$ , la cual consta de valores constituyendo la escala de grises de la imagen en el intervalo continuo de 0 a 1. Para graficar dicha imagen en el cuadrante del plano cartesiano  $x - y$  (delimitado al rango  $[0, 1]$  para ambas dimensiones) se usará la función *image()*. Esta función despliega la figura interpretando al renglón como la coordenada  $x$  y a la columna como la coordenada  $y$ . Sin embargo, esta es una interpretación distinta a la asignación que interpreta a los renglones como el largo (es decir, el eje  $y$ ) y a las columnas como el ancho (es decir, el eje  $x$ ) propia del output de *readPNG()[, , 1]*. De tal manera, si se computa la escala de grises de este output mediante *image()* se tendría como resultado la imagen rotada  $90^\circ$  en sentido contrario a las manecillas del reloj (hacia la izquierda), como se muestra en la siguiente imagen.



**Figura 5.1. Despliegue `image(.)`, matriz original**

Para solventar este problema se usará una versión auxiliar de la matriz original, rotándola  $90^\circ$  en sentido de las manecillas del reloj. El resultado se despliega a continuación.



**Figura 5.2. Despliegue `image(.)`, matriz rotada 90 a la derecha**

El siguiente paso consiste en usar las dimensiones de píxeles,  $m$  y  $n$ , para delimitar el tamaño de la matriz tal que se permita generar una partición de la misma en submatrices que sea factible, es decir, que sean submatrices de dimensiones idénticas y que, posicionadas estratégicamente, generen la matriz grande (o matriz total). Para ello se requiere que la cantidad de renglones de la nueva matriz acotada sea 0 módulo  $m$  y, la cantidad de columnas, 0 módulo  $n$ . Si bien puede haber más de una combinación renglón  $\times$  columna consistente con las propiedades mencionadas, para nuestra regla de recortar la matriz mayor se escogerá una nueva dimensión  $\mu' \times \nu'$  que corresponda al máximo valor factible que no exceda  $\mu \times \nu$ .

$$DIM = \{(\mu', \nu') = (\max(a), \max(b)) | 0 = a \text{mod}(m), 0 = b \text{mod}(n), a \in [0, \mu] \cap \mathbb{N}, b \in [0, n] \cap \mathbb{N}\} \quad (5.1)$$

Es trivial notar que se requiere que  $m \leq \mu$  y  $n \leq \nu$ . El acotamiento arriba expresado se realiza de la manera más simétrica posible. Esto es, si  $\mu' < \mu$ , se recortarán  $\epsilon = \lfloor \frac{1}{2}(\mu - \mu') \rfloor$  renglones por arriba y  $\delta = (\mu - \mu') - \epsilon$  renglones por abajo. Así, se tendrá  $\epsilon - \delta \in \{0, 1\}$  (0 si  $\frac{1}{2}(\mu - \mu')$  es par, 1 e.o.c), lo cual quiere decir que, en el peor caso, se recortará un pixel más por un lado que por otro. Análogamente, si  $\nu' < \nu$ , se recortarán  $\rho = \lfloor \frac{1}{2}(\nu - \nu') \rfloor$  columnas por la izquierda y  $\sigma = (\nu - \nu') - \rho$  por la derecha (siendo también la diferencia de, a lo mucho, uno). El siguiente paso consiste en convertir la matriz de la escala  $[0, 1]$  a la escala  $\{0, \dots, 255\}$ . Luego, se procede a fraccionar la matriz en un conjunto de  $m \times n$  submatrices mediante la función *matsplitter(.)* de R, la cual recibe como entradas la matriz a fraccionar, la cantidad de renglones por submatriz,  $k = \mu'/m$  y la cantidad de columnas,  $h = \nu'/n$ . *matsplitter(.)* entregará una lista de  $m \times n$  elementos. Los elementos 1 a  $n$  de la lista consistirán en las primeras  $n$  submatrices superiores de la matriz, los elementos  $n + 1$  a  $2n$  consistirán en las segundas  $n$  submatrices superiores. Así sucesivamente, hasta llegar a las  $n$  submatrices inferiores, enumeradas  $n \times m - n + 1$  a  $n \times m$ . En el supuesto de que cada submatriz representara un único valor numérico (o arreglo  $1 \times 1$ ) dentro de una matriz  $m \times n$ ; entendemos que este tipo de ordenamiento es de clase *rowwise* (o por renglón), ya que se va llenando renglón por renglón, empezando por los renglones de arriba y continuando hacia abajo. Cada renglón se llena de izquierda a derecha.

El otro tipo común de ordenamiento es *columnwise*, o por columna. Conociendo esto, podemos calcular la luminosidad promedio de cada elemento del output de *matsplitter(.)*. Dicho conjunto de luminosidades puede sintetizarse en un vector de dimensión  $m \times n$ .

## 5.2. Arte con el agente viajero

### 5.2.1. Arte con el agente viajero, caso Twitter

Tras haber seguido los pasos de la sección 5.1 lo siguiente consiste en convertir la escala del conjunto  $\{0, \dots, 255\}$  a la escala  $\{0, \dots, \gamma\}$ , empleando la ecuación, ec. (3.2) desglosada en la sección 3.2 de la presente tesina. Así pues, obtenemos un vector de dimensión  $m \times n$ , llámese  $g$ . Para el primer ejercicio, asígnese  $\gamma = 7$ . Bajo esta  $\gamma$ , las escalas primera a treintaiseisava de luminosidad coinciden con la escala 7 en oscuridad (o siete “ciudades”). Las siguientes luminosidades (por orden) corresponden a seis ciudades; sucesivamente, hasta llegar a la escala 255, que corresponde a cero ciudades. Es obvio que situar más puntos, o ubicar más aristas que conecten éstos en un rectángulo generará una mayor oscuridad en el mismo, de ahí que se considere a  $\gamma$  como una escala de oscuridad. Lo siguiente consta en ubicar las ciudades en el plano  $x - y$ , haciendo un loop a través de las  $m \times n$  matrices.

1. La asignación del  $k$ -ésimo elemento de la lista  $\{1, \dots, m \times n\}$  a una coordenada de matriz  $(i, j)$  se rige por el siguiente pseudocódigo, el cual tiene como base el tipo de arreglo *rowwise*.

---

<sup>0</sup>Arreglo por columna, o *columnwise* donde se llenan las columnas de izquierda a derecha, cada columna se llena de arriba a abajo

$$mod2 = k \bmod(n)$$

si  $mod2 = 0$

entonces  $i = mod2$

e.o.c.

$$i = mod2$$

$$j = \frac{(k-i)}{n} + 1$$

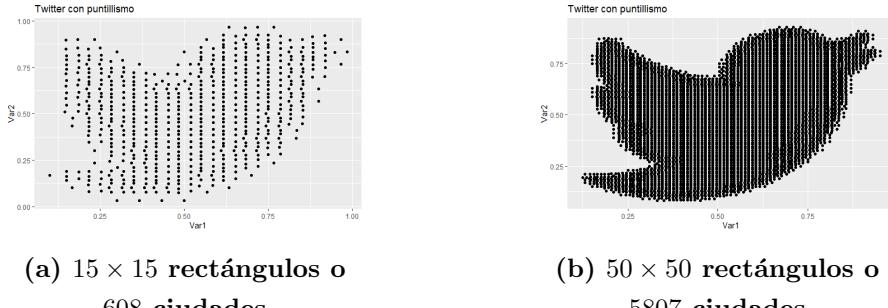
2. Una vez que tenemos la asignación  $k \mapsto (i, j)$ ,  $(i(k), j(k))$ , el siguiente paso es plasmar los puntos que asemejarán a la imagen objetivo sobre el rectángulo  $[0, 1] \times [0, 1]$  en el plano  $x - y$ . Para ello, vale la pena considerar que el extremo superior izquierdo de la imagen corresponde al elemento  $(1, 1)$  de la matriz y a la coordenada  $(0, 1)$  del plano  $x - y$ . Partiendo de aquí, deducimos la posición del extremo superior izquierdo de la coordenada genérica  $(i, j)$  de la matriz. La coordenada  $x$  corresponde al pixel con la columna  $j$ , representando un paso de  $\frac{(j-1)}{n}$  a la derecha de  $x = 0$ . La coordenada  $y$ , con el renglón  $i$ , corresponde su ubicación en el eje  $y$  a un paso de  $\frac{(i-1)}{m}$  unidades abajo de  $y = 1$ . Bajo esta formulación, derivamos que cada rectángulo (asociado a una submatriz) tendrá un ancho  $\frac{1}{n}$  y un largo de  $\frac{1}{m}$ . Bajo este planteamiento se puede demostrar que para todo par de coordenadas de matriz de tipo  $\{(i, j), (i + 1, j)\}$  o  $\{(i, j), (i, j + 1)\}$  existirá una relación de adyacencia entre sus correspondientes rectángulos. Además, las esquinas de la figura que envuelve a este conjunto de  $m \cdot n$  rectángulos serán las coordenadas  $\{(0, 1), (1, 1), (0, 0), (1, 0)\}$ . Con esto, garantizamos que los rectángulos mencionados conforman una partición del cuadrado  $[0, 1] \times [0, 1]$
3. El siguiente paso es usar el valor entero  $g_k = g[k]$  para colocar

esa cantidad de ciudades en el rectángulo asociado a la coordenda  $(i(k), j(k))$ . Una alternativa inmediata es colocarlas aleatoriamente, lo cual muy probablemente permitiría reproducir la imagen con buena similaridad. Sin embargo, se propuso hacer una asignación de coordenadas más razonada que obedece a la idea de hacer una buena distribución de las ciudades a lo largo y ancho de su correspondiente rectángulo. Con una “buena” distribución se busca dar a entender que se quiere evitar una excesiva concentración de las ciudades en una zona del rectángulo, dejando a otras zonas muy blancas (o sin ciudades). La idea propuesta, pues, es generar un *grid* de coordenadas  $\{x_0, x_1, \dots, x_{10}\}$  donde  $x_0$  y  $x_{10}$  corresponden a los extremos izquierdo y derecho del rectángulo y  $x_{l+1} - x_l = q \forall l \in \{0, \dots, 9\}$  ( $q = \frac{1}{10n}$ ); otro grid para las coordenadas  $y$ :  $\{y_0, y_1, \dots, y_{10}\}$ , con  $y_{p+1} - y_p = r \forall p \in \{0, \dots, 9\}$  ( $r = \frac{1}{10m}$ ) con  $y_0$  y  $y_{10}$  extremos superior e inferior. Con esto, ya habremos construido el *grid* 2-dimensional  $G = \{(x_i, y_i) | (i, j) \in \{0, \dots, 10\}^2\}$  que consiste en 121 coordenadas. Sobre este conjunto de coordenadas aplicar un *k-means* con  $g_k$  centroides. Una buena asignación de ciudades que cumple la condiciones deseadas (y previamente descritas) es aquella que asocia un centroide a una ciudad. Se procede a guardar esta asignación de ciudades y añadirlas a la selección correspondiente a submatrices anteriores (si es que hay anteriores).

Al iterar este proceso las  $m n$  veces, se obtienen las  $\sum_{k=1}^{m n} g_k = \tau$  ciudades objetivo y podemos resolver el TSP sobre éstas. Los resultados

<sup>0</sup>esta es una expresión equivalente a la ecuación ec.(3.3) , pero en la versión que mapea  $(i, j) \rightarrow k(i, j)$

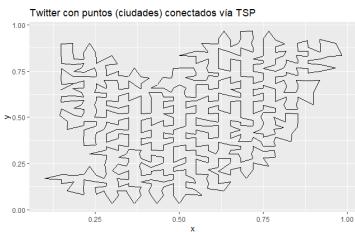
de escribir todas las ciudades como puntos bajo los ejercicios con  $15 \times 15$  rectángulos y (608 ciudades) y con  $50 \times 50$  (5807 ciudades) se desglosan a continuación.



**Figura 5.3. Gráfica de los puntos sobre el cuadrante  $x - y$**

Existe un tipo de archivo cuya extensión es “.tsp”. Éste se basa en el conjunto de ciudades sobre el que se ha de resolver una instancia del problema lineal. Dicha solución puede procederse mediante distintos métodos o heurísticas y puede realizarse mediante distintos *softwares*, algunos de los cuales son R, Python, Java y Concorde. En aras de practicidad, y en vista de que hasta el momento la construcción de los rectángulos y las ciudades se ha llevado a cabo con R, se usará este lenguaje para manipular tanto objetos como archivos de tipo TSP. Para este efecto se usará la librería homónima cuya función, *ETSP()* debe su nombre a *Euclidean TSP* y convierte el *data frame* de 2 columnas (una por coordenada) y  $\tau$  renglones (uno por ciudades) en un problema del agente viajero. El *Euclidean* obedece a que, en otros potenciales ejercicios del agente viajero, las coordenadas podrían ser de otra métrica sobre un espacio normado o incluso, coordenadas terrestres. La función encargada de resolver un *etsp* es *solveTSP()*. Si bien se le puede ingresar el método o algoritmo a emplear como

entrada de la función, el que se usa por *default* es el algoritmo de inserción arbitraria con refinamiento de tipo *two opt*. El tipo de datos del *output* es doble: uno es de la clase *tour* y otro es entero. La versión como entero consiste en el orden de las ciudades que se anidan para construir el *tour*. Con ello se puede graficar la solución con ayuda de *ggplot()*. Los resultados con  $fv15 \times 15$  y  $50 \times 50$  son los siguientes.



(a)  $15 \times 15$  rectángulos o  
608 ciudades; solución vía  
TSP



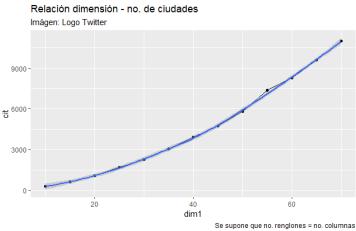
(b)  $50 \times 50$  rectángulos o  
5807 ciudades; solución vía  
TSP

**Figura 5.4.** Gráfica de las líneas uniendo a los puntos de manera eficiente sobre el cuadrante  $x - y$

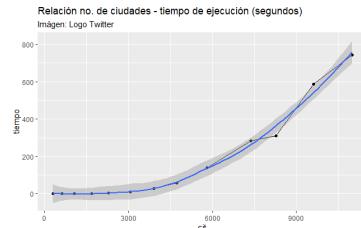
### 5.2.2. Arte con el agente viajero, complejidad y exploración de instancias más complejas

Un siguiente paso interesante consiste en evaluar la potencia computacional del solver del *tsp* en *R* que se ha ocupado (módulo *tsp*, función *solveTSP()*) al registrar el tiempo que lleva resolver una determinada imagen conforme se agregan más dimensiones o rectángulos y, por conseciente, más ciudades. Las siguientes gráficas reflejan la relación que tiene la dimensión (suponiendo que hay tantos rectángulos a lo largo como a lo ancho) con el número de ciudades en la imagen de Twitter; asimismo describen la relación del número de

ciudades con el tiempo de solución del tsp; claramente esta relación es de una complejidad superior a la lineal.



(a) Dimensión: no.  
ciudades



(b) no. ciudades vs tiempo

La siguiente es la tabla asociada a la gráfica anterior. Vale la pena mencionar que la columna de restricciones supone la técnica de reducción de restricciones a los subtours de Miller-Tucker-Zemlin que asumen que son  $n^2$  y no  $2^n$  de estas restricciones

Renglones (= Columnas)	Total ciudades	Tiempo de ejecución (s)	Variables	r
10	286	0.00	81796	
15	608	0.03	369664	
20	1051	0.25	1104601	
25	1670	1.35	2788900	
30	2284	4.65	5216656	
35	3058	10.26	9351364	
40	3894	28.77	15163236	
45	4723	57.33	22306729	
50	5807	139.93	33721249	
55	7360	283.53	54169600	
60	8271	310.83	68409441	
65	9616	586.88	92467456	
70	10987	744.08	120714169	

Una cuestión que vale la pena abordar es la relativa a si el algoritmo programado es capaz de reproducir imágenes de un grado más elevado de complejidad, o se limita a ser aplicable solo a imágenes sin relieve, ni dimensión, ni variedad de colores. Para este efecto buscaré reproducir una foto mía en mi escritorio de casa, la foto tendrá por nombre “*home office*” y es la siguiente

Lo que sigue es reproducir la imagen. Para esto, se hizo uso de un truco: atenuar los colores más claros. Es decir, a partir de un parámetro determinado de escala de luminosidad en adelante se aclarará la escala al máximo valor posible, es decir, 1. Para esta imagen el parámetro elegido es de 0.70. Esto permitió que la imagen resultara más distinguible, sobre todo en lo que concierne a las facciones faciales: cara, ojos, boca y al implemento electrónico que se tiene sobre las orejas (audífonos). La razón de este recurso puede aclararse si se muestra el resultado de ejecutar el programa sin hacer



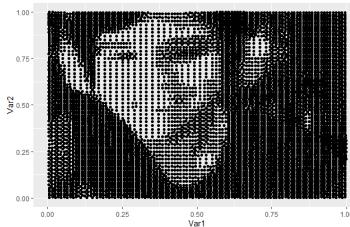
**Figura 5.6. Catálogo de imágenes seleccionadas para el fotomosaico**

esta modificación de la luminosidad. Ambos resultados se muestran en los dos pares de imágenes siguientes.

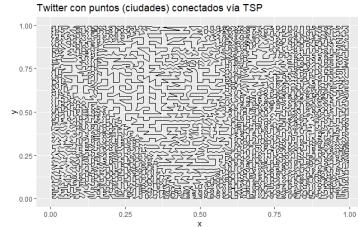
### 5.3. Fotomosaicos

#### 5.3.1. Fotomosaicos en blanco y negro

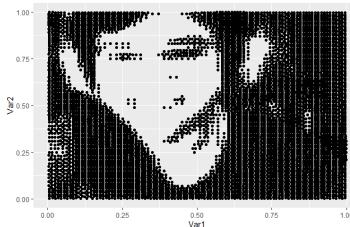
Posteriormente a reproducir los pasos descritos en la sección 5.1, se procede a convertir la escala del conjunto  $\{0, \dots, 255\}$  a  $[0, 1]$  dividiendo entre 255 (ver primer párrafo de la sección referida). Llámese  $h$  al vector resultante de extraer estas escalas promedio de las submatrices ordenadas con base en el esquema *rowwise*. Supóngase que las dimensiones de las imágenes que compondrán el fotomosaico son cuadradas, de valor  $pi \times pi$ . Entonces, generaremos tres arreglos importantes: uno de parámetros de costos, dos de escalas de grises. El



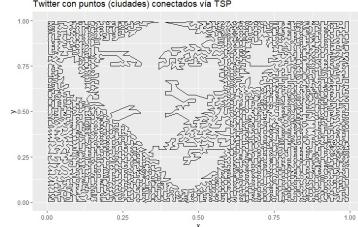
**(a) Ciudades o puntillismo con imagen *home office*:**  
9768 ciudades



**(b) TSP con imagen *textithome office*:**  
9768 ciudades



**(a) Ciudades o puntillismo con imagen *home office* atenuada:**  
8726 ciudades



**(b) TSP con imagen *home office* atenuada:**  
8726 ciudades

primero será uno numérico 3-dimensional de tipo  $cf \times m \times n$ , llámese  $Arr3$ , con  $Arr3[f, i, j] = c_{fij}$ , donde el lado derecho es el mismo ponderador de costos de la expresión (4.1a). Uno de los arreglos con *greyscales* será de dimensión  $m \times n$ , tendrá por nombre  $mp$ , por “matriz perfecta” el cual es un eufemismo por tratarse de la mejor representación factible de una imagen con las dimensiones que se le entregan; el otro, será un arreglo de  $(m \text{ pi}) \times (n \text{ pi})$  y será  $MF$ , por “matriz final”. Además, se contará con una dotación de  $\sum_{f=1}^{cf} u_f$  imágenes de solo  $cf$  tipos, cada uno denotado mediante  $Mgf$  por las iniciales de “matriz de *grayscale*” de la imagen tipo  $f$ . A continuación se describe el bucle a través de  $m \text{ } n$  iteraciones que es la base para

llenar los arreglos mencionados.

1. Tomamos el número de iteración,  $k$ , y le asignamos su correspondiente coordenada matricial,  $(i(k), j(k))$  deducida por el esquema *rowwise* (previamente descrito, ver pseudo-algoritmo del primer paso del *loop* en el capítulo 5.2)
2. Obtener la escala  $[0, 1]$  de luminosidad del elemento  $(i, j)$ ,  $h_k$ ; es decir, el valor promedio de la correspondiente submatriz.
3. Obtener la diferencia cuadrática de las luminosidades; es decir, la que hay entre  $h_k$  y cada elemento del vector de luminosidades promedio de las  $|F|$  figuras disponibles, llámese *lumP*. Esto es hacer  $c_{ij} = [(h_k - \text{lumP}_f)^2]_{f \in F} \in [0, 1]^{cf}$ . Asignar el valor de este vector a los elementos  $\text{Arr3}[i, j]$  del arreglo *Arr3*.
4. Asignar el valor  $h_k$  al elemento  $mp[i, j]$  de la matriz perfecta.

Posteriormente, hay que construir un vector que cuantifique las dotaciones disponibles de las  $cf$  figuras. Para ello, nos valemos de dos parámetros: uno,  $hd$ , de 'holgura', que nos dirá qué tantas más de las fichas estrictamente necesarias tendremos. Si  $hd = 0.1$ , tendremos aproximadamente 10 % más de las fichas exactamente suficientes.

Si deseáramos que las disponibilidades de cada ficha fueran uniformes, tendríamos  $\lceil \frac{m n(1-hd)}{cf} \rceil$  imágenes de cada tipo. Sin embargo, para darle mayor variabilidad a estas disponibilidades introducimos  $vd$ , un parámetro de variabilidad que nos permitirá tener una dotación de la ficha  $f$  extraída aleatoriamente de entre todos los enteros en  $u_f \in \{\lceil \frac{m n(1-hd)}{cf} (1 - vd) \rceil, \dots, \lceil \frac{m n(1+hd)}{cf} (1 + vd) \rceil\}$ . A mayor  $vd \in [0, 1]$ , mayor será la varianza del vector  $u$  de disponibilidades.

Con los pasos 1 a 4 del *loop* y la construcción de  $u$  que le es posteriormente descrita se habrán generado los parámetros necesarios para definir el modelo. El siguiente paso radica en reestructurar la información plasmada en el arreglo  $Arr3$  en un *data frame*,  $df3$  que consiste en los campos:  $t, i, j, f, ce$  y cuya forma genérica se muestra a continuación.

t	f	i	j	costo (ce)
1	1	1	1	A[1,1,1]
2	1	1	2	A[1,1,2]
:	:	:	:	:
n	1	1	n	A[1,1,n]
n+1	1	2	1	A[1,2,1]
n+2	1	2	2	A[1,2,2]
:	:	:	:	:
2n	1	2	n	A[1,2,n]
:	:	:	:	:
(m-1)n+1	1	m	1	A[1,m,1]
(m-1)n+2	1	m	2	A[1,m,2]
:	:	:	:	:
mn	1	m	n	A[1,m,n]
mn+1	2	1	1	A[2,1,1]
mn+2	2	1	2	A[2,1,2]
:	:	:	:	:
mn+n	2	1	n	A[2,1,n]
mn+n+1	2	2	1	A[2,2,1]
mn+n+2	2	2	2	A[2,2,2]
:	:	:	:	:
mn+2n	2	2	n	A[2,2,n]
:	:	:	:	:
mn+(m-1)n+1	2	m	1	A[2,m,1]
mn+(m-1)n+2	2	m	2	A[2,m,2]
:	:	:	:	:
2mn	2	m	n	A[2,m,n]
:	:	:	:	:
(cf-1)mn+1	cf	1	1	A[cf,1,1]
(cf-1)mn+2	cf	1	2	A[cf,1,2]
:	:	:	:	:
(cf-1)mn+n	cf	1	n	A[cf,1,n]
(cf-1)mn+n+1	cf	2	1	A[cf,2,1]
(cf-1)mn+n+2	cf	2	2	A[cf,2,2]
:	:	:	:	:
(cf-1)mn+2n	cf	2	n	A[cf,2,n]
:	:	:	:	:
(cf-1)mn+(m-1)n+1	cf	m	1	A[cf,m,1]
(cf-1)mn+(m-1)n+2	35cf	m	2	A[cf,m,2]
:	:	:	:	:
m n cf	cf	m	n	A[cf,m,n]

Como se muestra, el ordenamiento de  $df3$  sigue la jerarquía  $f, i, j$ . Con ayuda de esta tabla, podemos definir el modelo en un objeto de clase “lp”, auxiliándonos en el módulo *lpSolve* del lenguaje *R*. Dicho objeto se basa en (a) un vector de pesos,  $ce$ , (b) una instrucción de optimización,  $io$ : “minimiza” o “maximiza”, (c) una matriz  $RM$  de restricciones con dimensiones  $re \times (m \ n \ f)$  que representa a las  $re$  restricciones existentes sobre las variables (d) un vector de lados derechos,  $rh \in \mathbb{R}^{re}$  con el valor contra el cual será contrastada la combinación lineal de  $x$ ,  $RM[row,] \cdot x$ , (e) un vector de caracteres,  $sign$ , con el cual la restricción  $t$ -ésima será  $RM[row,] \cdot x \ "sign[t]" \ rh_t$ , con  $sign \in \{"<", "\leq", ">", "\geq", "="\}$ .

La estructura descrita redefine el problema, para este caso, pasando de uno sobre variables de decisión indexadas sobre tres componentes  $x_{fij}$  a su análogo 1-dimensional,  $x_t$ . El ejercicio requiere de una reasignación  $(f, i, j) \mapsto t(f, i, j)$  definida por  $t = (f - 1)(m \ n) + m(i - 1) + j$  que representa el orden de los renglones que sigue  $df3$ . Ahora, defínanse en *R* los parámetros requeridos en el párrafo anterior

1.  $ce = df3["costo"]$
2.  $io = "min"$
3. Restricciones  $RM \in \mathbb{R}^{re \times m \ n \ cf}$  con  $re = cf + m \ n$ , los primeros  $cf$  renglones representando a las restricciones de disponibilidad de las fichas que hay, y las restricciones restantes representando el correcto llenado de cada celda genérica  $(i, j)$ . La matriz consta únicamente de valores 0 y 1. Los 1's se asignan fácilmente conociendo, con la ayuda de la asignación  $t(.)$  qué posiciones corresponden a cada  $f$  para el primer conjunto de restricciones; similarmente, para el segundo conjunto de restricciones basta reconocer qué posiciones le corresponden a cada celda  $(i, j)$ .

$$4. \ rh_f = u_f \quad \forall f \in \{1, \dots, cf\}$$

$$rh_t = 1 \quad \forall f \in \{cf + 1, \dots, cf + m n\}$$

$$5. \ sign_f = “\leq” \quad \forall f \in \{1, \dots, cf\}$$

$$sign_t = “=” \quad \forall f \in \{cf + 1, \dots, cf + m n\}$$

Si bien el problema no fue restringido a enteros, el resultado es entero y consta únicamente de 0's y 1's. Si extraemos únicamente los valores  $T = \{t|x_t = 1\}$  en una tabla  $dfSel$  con campos  $t, f, i, j$  y obtenemos la imagen de función inversa de  $t(.)$  sobre  $T$ , se valida que las  $m n$  combinaciones  $(i, j)$  son cubiertas exactamente una única vez cada una y que cada figura en  $F$  se ocupa en una cantidad de ocasiones que no excede su correspondiente límite definido en  $u$ . Habiendo supuesto que la dimensión de las  $cf$  figuras es de  $pi \times pi$  y sea  $MG_f \in \mathbb{R}^{pi \times pi}$  la matriz de *greyscales* de la figura  $f$ , entonces procedemos a llenar el fotomosaico mediante el siguiente *loop*: a través de  $m n$  iteraciones hacer lo siguiente.

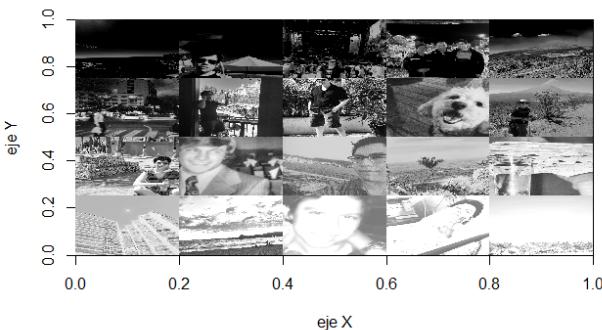
1. Extraemos el  $\kappa$ -ésimo vector-renglón.  $S_\kappa = dfSel[\kappa, :]$ . Así, obtenemos  $S_\kappa[“t”] = t$ ,  $S_\kappa[“i”] = i(t)$ ,  $S_\kappa[“j”] = j(t)$ ,  $S_\kappa[“f”] = f(t)$ , con el renglón, la columna y la figura inferidas de la función inversa de  $t(.)$ )
2. Definimos el extremo superior izquierdo de la imagen  $f(t)$  sobre la matriz final,  $MF[rI, cI]$ . Sígase que  $rI(t) = (i(t) - 1)pi + 1$  y  $cI(t) = (j(t) - 1)pi + 1$
3. Luego, se asigna el valor de  $MF_{f(t)}$  a la submatriz con el extremo superior izquierdo en  $(rI, cI)$ ; esto es,  $MF[\{rI, \dots, rI + pi - 1\}, \{cI, \dots, cI + pi - 1\}] = MG_f$

Se puede validar que este procedimiento llena por completo y con valores factibles a la matriz  $MF$ .

El siguiente paso es ejecutar los pasos descritos hasta este punto de la sección usando una gama de imágenes cuadradas y buscando emular el logo de Twitter. Para ello, se generó un código de *R* que lee todos los archivos png de una subcarpeta y extrae sus dimensiones (pixeles). Posteriormente, elige la mínima dimensión (de entre columnas y renglones) del conjunto de imágenes, acota a todas las imágenes de manera que cumplan con esta dimensión a lo largo y ancho y guarda su luminosidad promedio. Además, define un intervalo de luminosidad,  $i = 1/cf$ , con  $cf \in \mathbb{N}$  sobre el cual se seleccionará la imagen. Esto es, si  $i = 0.05$  entonces las figuras se clasificarán en  $cf = 20$  categorías, a saber,  $\{[0, 0.05), [0.05, 0.10), [0.10, 0.15), \dots, [0.90, 0.95), [0.95, 1]\}$ . Para el primer ejercicio extraemos las imágenes de un conjunto de 60 fotos de una cuenta de Instagram. Al computar su luminosidad se encuentra que las 20 categorías no son todas cubiertas, estando las imágenes acotadas en luminosidades entre 0.30 y 0.75. Para hacer que la cantidad de categorías cubiertas sea de 20 entonces se oscurecen o iluminan tantas más imágenes como categorías faltan por cubrir. El resultado de hacer la selección de 20 imágenes, ordenadas por luminosidad de menor a mayor, se muestra a continuación.

Habiendo seleccionado las imágenes, se procede a reproducir el ejercicio de los fotomosaicos. Al hacerlo con  $dp = 0.3$  y  $vp = 0.3$ , podemos desplegar la imagen resultante con dimensiones  $15 \times 15$  y  $20 \times 20$

Para instancias con los parámetros indicados, el problema de  $20 \times 20$  fue el más complejo de computar para el desplegado de la imagen mediante *image()*. En dimensiones más grandes, la función arrojaba errores por sobrepasarse el máximo peso de input (matriz) computable. Sin embargo, una función similar, que acepta parámetros de la misma clase, *display()* de la librería *EBImage*, logró desplegar el resultado



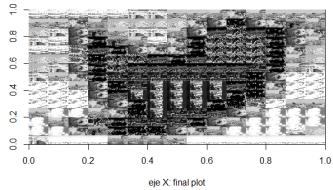
**Figura 5.9. Catálogo de imágenes seleccionadas para el fotomosaico**

del ejercicio de  $25 \times 25$ , tal como se muestra a continuación.

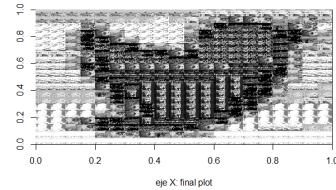
Claramente el resultado ha mostrado ser, visualmente, eficiente. La matriz  $mp$  sirve como referencia para ver cómo se vería la imagen si se deseara desplegar la mejor representación posible de la imagen original contando exactamente con dimensiones de pixel  $m \times n$ . El resultado se muestra a continuación.

Procedemos a evaluar la complejidad computacional del algoritmo de resolución del programa lineal. Para ello, registramos el tiempo que lleva resolver una determinada imagen conforme se agregan más dimensiones o rectángulos y, por conseciente, más ciudades.

Las siguientes gráficas y tablas reflejan la relación que tiene la dimensión (suponiendo que hay tantos rectángulos a lo largo como a lo ancho) con el tiempo de cómputo requerido para generar la imagen de Twitter; también, la que hay entre el total de rectángulos (la dimensión al cuadrado) y el tiempo.

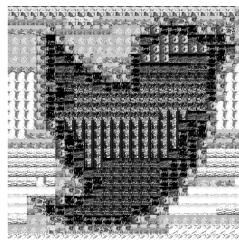


**(a) Fotomosaico: 20 tipos de figuras,  $15 \times 15$  mosaicos**



**(b) Fotomosaico: 20 tipos de figuras,  $20 \times 20$  mosaicos**

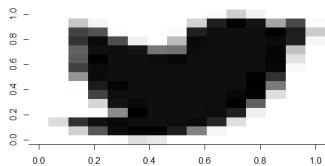
**Figura 5.10. Soluciones gráficas al problema de los fotomosaicos con  $dp = 0.3$  y  $vp = 0.3$ ; dimensiones  $15 \times 15$  y  $20 \times 20$**



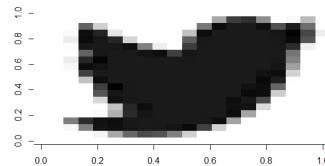
**Figura 5.11. Soluciones gráficas al problema de los fotomosaicos con  $dp = 0.3$  y  $vp = 0.3$ ; dimensiones  $25 \times 25$**

Renglones (= Columnas)	Total mosaicos	Tiempo de ejecución (s)	Variables	re
15	225	0.25	4500	
20	400	0.50	8000	
25	625	1.22	12500	
30	900	2.93	18000	
35	1225	6.65	24500	
40	1600	14.65	32000	

Una cuestión interesante de abordar es el atributo de escasez de fichas en el problema, el cual se concentra en el vector de parámetros,

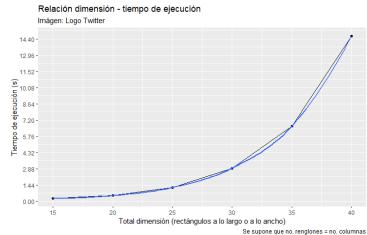


(a) Matriz perfecta,  
dimensiones  $15 \times 15$

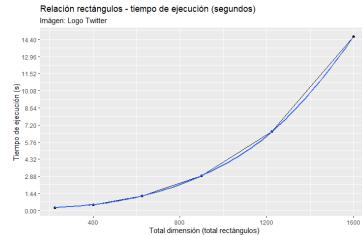


(b) Matriz perfecta,  
dimensiones  $20 \times 20$

**Figura 5.12.** Matrices perfectas basadas en luminosidades promedio de submatrices



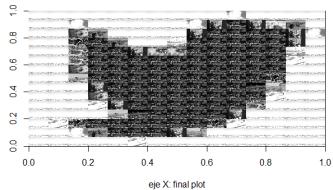
(a) Dimensión vs tiempo



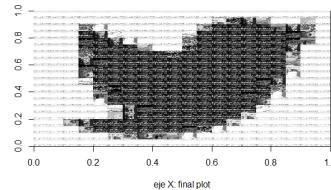
(b) Rectángulos vs tiempo

u. Inspeccionando visualmente la imagen original, o bien, incluso a la versión de *mp* de la misma, se puede observar que todos los contornos de la imagen (alrededor del pajaro) tienen la máxima luminosidad disponible. Esto incluye, naturalmente a los extremos inferior derecho y superior derecho. Sin embargo, al ver el fotomosaico en figura 5.10, se observa claramente que las celdas inferiores derechas tienen una luminosidad muy elevada (cercana a 1), en tanto que la luminosidad de las superiores derechas es alta, pero claramente menor. Esto ocurre porque, si bien pudiera ser deseable que todos los contornos tuvieran el máximo de luminosidad, existe una disponibilidad limitada de la ficha más blanca, así como de todas en general. Por lo tanto, se recurren a

otras fichas, lo menos oscuras posibles que minimicen la suma de diferencias cuadráticas de luminosidades para cubrir el contorno del ave sin exceder los recursos existentes. Haciendo uso de los parámetros introducidos y explicados en el presente escrito, podríamos hacer el ejercicio sin limitar estos recursos. Basta indicar que  $hd = m \cdot n$ , queriendo decir que, si tuviéramos dimensiones  $m \times n = 15 \times 15$ , entonces las fichas disponibles son  $15^2 = 225$  y, si asignamos  $hd = 225$ , entonces dispondremos de 22500 % más de las fichas estrictamente necesarias. Además, si  $vd = 0$  las disponibilidades se distribuirán homogéneamente, con lo cual garantizamos que cada tipo de ficha tiene el potencial de llenar todo el fotomosaico con fichas de un solo tipo. Así pues, a continuación se muestra el resultado de hacer  $hd = m \cdot n$  y  $vd = 0$



**(a) Fotomosaico: 20 tipos de figuras,  $15 \times 15$  mosaicos**



**(b) Fotomosaico: 20 tipos de figuras,  $20 \times 20$  mosaicos**

**Figura 5.14. Soluciones gráficas al problema de los fotomosaicos con  $dp = m \cdot n$  y  $vp = 0$ ; dimensiones  $15 \times 15$  y  $20 \times 20$**

Claramente, estas imágenes tienen contornos más blancos y los fotomosaicos asemejan más a los resultados de usar los valores de matriz perfecta para *greyscales*, 5.12. Sin embargo, la observación de que no todos los 20 tipos de fichas disponibles se están empleando, lo

cual es indicador de que la imagen carece de manera importante de variedad en escalas de intensidad lumínica, nos podría hacer perder interés en esta instancia específica del problema.

### 5.3.2. Fotomosaicos a color

En el capítulo de planteamiento del problema de los fotomosaicos se explicaba que para la aplicación a color el concepto en términos de programación lineal es idéntico y los únicos retos puntuales son: la variedad de fichas a elegir, ya que representan una dimensión superior al del escala de grises en magnitud cúbica; el particionamiento de las  $mn$  submatrices en tres dimensiones (es decir, lo que hace la previamente referida función *matsplitter(.)* de *R*) y el posicionamiento del resultado final en un arreglo de tres dimensiones. Sobre todo esta primera diferencia hace que no sea tan trivial el problema de reproducir el ejercicio de los fotomosaicos con cualquier imagen. Se trabaja entonces partiendo de una imagen relativamente limitada en cuanto a colores, digase la bandera de México, y a raíz de eso, selecciónense los colores a ocupar. La propuesta iniciada de la foto enfocada de una flor con fondo mesurado en la subsección *referir* se extiende a otros colores: rojo, verde, café; complementariamente añádese un rosa y un anaranjado para verificar que el algoritmo acierte al reserva su uso, ya que no figuran como colores en la bandera nacional. A continuación la foto a reproducir.

A continuación se desglosan dos conjuntos de imágenes de flores y sus respectivos colores dominantes generados de reproducir el algoritmo *kmeans* con  $k = 2$



**Figura 5.15. Imagen objetivo: bandera de la República Mexicana**



(a) Flores verdes



(b) Flores rojas



(c) Flores de café



(d) Flores anaranjadas



(e) Flores rosas

**Figura 5.16. Desglose de flores basada en selección manual de colores**



(a) Flores verdes



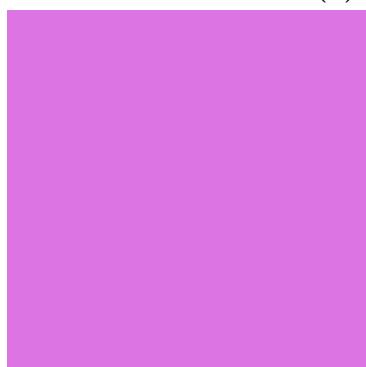
(b) Flores rojas



(c) Flores café



(d) Flores anaranjadas



(e) Flores rosas

**Figura 5.17.** Desglose de color promedio o predominante de 5 tipos de flores que, junto con las de tipo blanco conforman las fichas a asignar<sup>46</sup> de forma óptima

# Índice de cuadros

# Índice de figuras

4.1. Simplificación de imágenes basado en selección de pixeles centroides del modelo k-means . . . . .	17
5.1. Despliegue <code>image(.)</code> , matriz original . . . . .	22
5.2. Despliegue <code>image(.)</code> , matriz rotada 90 a la derecha . . . . .	22
5.3. Gráfica de los puntos sobre el cuadrante $x - y$ . . . . .	27
5.4. Gráfica de las líneas uniendo a los puntos de manera eficiente sobre el cuadrante $x - y$ . . . . .	28
5.6. Catálogo de imágenes seleccionadas para el fotomosaico	31
5.9. Catálogo de imágenes seleccionadas para el fotomosaico	39
5.10. Soluciones gráficas al problema de los fotomosaicos con $dp = 0.3$ y $vp = 0.3$ ; dimensiones $15 \times 15$ y $20 \times 20$ . . .	40
5.11. Soluciones gráficas al problema de los fotomosaicos con $dp = 0.3$ y $vp = 0.3$ ; dimensiones $25 \times 25$ . . . . .	40
5.12. Matrices perfectas basadas en luminosidades promedio de submatrices . . . . .	41
5.14. Soluciones gráficas al problema de los fotomosaicos con $dp = m n$ y $vp = 0$ ; dimensiones $15 \times 15$ y $20 \times 20$ . . .	42
5.15. Imagen objetivo: bandera de la República Mexicana . . .	44
5.16. Desglose de flores basada en selección manual de colores	45

5.17. Desglose de color promedio o predominante de 5 tipos de flores que, junto con las de tipo blanco conforman las fichas a asignar de forma óptima . . . . .	46
---	----

*Diseño, Construcción y Validación  
de un Modelo de Riesgo Crediticio*  
escrito por Julio César Espinosa León,  
se terminó de imprimir en enero de 2023  
en los talleres de Tesis Martínez.  
República de Cuba 99, colonia Centro,  
Ciudad de México.