

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. Fotomosaicos</b>	<b>7</b>
3.1. Fotomosaicos: concepto y contexto histórico . . . . .	7
3.2. Fotomosaicos en escalas de grises . . . . .	9
3.3. Fotomosaicos en la triada RGB, o espacios de color . . .	13
3.3.1. Programa lineal . . . . .	13
3.3.2. Idea del color promedio . . . . .	15
<b>4. TSP: Agente viajero</b>	<b>20</b>
4.1. Problema de Agente viajero, descripción general . . . .	20
4.2. Arte con el agente viajero, o arte con línea continua . .	22
<b>5. Implementación</b>	<b>25</b>
5.1. Partición de la matriz en submatrices . . . . .	25
5.2. Fotomosaicos . . . . .	28
5.2.1. Fotomosaicos en blanco y negro . . . . .	28
5.2.2. Fotomosaicos a color . . . . .	42
5.2.3. Fotomosaicos con librería de R . . . . .	48

5.3.	Arte con el agente viajero . . . . .	51
5.3.1.	Arte con el agente viajero, caso Twitter . . . . .	51
5.3.2.	Arte con el agente viajero, complejidad y exploración de instancias más complejas . . . . .	57
<b>6.</b>	<b>Conclusiones</b>	<b>67</b>
<b>7.</b>	<b>Apéndice</b>	<b>71</b>
7.1.	Adecuando largo y ancho de pixeles para adaptarse a parámetros ingresados . . . . .	71
7.2.	Reduciendo dimensionalidad del vector de costos . . . . .	72
	<b>Bibliografía</b>	<b>74</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, un problema lineal con variables de decisión en tres dimensiones (siendo éstas asociadas a renglón, columna y tipo de imagen); y el segundo, una aplicación del problema del agente viajero

(TSP por sus siglas en inglés).

Dicho esto, los capítulos del presente escrito se centran, primero (3), en explicar el planteamiento de los fotomosaicos en escalas de grises como problema de programación lineal y la posterior extensión del problema al color (para este efecto, se desarrolla un paréntesis para dilucidar el concepto del color promedio a través de una técnica estadística, el kmeans); posteriormente (4), explicar el *TSP*, algunas complejidades asociadas y sus soluciones y aplicaciones al conectar ciudades para replicar una imagen; finalmente (5) un capítulo se enfoca en desarrollar los resultados y comentar los procedimientos que fueron seguidos en el cómputo de los problemas y las correspondientes imágenes desplegadas. Finalmente se concluye repasando teoría, aplicación realizada y probables alcances futuros.

# 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

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

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

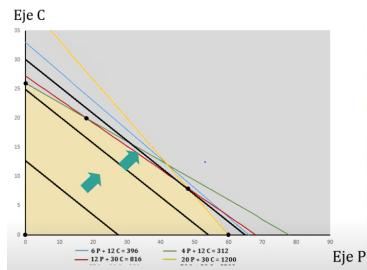
$$x \geq 0 \quad (2.1c)$$

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 los recursos, relacionados vía una ecuación lineal. Algunos de sus primeros usos se remontan a la Segunda Guerra Mundial en la búsqueda de planificar gastos y recursos para reducir costos propios 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



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

igualdades y ejecuta operaciones lineales simples sobre la función objetivo y las restricciones para encontrar el óptimo del problema. Este método irá llevando a quien lo resuelve de vértice en vértice hacia mejores soluciones (mejores valores de la función objetivo) 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 puede alejar uno del óptimo entero, por lo que hay que explorar distintas zonas de la región factible usando 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.

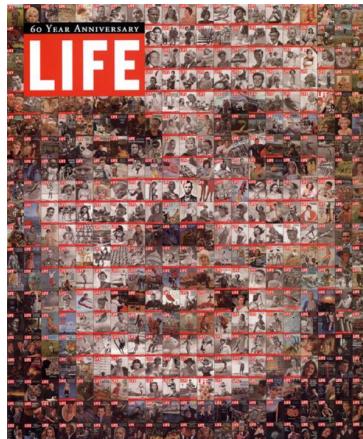
# Capítulo 3

## Fotomosaicos

### 3.1. Fotomosaicos: concepto y contexto histórico

En el área del *marketing* han abundado técnicas de *advertising* que consisten en reproducir con elevada similaridad una imagen de interés conformada por un conjunto de imágenes o grupos de píxeles. Un fotomosaico —o mosaico fotográfico— es una imagen dividida en secciones (las cuales pueden entenderse o interpretarse como submatrices) donde cada sección se reemplaza por otra imagen que se le asocia por algún criterio de similaridad (o distancia). Para construir un fotomosaico, se requieren tres insumos: una foto objetivo, un *set* de fotos a usar para una dada subsección y un *software*-lógica que ejecute la tarea. La idea del fotomosaico tiene una historia reciente en el ámbito de la computación, cuando en fechas similares dos personajes implementaron lógicas y softwares con los cuales podía llevarse a cabo esta tarea. En 1993 Joseph Francis, diseñador, desarrolló lo propio trabajando en el departamento de animación por computadora de la

agencia de publicidad R/Greenberg. Por otra parte, en 1995, Robert Silvers, como estudiante de ingeniería en el MIT, desarrolló una lógica que asumiría proyección e importante audiencia cuando construyó la imagen de la actriz Marilyn Monroe a partir de portadas de la revista LIFE como parte de una edición para el sexoagésimo aniversario de la revista LIFE (portada de la misma), haciendo extensivo su desarrollo a un algoritmo *software* cuya patente registró años más tarde.



(a) Marilyn Monroe  
formulada como conjunto  
de revistas LIFE; portada  
de aniversario 60. Por  
Robert Silvers, primera  
implementación relevante  
de algoritmo patentado.

El programa desarrollado por Silvers se vio reflejado en réditos económicos para él, principalmente en el medio del *marketing*. Los algoritmos, planteamientos matemáticos / computacionales que planteó no fueron transparentados con el objetivo de explotar la

patente. Sin embargo en los años sucesivos se implementaron otras lógicas que desarrollaban fotomosaicos. Algunos ejemplos son, en el campo informático del *open source*, lenguajes como *R* y *python* han desarrollado módulos que permiten generar fotomosaicos con los tres tipos de insumos ya mencionados; y todo esto, como da a entender el nombre de *open source*, por un costo cero para el usuario. La NASA, asimismo, en 2014 generó un fotomosaico de una vista del planeta tierra conformada por más de 36,000 fotos de personas tomándose *selfies* totalizadas en 3.2 mil millones de píxeles (o giga-píxeles) como parte de un programa orientado a concientizar sobre el cambio climático.

### 3.2. Fotomosaicos en escalas de grises

En la introducción se hace referencia a dos maneras de replicar imágenes con optimización lineal que se desarrollarán en el presente estudio. La primera implementación es el de los fotomosaicos en blanco y negro. Parte del supuesto de que se cuenta con una imagen en blanco y negro con una gama de *greyscales* (o escala de grises<sup>1</sup>). Asociamos una mayor *greyscale* a una mayor luminosidad o blancura. Hay una serie de pasos estándar que deben de seguirse si se buscan generar las instancias de programación lineal por fotomosaicos, o bien por el TSP. A continuación se enlistan

- 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

---

<sup>1</sup>En adelante se emplearan indistintamente “*greyscales*” o “escala de grises”

que la matriz grande cuenta con dimensiones de pixeles  $\mu' \times \nu'$  con  $\mu' = k \ m$  y  $\nu' = h \ n$ ; asímismo, obtenemos una escala de grises para cada pixel en el rango discreto de 0 a 255. Llámesele  $\psi$ .

- Obtenemos la *greyscale* promedio de cada rectángulo en coordenada  $(i, j)$  y dividimos el valor entre 255, para efectos de determinar un cómputo de distancias más sencillo (reducido a niveles de 0 a 1),  $\beta_{ij} = \psi_{ij}/255$ . Esta será una escala de luminosidad del rectángulo.
- Supóngase que se cuenta 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 a lo menos un rectángulo con la fotografía  $f$  impresa, pero podemos tener hasta  $u_f$ , que es la cantidad de rectángulos de tipo  $f$  disponibles para su uso. De tal manera contamos con un total de  $\sum u_f = U \geq m \ n$  fichas disponibles. La formulación con programación lineal del problema de los fotomosaicos se muestra, en 3.1.
- Se posicionan las fichas de acuerdo al orden que derivó en la solución óptima de 3.1 y se despliega la imagen resultante de colocarlos, que es el fotomosaico.

En 3.1 se muestra el planteamiento del vector de costos y el conjunto de restricciones introducido para el artículo de divulgación científico - matemática “Laberintos e Infinitos” de nombre “Puentes, agentes viajeros y mosaicos, modelando usando graficas y programación lineal” (Possani, 2012, p. 4).

$$\min_{x_{fij}} \sum_{f \in F} \sum_{i=1}^n \sum_{j=1}^m (b_f - \beta_{ij})^2 x_{fij} \quad (3.1a)$$

$$\text{s.a. } \sum_{i=1}^n \sum_{j=1}^m x_{fij} \leq u_f \quad \forall f \in F \quad (3.1b)$$

$$\sum_{f \in F} x_{fij} = 1 \quad \forall (i, j) \in \{1, \dots, m\} \times \{1, \dots, n\} \quad (3.1c)$$

La expresión (3.1a) 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, 2012). La función de costos será de tipo tridimensional:  $c_{fij} = (b_f - \beta_{ij})^2$ . (3.1b) garantiza que se no se excedan las dotaciones disponibles de cada figura. La restricción (3.1c) 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 de las variables de decisión en planteamiento son solo 0 y 1), el mismo se puede plantear como un problema con variables de decisión continuas y aún así se de lugar a un resultado óptimo entero y factible. Esto es una noticia buena, ya que añadir la propiedad de “entero” o “entero” implica en la solución restricciones asociadas con una complejidad computacional y de planteamiento. La restricción de enterez dificulta el cálculo de soluciones; el algoritmo de ramificación y acotamiento para problemas enteros expresado en el capítulo 2 es bastante ilustrativo para dar fe de esta aseveración, pues la ramificación podría iterarse hasta  $mn$  veces.

El planteamiento referido cuenta con tres dimensiones en su vector de costos,  $c_{fij}$ : las primeras dos corresponden cada una a la coordenada de submatriz (horizontal, vertical) y la tercera a la

distancia en escala de grises respecto a la ficha de tipo  $f$ . Una variación a este problema puede valerse de un problema con fichas de dominó con dos pares de valores ordenados  $(w, z) \in \{0, \dots, 9\} \times \{0, \dots, 9\}$  que sustituyen a la dimensión de fichas  $f$ , donde  $w < z$  y donde cada ficha puede cubrir dos submatrices yuxtapuestas vertical u horizontalmente. La sustitución de  $f$  por el par ordenado de valores de las fichas implica pasar de tres a cuatro dimensiones en el vector de costos. Ahora bien, la orientación de la ficha de dominó implica una nueva dimensión, la quinta:  $o$  que serían las cuatro posibles orientaciones de la ficha de dominó. De tal forma que el vector de costos resultantes sería de tipo  $c_{wzijo}$ . Adicional se tiene una restricción de tipo 3.1c pero que considera que los  $mn$  espacios deben llenarse una única vez. Este planteamiento fue introducido por Bosch en su artículo “Constructing Domino Portraits” de 2004. Es natural intuir que el problema planteado para mosaicos con fichas de dominó tiene más complejidad combinatoria que el planteamiento 3.1, ya que se tienen dos dimensiones más en los costos, así como más restricciones. Si bien este problema no será objeto de estudio ni aplicaciones en este trabajo, sí nos ayuda a comprender las cantidad de variaciones con las que se puede concebir el modelo de fotomosaicos, transformando el vector de costos y las clases de restricciones.

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

#### **3.3.1. Programa lineal**

Una pregunta interesante sería ¿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 en una dimensión de 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” de tal manera que a más colores se superponen, más oscuro es el resultado, esto implica que se resta luminosidad, de este principio se deriva el nombre de “sustractivo” (Castañeda, 2005). Dentro de la categoría de este tipo de modelos; el RYB es el más antiguo registrado e inclusive, el que se pone en práctica desde los niveles de educación primaria usando acuarelas. 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 realizada por el dispositivo, 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 para formar 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, por sus siglas en inglés). A este espacio de color 3-dimensional también se le llamará triada *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 las distancias euclídeanas, pero esta vez ya no en  $\mathbb{R}^1$ , sino en  $\mathbb{R}^3$ . Para ello, la fórmula enunciada en (3.1a) puede modificarse incorporando tres componentes: la primera dimensión,  $r$  (que antes existía, sin embargo se omitía / obviaba al tratarse de 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”,  $\beta_{ij}^A = (\beta_{ijr}, \beta_{ijr}, \beta_{ijg})$  es el color del pixel “promedio” (si se puede decir así, y en efecto se puede comprobar que semejante pixel existe) y  $b_f^A = (b_{fr}, b_{fr}, b_{fg})$  es el color de la ficha  $f$ . Habiendo definido esto, generamos el problema lineal siguiente

$$\min_{x_{fij}} \sum_{f \in F} \sum_{i=1}^n \sum_{j=1}^m [(b_{fr} - \beta_{ijr})^2 + (b_{fg} - \beta_{ijg})^2 + (b_{fb} - \beta_{ijb})^2] x_{fij} \quad (3.2a)$$

$$\text{s.a. } \sum_{i=1}^n \sum_{j=1}^m x_{fij} \leq u_f \quad \forall f \in F \quad (3.2b)$$

$$\sum_{f \in F} x_{fij} = 1 \quad \forall (i, j) \in \{1, \dots, m\} \times \{1, \dots, n\} \quad (3.2c)$$

Podemos simplificar la expresión del vector de costo unitario  $(b_{fr} - \beta_{ijr})^2 + (b_{fg} - \beta_{ijg})^2 + (b_{fb} - \beta_{ijb})^2$  como  $d^2(b_f^A, \beta_{ij}^A)$  con  $d$  la norma euclídea en  $\mathbb{R}^3$ . Este costo, que antes asumía valores en el rango  $[0, 1]$  ahora los toma en  $[0, 3]$ . 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 distribuidas 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 de una imagen compleja. Dichas estas salvedades. Al incrementar la cardinalidad del conjunto de fichas a colocar, los requisitos de procesamiento de cómputo para resolver el programa lineal aumentan considerablemente.

### 3.3.2. Idea del color promedio

En el subcapítulo anterior, 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 la 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 óptimalmente 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 una cosa o 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*  $\times$  *columna*) de pixeles 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 éste si se pasa a  $k = 10$ . Dada la  $k$ , el algoritmo va a definir esta cantidad de centroides para agrupar la totalidad de pixeles. Se desplegará cada centroide reemplazando al pixel original dando lugar a una imagen con la misma dimensión pero menor variedad en sus colores (Aqil Burney, 2014). El caso trivial es cuando  $k$  equivale al total de distintos valores de pixeles que hay, pues entonces tendríamos como resultante la misma imagen que la original. En la figura 3.2 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 distintos 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



(a) Imagen original



(b) 2 tipos de píxeles



(c) 3 tipos de píxeles



(d) 10 tipos de píxeles

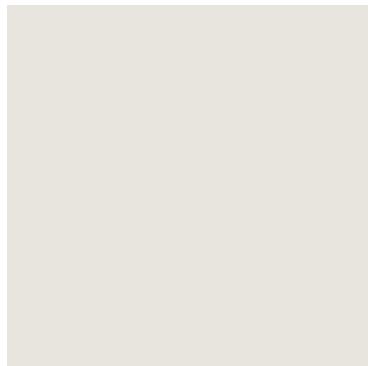
**Figura 3.2. Simplificación de imágenes basado en selección de píxeles-centroides del modelo k-means**

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 0.90. 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 en la triada *RGB*? Así pues, los ejercicios en 3.2 reducen esta diversidad de píxeles 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 (considérese “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, llamarle el “pixel promedio” de la ficha y asegurar que una  $k$  pequeña sea suficiente. 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 existan. En el capítulo de la implementación esta inventiva se llevará a cabo análogamente, precisamente con flores. 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 3.2b, pues es aquél visualmente más abundante de ésta.



(a) Color promedio de  
flores blancas derivadas de  
 $k = 2$

# Capítulo 4

## TSP: Agente viajero

### 4.1. 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, representado por una flecha), o bien, no dirigido (representado por una simple arista o recta). 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.

$$\min_{x_{i,j}} \sum_{i=1}^{n_*} \sum_{j=1}^{n_*} c_{i,j} x_{i,j} \quad (4.1a)$$

$$\text{s.a. } \sum_{i=1}^{n_*} x_{i,j} = 1 \quad \forall j \in \{1, \dots, n_*\} \quad (4.1b)$$

$$\sum_{j=1}^{n_*} x_{i,j} = 1 \quad \forall i \in \{1, \dots, n_*\} \quad (4.1c)$$

$$\sum_{i \in S} \sum_{j \in S} x_{i,j} \leq |S| - 1 \quad \forall S \subset \{1, \dots, n_*\} \quad (4.1d)$$

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 (4.1a) es la función objetivo: el recorrido total para arribar a las  $n_*$  ciudades. (4.1b) exige que de cada ciudad se parte hacia una única ciudad; (4.1c), análogo a (4.1b), 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 (4.1d), ya que sin éste podrían haber *tour*s 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 (4.1b) y (4.1c) 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 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 (4.1d) elimina la posibilidad de lo que llamaremos *subtours*. Sin embargo, al tratarse de una restricción que explora 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 operan algún tipo de “algoritmo de optimización combinatoria”.

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

La segunda implementación de una imagen a emular es vía el TSP: parte del supuesto de que se cuenta con una imagen en blanco y negro

con una gama de *greyscales*. 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  $\nu' = h n$ ; así mismo, 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:  $\gamma \in [4, 9] \cap \mathbb{N}$ <sup>1</sup>. 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. Entonces constrúyase

$$g_{ij} = \lfloor \gamma - \gamma \psi_{ij} / 255 \rfloor \quad (4.2)$$

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

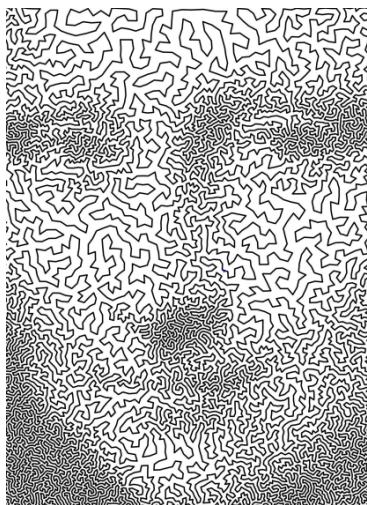
$$\tau = \sum_{i=1}^m \sum_{j=1}^n g_{ij} \quad (4.3)$$

---

<sup>1</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

- 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. Conectamos las ciudades por aristas dando lugar a una representación de líneas en el plano cartesiano; este será el producto final.

Siendo que ahora el planteamiento se esboza sobre  $\tau$  ciudades 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 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 a 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 valores. Por eso, y en aras de respetar 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 4. Para iniciar los ejercicios se ha decidido usar una imagen de la extinta red social Twitter (ahora X), la cual se muestra en 5.1. 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, cada dimensión representando la escala de cada unidad del color primario en el modelo *RGB*. De los elementos extrapolados del primer componente de la dimensión tres (i.e., rojo), `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 `grid.raster(.)` perteneciente al paquete *grid*.

El siguiente paso consiste en usar las dimensiones de pixeles,  $m$  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 analizamos residuos de divisiones con aritmética modular para pasar



**Figura 5.1.** Despliegue con *grid.raster(.)*

de  $(\mu, \nu)$  a  $(\mu', \nu')$ <sup>1</sup>.

Posteriormente, se convierte la matriz de la escala  $[0, 1]$  a la escala  $\{0, \dots, 255\}$ . Luego, se fracciona 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  $m \times n$ . 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<sup>2</sup>. Conociendo esto, podemos calcular la

---

<sup>1</sup>Véase más detalle sobre las propiedades de la regla de asignación para generar esta transformación en apéndice

<sup>2</sup>Arreglo por columna, o *columnwise* donde se llenan las columnas de izquierda a

luminosidad promedio de cada elemento del output de *matsplitter(.)* computado como un promedio aritmético del set de pixeles que compone a cada submatriz. Dicho conjunto de luminosidades puede sintetizarse en un vector de dimensión  $m \times n$ , bajo la ya mencionada lógica del ordenamiento *rowwise*.

## 5.2. Fotomosaicos

### 5.2.1. Fotomosaicos en blanco y negro

#### Construcción del vector de costos y la Matriz Perfecta

Posterior 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). Llámeselos  $h$  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 primero será uno numérico 3-dimensional de tipo  $cf \times m \times n$ , llámesele *Arr3*, con  $Arr3[f, i, j] = c_{fij}$ , donde el lado derecho es el mismo ponderador de costos de la expresión (3.1a). Uno de los arreglos con *greyscales* será de dimensión  $m \times n$ , tendrá por nombre *mp*, por “matriz perfecta”; esto es un eufemismo por tratarse de la mejor representación factible de una imagen con las dimensiones que se ingresan como *input*; el otro, será un arreglo de  $(m \times pi) \times (n \times pi)$  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 matricialmente denotado

---

derecha, cada columna se llena de arriba a abajo

mediante  $Mg_f$ , o “matriz de *grayscale*” de la imagen tipo  $f$ .

Existe un bucle que ayuda convertir la información plasmada en el arreglo  $Arr3$  (de tres dimensiones) en un *data frame*,  $df3$  (de una dimensión) con los campos:  $t, i, j, f, ce$ <sup>3</sup> donde cada  $t$  se asocia a exactamente una combinación  $(i, j, k)$  y a cada terna  $(i, j, k)$  le corresponde una sola  $t$  (puede pensarse como un mapeo biyectivo). Esto liberará una tarea, permitiéndose que la librería elegida (*lpSolve*), para computar el problema de programación lineal, admita los argumentos aportados. A continuación se describe cómo se llenan 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).
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  $c_{ij} = [(h_k - lumP_f)^2]_{f \in F} \in [0, 1]^{cf}$ . Asignar el valor de este vector a los elementos  $Arr3[i, j, ]$  del arreglo  $Arr3$ .
4. Asignar el valor  $h_k$  a la posición  $(i, j)$  de la matriz perfecta.

Es importante considerar que el orden del mapeo  $i, j, f$  a  $t$  también debe guiar la construcción de la matriz con los dos tipos de restricciones denotados en la formulación del problema.

---

<sup>3</sup>La forma genérica de dicho arreglo 2-dimensional se explica y se esboza con mayor detalle en el apéndice

## Adaptación a módulo *lpSolve* y redimensionamiento

La mayoría de módulos en el *open source* perciben el vector de costos de un programa lineal como de una única dimensión; el planteamiento desglosado en el capítulo 3 plantea tres dimensiones. Por lo tanto hay que construir un arreglo que descomponga toda la información en uno solo pasando de *Arr3* a *df3*, la construcción de este arreglo se ve explicada en el apéndice. Llevado al universo de programación en *R*, y con ayuda de esta *df3* podemos definir el modelo en un objeto de clase “*lp*”, auxiliándonos en el módulo *lpSolve*. Dicho objeto se basa en (a) un vector de pesos, *ce*, (b) una instrucción de optimización, *io*: “minimiza” o “maximiza” (en este caso sera la segunda), (c) una matriz *RM* de restricciones con dimensiones *re* × (*m n f*) que representa a las *re* restricciones existentes sobre las variables (d) un vector de lados derechos, *rh* ∈ ℝ<sup>*re*</sup> con el valor contra el cual será contrastada la combinación lineal de *x*, *RM*[*t*,] · *x*, (e) un vector de caracteres, *sign*, con el cual la restricción *t*-ésima será *RM*[*t*,] · *x* “*sign*[*t*]” *rh*<sub>*t*</sub>, con *sign* ∈ {“<”, “≤”, “>”, “≥”, “=”}.

## Holgura y variabilidad: Construcción del vector de disponibilidades

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. Ejemplo: si *hd* = 0.1, tendremos aproximadamente 10% más de las fichas exactamente suficientes para llenar los *mn* espacios habilitados. Para garantizar la viabilidad del problema, *hd* debe ser estrictamente no negativo ya que si se tienen menos fichas de las necesarias el problema tendría una región factible

vacía.

Si deseáramos que las disponibilidades de cada ficha fueran las mismas para todas, 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 (utilizando distribución uniforme) 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  $vd \in [0, 1]$ , y suponiendo un correcto ejercicio aleatorio, mayor será la varianza de las disponibilidades de cada tipo de ficha en el vector  $u$ . Es decir que habrá una mayor diferencia entre dotaciones del tipo de ficha más común (o poblado) versus el menos común.

Con los mencionado hasta este punto se han generado los parámetros necesarios y suficientes para definir el modelo. Sigue discutir cómo se despliega la solución y evaluar la eficiencia de la misma.

## Identificación de solución para cada submatriz y despliegue del resultado

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  $T = \{t | x_t = 1\}$  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 el vector  $u$ . 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, o matriz final  $MF$ , mediante el siguiente *loop*: a través de  $m n$  iteraciones hacer lo siguiente únicamente a través de las soluciones  $\kappa \in T$ .

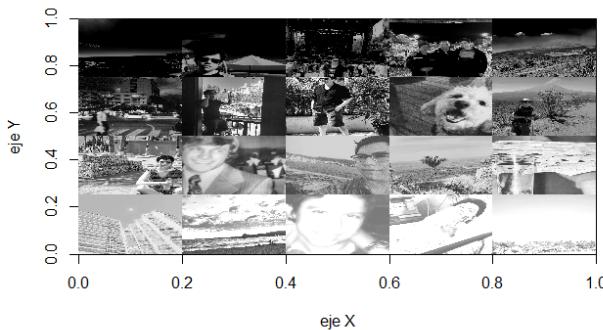
1. Extraemos el  $\kappa$ -ésimo vector-renglón.  $S_\kappa = df Sel[\kappa, \cdot]$ . Esto implica una  $(i, j, f)$  correspondiente.
2. Ahora hay que definir la posición de la figura,  $f$  en la matriz final. Definimos el pixel superior izquierdo de la figura  $f$  sobre la misma,  $MF[rI, cI]$ . Sígase que  $rI = (i - 1)pi + 1$  y  $cI = (j - 1)pi + 1$
3. Sigue definir al resto de la figura que llenará la posición  $(i, j)$ . Se asigna el valor de  $MF_f$  a la submatriz de píxeles 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  $MF$ , y de manera óptima desde el enfoque de la distancia de *grayscale*.

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 (píxeles). 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 falten por cubrir. El resultado de hacer la selección de 20 imágenes, ordenadas por luminosidad de menor a mayor, se muestra a continuación.



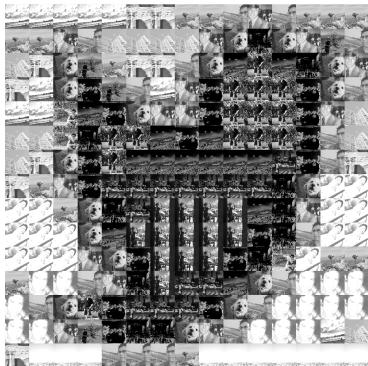
**Figura 5.2. Catálogo de imágenes seleccionadas para el fotomosaico**

Habiendo seleccionado las imágenes, se procede a reproducir el ejercicio de los fotomosaicos. Al hacerlo con  $hd = 0.3$  y  $vp = 0.3$ , podemos desplegar la imagen resultante con dimensiones  $15 \times 15$  y  $20 \times 20$  en la figura 5.3 usando la función *grid.raster(.)*.

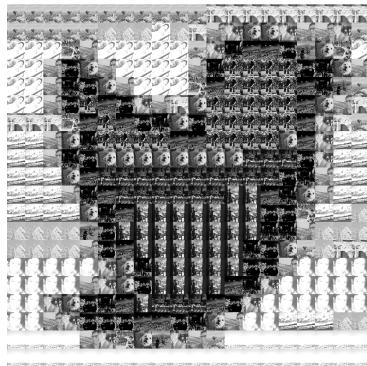
Para instancias con dimensiones mayores también hubo despliegues, los cuales se encuentran en 5.4.

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 en figura 5.5.

Procedemos a evaluar la complejidad computacional del algoritmo



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

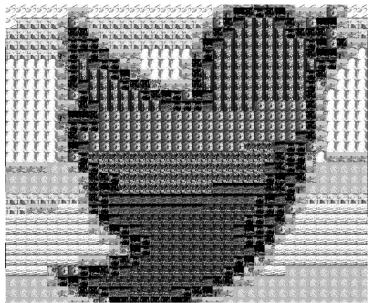


(b) Fotomosaico: 20 tipos de figuras,  $20 \times 20$  mosaicos

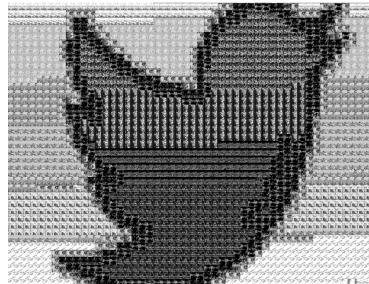
**Figura 5.3. Soluciones gráficas al problema de los fotomosaicos con  $dp = 0.3$  y  $vp = 0.3$ ; dimensiones  $15 \times 15$  y  $20 \times 20$**

de resolución del programa lineal. Para ello, registramos el tiempo que lleva resolver para la función  $lp()$  una determinada imagen conforme se agregan más dimensiones o rectángulos. Cabe recalcar que los tiempos registrados contabilizan el esfuerzo del algoritmo de  $lp()$  y no los procesamientos (que pueden ser bastante robustos) para construir las matrices y vectores que definen el problema, así como para desplegar el resultado final.

Las gráficas 5.6 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. Vale la pena mencionar que con la función  $lpSolve$ ,  $R$  abortó al llegar a la ejecución de dimensión 60; sin embargo las gráficas desplegadas y sus estimaciones lineales,



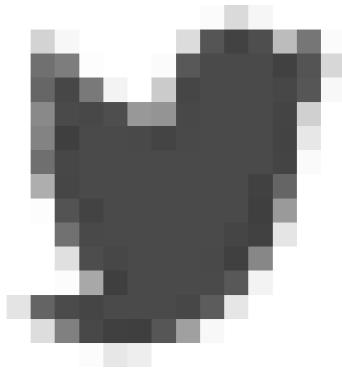
(a) Dimensiones  $35 \times 35$



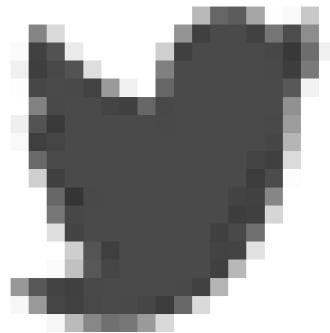
(b) Dimensiones  $60 \times 60$

**Figura 5.4. Soluciones gráficas al problema de los fotomosaicos con c; dimensiones  $35 \times 35$  y  $60 \times 60$**

cuadráticas y cúbicas dan a entender que la ejecución es cuando menos de orden 3 en el número de submatrices ingresadas como parámetro objetivo. Esto se explica por las gráficas de *fit* (en particular 5.6b), donde es eviente que el comportamiento dista de ser lineal y tiene un ajuste cuadrado extraño (el que tenga su vértice en valores de rectángulos “negativos” es absolutamente contraintuitivo). No es sino hasta el estimado cúbico que se distingue una relación asociable.



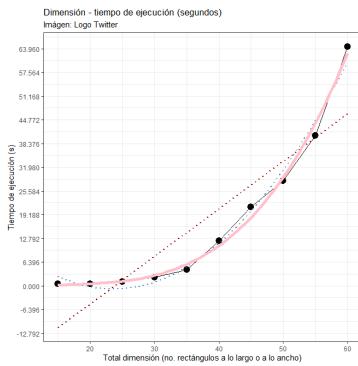
(a) Matriz perfecta,  
dimensiones  $15 \times 15$



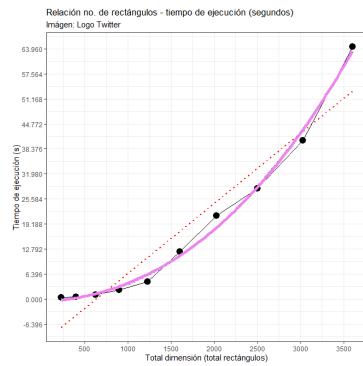
(b) Matriz perfecta,  
dimensiones  $20 \times 20$

**Figura 5.5. Matrices perfectas basadas en luminosidades promedio de submatrices**

Renglones (= Columnas)	Total mosaicos	Tiempo de ejecución (s)	Variables	restricciones
15	225	0.45	4,500	245
20	400	0.52	8,000	420
25	625	1.12	12,500	645
30	900	2.36	18,000	920
35	1,225	4.42	24,500	1,245
40	1,600	12.12	32,000	1,620
45	2,025	21.28	40,500	2,045
50	2,500	28.29	50,000	2,520
55	3,025	40.49	60,500	3,045
60	3,600	64.45	72,000	3,620



(a) Dimensión vs tiempo



(b) Rectángulos vs tiempo

Figura 5.6. Graficaciones tiempos de ejecución

Renglones (= Columnas)	Total mosaicos	Distancia resultante	distancia result, vs valor max (%)
15	225	6.77	3.0 %
20	400	10.3	2.6 %
25	625	20.3	3.2 %
30	900	26.3	2.9 %
35	1,225	29.9	2.4 %
40	1,600	40.4	2.5 %
45	2,025	64.4	3.2 %
50	2,500	88.3	3.5 %
55	3,025	99.6	3.3 %
60	3,600	117	3.3 %

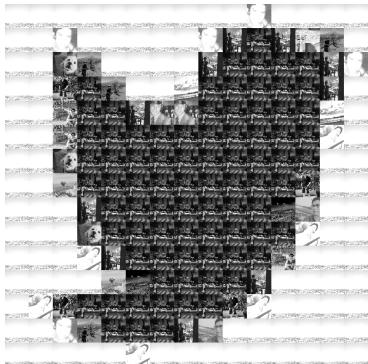
La tabla de distancias introduce una interrogante interesante; ¿qué tan buena o mala es la distancia resultante de las instancias trabajadas? La respuesta para estos ejercicios depende: es natural

pensar que a mayor sea la cantidad de rectángulos a reproducir, más amplio será el rango de valores de la función objetivo ya que cada distancia cuadrada tendrá un valor máximo de 1. El rango de valores de la función objetivo es de 0 al número total de rectángulos. De tal manera que, tal como podría intuirse, el valor de la función objetivo muy probablemente subirá en la medida que se amplían las dimensiones del problema. Sin embargo, la evaluación de si ello representa o una buena o mala reproducción dependerá de la óptica con la que se aborde el problema. Si tomamos la distancia resultante contra el peor escenario (aquél donde se iguala la distancia al número total de mosaicos) y midiendo eso como porcentaje, entonces puede llevarse a cabo un ejercicio más equiparable con resultados evaluados en función de la dimensionalidad del conjunto de submatrices. Y visto desde esta métrica, en la cuarta columna de se puede observar que el valor de la función objetivo es de entre 2.5% y 3.5% de su máximo posible.

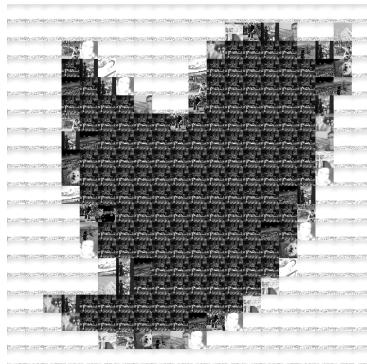
Una cuestión interesante de abordar es el atributo de escasez de fichas en el problema, el cual se concentra en el vector de disponibilidades,  $u$ . Inspeccionando visualmente la imagen original, o inclusive la versión de  $mp$  de la misma, se puede observar que todos los contornos de la imagen (alrededor del pájaro) tienen la máxima luminosidad disponible (es decir, blanco). Esto incluye, naturalmente, a los extremos inferior derecho y superior derecho. Sin embargo, al ver el fotomosaico en figura 5.3, 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 recurre 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. Cabe mencionar que esta reparametrización tiene una alternativa: eliminar las restricciones del tipo definido en las expresiones 3.1a. Hacer esto implicaría reducir la dimensionalidad de la matriz de restricciones, conllevando inherentemente ahorros en el cálculo de soluciones; sin embargo no se hará así, porque eso cambiaría drásticamente la estructura del código de la solución computada y podría perderse la sensibilidad del lector sobre la importancia de la holgura de disponibilidades que este escrito busca enfatizar. En 5.10 se muestra el resultado de hacer  $hd = m \cdot n$  y  $vd = 0$ .

Claramente, estas imágenes tienen contornos más blancos y los fotomosaicos se asemejan a los resultados de usar los valores de matriz perfecta para *greyscales*, 5.5, puesto que el costo total de diferencias de luminosidades promedio (valor de la función objetivo) se reduce al relajar las restricciones. Para estas instancias tenemos resultados de distancias de 0.30 y 0.59 representando en ambos casos 0.13 % y 0.15 % del supremo del rango de la función objetivo. Recordemos que en ejercicios previos con holguras de disponibilidad más amplias estos



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



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

**Figura 5.7. Soluciones gráficas al problema de los fotomosaicos con  $dp = m n$  y  $vp = 0$ ; dimensiones  $15 \times 15$  y  $20 \times 20$**

porcentajes eran del orden del 3 %; es decir, hasta 20 veces menor. De esta modificación estratégica de las restricciones realizadas en ejercicios de mismas dimensiones podemos dilucidar la sensibilidad del algoritmo diseñado y sus variaciones paramétricas sobre la calidad visual y cuantitativa de los resultados. No obstante lo anterior, es claro que no todos los 20 tipos de fichas disponibles se están empleando, lo cual es indicador de que la imagen carece notablemente de variedad en escalas de intensidad lumínica, nos podría hacer perder interés en esta clase específica de instancias.

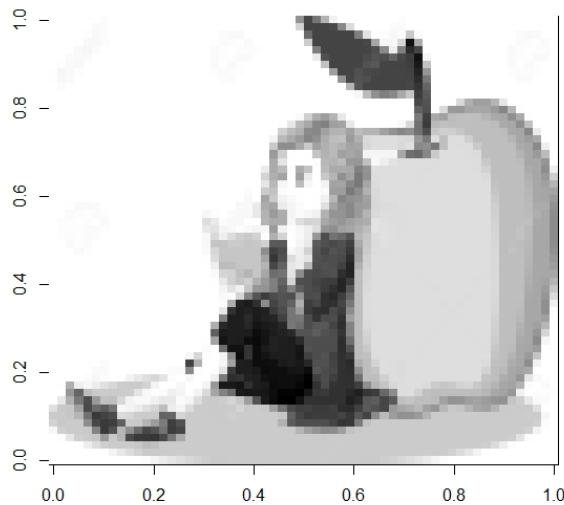
Finalmente, consideramos que vale la pena explorar la posibilidad de construir una imagen más elaborada, con mayor relieve y color que el logo de Twitter. Para ello, se propuso reproducir el ejercicio de los fotomosaicos con una imagen caricaturizada del connotado científico británico, Isaac Newton, la cual se puede encontrar en 5.8.



**Figura 5.8. Fotomosaico de Isaac Newton**

En ejercicios anteriores ya se demostró que el algoritmo computado para este trabajo soporta composiciones de imágenes de hasta  $60 \times 60$  submatrices. Por ello, se optó por una asignación poco por debajo de ese límite, de  $50 \times 50$ . Reproduciendo de nueva cuenta la escala de gris del promedio en cada rectángulo, se genera el resultado de la matriz objetivo (o matriz perfecta).

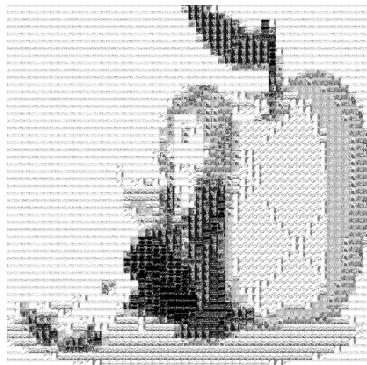
En 5.10, se tienen dos resultados de nueva cuenta, de forma similar a como se trabajó con la imagen del pájaro. Un ejercicio con holgura indefinida en las dotaciones de fichas y otro ejercicio que exige variabilidad en las asignaciones de fichas sin importar que exista una ficha geométricamente más cercana a un dado rectángulo sobre el espacio de grises.



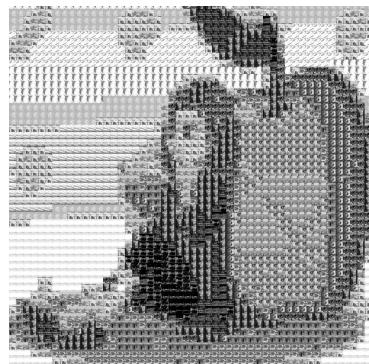
**Figura 5.9. Fotomosaico de Isaac Newton, matriz perfecta**  
 $50 \times 50$

### 5.2.2. 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 elección de variedad de fichas a elegir, ya que éstas representan una dimensión superior al del escala de grises en proporciones cúbicas; 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 haciendo yuxtapuestos varios arreglos matriciales de dimensión tres. Sobre todo esta primera diferencia hace que no sea tan trivial el problema de reproducir el



(a) Fotomosaico Newton:  
20 tipos de figuras,  
Disposiciones no limitadas



(b) Fotomosaico Newton:  
20 tipos de figuras,  
Disposiciones limitadas

**Figura 5.10.** Soluciones gráficas al problema de los fotomosaicos con dimensiones  $50 \times 50$ : Versión con holgura de  $dp = m n$  y  $vp = 0$ ; y sin holgura de  $dp = 0.3$  y  $vp = 0.3$

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 de intensidad medida en la figura 3.2 se extiende a otros colores perceptibles en la bandera nacional: rojo, verde, café; complementariamente añádese un rosa y un anaranjado para verificar que el algoritmo acierte al reservar su uso, ya que a simple vista no parecieran figurar como colores en el símbolo patrio. En 5.11, está la foto a reproducir.

En 5.12 y 5.13 respectivamente, se desglosan dos conjuntos de imágenes de flores y sus respectivos colores dominantes generados al reproducir el algoritmo *kmeans* con  $k = 2$



**Figura 5.11. Imagen objetivo: bandera de la República Mexicana**

Habiendo seleccionado la imagen a computar y las fichas, se procede a generar el algoritmo de fotomosaico. Vale la pena sugerir empezar con dimensiones pequeñas, diganse  $m$  y  $n$  con  $n > m$  ya que la bandera tiene una forma horizontal. Así pues, sea  $m = 15$  y  $n = 20$  y posteriormente  $m = 20$  y  $n = 30$  se generan los resultados de 5.18.

La imagen se ha computado con éxito. Como se preveía, las asignaciones están concentradas principalmente en los tres colores especificados que integran la bandera (verde, blanco, rojo y café); al computar el caso de  $20 \times 30$  se observa que las flores naranjas están desplegadas en 3 ocasiones; esto se explica porque los contornos de la imagen cuyas submatrices tienen centroides en el hiperplano *RGB* ubicado entre el verde de la encina, el laurel y el nopal y el café del águila. El rosa se aleja de las posibilidades de ser seleccionado en la medida de que es un color distante de los que componen la bandera.

Extiéndase el ejemplo de la visualización de Isaac Newton a la versión a color. Las versiones degradadas de la imagen original, en sus versiones de  $40 \times 40$  se presentan en 5.9, con dos despliegues que con



(a) Flores verdes



(b) Flores rojas



(c) Flores de café



(d) Flores anaranjadas



(e) Flores rosas

**Figura 5.12. Desglose de flores basada en selección manual de colores**



(a) Flores verdes



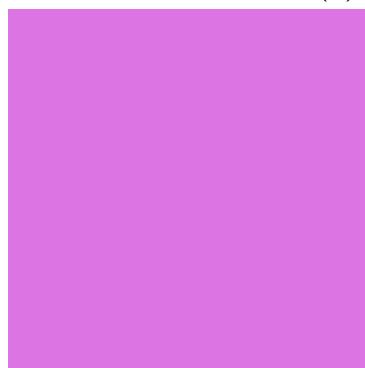
(b) Flores rojas



(c) Flores café

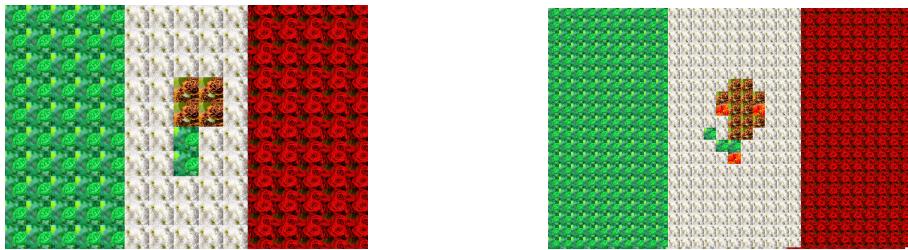


(d) Flores anaranjadas



(e) Flores rosas

**Figura 5.13.** 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



(a) Dimensiones  $15 \times 20$

(b) Dimensiones  $20 \times 30$

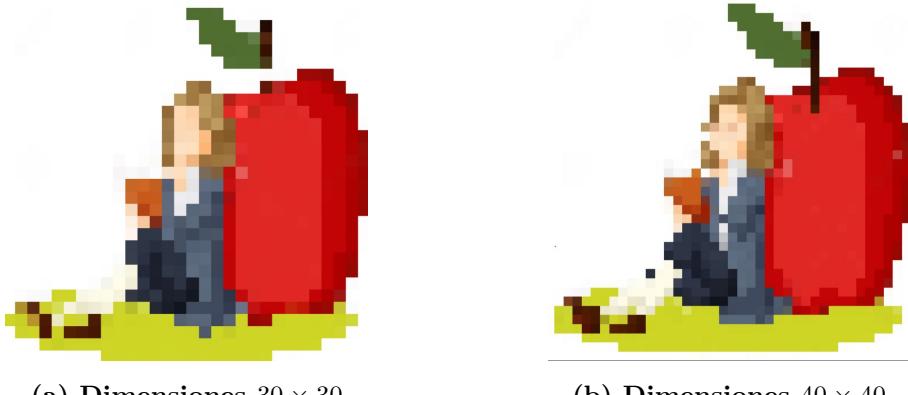
**Figura 5.14. Resultado de fotomosaicos a color: bandera de la república mexicana a base de flores**

facilidad son fácilmente identificables

El primer ejercicio se realiza con la instancia de  $30 \times 30$  con las mismas flores que 5.12 y los resultados se muestran en 5.16

Con algo de imaginación se puede identificar que el desarrollo de esta instancia da un buen resultado. Ajusta el verde de la hoja de manzana con el verde, el rojo de la manzana con las flores rojas y naranjas y las flores cafés se posicionan en donde iría el pelo castaño del personaje. Sin embargo, no deja de ser un ejercicio limitado ya que la variedad disponible es de solo seis colores. De tal manera que se amplió la gama de flores de seis a veinte y se generaron las correspondientes ejecuciones: la de máxima holgura y la de la variación forzada en los parámetros.

Se observa en los resultados en 5.8 que ambos ejercicios ocupan una mayor variedad de fichas que las instancias anteriores de Newton. Sin embargo, resulta interesante cómo en el primer ejercicio el algoritmo identificó el color del pétalo con las flores grises más que con las propias verdes claras ya mostradas. Esto se debe a que el pixel “promedio” de la flor gris, es decir, el gris, se encuentra cercano a muchos colores, entre ellos el verde. El segundo ejercicio tuvo como resultado una seriación de colores variados plasmados en el fondo que



**Figura 5.15. Resultado de arreglos objetivos, versión Newton**

originalmente es color blanco. El hecho de que se colocaran yuxtapuestas bajo un patrón horizontal siendo que no existe similaridad con el blanco de algunos de estos colores (rosa, azul, vino) es un indicativo de que estas imágenes se ocuparon ahí como recurso bajo el objetivo de cumplir la asignación las fichas de los veinte tipos de colores en similar cuantía. Si bien el resultado es imperfecto, sí es visualmente atractivo e interesante en cuanto a problema artístico y de programación. Además, analizando visualmente la imagen objetivo y el resultado, sí es factible relacionar que una imagen es un intento de replicar creativamente a la otra. El resultado en general sí puede considerarse exitoso puesto que se generaron imágenes con características similares.

### 5.2.3. Fotomosaicos con librería de R

En el capítulo 3 se menciona que existen módulos o librerías de lenguaje de programación *Open Source* con los cuales se pueden reproducir ejercicios con imágenes. Tal es el caso de R, que hace uso de



**Figura 5.16. Imagen objetivo: Newton en dimensiones  $30 \times 30$**

su librería *RSimMosaic* para, vía el módulo *composeMosaicFromImageRandomOptim* generar la imagen objetivo con tres insumos mínimos: imagen objetivo, conjunto (folder) de imágenes a colocar y nombre del archivo jpg a crear como resultado. A diferencia del algoritmo que se desarrolló en este trabajo, esta librería se fija como objetivo usar tantas imágenes como sea posible. Además, iterativamente se ejecuta una acción sobre cada subrectángulo, aleatoriamente elegido, que consiste en asignarle una etiqueta mediante un algoritmo de clasificación supervisada llamado “k-vecinos más cercanos” donde cada elemento, o pixel, en el espacio 3 dimensional se asocia con sus k coordenadas más cercanas, cada una de éstas eligiendo por su categoría y optándose por el voto



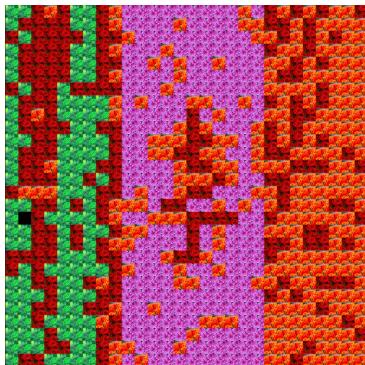
(a) Dimensiones  $40 \times 40$



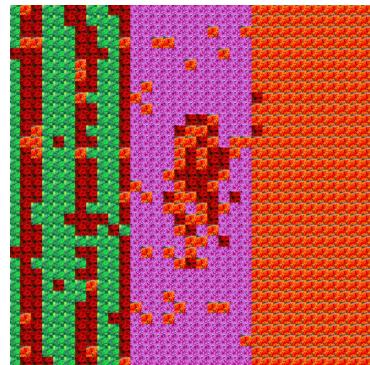
(b) Dimensiones  $40 \times 40$

**Figura 5.17. Resultado de arreglos objetivos, versión Newton**

mayoritario. Tanto el componente de selección y clasificación aleatorio, como el hecho de que no se garantiza un resultado óptimo en cuanto a distancia euclidiana hacen que este programa trabaje sobre una heurística. Una de las ventajas de esto es que se puede desplegar una solución sin un costo computacional tan elevado. Una desventaja se da en cuanto a la solución ya que como se puede revisar, el resultado es menos bueno cualitativamente. De esto podría derivarse otro reto, de identificar qué parámetros pueden ajustarse para hacer una mejor reproducción de la imagen de la bandera (número de vecinos, reemplazar fichas en el proceso de selección, etc.). Vale la pena rescatar que este algoritmo procesa cada pixel de la imagen que se le dé, siendo esta una tarea compleja en imágenes de más de 400 pixeles por dimensión. De manera que tuvo que hacerse el ejercicio de reducción de dimensionalidad desarrollado anteriormente de agrupar un conjunto de pixeles y asignarles su color promedio. Haciendo esto, el algoritmo logra computarse rápidamente, generando imágenes tales como se muestra a continuación.



(a) Dimensiones  $30 \times 30$



(b) Dimensiones  $35 \times 35$

**Figura 5.18. Resultado de fotomosaicos a color: bandera de la república mexicana a base de flores**

Vale la pena notar que este algoritmo se fija como objetivo desplegar la mayor variedad posible de fichas haciendo que en la misma se haga uso de los colores naranja y rosa, que tienen poca relación con la bandera de México. Además la asignación es, como dice el nombre aleatoria, por lo cual sucesivas reproducciones del mismo código darán un resultado diferente.

### 5.3. Arte con el agente viajero

#### 5.3.1. 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 (4.2) desglosada en la sección 4.2 de la presente tesisina. Así pues, obtenemos un vector de dimensión  $m \ n$ , llámeselo  $g$ . 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*.

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

$$\text{si } mod2 = 0$$

$$\text{entonces } i = mod2$$

*e.o.c.*

$$i = mod2$$

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

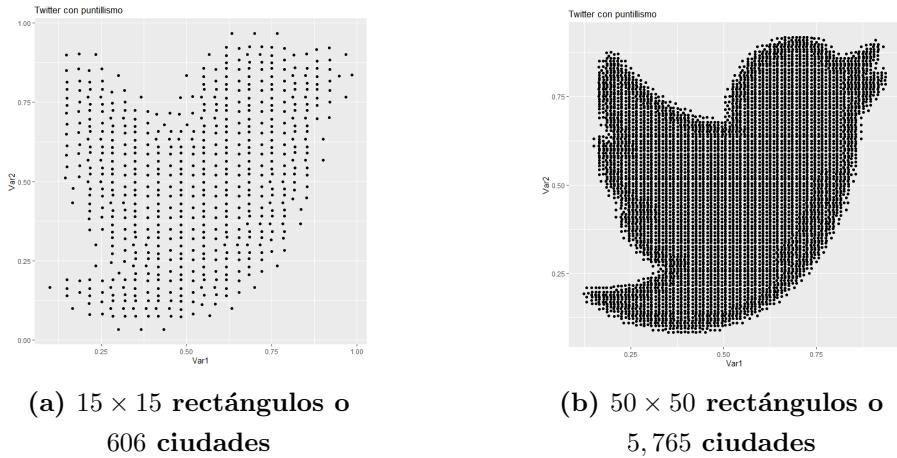
2. Una vez que tenemos la asignación  $k \mapsto (i, j)$ ,  $(i(k), j(k))$ , el siguiente paso es plasmar los puntos que asemejarán a la imagen objetivo sobre el rectángulo  $[0, 1] \times [0, 1]$  en el plano  $x - y$ . Para ello, vale la pena considerar que el extremo superior izquierdo de la imagen 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  $m \times n$  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 coordendas  $y$ :  $\{y_0, y_1, \dots, y_{10}\}$ , 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  $\sum_{k=1}^{m n} g_k = \tau$ <sup>4</sup> ciudades objetivo y podemos resolver el TSP sobre éstas. Los resultados de escribir todas las ciudades como puntos bajo los ejercicios con  $15 \times 15$  rectángulos y (606 ciudades) y con  $50 \times 50$  (5,765 ciudades) se desglosan en la imagen que se nombra “puntillismo”, 5.19, por su similitud con la técnica artística que despuntó de manera importante en Francia a inicios del siglo XX y consiste en generar una obra con puntos diminutos.



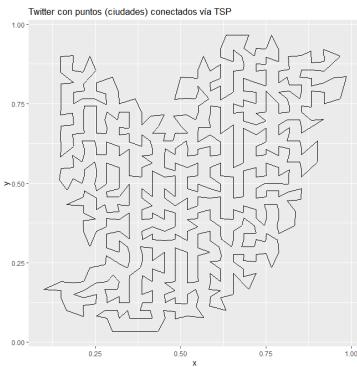
**Figura 5.19. Gráfica de los puntos sobre el cuadrante  $x - y$**

---

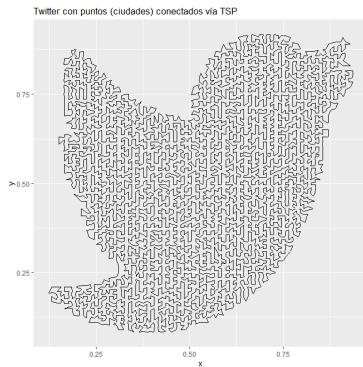
<sup>4</sup>esta es una expresión equivalente a la ecuación ec.(4.3) , pero en la versión que mapea  $(i, j) \rightarrow k(i, j)$

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 diversos *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  $15 \times 15$  y  $50 \times 50$  se despliegan en 5.20.

En el punto 3 de los pasos del instructivo en la subsección 5.3.1 se describe la lógica de la distribución de ciudades “uniforme” propuesta. Sin embargo, alternativamente se puede evaluar el resultado de aleatorizar las coordenadas de puntos del rectángulo, respetando las escalas de color definidas por submatriz (y por lo tanto, manteniendo el número de ciudades). Para ello se ocupó la función de distribución uniforme desarrollada por R, *runif()* dos veces por rectángulo: una



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

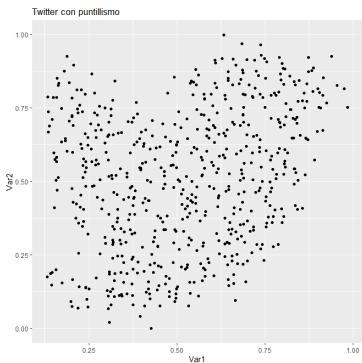


**(b)  $50 \times 50$  rectángulos o  
5,765 ciudades; solución  
vía TSP**

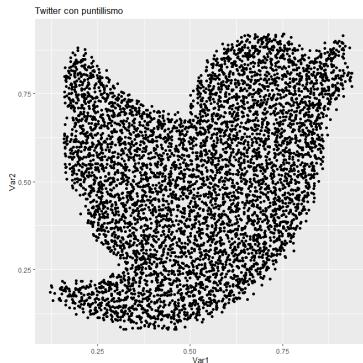
**Figura 5.20. Gráfica de las líneas uniendo a los puntos de  
manera eficiente sobre el cuadrante  $x - y$**

vez para la dimensión  $x$  y otra para la dimensión  $y$ . Los resultados de puntillismo y conexión de ciudades para dimensiones 15 y 50 con 606 y 5,765 ciudades respectivamente se despliegan en los conjuntos de imágenes 5.21 y 5.22.

El nivel de detalle en la dimensión 50 da pie a que la calidad y propiedades de los ejercicios sean visualmente similares entre los casos de distribución “uniforme” regida por *kmeans* frente al de distribución aleatoria (contrastar imagen 5.20b contra 5.22b). La diferencia importante se detecta al comparar ambos tipos de planteamientos con dimensión 15: se observa cómo la conexión entre puntos en el caso aleatorio no tiene una forma visiblemente tan bien definida; es decir, no es trivial deducir que se trata de una silueta de un pájaro, cosa que en el caso “uniforme” sí se observa de forma más evidente (contrastar imagen 5.20a contra 5.22a).



(a)  $15 \times 15$  rectángulos o  
606 ciudades, ejercicio  
aleatorio



(b)  $50 \times 50$  rectángulos o  
5,765 ciudades, ejercicio  
aleatorio

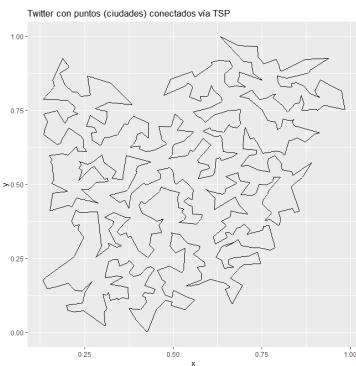
**Figura 5.21. Gráfica de los puntos sobre el cuadrante  $x - y$ , ejercicio aleatorio**

Finalmente, como se ve en 5.23 el ejercicio se hace extensivo a la versión más grande que se logró correr del *TSP* con 85 rectángulos y 15,311 ciudades.

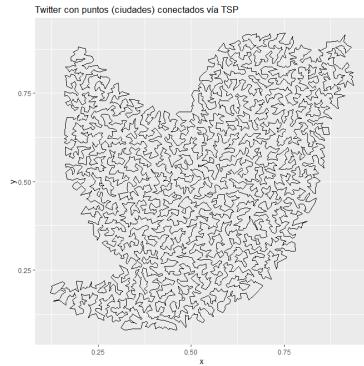
### 5.3.2. Arte con el agente viajero, complejidad y exploración de instancias más complejas

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, naturalmente, más ciudades.

La tabla 5.3.2 asocia la dimensión única (rectángulos a lo largo igualan a los que hay a lo ancho) con el número de ciudades en la imagen de Twitter; asimismo describe la relación del número de



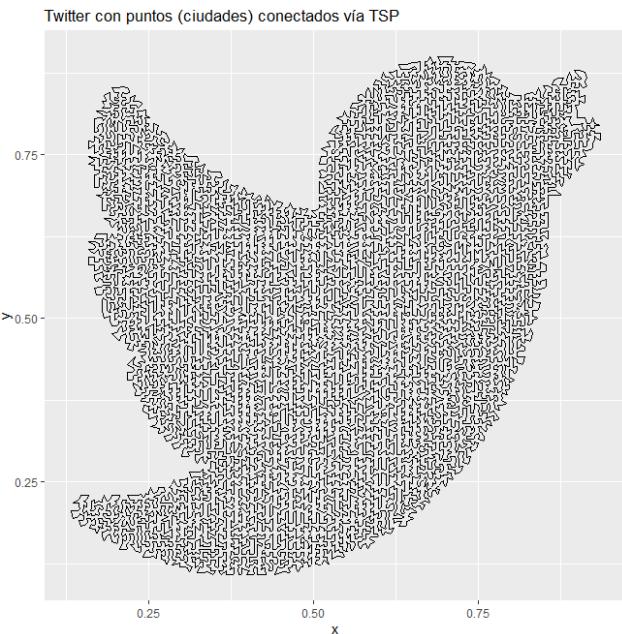
(a)  $15 \times 15$  rectángulos o  
606 ciudades, ejercicio  
aleatorio; solución vía TSP



(b)  $50 \times 50$  rectángulos o  
5,765 ciudades, ejercicio  
aleatorio; solución vía TSP

**Figura 5.22.** Gráfica de las líneas uniendo a los puntos de manera eficiente sobre el cuadrante  $x - y$ , ejercicio aleatorio

ciudades con el tiempo de solución del tsp; claramente esta relación es de una complejidad superior a la lineal. Vale la pena mencionar que la columna de restricciones supone la técnica de reducción de restricciones a los subtours de Miller-Tucker-Zemlin que asume que son  $n^2$  y no  $2^n$  de estas restricciones



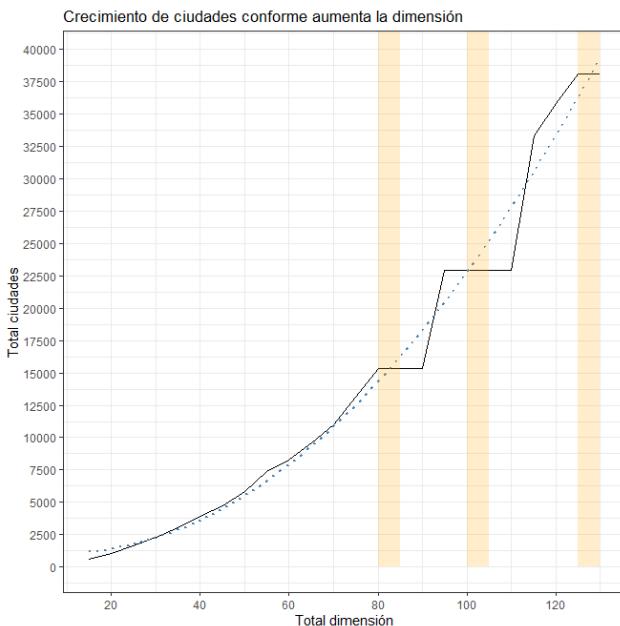
**Figura 5.23.** **85 × 85 rectángulos o 15,311 ciudades; solución vía TSP**

Renglones (= Columnas)	Total ciudades	Tiempo de ejecución (s)	Variables	restricciones
10	286	0.00	81,796	82,368
15	606	0.04	369,664	370,880
20	1,051	0.16	1,104,601	1,106,703
25	1,670	0.83	27,889,00	2,792,240
30	2,284	2.46	52,166,56	5,221,224
35	3,058	5.83	9,351,364	9,357,480
40	3,894	15.67	15,163,236	15,171,024
45	4,723	29.56	22,306,729	22,316,175
50	5,765	61.31	33,721,249	33,732,863
55	7,360	182.34	54,169,600	54,184,320
60	8,271	221. <sup>59</sup> 77	68,409,441	68,425,983
65	9,616	309.78	92,467,456	92,486,688
70	10,987	433.37	120,714,169	120,736,143

### Cuadro 5.1. Tiempos de ejecucion de la función $solveTSP()$ a las instancias definidas por la dimensión

Algo que resalta a la vista de observar la tabla de arriba es que el número de ciudades crece respecto a la dimensión a tasas crecientes en una curva “suave” hasta antes de pasar de 80 a 85; vale la pena observar la gráfica en 5.24 para concluir que es en este intervalo donde ésta empieza a manifestar quiebres. Éstos podrían deberse a que en la medida de que el largo y ancho en submatrices de la imagen se aproximan a los pixeles reales de la misma van a existir transiciones de dimensiones donde algunos pixeles reciben asignaciones de *greyscales* mayores en tanto que otros disminuyen. En el balance, a pesar de que se tienen más rectángulos, serán más los rectángulos que reducen su métrica de oscuridad que aquellos que la aumentan. Esto al grado de que en algunas transiciones se disminuya el total de ciudades resultante (la gráfica muestra cómo eso ocurre tres veces en el rango analizado). Otra de las consideraciones es que, tal como se explica en el apartado de partición de en submatrices, 5.1, la dimensión original se va recortando a lo largo y a lo ancho de tal manera que ambas sean divisibles entre el *input* entregado. Esto da lugar a que las matrices objetivo difieran en dimensión. La dimensión original de la imagen de Twitter es de  $450 \times 550$  en tanto que para ajustarse a dimensiones 80 y a 85 se tuvo que pasar a  $400 \times 480$  y  $425 \times 510$  respectivamente. Por consiguiente las submatrices tienen formas, oscuridades y estructuras distintas dando lugar a ciudades con cardinalidades poco predecibles.

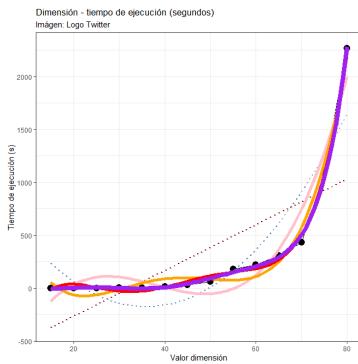
Similar a lo realizado en el capítulo de implementación para fotomosaicos, 5.2.1, se reproducen las gráficas asociadas a la tabla anterior en 5.25. Se añaden los correspondientes ajustes polinomiales. A



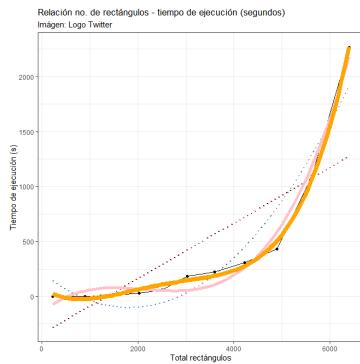
**Figura 5.24. Dimensión vs ciudades generadas. Rangos donde se reduce la cantidad de ciudades se resalta en amarillo**

diferencia de el caso de los fotomosaicos donde bastaban los ajustes cúbicos, en este ejercicio fue hasta llegar a los ajustes de orden 6 y 4 para dimensión y rectángulos respectivamente, que se logró un ajuste suave, consistente e intuitivo.

Sin embargo existe un componente que vale la pena precisar si lo que se busca es comparar el tiempo de ejecución de las instancias de programación lineal genérico vs el del módulo que trabaja el TSP. Y es que el problema aquí versa sobre las ciudades distribuidas sobre los rectángulos (y sus correspondientes distancias mutuas) y no sobre el número de rectángulos en sí. Adicionalmente, se puede decir que los primeros crecen polinomialmente en función de las segundas. Interesa



(a) Dimensión vs tiempo



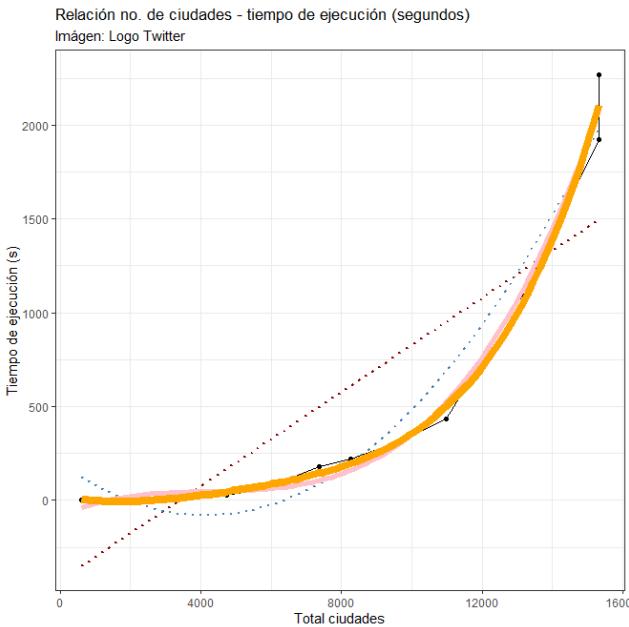
(b) Rectángulos vs tiempo

**Figura 5.25. Graficaciones tiempos de ejecución**

por lo tanto conocer el comportamiento del tiempo de ejecución conforme crecen las ciudades. La gráfica demuestra cómo, similar al caso de la relación rectángulos vs tiempo, basta el polinomio de grado 4 para estimar una relación en ciudades vs tiempo.

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 5.27.

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 distingible, sobre todo en lo que concierne a las facciones faciales: cara, ojos, boca y al implemento electrónico que se



**Figura 5.26. Relación ciudades vs tiempo tsp**

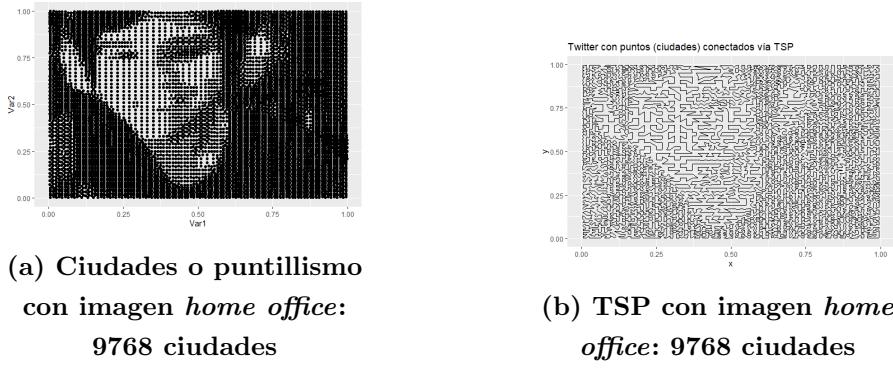
tiene sobre las orejas (audífonos). La razón de este recurso puede aclararse si se muestra el resultado de ejecutar el programa sin hacer esta modificación de la luminosidad. Ambos resultados se muestran en los dos pares de imágenes indexados como 5.28 y 5.29, respectivamente.

Ahora hagamos la instancia del Newton que se mencionó en los apartados del ejercicio de los fotomosaicos. La instancia que se muestra es la procesada en  $50 \times 50$ . El resultado genera una imagen que se distingue con claridad como una manzana tanto por la forma de la cáscara como por la hoja, acompañada de una figura. La parte del hombre se identifica según el observador, cuando el escritor preguntó, las respuestas fueron variadas. Algunos empezaron por identificar primero

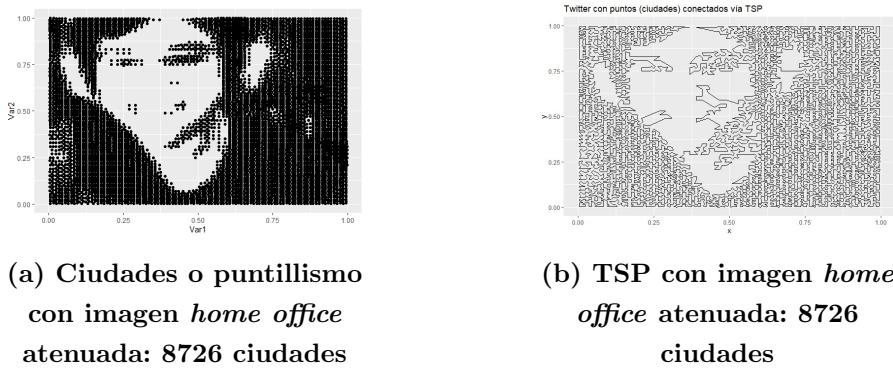


**Figura 5.27. Selfie en pandemia**

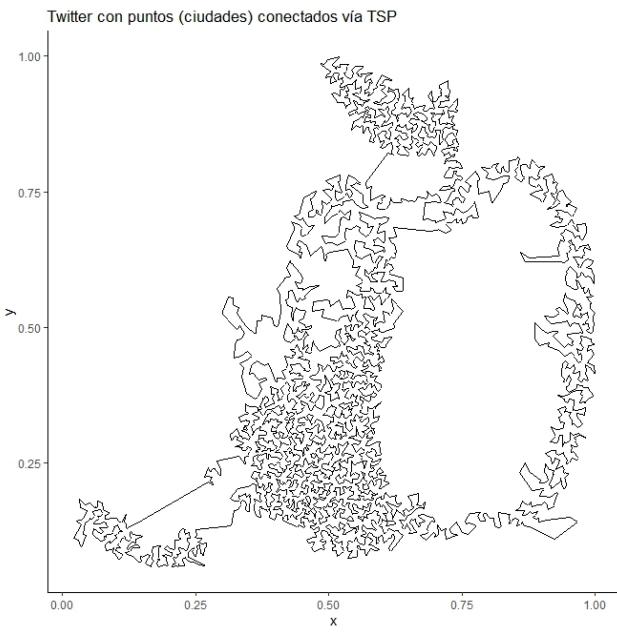
las piernas para luego dilucidar que el resto de ese segmento de imagen es de una figura humana; otros, en tanto, lograron identificar la forma, ya concibiendo el libro que sostiene. Esto, fue tras entrecerrar los ojos.



**Figura 5.28.** Puntillismo y TSP para autorretrato (sin agudizar brillos)



**Figura 5.29.** Puntillismo y TSP para autorretrato (agudizado brillos)



**Figura 5.30. Ejercicio de Newton con el problema del ajente viajero, versión  $50 \times 50$**

# Capítulo 6

## Conclusiones

La presente tesis ha ahondado en la manipulación de las escalas de grises, así como en el modelo aditivo de color de tipo RGB para construir resultados interesantes, tanto del punto de vista de programación como en ámbitos artísticos o creativos. Se determinó que la interpretación matricial de las imágenes dan pie a plantear tantos problemas de programación lineal como se puedan idear con un resultado que derive en hallazgos visibles. Tanto los fotomosaicos como el TSP para replicar la imagen objetivo son formas creativas de interpretar el arreglo matricial; asimismo, la segmentación de k means para deducir un color principal de la imagen resultó una buena aplicación del modelo RGB. En particular la aplicación de fotomosaicos resulta ser interesante desde un punto de vista de marketing para demostrar los efectos de colocar imágenes para alterar la percepción del observador y contemplar un objeto asociable. Sin embargo, la concepción y entendimiento de los pixeles como componentes en un espacio matricial da lugar a procesar diversos algoritmos, como la identificación de formas / siluetas, detección de

colores, o inclusive se puede extender a archivos de video, detectando patrones de movimiento, como ocurre en los automóviles que detectan proximidad al estacionarse, en o los avanzados algoritmos de inteligencia artificial que incluso se desarrollan para programar la conducción automática del vehículo (sin un usuario humano).

Los programas de R para particionar las imágenes objetivo que se codificaron para este escrito resultaron prácticas y robustas a las dos clases de problemas abordados: fotomosaico y TSP.

Las instancias trabajadas de los fotomosaicos en *grayscale*s resultaron efectivas para replicar la imagen objetivo con relativa simplicidad, ya que previamente se generó un *grid* con imágenes promedio con una variedad que aproxima de buena manera al espectro completo de *grayscale*s; sin embargo para la versión a color debió hacerse una selección manual de flores de colores que no abarcan satisfactoriamente el espacio tridimensional. La selección funcionó para la selección de imágenes objetivo, ya que se eligieron fichas pensando en generar resultados simples que replicaran la imagen objetivo. Significa un reto computacional relevante el crear un *grid* con respectivas imágenes asociadas que satisfaga la variedad exhaustiva que se cumplió en las escalas de grises. En contraste con la librería de R especializada, los resultados fueron visiblemente más estéticos, ya que la librería no trabaja un problema de minimización de distancias sobre cada submatriz, sino de asignación de fichas mediante heurísticas de rápida —no necesariamente eficiente— solución.

Las instancias de TSP también resultaron en despliegues con una buena aproximación a las imágenes objetivo incluso ante incrementos de complejidad de los mismos: pasando desde la imagen del logo de Twitter hasta el Newton o el autorretrato. La complejidad de este modelo, no obstante, se demostró en las gráficas correspondientes, con

un tiempo de procesamiento que es notablemente más creciente ante cambios en dimensionalidad de submatrices elegidas que lo que resultó ser en la solución de fotomosaicos. Esto tiene que ver con la creciente complejidad inherente del problema de TSP que se refleja en las restricciones indicadas en el planteamiento y su codificación en la librería de R que, aún con heurísticas o relajación de restricciones para resolverse de manera computacionalmente viable, no puede resolverse de manera más económica.

Si bien el procesamiento de imágenes como arreglos matriciales trae consigo una amplia de variedad de aplicaciones en múltiples ramas como la ingeniería, la medicina, la industria o el transporte, la ciencia es válida de aplicarse en producir arreglos estéticos. Como indica Robert Silvers en su obra Photomosaics, los resultados de esta clase de problemas de optimización con un enfoque visual nos da una linda conciliación o “matrimonio entre arte y tecnología; de fotografía y las computadoras, de la belleza y la ciencia.”

# Bibliografía

- [1] Silvers, R. (1997). Photomosaics. New York: Henry Holt and Company.
- [2] Cole, S. (2014). Image: NASA releases Earth Day “global selfie” mosaic of our home planet. Isla de Man: PhysOrg.
- [3] Possani, S. (2012). Puentes, agentes viajeros y mosaicos: modelando usando gráficas y programación lineal. Ciudad de México: ITAM.
- [4] Bosch, R. (2004). Constructing Domino Portraits. Ohio: Oberlin College.
- [5] Castañeda, W. (2005). Color. Caldas: Universidad de Caldas.
- [6] Aqil Burney, SM. (2014). K-Means Cluster Analysis for Image Segmentation. Sind: University of Karachi.

# Capítulo 7

## Apéndice

### 7.1. Adecuando largo y ancho de píxeles para adaptarse a parámetros ingresados

Para la regla de asignación consistente en hacer la  $\mu'$  múltiplo de  $m$  y  $\nu'$  múltiplo de  $n$  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  $\mu \times \nu$ .

$$DIM = \{(\mu', \nu') = (\max(a), \max(b)) | 0 = a \bmod(m), 0 = b \bmod(n), a \in [0, \mu] \cap \mathbb{N}, b \in [0, \nu] \cap \mathbb{N}\} \quad (7.1)$$

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).

## 7.2. Reduciendo dimensionalidad del vector de costos

La forma genérica que ayuda a convertir el arreglo 3-dimensional de costos en los fotomosaicos en uno de 2 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	73f	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  $f, i, j$ .

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 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  $(i, j)$ .

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

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

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

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

# Índice de cuadros

5.1. Tiempos de ejecucion de la función <i>solveTSP(.)</i> a las instancias definidas por la dimensión . . . . .	60
--	----

# Índice de figuras

3.2. Simplificación de imágenes basado en selección de pixeles centroides del modelo k-means . . . . .	17
5.1. Despliegue con <i>grid.raster()</i> . . . . .	27
5.2. Catálogo de imágenes seleccionadas para el fotomosaico	33
5.3. Soluciones gráficas al problema de los fotomosaicos con $dp = 0.3$ y $vp = 0.3$ ; dimensiones $15 \times 15$ y $20 \times 20$ . . .	34
5.4. Soluciones gráficas al problema de los fotomosaicos con $c$ ; dimensiones $35 \times 35$ y $60 \times 60$ . . . . .	35
5.5. Matrices perfectas basadas en luminosidades promedio de submatrices . . . . .	36
5.6. Graficaciones tiempos de ejecución . . . . .	37
5.7. Soluciones gráficas al problema de los fotomosaicos con $dp = m n$ y $vp = 0$ ; dimensiones $15 \times 15$ y $20 \times 20$ . .	40
5.8. Fotomosaico de Isaac Newton . . . . .	41
5.9. Fotomosaico de Isaac Newton, matriz perfecta $50 \times 50$ .	42
5.10. Soluciones gráficas al problema de los fotomosaicos con dimensiones $50 \times 50$ : Versión con holgura de $dp = m n$ y $vp = 0$ ; y sin holgura de $dp = 0.3$ y $vp = 0.3$ . . . . .	43
5.11. Imagen objetivo: bandera de la República Mexicana .	44
5.12. Desglose de flores basada en selección manual de colores	45

5.13. 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
5.14. Resultado de fotomosaicos a color: bandera de la república mexicana a base de flores . . . . .	47
5.15. Resultado de arreglos objetivos, versión Newton . . . . .	48
5.16. Imagen objetivo: Newton en dimensiones $30 \times 30$ . . . . .	49
5.17. Resultado de arreglos objetivos, versión Newton . . . . .	50
5.18. Resultado de fotomosaicos a color: bandera de la república mexicana a base de flores . . . . .	51
5.19. Gráfica de los puntos sobre el cuadrante $x - y$ . . . . .	54
5.20. Gráfica de las líneas uniendo a los puntos de manera eficiente sobre el cuadrante $x - y$ . . . . .	56
5.21. Gráfica de los puntos sobre el cuadrante $x - y$ , ejercicio aleatorio . . . . .	57
5.22. Gráfica de las líneas uniendo a los puntos de manera eficiente sobre el cuadrante $x - y$ , ejercicio aleatorio . .	58
5.23. $85 \times 85$ rectángulos o 15,311 ciudades; solución vía TSP	59
5.24. Dimensión vs ciudades generadas. Rangos donde se reduce la cantidad de ciudades se resalta en amarillo . .	61
5.25. Graficaciones tiempos de ejecución . . . . .	62
5.26. Relación ciudades vs tiempo tsp . . . . .	63
5.27. Selfie en pandemia . . . . .	64
5.28. Puntillismo y TSP para autorretrato (sin agudizar brillos) . . . . .	65
5.29. Puntillismo y TSP para autorretrato (agudizado brillos)	65
5.30. Ejercicio de Newton con el problema del ajente viajero, versión $50 \times 50$ . . . . .	66

*Foto-mosaicos*  
*y optimización para el arte (Opt-Art)*  
escrito por Julio César Espinosa León,  
se terminó de imprimir en marzo de 2025  
en los talleres de Tesis Martínez.  
República de Cuba 99, colonia Centro,  
Ciudad de México.