

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



**Foto-mosaicos: optimización en el arte**

TESINA

QUE PARA OBTENER EL TÍTULO DE  
LICENCIADO EN MATEMÁTICAS APLICADAS  
PRESENTA

JULIO CÉSAR ESPINOSA LEÓN

ASESOR

DR. EDGAR POSSANI ESPINOSA

«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 **“Foto-mosaicos: optimización en el arte”**, 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>
2.1. Definición general . . . . .	3
2.2. Programación entera . . . . .	6
<b>3. Fotomosaicos</b>	<b>11</b>
3.1. Fotomosaicos: concepto y contexto histórico . . . . .	11
3.2. Fotomosaicos en escalas de grises . . . . .	13
3.3. Fotomosaicos en la triada RGB, o espacios de color . . .	18
3.3.1. Programa lineal . . . . .	18
3.3.2. Idea del color promedio . . . . .	21
<b>4. El problema del agente viajero (TSP)</b>	<b>27</b>
4.1. Descripción general del problema . . . . .	27
4.2. Arte con el agente viajero, o arte con línea continua . .	33
<b>5. Implementación y resultados</b>	<b>37</b>
5.1. Partición de la matriz en rectángulos . . . . .	37
5.2. Fotomosaicos . . . . .	40
5.2.1. Fotomosaicos en blanco y negro . . . . .	40

5.2.2. Fotomosaicos a color . . . . .	56
5.2.3. Fotomosaicos con librería de R . . . . .	62
5.3. Arte con el agente viajero . . . . .	65
5.3.1. Arte con el agente viajero, caso Twitter . . . . .	65
5.3.2. Arte con el agente viajero, complejidad y exploración de instancias más complejas . . . . .	72
<b>6. Conclusiones</b>	<b>82</b>
<b>A. Apéndice</b>	<b>90</b>
A.1. Adecuando largo y ancho de pixeles para adaptarse a parámetros ingresados . . . . .	90
A.2. Reduciendo dimensionalidad del vector de costos . . . . .	91
<b>Bibliografía</b>	<b>93</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 modelos que concilian la optimización, particularmente del tipo lineal, con problemas de carácter estético más vinculados con ramas como el arte o la composición fotográfica. El presente trabajo abordará el problema de reproducir o replicar con buena similaridad una imagen fuente (o imagen objetivo) mediante el empleo, ya sea de trazos continuos a lo largo de un cuadro, o con imágenes diversas que, posicionadas estratégicamente, lograrán construir un producto que, a ojo humano, tendrá una forma fácilmente asociable con la imagen objetivo. Se presentan dos modelos de optimización lineal entera: el primero, un problema con variables en tres dimensiones (renglón, columna y tipo de imagen); el segundo, el problema del agente viajero (TSP, por sus siglas en inglés, *Traveling Salesman Problem*), un clásico

de la Investigación de Operaciones planteado en el siglo XIX por William Hamilton y Thomas Kirkman (González-Santander, 2020), en una aplicación que traza un camino hamiltoniano similar a la imagen.

En el Capítulo 2 se presenta una breve explicación de la programación lineal y su principal método de solución, el algoritmo Simplex, junto con un análisis del método de ramificación y acotamiento para problemas que requieren variables de decisión enteras, el Capítulo 3, explica el planteamiento de los fotomosaicos en escalas de grises como un problema de programación lineal y la posterior extensión del problema a la noción de color (para este efecto, se desarrolla un paréntesis para dilucidar el concepto del color promedio a través de la técnica estadística k-medias); después, en el Capítulo 4, se explica el Problema del Agente Viajero (TSP), una formulación de programación lineal y las heurísticas que se han usado para obtener soluciones aproximadas para este problema, después se muestran aplicaciones al conectar ciudades para replicar una imagen; posteriormente, el Capítulo 5 se enfoca en desarrollar los resultados y comentar los procedimientos que fueron seguidos en el cómputo de las soluciones de los problemas y las correspondientes imágenes desplegadas con su correspondiente tiempo de procesamiento computacional para estimar una complejidad. Finalmente, en la conclusión (Capítulo 6) se dan algunas observaciones finales y probables alcances futuros.

# Capítulo 2

## La programación lineal

### 2.1. Definición general

La programación lineal es un área de la optimización que busca minimizar o maximizar una función lineal sobre múltiples variables, de tal manera que dichas variables cumplan un conjunto de restricciones (ecuaciones o inecuaciones), también lineales. La expresión estándar de un problema lineal es la siguiente:

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

$$\text{s.a. } \sum_{i=1}^n a_{ij} x_i \leq b_i \quad \forall j \in \{1, \dots, m\} \quad (2.1b)$$

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

Las desigualdades de la expresiones (2.1b) y (2.1c) especifican un poliedro convexo, denominado región de factibilidad, que representa un espacio geométrico cerrado.

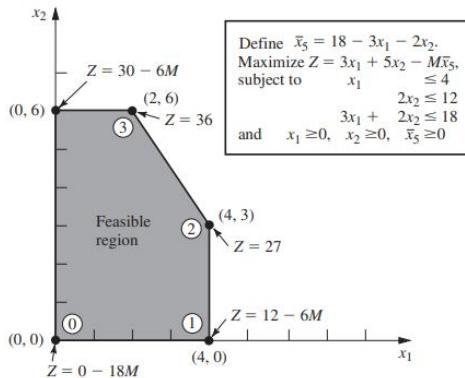
La disciplina y teoría de la programación lineal se manifiesta en problemas que buscan maximizar un beneficio o minimizar un costo bajo circunstancias que restringen la disponibilidad de recursos existentes. El costo o beneficio, según el problema de optimización se denomina función objetivo y es especificado en (2.1b) por una expresión lineal (alternativamente se le referirá como  $f(x)$ ). Dicho beneficio o costo depende de las cantidades de los recursos, además, las dotaciones utilizadas se restringen en un sistema de ecuaciones o inecuaciones lineales definiendo una región de factibilidad. Algunos de sus primeros usos se remontan a la Segunda Guerra Mundial en aplicaciones de estrategias de asignación para reducir costos propios y aumentar las pérdidas enemigas. Otras aplicaciones 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.

Se puede garantizar la resolución de los programas lineales en la medida de que la región factible sea no vacía y acotada en la dirección del gradiente de la función objetivo. Aquí el gradiente, definido como el vector  $\nabla f = (c_1, c_2, \dots, c_n)$ , representa la dirección de mayor incremento de la función objetivo. La acotación en esta dirección asegura que el valor óptimo de la función sea finito. El algoritmo Simplex, propuesto por Dantzig en 1947, es uno de los algoritmos más utilizados para resolver programas lineales de manera óptima, con la solución óptima representada gráficamente como el contacto de una recta de nivel de la función objetivo lineal con el poliedro convexo de la región factible (Fernandez, 2011). Dado que la función objetivo es lineal (cónica no estricta) y la región factible es convexa no estrictamente, puede haber múltiples soluciones óptimas, incluso infinitas, si la recta de nivel coincide con una cara del poliedro. Este

comportamiento es análogo a la optimización no lineal, donde una función cóncava no estricta sobre un conjunto convexo puede tener múltiples óptimos. Lo que se puede asegurar es que cuando menos un vértice de la región factible es una solución óptima.

El método sistemático Simplex emplea las características geométricas mencionadas de los vértices, y optimiza la función objetivo desplazando iterativamente las curvas de nivel de vértice en vértice con valores crecientes de  $f(x)$  hasta alcanzar un óptimo. La gráfica en Figura 2.1 muestra un ejemplo clásico de un programa lineal en el espacio bidimensional, basado en Hillier y Lieberman (2015), donde se ilustra la región factible en gris y los vértices  $(0, 0)$ ,  $(4, 0)$ ,  $(4, 3)$  y  $(0, 6)$  sobre los cuales itera el algoritmo en orden  $(0, 1, 2, 3)$ . Si un programa lineal de la clase especificada en (2.1) es resolvible (esto ocurre si satisface que la región definida por las restricciones está acotada en la dirección del gradiente de la función objetivo), entonces puede ser solucionable vía Simplex. Para complementar esta visión teórica, la Figura 2.2 despliega una representación más didáctica generada con la librería *matplotlib* de Python, donde se muestran las rectas de restricción, la región factible sombreada en verde y las flechas que conectan los puntos iterativos del algoritmo simplex ( $A(0, 0) \rightarrow B(7.5, 0) \rightarrow C(3, 6)$ ), así como las curvas de valor asociadas a estos puntos ( $Z = 0 \rightarrow 21 \rightarrow 22.5$ ). Esta visualización permite apreciar el proceso iterativo y su aplicabilidad a más dimensiones o variedades de ( $m$ ) restricciones de tamaño arbitrario.

El método Simplex transforma desigualdades en igualdades mediante variables de holgura, aplicando operaciones lineales para optimizar la función objetivo sujeta a las restricciones. Este proceso, basado en álgebra lineal, garantiza la convergencia hacia un óptimo global si la región factible está acotada, resolviendo eficientemente

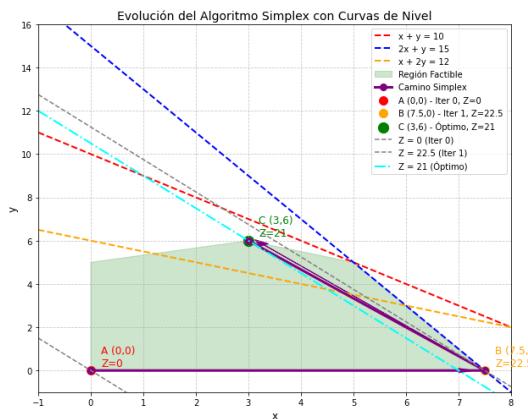


**Figura 2.1.** Ejemplo gráfico de un problema de programación lineal en 2 dimensiones basado en Hillier y Lieberman (2015), mostrando la región factible y los vértices iterativos del método Simplex (0, 1, 2, 3).

problemas de programación lineal.

## 2.2. Programación entera

Algunos problemas requieren soluciones enteras, es decir, incorporar al planteamiento (2.1) la restricción  $x_i \in \mathbb{Z} \forall i \in \{1, \dots, n\}$  (esto es equivalente a expresar  $x \in \mathbb{Z}^n$  o “restricciones de integralidad” sobre las  $n$  variables de decisión). Si bien  $x_i \in \mathbb{Z}$  puede aplicarse a un subconjunto de variables con cardinalidad menor a  $n$ , estos casos no se abordan en este trabajo. Incorporar la integralidad a las restricciones complica la búsqueda del óptimo en un problema lineal. Una estrategia intuitiva es identificar la solución entera más cercana a la del problema relajado, que no incluye la exigencia de integralidad en la región factible. Sin embargo, el valor óptimo del problema relajado es igual o superior al del problema entero asociado, y no hay garantía de que las



**Figura 2.2.** Representación didáctica generada por el autor con *matplotlib*, mostrando las rectas de restricción, la región factible sombreada y el camino iterativo del algoritmo Simplex ( $A(0,0) \rightarrow B(7.5,0) \rightarrow C(3,6)$ ).

soluciones relajada y entera sean próximas, ni de que técnicas simples como el redondeo conduzcan a la solución óptima entera. Considérese un problema de programación lineal dado por  $\max z = 5x_1 + 3x_2$ , sujeto a  $x_1 + x_2 \leq 4$ ,  $2x_1 + x_2 \leq 6$ ,  $x_1, x_2 \geq 0$ . La solución relajada óptima es  $(x_1, x_2) = (2.5, 1.5)$ , con  $z = 17$ . Al imponer la restricción de integralidad ( $x_1, x_2 \in \mathbb{Z}$ ), la solución óptima entera es  $(2, 2)$ , con  $z = 16$ . La solución entera más cercana al óptimo relajado, como  $(2, 1)$ , con  $z = 13$ , es factible pero no óptima, ilustrando esto un caso de no-optimalidad de un recurso como el redondeo.

Ante las dificultades mencionadas, se vuelve necesario explorar distintas zonas de la región factible usando algoritmos, el más conocido de éstos siendo el de ramificación y acotamiento.

En su capítulo para cursos de optimización, Goic (2009) define los elementos más básicos para la definición matemática de esta clase de

problemas. En el mismo, establece la importancia de elegir de manera estratégica las coordenadas a explorar ya que, si bien esta elección es indistinta en función de encontrar el óptimo (la ramificación y acotamiento *per se* encuentra el óptimo), el iterar sobre las coordenadas en un buen orden permite ahorrar, dependiendo del problema en cuestión, ciclos del algoritmo que implicarían costos computacionales sustantivamente elevados. En la formulación de Goic, el planteamiento parte inicialmente de la resolución del problema lineal relajado y, dada esta primera aproximación, va incorporando dos restricciones sobre alguna variable no entera (indistintamente): 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 entre ambas alternativas la 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 a la elección de una nueva variable no entera a ramificar. Dada la solución más reciente, el algoritmo reproduce y evalúa las ramificaciones de distintas variables fraccionales hasta dar con una solución con todas sus variables enteras. En términos de formulación matemática, lo que se hace es transitar de un problema,  $P$  (el problema original) a un problema  $P_0$  (el problema relajado), el cual sería el primer nodo en el conjunto total de ramas a explorar, de tal manera que, como definición inicial,  $L = \{P_0\}$

$$(P) \min_{x_i} z = \sum_{i=1}^n c_i x_i \quad (2.2a)$$

$$\text{s.a. } Ax \leq b \quad (2.2b)$$

$$x \geq 0, x \in \mathbb{Z}^n \quad (2.2c)$$

$$(P_o) \min_{x_i} z = \sum_{i=1}^n c_i x_i \quad (2.3a)$$

$$\text{s.a. } Ax \leq b \quad (2.3b)$$

$$x \geq 0, x \in \mathbb{R}^n \quad (2.3c)$$

Luego, se deriva un subproblema,  $P_k$ , el cual elige una coordenada fraccional-no entera ( $i$ ) del óptimo de  $P_0$ , llámesele  $x^*$ , i.e.,  $f_i = x_i^*$  y optimiza los siguientes problemas subyacentes.

$$(P_k^-) \ min z = \sum_{i=1}^n c_i x_i \quad (P_k^+) \ min z = \sum_{i=1}^n c_i x_i \quad (2.4)$$

$$x \in \mathbb{R}^n \quad (2.5)$$

$$x_i \leq \lfloor f_i \rfloor \quad x_i \geq \lfloor f_i \rfloor + 1 \quad (2.6)$$

$$Ax \leq b \quad Ax \leq b \quad (2.7)$$

$$x \geq 0 \quad x \geq 0 \quad (2.8)$$

Los cuales se llaman respectivamente, rama inferior y rama superior. De tal manera que, si existe factibilidad en ambos problemas, se incorporan como nodos  $P_k^+$  y  $P_k^-$  de la lista  $L$ . Iterativamente, y dependiendo de la integralidad de las coordenadas en la solución de la ejecución corriente, se irá corriendo esta misma lógica sobre las ramas del árbol de problemas descartando o “podando / acotando” problemas (lo cual se refleja en una reducción del conjunto  $L$ ) sin solución factible o de solución entera y actualizando las soluciones

óptimas integrales que existan de manera temporal en los nodos hasta el momento explorados (este óptimo se define como  $\hat{Z}$ ). El algoritmo deja de correr cuando el conjunto de problemas,  $L$ , a ramificar sea vacío.

La complejidad computacional de un algoritmo se refiere al tiempo o espacio requerido para ejecutarse en función del tamaño de la entrada,  $n$ . Se suele expresar por la notación *Big O* que describe la relación tamaño - tiempo; algunos casos comunes de orden son  $O(1)$  (no dependiente de  $n$ ),  $O(n^a)$  (polinomial de orden  $a$ ) o  $O(2^n)$  (exponencial). El hecho de que una solución hallada por método Simplex a un problema lineal sea de una complejidad determinada y que el algoritmo de ramificación y acotamiento resuelve sucesivamente varios algoritmos de este tipo, vuelve a la complejidad computacional del problema lineal entero naturalmente mayor al problema lineal relajado. Se trata de una complejidad que puede pasar de un orden polinomial específico a uno polinomial más grande o inclusive exponencial. Cuando se escala la proporción en la dimensión del vector de variables de decisión y dimensión de restricciones, suele volverse complejo abordar esta clase de problemas, por lo cual se recurre a las heurísticas, métodos aproximados para encontrar soluciones óptimas o cercanas a la óptima de manera eficiente, sin garantizar la optimalidad global de las mismas.

# Capítulo 3

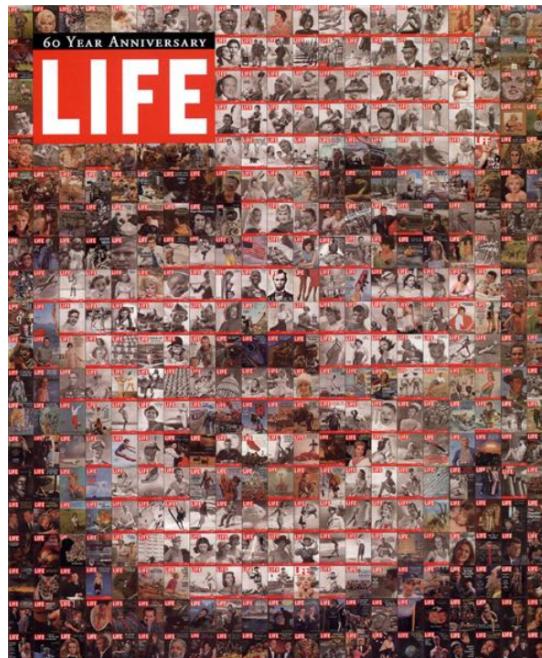
## Fotomosaicos

### 3.1. Fotomosaicos: concepto y contexto histórico

Una imagen digitalmente desplegada parte comúnmente de la lectura / interpretación de un archivo JPEG o PNG. Ambas clases de archivos pueden traducirse como un arreglo numérico de píxeles, entendiendo al pixel (abreviación de *picture element*) como la unidad mínima de información en la imagen. Esto se ve con mayor detalle más adelante en este capítulo. En la mercadotecnia han abundado técnicas de publicidad que consisten en reproducir con elevada similaridad una imagen de interés conformada por un conjunto de imágenes yuxtapuestas que, individualmente vistos, no presentan asociación con la imagen aproximada. Un fotomosaico —o mosaico fotográfico— es una imagen dividida en secciones rectangulares, las cuales pueden entenderse o interpretarse como submatrices de la matriz: en el presente texto se habla indistintamente también de “rectángulos” o “subáreas”. En el fotomosaico cada sección se reemplaza por otra

imagen que se le asocia por algún criterio de similaridad (o distancia), mediante este arreglo se logra hacer reconocible una imagen. Para construir un fotomosaico, se requieren tres insumos: una foto objetivo, un conjunto de mosaicos a usar para una dada subárea y un procedimiento o algoritmo que asigne a cada rectángulo, un mosaico. La idea del fotomosaico tiene una historia reciente en el ámbito de la computación, cuando en fechas similares dos personajes implementaron procedimientos con los cuales podía llevarse a cabo esta tarea. En 1993 Joseph Francis, diseñador, desarrolló un *software* 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 otro algoritmo que logró gran impacto y 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 LIFE, haciendo extensivo su desarrollo a un *software* cuya patente registró años más tarde (Fraga, 2024).

El programa desarrollado por Silvers se vio reflejado en réditos económicos para él, principalmente en el campo de la mercadotecnia. Los algoritmos, planteamientos matemáticos / computacionales que planteó no se hicieron públicos con el objetivo de explotar la patente. Sin embargo, en los años sucesivos se implementaron otros procedimientos que desarrollaban fotomosaicos. Como algunos ejemplos, 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; esto, como implica 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



**Figura 3.1.** Marilyn Monroe formulada como conjunto de portadas de revistas LIFE; portada de aniversario 60. Por Robert Silvers, primera implementación relevante de algoritmo patentado.

*selfies* totalizadas en 3.2 mil millones de pixeles (o giga-pixeles) como parte de un programa orientado a concientizar sobre el cambio climático (Cole, 2014).

### 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. El primer modelo 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 escalas de grises (o *greyscales* en inglés). Se asocia una menor escala de gris a una mayor luminosidad o blancura. Hay una serie de pasos estándar que deben de seguirse si se busca generar fotomosaicos con instancias de programación lineal (o para resolver con TSP como se verá en el Capítulo 4). La secuencia de procedimientos se detalla a continuación:

- La imagen fuente se divide en una cuadrícula de  $m \times n$  bloques rectangulares, donde cada bloque tiene  $k$  píxeles de ancho por  $h$  píxeles de alto. La imagen compuesta resultante, o fotomosaico, tiene dimensiones de  $\mu' \times \nu'$  píxeles, donde  $\mu' = k \cdot m$  y  $\nu' = h \cdot n$ . Cada bloque se identifica por un par ordenado  $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$ , representando su posición en la cuadrícula. Además, cada píxel de la imagen fuente se asocia a una escala de grises en el rango discreto de 0 (negro puro) a 255 (blanco puro), denotada como  $\psi_{i'j'}$ .
- Se obtiene la escala de grises promedio de cada rectángulo en la entrada  $(i, j)$ , es decir  $\psi_{ij} = \frac{1}{h \cdot k} \sum_{i'=1}^k \sum_{j'=1}^h \psi_{i'j'}$ . Notar que aquí  $(i, j)$  denota posición del rectángulo, y  $(i', j')$  denota posición del pixel (unidad mínima de información en imágenes digitales) dentro de cada uno de tales rectángulos. Se divide el promedio generado entre 255, para efectos de determinar un cálculo de distancias más sencillo (acotado entre 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 mosaicos (con cardinalidad  $cf$ ), cada rectángulo representa un una fotografía (según la bibliografía, se les llama fichas, fotos,

mosaicos / *tiles* en el planteamiento original de Robert Bosch). Cada mosaico de tipo  $f \in F$  cuenta con una intensidad lumínica con valor  $b_f$ . Cada mosaico  $f$  no se puede usar más de  $u_f$  veces y puede usarse desde cero ocasiones siempre que existan otros mosaicos en  $F$  todas las subáreas. Bajo este planteamiento, se espera que el total sea de  $\sum_{f \in F} u_f = U \geq m \cdot n$  mosaicos disponibles.

- Se posicionan los mosaicos de acuerdo a la solución óptima del problema presentado a continuación, (3.1), y se despliega la imagen resultante de colocarlos, dicha imagen es el fotomosaico.

En (3.1) se muestra el planteamiento del vector de costos y el conjunto de restricciones introducido para el artículo realizado en la *Orbelin College*, “Opt Art” (Bosch, 2006).

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

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

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

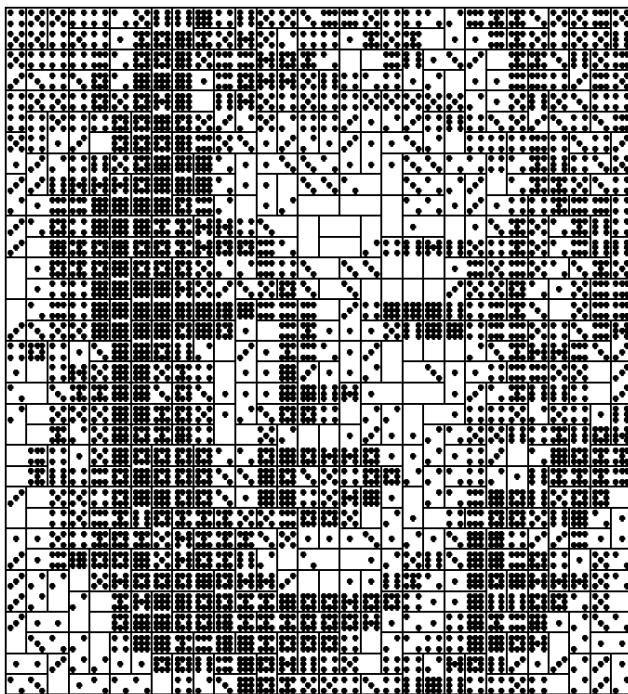
$$x_{fij} \geq 0 \quad \forall (f, i, j) \in \{1, \dots, cf\} \times \{1, \dots, m\} \times \{1, \dots, n\} \quad (3.1d)$$

Llámese  $z^* = \arg \min z$  al valor de la función objetivo en el óptimo del problema (3.1). 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 mosaico y el

rectángulo en el área subyacente sobre la cual se coloca el mosaico  $f$  en la posición  $(i, j)$  (Possani, 2012). La función de costos será de tipo tridimensional:  $c_{fij} = (bf - \beta_{ij})^2$ . La condición en (3.1b) garantiza que 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 admitidos por el modelo son dicotómicos: 0 o 1), el mismo se puede plantear como un problema con variables de decisión continuas y aún así se deriva de éste un resultado óptimo entero. Bosch, en su artículo *Opt Art* asegura esta integralidad, la razón de la misma es que el problema en cuestión es una instancia específica de una categoría de planteamientos de optimización combinatoria catalogados como “El Problema de Asignación” que busca asignar un número de objetos (tareas, recursos) a un número igual de destinatarios (personas, máquinas o, en este caso, rectángulos ordenados), de tal manera que se minimice el costo total de la asignación. La matriz de restricciones de “El Problema de Asignación” es totalmente unimodular, lo cual quiere decir que cada una de sus submatrices cuadradas son invertibles, con dichas submatrices teniendo un determinante igual a 0 o  $\pm 1$ . En programación lineal, esta propiedad garantiza que las soluciones básicas del problema relajado sean enteras. Esto es una noticia buena, ya que añadir la propiedad de “entero” o “integralidad” implica en la solución complejidades sustanciales que dificultan mucho la tarea de optimización; el algoritmo de ramificación y acotamiento para problemas enteros expresado en la sección 2.1 ilustra esta aseveración, pues la ramificación podría iterarse hasta  $m \cdot n$  veces.

El planteamiento referido cuenta con tres dimensiones en su vector de costos,  $c_{fij}$ : la primera corresponde al mosaico de tipo  $f$  (que va a

contrastarse en escala de grises con el rectángulo en cuestión) y la segunda y tercera a la coordenada del rectángulo (horizontal y vertical, respectivamente). Una variación a este problema consiste en fichas de dominó con un par 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 cada ficha puede cubrir dos rectángulos yuxtapuestos 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  $c_{wzijo}$ . Adicionalmente, 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; en el mismo, se muestran fotomosaicos del programa resultante, de los cuales en Figura 3.2 se despliega un ejemplo de retrato de Marylin Monroe que sigue el patrón de representar a la actriz con optimización lineal (tal como el del fotomosaico de LIFE mencionado antes en este capítulo). Naturalmente el problema planteado para mosaicos con fichas de dominó tiene más complejidad combinatoria que el planteamiento de esta sección (3.1), ya que se tienen dos dimensiones más en los costos, así como más restricciones. Si bien el desarrollo matemático de este problema no es objetivo de este escrito, sí ayuda a comprender el tipo de variaciones que puede un programador ingeniarse creativamente sobre el modelo de fotomosaicos al transformar el vector de costos y las especificaciones de las restricciones.



**Figura 3.2.** Ejercicio de Monroe por Robert Bosch usando 9 juegos (con 55 fichas por juego)

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

#### 3.3.1. Programa lineal

Una pregunta interesante es ¿cómo se construiría un fotomosaico usando el principio de optimización lineal planteado al inicio del capítulo, pero ahora en lugar de optimizar la asignación de escalas de grises, hacerlo en un contexto 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; este modelo se expresa como RYB por sus siglas en inglés, identificado como un modelo de tipo sustractivo (de hecho es el más común dentro de esta clase). 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, restándosele luminosidad, de esto se deriva el nombre de “sustractivo” (Castañeda, 2005). 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, la representación de un objeto por una pantalla depende de la emisión de la luz realizada por el dispositivo, de ahí que se vincule con un modelo de naturaleza opuesta, como 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 incorporación de valores numéricos de cada pixel que, en su conjunto, forman un espacio 3-dimensional. El modelo aditivo mas común en la informática y la fotografía 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 (estos son dos ejemplos: el modelo se elige dependiendo de los requerimientos del problema en curso). Una extensión del principio de la solución de fotomosaicos es minimizar la suma de los cuadrados de las distancias euclidianas, 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”; para este caso,  $\beta_{ij}^A = (\beta_{ijr}, \beta_{ijg}, \beta_{ijb})$  es el color del pixel “promedio” (concediendo que pudiera llamársele así y, en efecto, se puede comprobar que semejante pixel existe) y  $b_f^A = (b_{fr}, b_{fg}, b_{fb})$  es el color del mosaico  $f$ . Habiendo definido esto, generamos el problema lineal siguiente:

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

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

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

$$x_{fij} \geq 0 \quad \forall (f, i, j) \in \{1, \dots, c_f\} \times \{1, \dots, m\} \times \{1, \dots, n\} \quad (3.2d)$$

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 se encontraba en el intervalo de valores  $[0, 1]$  ahora los toma en  $[0, 3]$ . Vale la pena observar que, si bien la expresión parece más complicada debido a cómo se expresa el coeficiente de costos, en realidad el problema es en esencia idéntico con dos salvedades: la primera, que el cálculo de los costos es más largo; la segunda, que encontrar mosaicos 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 —adelantándonos a la implementación y la elección de mosaicos— si en el planteamiento en escalas de grises se deseaba tener 10 imágenes con escalas de grises promedio homogéneamente distribuidas sobre el rango  $[0, 1]$  (separadas por un *grid* de 0.1) ahora se necesitarían  $10^3 = 1000$  imágenes con escalas promedio distintas para cumplir la tarea análoga; por lo tanto la cardinalidad del conjunto mosaicos  $F$ , debe de aumentar considerablemente si se desea generar una representación legible de una imagen con tonalidades diversas, afectando por consiguiente la complejidad computacional de la solución del problema. En síntesis, la base detrás del planteamiento es idéntico, pero la inherente necesidad de incrementar la cardinalidad del conjunto de mosaicos a colocar implica incrementos en los requerimientos computacionales para la solución.

### 3.3.2. Idea del color promedio

En la subsección anterior se anticipaba que sí existe el concepto de un color dominante o promedio en una imagen. En efecto, desde la década de 1970 existe una teoría en torno a la cuantificación del color que busca comprimir una variedad de tonos de color a uno solo y que dió luz a nociones que hoy se manejan recurrentemente, relacionados, por ejemplo, con la popular extensión de archivo digital, JPEG. Hay dos conceptos centrales en torno a la cuantificación del color. El primero, es que esta representación se usa para reducir costos computacionales, en especial en dispositivos con capacidad limitada, al reducir óptimalmente la información desplegada por un archivo de imagen (tal es el caso del JPEG). El segundo, es que da al analista los elementos para entender

las variedades de color, ayudando a categorizar el color de una cosa o producto, o bien a derivar una paleta optimizada de una imagen.

Relacionado con la segunda cuestión, destaca el algoritmo de k-medias (universalmente conocido como *k-means* por su denominación en inglés) que permite usar exactamente la misma dimensión (renglon por columna,  $k \times h$  en términos del planteamiento explicado) de pixeles pero donde el total de distintos valores de los mismos en el hiperplano RGB sea limitado manualmente por el usuario, como un hiperparámetro. Antes de ahondar en este enfoque, conviene hacer una breve recapitulación de dicho algoritmo, elemental en el estudio del aprendizaje supervisado, cuyo objetivo es segmentar el universo en un agrupamiento que opera en función de la proximidad vista como distancia en espacios normados, comúnmente desde el enfoque euclídeo (Chong, 2021). En síntesis, para un conjunto de  $n$  observaciones en  $x$ , en un espacio euclídeo (indistintamente de su dimensión, para el análisis del espacio de color RGB la dimensión es 3), el objetivo del algoritmo es seleccionar, primeramente, el hiperparámetro  $k'$ , o número total de *clusters* / segmentos a generar; posteriormente se enfoca en seleccionar los mejores centroides, asignando a cada punto con su centroide más cercano (esto implica una asignación de *cluster*). De tal manera que el promedio sobre el total de observaciones de la suma cuadrática de la distancia de cada punto a su correspondiente centroide se vea minimizado. El algoritmo se ejecuta en bucle; sin embargo, en el resultado final computado, el centroide debe de ser igual al promedio,  $\mu_i$  del i-ésimo cluster generado. Visto matemáticamente, se optimiza la expresión (3.3), la cual se encuentra en el artículo “*K-means clustering algorithm: a brief review*” de Chong (2021).

$$z^* = \operatorname{argmin}_S \sum_{i=1}^{k'} \sum_{x \in S_i} \|x - \mu_i\|^2 \quad (3.3)$$

Donde  $S = \{S_1, \dots, S_{k'}\}$  constituye la segmentación óptima de *clusters* y la unión de los mismos compone el universo total de observaciones,  $\bigcup_{i=1}^{k'} S_j = \{x_1, \dots, x_n\}$ . Supóngase inicialmente el caso con un *cluster*, *i.e.*  $k' = 1$ . En esta casuística hay un único centroide  $\mu_1$ , y puede considerarse un ejercicio trivial donde el color “promedio” es el promedio aritmético en cada una de las tres dimensiones de los  $k \times h$  pixeles. Ejercicios empíricos pueden conducir a la conclusión de que este no es un muy buen valor para instanciar  $k'$ , ya que en una imagen con cierta variedad en su gama de colores, este promedio aritmético estaría capturando cierto ruido de aquellos segmentos de la imagen que, si bien no son muy grandes, generan contraste, haciendo que el promedio computado se desvíe del color que tiene el objeto central de la imagen. Regresando al problema de la imagen, si se asigna  $k' = 2$  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 valores de pixeles. Se desplegará cada centroide reemplazando al pixel original dando lugar a una imagen con la misma dimensión (renglón por columna,  $k \times h$ ) 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 en el espacio RGB, pues entonces el resultado es la misma imagen original. En la figura 3.3 se desarrolla la simplificación con k-medias, en un ejemplo con tres despliegues, cada uno con una  $k'$  distinta.

El caso aquí analizado corresponde a una imagen de flores blancas, presentada en la Figura 3.3a, con el objetivo de identificar el tono de



(a) Imagen original



(b) 2 tipos de píxeles



(c) 3 tipos de píxeles



(d) 10 tipos de píxeles

**Figura 3.3. Simplificación de imágenes basado en selección de píxeles-centroides del modelo k-medias (cuatro configuraciones para  $k'$ )**

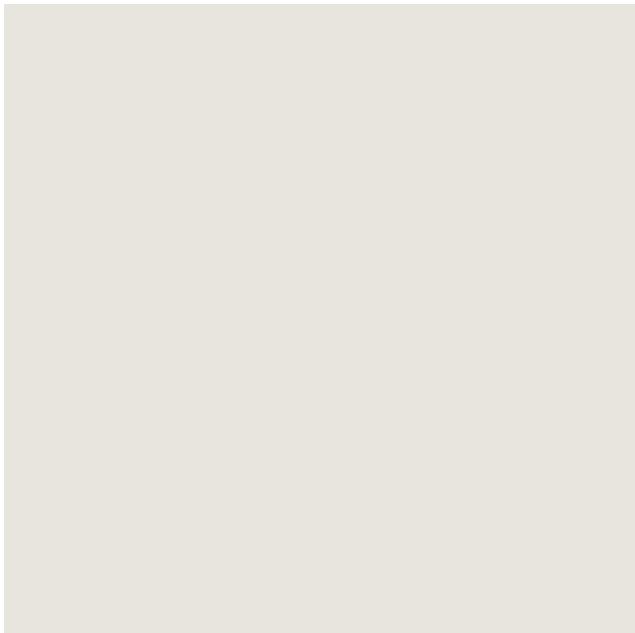
blanco predominante. La imagen tiene una resolución de  $517 \times 517$  píxeles, lo que equivale a  $517^2 = 267,289$  píxeles en total. Un análisis de agrupación (`COUNT(*) GROUP BY r, g, b`) revela que solo existen 58,168 combinaciones únicas de valores  $r, g, b$  (en el rango  $[0, 1]$ ), lo que indica una repetición significativa de colores. De estas combinaciones, solo 22 aparecen en más de 500 píxeles cada una, representando más del 0.2 % del total de píxeles. Notablemente, todas estas 22 combinaciones tienen valores de  $r, g, b > 0.90$ , lo que confirma que corresponden a diferentes tonalidades cercanas al blanco puro. Esto sugiere la posibilidad de agrupar estas tonalidades en un único *cluster*, junto con otras combinaciones de  $r, g, b$  en una vecindad cercana en el espacio RGB. Para explorar esta idea con un enfoque geométrico más formal, los experimentos en la Figura 3.3 aplican un algoritmo de segmentación k-medias para reducir la diversidad de

colores a  $k' = 2, 3$ , o 10 segmentos.

Ahora, la asignación de la  $k'$  depende de la clase de problema a abordar. En general para k-medias (en aplicaciones más allá de la cuantificación del color) existen varios criterios que permiten determinar una  $k'$  que satisfaga alguna condición “óptima”; dos métodos comunes son el método del codo y el análisis de la silueta. Como la participación del algoritmo k-medias en este escrito es meramente auxiliar y complementario para ilustrar el problema de optimización lineal en la asignación de mosaicos de color, se desea minimizar la discusión de pormenores del algoritmo. Para el caso específico de interés, a más homogénea es una imagen, más sencillo se espera que una  $k'$  “pequeña” 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 en imágenes complejas (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 cada mosaico a asignar del conjunto  $F$  sea una imagen con colores homogéneos, para así elegir el valor del centroide con más población de pixeles asociada, llamarle el “pixel promedio” del mosaico y asegurar que una  $k'$  pequeña (la cual será  $k' = 2$ ) sea suficiente para determinar rápida y eficientemente un color promedio. Tantas imágenes con atributos similares al de la flor blanca mostrada se pueden imaginar como colores de flores existan. Esto es, imágenes simples, uni-color, con un fondo nulo o chico para minimizar el ruido. En el capítulo de implementación esta inventiva se llevará a cabo análogamente, precisamente con flores. Similarmente, bajo esta idea de color promedio determinado con k-medias existirá un

“pixel promedio” en cada uno de los  $m \cdot n$  rectángulos, que posicionados contiguamente, conforman la imagen objetivo.

En relación a la imagen de la flor blanca, la presentación de la Figura 3.4 muestra el color dominante derivado de  $k' = 2$ , este color ya se conoció anteriormente en la Figura 3.3b, pues es aquél visualmente más abundante de ésta, o expresado con formalidad, el centroide determinado por el algoritmo con más pixeles asociados.



**Figura 3.4. Color promedio (blanco-gris dominante) de flores blancas derivadas de  $k' = 2$ .**

# Capítulo 4

## El problema del agente viajero (TSP)

### 4.1. Descripción general del problema

La teoría de grafos estudia las propiedades y relaciones de un grafo que es un par  $(V, E)$  con  $V$  un conjunto de vértices y  $E$  conjunto de aristas que los une. Siendo la arista modelada, ya sea como un par dirigido, donde el orden de los vértices importa (digrafo o arco, representado por una flecha), o bien, no dirigido (representado por una simple arista o recta). Existen problemas de optimización planteados sobre grafos que pueden ser resueltos con programación lineal, siendo el problema del agente viajero (*Traveling Salesman Problem*, TSP) uno de los más interesantes y populares (Desrochers, 1991). Este problema se define dentro del marco de los caminos hamiltonianos: dicho tipo de ruta se enfoca en visitar cada vértice de un grafo exactamente una vez. El TSP plantea que un individuo (el agente viajero) debe visitar una cantidad  $n'$  de ciudades partiendo de una ciudad origen y finalizando

el recorrido en dicha ciudad. El objetivo es minimizar la distancia total recorrida. Existe un tipo de parámetros necesario para definir el problema: la distancia que hay entre cada par de ciudades existente; para ello defínase  $c_{ij}$  como el valor de la distancia que hay entre las ciudades (o vértices)  $(i, j)$  para cada par  $(i, j) \in \{1, \dots, n'\}^2$ . Alternativamente, cada distancia  $c_{ij}$ , puede derivarse indirectamente si se tiene la ubicación de cada ciudad. Con lo explicado en el presente párrafo, se puede desarrollar intuitivamente una formulación matemática mediante la programación lineal entera en (4.1).

$$\min_{x_{ij}} \sum_{i=1}^{n'} \sum_{j=1}^{n'} c_{ij} x_{ij} \quad (4.1a)$$

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

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

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

$$x_{ij} \geq 0 \quad \forall (i, j) \in \{1, \dots, n'\}^2 \quad (4.1e)$$

Donde  $S$  es el conjunto de todos los subconjuntos posibles (conjunto potencia) generado con  $\{1, \dots, n'\}$ . El problema lineal esbozado en 4.1 se define en tres tipos de restricciones. Para entenderlo, defínase antes el conjunto de variables de decisión como  $\{x_{ij} | (i, j) \in \{1, \dots, n'\}^2\}$  con dos asignaciones de valor posibles: 0 y 1. Bajo este planteamiento,  $x_{ij} = 1$  describe que el agente realizó un recorrido directo partiendo de la ciudad  $i$  hacia la ciudad  $j$  con una distancia correspondiente de  $c_{ij}$ ;  $x_{ij} = 0$ ,

por el contrario, indica el caso opuesto: no se viaja de  $i$  a  $j$ . En términos técnicos, el agente viajero habría ocupado (o no) el arco 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; por su parte la condición (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 *tours* donde a cada ciudad se llega una única vez y cada ciudad parte hacia un único destino, pero dos o más subconjuntos resultado de la partición de  $\{1, \dots, n'\}$  generan cada uno un recorrido o *tour* cerrado. Supongamos que se generara 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  $T_1$ , sugiriendo que el agente pasó de  $T_1$  a  $T_2$  o  $T_3$  sin un arco de conexión, algo inviable en un tour continuo. En este escenario se viola el supuesto de que el agente parte de un punto y termina en el mismo tras recorrer todas las ciudades. Resumidamente, 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  $\{1, \dots, n'\}$  (también caracterizado como “conjunto potencia”), es una restricción con una elevada complejidad computacional asociada que debe simplificarse con adecuadas alternativas, como se hace en las restricciones de Miller, Tucker & Zemlin introducidas en 1960 en el artículo “*Integer programming formulations and traveling salesman problems*”. Sumado al planteamiento descrito, el TSP tiene como conjunto factible los  $n'!$  posibles ordenamientos de  $\{1, \dots, n'\}$ , de tal manera que los programas

que lo estudian operan sobre el campo de los “algoritmos de optimización combinatoria”.

A continuación, el desarrollo del enfoque propuesto por Miller, Tucker & Zemlin (indistintamente también referido como “MTZ”). Introdúzcanse  $n' - 1$  variables enteras,  $u_i, i \in \{2, \dots, n'\}$  que representan la posición / orden en que se visita la ciudad  $j$ , de tal manera que el mapeo de  $i \neq 1$  a  $u$  es inyectivo. Se formulan las  $(n' - 1)^2 - n'$  restricciones (4.2a), de subtour y las  $n' - 1$  restricciones (4.2b), especificadas para variables de orden. Ambas expresiones sustituyen de manera ahorrativa a la restricción (4.1d) que explora el conjunto potencia (de orden exponencial) de ciudades pasando a explorar el producto cartesiano de las mismas (significativamente más pequeño, de orden cuadrático). Complementariamente, este enfoque permite relajar las restricciones de integralidad:

$$u_i - u_j + (n' - 1)x_{ij} \leq n' - 2, i \neq j \quad (4.2a)$$

$$1 \leq u_i \leq n' - 1 \quad (4.2b)$$

La idea detrás de estas restricciones es que, si el agente viaja de la ciudad  $i$  a la ciudad  $j$  entonces el orden en que se visita la ciudad  $j$  debe ser mayor al orden en que se visita la ciudad  $i$ . En caso contrario, se tendría  $x_{ij} = 0$ , por lo cual es trivial deducir que se cumple  $u_i - u_j \leq n' - 1$ , pues  $u_i, u_j \in \{2, \dots, n'\}$ . Por lo cual la restricción garantiza que exista una solución factible con un único ciclo hamiltoneano (no hay multiplicidad de *subtours*); no obstante, las restricciones no garantizan el óptimo global, de donde se derivan alternativas de mejora para fortalecer las restricciones, alcanzando con las mismas un mejor óptimo respecto al enfoque MTZ original. Las

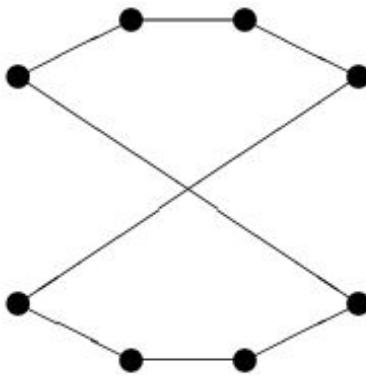
restricciones MTZ son una alternativa que reemplaza el número exponencial de restricciones de eliminación de subtours por un número polinomial (del orden de  $O(n'^2)$  ), lo cual representa un ahorro computacional sustantivo desde una formulación más compacta; esto, sin eliminar la complejidad NP-Dura inherente al problema. Este reforzamiento de la restricción puede extenderse a problemas de ciclos hamiltonianos más generales (enfoque más allá del TSP), tal como desarrolla Desrochers (1991) en su artículo que estudia problemas de rutas vehiculares (VRP's por sus siglas en inglés) “*Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints*”.

Aun con esta reducción con las restricciones de Miller, Tucker & Zemlin, el ritmo en que escala el tiempo de procesamiento computacional el programa lineal (*i.e.*, la complejidad) vuelve necesario el uso de heurísticas para hacer el cálculo con esfuerzos relativamente moderados. Esto ha sido comprobado tanto desde el enfoque empírico como desde la ciencia de la computación, demostrándose que no existe algoritmo alguno que resuelva el problema en tiempo polinomial; o, dicho en otros términos, se trata de un problema NP-Duro. En el artículo de Nilsson (2003) se explica un enfoque en la construcción de algoritmos para la resolución del TSP. Se basa en cuatro ejes conceptuales para discutirlos: *i*) construcción de tours, *ii*) mejora de tours, *iii*) tiempo de cómputo (asociado a la complejidad) y *iv*) la lejanía de la función objetivo en la solución respecto a la cota inferior de *Held-Karp* (asociado a la optimalidad del algoritmo). La *Held-Karp* es el valor óptimo del problema relajado asociado al TSP original, haciéndolo resolvable vía Simplex, o bien, aproximable con algoritmos. De tal manera, Nilsson discute los algoritmos con los cuales se construye una solución o tour factible,

abordando su complejidad y distancias promedio resultantes (dentro de un conjunto de instancias de TSP simuladas) respecto a la cota *Held-Karp*. Esta distancia varía desde el 25 % en el algoritmo de vecino más cercano hasta el 10 % en el algoritmo de Christofides.

Posteriormente, se mejora el tour construido tras la selección del método de construcción, habiendo algoritmos de *2-opt*, *3-opt*, *k-opt* (la expresión general con  $k \in \mathbb{N}$ ) y sus correspondientes derivaciones. Para el *2-opt* se seleccionan dos ciudades del tour y se reconectan siempre y cuando el tour resultante tenga una mejor solución. Este movimiento puede ser también interpretado como un reverso a un segmento del tour inicial. La selección de dos ciudades para este reverso tiene la ventaja de solo contar con una posible combinación en la reconexión de nodos, tal como lo muestra la Figura 4.1. Nilsson estima una desviación respecto a la cota de *Held-Karp* de 5 % para el *2-opt*; para el *3-opt* (el cual sigue una lógica similar pero con dos alternativas para la reconexión de nodos) se reduce a 3 %. Si bien los grados de mejora a un tour realizados desde este enfoque son relevantes, el cómputo sigue siendo complejo, pues cada iteración implica una evaluación de cada pareja posible asociable a cada ciudad. Una manera efectiva de reducir esta complejidad es cotejar a cada ciudad versus sus  $m'$  vecinos más cercanos, con  $m'$  el hiperparámetro a elegir. Esta variación o aceleración del algoritmo *2-opt* requiere cierto nivel de sensibilidad del problema para elegir adecuadamente la  $m'$  y simplifica la optimización permitiendo reducir la complejidad del algoritmo por un grado polinomial (2 a 1), o de  $O(n'^2)$  a  $O(m' \cdot n')$  (Nilsson, 2003).

Finalmente, el principio de mejora por *k-opt* es extendible a una completa gama de variaciones que se le derivan, tales como el algoritmo de Lin-Kernighan, que elige la  $k$  óptima para cada iteración, o el algoritmo de búsqueda tabú, el cual asigna costos a optar por



**Figura 4.1. Única representación del movimiento  $2\text{-opt}$**

determinados movimientos en la elección de ciudades, catalogados como “movimientos tabú”. Asimismo, hay enfoques de probabilidad en los plantamientos heurísticos que implican simulaciones aleatorias y replican comportamientos biológicos; algunos ejemplos son los Algoritmos Genéticos y la Optimización de Colonia Hormiguera.

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

La segunda implementación de replicación de una imagen fuente vía optimización lineal es con TSP: parte del supuesto de que se cuenta con una imagen en blanco y negro, desplegado y expresado como escala de grises. Procedemos de la siguiente manera, basado en los pasos indicados por Bosch (2003) en su artículo “*Continuous line drawings via the traveling salesman problem*”

- Particionamos la imagen en rectángulos conformados por píxeles.

Cada rectángulo será de  $k$  pixeles a lo largo por  $h$  pixeles a lo ancho. Los rectángulos se identificarán en función de sus coordenadas o posición: horizontal —o columnas— siendo  $n$  el total de éstos; y vertical —o renglones— totalizando  $m$ . De tal manera que la matriz final resultante cuenta con dimensiones  $\mu' \times \nu'$  con  $\mu' = k m$  y  $\nu' = h n$ ; asimismo, obtenemos una escala de grises para cada pixel en el rango discreto de 0 a 255 (empleando la conocida convención de 256 escalas mencionada en la subsección 3.3.2).

- Obtenemos la escala de gris promedio de cada rectángulo y fijamos un parámetro que ponderará el nivel de saturación máxima de ciudades viable para cada rectángulo:  $\gamma \in [4, 9] \cap \mathbb{N}$ <sup>1</sup>. De tal manera, para el rectángulo  $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$  la escala de gris del pixel promedio será de  $\psi_{ij}$ , con valores entre 0 y 255. Entonces constrúyase

$$g_{ij} = \lfloor \gamma - \gamma \psi_{ij} / 255 \rfloor \quad (4.3)$$

Ésta será una escala de oscuridad del rectángulo y el parámetro  $\gamma$  introducido por Bosch definirá el rango de saturación de ciudades o de detalle admisible en la imagen.

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

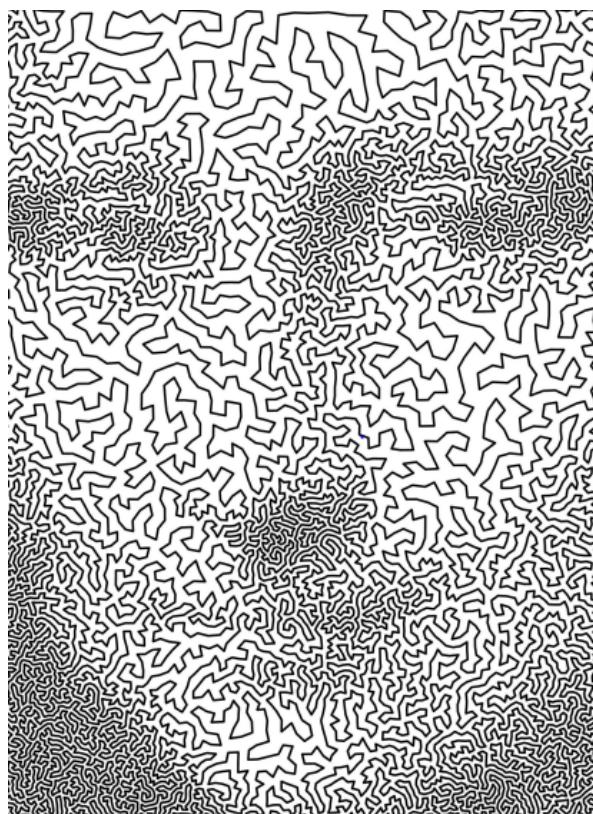
---

<sup>1</sup>Este rango de  $[4, 9]$  no tiene otra justificación más allá de la visual, *i.e.* se define a la luz de ejecuciones empíricas del planteamiento y las determinó Bosch en el artículo. Cabe destacar que valores superiores a 9 podrían generar imágenes sobre-saturadas de puntos

$$\tau = \sum_{i=1}^m \sum_{j=1}^n g_{ij} \quad (4.4)$$

- Computamos las distancias (o “costos”) entre las  $\tau$  distintas coordenadas en una matriz  $C = [c]_{ij} \in \mathbb{R}^{\tau \times \tau}$ . Resolvemos el TSP sobre estas distancias (se recomienda usar alguna heurística para hacer el problema escalable). Se conectan las ciudades por aristas dando lugar a una representación de aristas en el plano cartesiano; este es el resultado final.

Siendo que ahora el planteamiento se esboza sobre  $\tau$  ciudades podemos entender que ahora la  $n'$  introducida en la sección anterior tiene un determinado valor en el problema del arte con línea continua:  $n' = \tau$ .



**Figura 4.2. Rostro de la Monalisa reproducido por Robert Bosch usando el problema del agente viajero**

# Capítulo 5

## Implementación y resultados

### 5.1. Partición de la matriz en rectángulos

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) presente. 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, con 8 bits por pixel. Si bien actualmente hay variaciones adicionales 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  $e_{255} = \text{round}(255e_1)$ ; la transformación en sentido inverso será con  $e_1 = \frac{1}{255}e_{255}$ .

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 los capítulos 3 y 4. Para iniciar los ejercicios se ha decidido usar una imagen de la extinta red social Twitter (ahora llamada X tras su adquisición por xAI en 2022), cuyo símbolo, un pájaro, se muestra en Figura 5.1. Una librería / paquete del reconocido lenguaje *open source* de programación matemática, *R*, encargada de procesar y graficar las imágenes de tipo png tiene por nombre “png”. La función *readPNG()* lee la imagen especificada (indicándole la ruta del archivo en la computadora) y la transforma en un arreglo numérico con tres ejes (dimensión  $\mu \times \nu \times 3$ ): el primero representa la cantidad de píxeles por fila ( $\mu$ ), el segundo por columna ( $\nu$ ), y el tercero codifica la intensidad de cada componente cromática del modelo tridimensional RGB (rojo, verde y azul). 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 que conforman 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*”.

Como siguiente paso, se usan las dimensiones de pixeles,  $m$  y  $n$ , para delimitar el tamaño de la matriz que permita generar una partición



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

factible de la misma en rectángulos; para las instancias a desarrollar se requerirán rectángulos de dimensiones idénticas que, posicionadas estratégicamente, generen la matriz grande (o matriz total). Para ello analizamos residuos de divisiones con aritmética modular para pasar 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  $m \cdot n$  rectángulos mediante la función *matsplitter(.)* de R, la cual recibe como entradas la matriz a fraccionar, la cantidad de renglones (largo) por rectángulo,  $k = \mu'/m$  y la cantidad de columnas (ancho),  $h = \nu'/n$ . Luego, *matsplitter(.)* entregará una lista de  $m \cdot n$  elementos. Los elementos 1 a  $n$  de la lista consistirán en los primeros  $n$  rectángulos superiores de la matriz, los elementos  $n + 1$  a  $2n$  consistirán en el segundo grupo de  $n$  rectángulos superiores. Así sucesivamente, hasta llegar a los  $n$  rectángulos inferiores, enumerados  $n \cdot m - n + 1$  a  $m \cdot n$ . Entendemos que este tipo de ordenamiento es de clase *rowwise* (o por renglón), ya que se va llenando renglón por renglón, empezando por

---

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

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 luminosidad promedio de cada elemento del output de *matsplitter()* computado como un promedio aritmético escalar de grises sobre el total de pixeles de cada rectángulo. Dicho conjunto de luminosidades puede sintetizarse en un vector de dimensión  $m \cdot n$ , 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). Llámese  $h$  al vector resultante de extraer estas escalas promedio de los rectángulos ordenados con base en el esquema *rowwise*, definido aquí por  $\phi : r \rightarrow (i, j)$ . Este mapeo asigna cada índice  $r \in \{1, \dots, m \cdot n\}$  a una coordenada  $(i, j)$  de una matriz  $m \times n$ , según:

$$\begin{aligned} j &= ((r - 1) \bmod n) + 1, \\ i &= \left\lfloor \frac{r - 1}{n} \right\rfloor + 1. \end{aligned}$$

Esta correspondencia, recorre la matriz por filas, asociando cada  $r$  a una única posición  $(i, j)$ .

<sup>2</sup>Arreglo por columna, o *columnwise* donde se llenan las columnas de izquierda a derecha, cada columna se llena de arriba a abajo.

Supóngase que las dimensiones de las imágenes / mosaicos que compondrán el fotomosaico son cuadradas, de tamaño  $p \times p$ . Entonces, generaremos tres arreglos importantes: uno de parámetros de costos, dos de escalas de grises. El primero será uno numérico 3-dimensional de tipo  $cf \times m \times n$ , llámese  $A$ , con  $A[i, j, f] = c_{fij}$ , donde el lado derecho es el mismo ponderador de costos de la expresión (3.1a). Uno de los arreglos en escala de grises 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 de cantidades de rectángulos,  $m \times n$ , que se ingresan como *input* (ya que sustituye cada rectángulo por el valor en escala de grises del pixel promedio en el mismo); el otro, será un arreglo de  $(m \cdot p) \times (n \cdot p)$  y se denotará  $MF$ , por “matriz final”, ya que se trata del arreglo que representa el resultado final del fotomosaico. Además, se contará con una dotación de  $\sum_{f=1}^{cf} u_f$  imágenes de solo  $cf$  tipos, cada uno matricialmente denotado mediante  $MG_f$ , o “matriz de *grayscale*” de la imagen tipo  $f$ .

Mediante un bucle, el arreglo tridimensional  $A$  se transforma en un *data frame* bidimensional  $df3$ , en un esquema que involucra a los campos  $t$ ,  $i$ ,  $j$ ,  $f$  y  $c'$ <sup>3</sup>. Este proceso establece un mapeo biyectivo  $\eta : (i, j, f) \rightarrow t$ , definido en el Apéndice A.2, que tiene su respectivo inverso  $\eta^{-1}$ . Dicho mapeo permite expresar los costos y variables de decisión unidimensionalmente, asociando a cada costo  $c_{fij} \cdot x_{fij}$  un costo equivalente  $c'_t \cdot x_t = c'_{\eta(i,j)} \cdot x_{\eta(i,j)}$ . Este remapeo permite algo fundamental: que la librería de  $R$  elegida para computar el problema de programación lineal, *lpSolve*, admita los argumentos aportados, ya que se exige al vector de costos,  $c_{fij} = (b_f - \beta_{ij})^2$ , indexarse a una

---

<sup>3</sup>La estructura del *data frame*  $df3$  se detalla en el Apéndice A.2.

única dimensión,  $[c']_t$ . A continuación se describe el procedimiento para llenar el arreglo de costos  $A$  computado en  $R$ , con  $A[i, j, f] = c_{fij}$  y la “Matriz Perfecta” que representa un fotomosaico con el valor de la función objetivo igual a 0. Este proceso se basa en un bucle guiado por  $r \in \{1, \dots, m \cdot n\}$ .

1. Asignar el número de iteración  $r$  a su correspondiente coordenada matricial  $\phi(r) = (i, j)$ , determinada por el esquema *rowwise* descrito previamente.
2. Obtener el valor escalado en  $[0, 1]$  de la luminosidad promedio del rectángulo  $(i, j)$ , denotado como  $h_r$ , calculado a partir de los píxeles asociados.
3. Calcular la diferencia cuadrática entre  $h_r$  (escalado en  $[0, 1]$ ) y cada componente del vector de intensidades promedio de las  $cf$  figuras disponibles, denotado como  $\mathbf{I}_{\text{prom}}$  (de dimensión  $cf$ ), resultando en un vector  $\mathbf{D}_{ij} = [(h_r - I_{\text{prom},f})^2]_{f=1}^{cf}$ . Asignar este vector  $\mathbf{D}_{ij}$  a la submatriz  $A[i, j, :]$ , llenando así los  $cf$  elementos de la tercera dimensión para cada par  $(i, j)$ ; iterando este proceso sobre las posiciones  $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$  se genera el array  $A$  de dimensiones  $m \times n \times cf$ .
4. Asignar el valor  $h_r$  a la posición  $(i, j)$  de la “Matriz Perfecta”  $MP$ , representando la luminosidad ideal en esa región.

### **Adaptación a módulo IpSolve y redimensionamiento**

La mayoría de los módulos de *software* de código abierto percibe los costos de un programa lineal como un vector unidimensional. En contraste, el planteamiento descrito en el Capítulo 3 utiliza tres dimensiones. Por ello, se construye un arreglo bidimensional  $df3$  a

partir de  $A$ , reorganizando la información en renglones y columnas, como se detalla en el Apéndice A.1 bajo el mapeo  $\eta$ . 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 o costos,  $c'$ ,
- (b) una instrucción de optimización,  $io$ : “minimiza” o “maximiza” (en este caso se minimiza),
- (c) una matriz  $RM$  de restricciones con dimensiones  $re \times (m \cdot n \cdot cf)$  que representa a las  $re$  restricciones existentes sobre las variables tal que  $\forall q \in \{1, \dots, re\}$  la  $q$ -ésima desigualdad se define con base en un lado izquierdo que es el producto punto entre el  $q$ -ésimo vector renglón de  $RM$ , denotado  $\mathbf{RM}_q$ , <sup>4</sup> y el vector de variables  $x$ , es decir,  $\mathbf{RM}_q \cdot x$ ,
- (d) un vector de lados derechos,  $rh \in \mathbb{R}^{re}$ , con el valor contra el cual será contrastada la combinación lineal de  $x$ , y tal que el lado derecho de la  $q$ -ésima desigualdad sea  $rh_q$  y,
- (e) un arreglo,  $sign$ , de caracteres que define la naturaleza de cada restricción como ecuación o inecuación, con el cual,  $\forall q \in \{1, \dots, re\}$ , la restricción  $q$ -ésima se define como la relación entre el producto punto del lado izquierdo descrito en (c), es decir,  $\mathbf{RM}_q \cdot x$ , y el lado derecho especificado en (d), es decir,  $rh_q$ , a través de la relación indicada por  $sign[q]$ , resultando en  $\mathbf{RM}_q \cdot x \text{ "sign}[q]" rh_q$ , donde  $sign[q] \in \{"<", "\leq", ">", "\geq", "="\}$ .

---

<sup>4</sup>Notación:  $\mathbf{RM}_q$  denota el  $q$ -ésimo renglón de la matriz  $RM$ , equivalente a  $RM[q, :]$  en la indexación de  $R$ .

## Holgura y variabilidad: Construcción del vector de disponibilidades

Hay que construir un vector,  $u$  que cuantifique las dotaciones disponibles de las  $cf$  figuras. Estas dotaciones pueden responder a tantas clases de formulaciones como pueda imaginarse el programador; como una observación trivial pero conveniente, siempre será necesario definirlo tal que la región factible sea no vacía. El planteamiento de  $u$  aquí presentado es una propuesta original de este trabajo. En éste, se emplean dos parámetros: el primero,  $hd$ , de holgura, que indica el exceso de mosaicos respecto a los estrictamente necesarios. Por ejemplo, si  $hd = 0.1$ , se incluyen aproximadamente un 10 % más de mosaicos para cubrir los  $m \cdot n$  espacios habilitados. Para garantizar la viabilidad del problema,  $hd$  debe ser estrictamente no negativo; de lo contrario, la escasez de mosaicos resultaría en una región factible vacía, haciendo el problema irresoluble.

Si deséaramos que las disponibilidades de cada mosaico fueran las mismas, tendríamos  $\lceil \frac{m \cdot n \cdot (1+hd)}{cf} \rceil$  imágenes de cada tipo. Sin embargo, para darle mayor variabilidad a estas disponibilidades introducimos el segundo parámetro  $vd$ , que denota variabilidad, y nos permitirá tener una dotación del mosaico  $f$  extraída aleatoriamente (utilizando distribución uniforme) de entre el siguiente conjunto de enteros consecutivos,  $u_f \in \{\lceil \frac{m \cdot n \cdot (1+hd)}{cf} \cdot (1 - vd) \rceil, \dots, \lceil \frac{m \cdot n \cdot (1+hd)}{cf} \cdot (1 + vd) \rceil\}$ . A mayor  $vd \in [0, 1]$ , y suponiendo una correcta simulación aleatoria, mayor será la varianza del vector ( $u$ ) de disponibilidades de los tipos de mosaico. Habiendo en dichos casos una mayor diferencia entre las dotaciones del tipo de mosaico más común (o poblado) versus el menos común.

Con lo 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 rectángulo y despliegue del resultado**

En la implementación en *R* del problema (3.1), no se impusieron restricciones de integralidad ni la no negatividad ((3.1d)), ya que la matriz de restricciones, totalmente unimodular según la sección 3.2, garantiza soluciones enteras ( $x_{fij} \in \{0, 1\}$ ). Además, los coeficientes no negativos de la función objetivo y la estructura de asignación aseguran soluciones no negativas, simplificando la optimización con *lpSolve*.

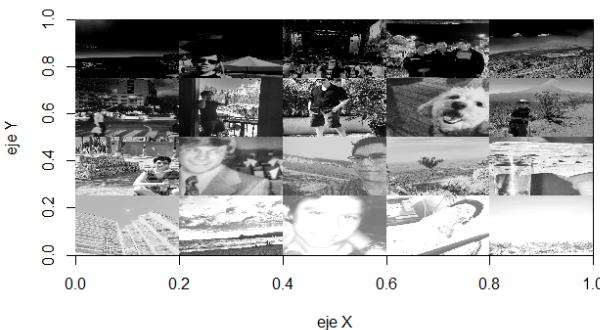
Extrayendo los elementos de la salida de *lpSolve* que cumplen  $T = \{t | x_t = 1\}$ , en una sola dimensión, y haciendo su mapeo a las posiciones de mosaicos, se valida que las  $m \cdot n$  combinaciones  $(i, j)$  son cubiertas exactamente una única vez cada una y que cada figura en  $F$  se ocupa 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  $p \times p$  y sea  $MG_f \in \mathbb{R}^{p \times p}$  la matriz de escala de grises de la figura  $f$ , se procede a llenar el fotomosaico, o matriz final  $MF$ , mediante el siguiente *loop*: a través de  $m \cdot n$  iteraciones hacer lo siguiente únicamente a través de las soluciones  $\kappa \in T$ .

1. Extraemos el  $\kappa$ -ésimo elemento. Sobre éste se asigna una  $(i, j, f)$  correspondiente, definida en función de  $\eta^{-1}$ , el inverso del mapeo mencionado al inicio de la sección 5.2.1. Con eso se define posición de rectángulo y ficha a colocar.
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,  $MF[rI, cI]$ , en donde  $rI = (i - 1) \cdot p + 1$  y  $cI = (j - 1) \cdot p + 1$

- Sigue definir al resto de la figura que llenará la posición  $(i, j)$ . Se asigna el valor de  $MG_f$  a la submatriz-rectángulo de pixeles con el extremo superior izquierdo en  $(rI, cI)$ ; esto es,  $MF[\{rI, \dots, rI + p - 1\}, \{cI, \dots, cI + p - 1\}] = MG_f$

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 en escalas de grises.

El siguiente paso es ejecutar los pasos descritos hasta este punto de la sección usando una gama de mosaicos de dimensiones idénticas  $p \times p$  y buscando emular el logo de Twitter. Para ello, se generó un código de  $R$  que lee todos los archivos png de una subcarpeta y extrae sus dimensiones (pixeles). Posteriormente, elige la mínima dimensión (de entre columnas y renglones) del conjunto de imágenes, acota a todas las imágenes de manera que cumplan con esta dimensión a lo largo y ancho y guarda su luminosidad promedio en un vector. Además, define una partición uniforme de luminosidad,  $cf \in \mathbb{N}$ . Esto es, si  $cf = 20$  entonces las figuras se clasifican en 20 luminosidades:  $\{[0, 0.05), [0.05, 0.10), [0.10, 0.15), \dots, [0.90, 0.95), [0.95, 1]\}$ . Para el primer ejercicio extraemos las imágenes de un conjunto de 60 fotos de una cuenta de Instagram. Un primer paso es reducir la dimensionalidad de estas 60 fotos (o mosaicos) a un  $cf \leq 60$  arbitraria. 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 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 en Figura 5.2.



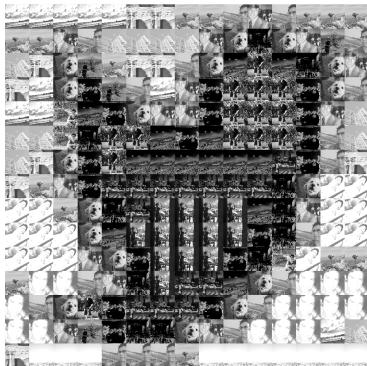
**Figura 5.2. Catálogo de imágenes seleccionadas para el fotomosaico**

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

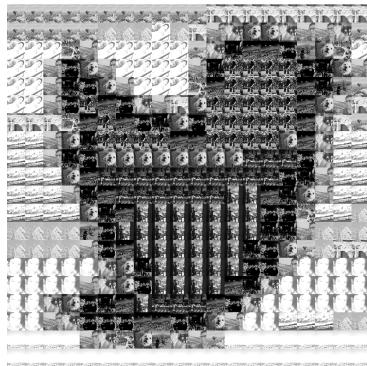
Para instancias con dimensiones mayores también hubo despliegues, los cuales se encuentran en Figura 5.4.

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 de resolución del programa lineal. Para ello, registramos el tiempo que lleva resolver para la función *lp()* el problema asociado a una determinada imagen conforme se agregan más dimensiones. Esto, en



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

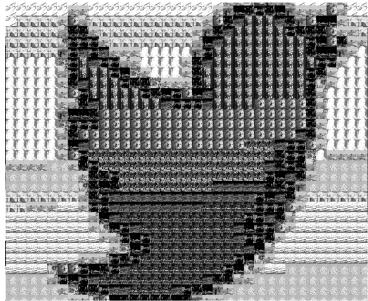


(b) Fotomosaico: 20 tipos de figuras,  $20 \times 20$  mosaicos

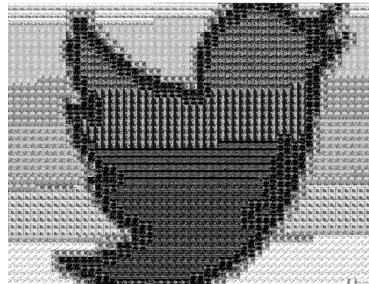
**Figura 5.3. Soluciones gráficas al problema de los fotomosaicos con  $dp = 0.3$  y  $vp = 0.3$ ; dimensiones  $15 \times 15$  y  $20 \times 20$**

una PC con sistema operativo Microsoft, procesador Inter Core i7 de octava generación y 8 Gigabytes (GB) de memoria RAM. 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 con *grid.raster(.)*.

Las gráficas en Figura 5.6 y tablas asociadas (Cuadro 5.1 y Cuadro 5.2) reflejan la relación que tiene la dimensión  $m = n$  (suponiendo que hay tantos rectángulos a lo largo como a lo ancho) con el tiempo de cómputo (en segundos) requerido para generar la imagen de Twitter, en subfigura 5.6a; también, la que hay entre el total de rectángulos (la dimensión al cuadrado) y el tiempo, reflejado en subfigura 5.6b. 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



(a) Dimensiones  $35 \times 35$

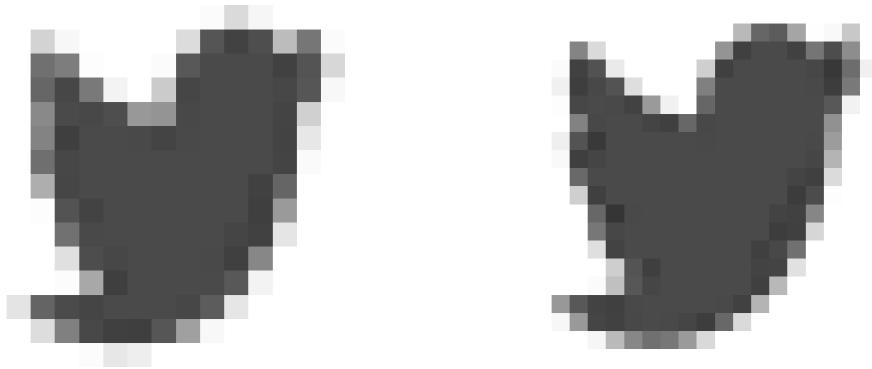


(b) Dimensiones  $60 \times 60$

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

sus estimaciones lineales, cuadráticas y cúbicas dan a entender que la ejecución es cuando menos de orden 3 en el número de dimensión,  $m = n$ . Esto se explica particular por el ajuste aproximado en la subfigura 5.6a, donde es evidente que el comportamiento es no lineal y tiene un ajuste cuadrático extraño al tener su vértice en valores de “rectángulos negativos”, cuestión absolutamente contraintuitiva. No es sino hasta el estimado cúbico que se distingue una relación asociable. Se estima entonces complejidad polinomial,  $O(n^3)$ .

El Cuadro 5.2 de valores óptimos de la función objetivo,  $z^*$ , introduce una interrogante interesante; ¿qué tan bueno o malo es el valor de la función objetivo en el óptimo, es decir  $z^*$ , para 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 “costo” de desplegar el mosaico  $f$  en el rectángulo  $(i, j)$  es una diferencia cuadrada de valores, ambos entre 0 y 1, teniendo por lo tanto un valor máximo de 1. Por lo tanto, el rango factible de valores

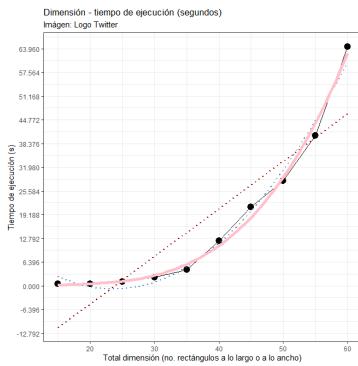


(a) Matriz perfecta,  
dimensiones  $15 \times 15$

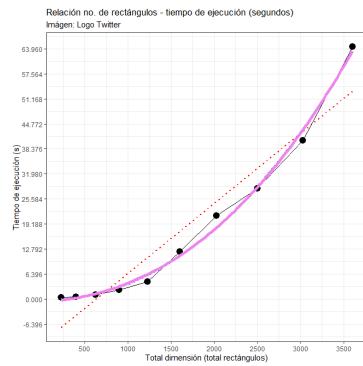
(b) Matriz perfecta,  
dimensiones  $20 \times 20$

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

de la expresión lineal  $z$  es de 0 al número total de rectángulos,  $m \cdot n$ . De tal manera que, como podría intuirse, el valor óptimo de la función objetivo muy probablemente subirá en la medida que se amplían las dimensiones del problema. Sin embargo, al tratarse de un desarrollo artístico con inherente subjetividad, el determinar si ello representa una buena o mala reproducción dependerá de la óptica con la que se evalúe el resultados. Al normalizar el valor óptimo  $z^*$  del problema (3.1) respecto al peor escenario, definido como el valor máximo teórico de la función objetivo, el cociente  $\frac{z^*}{m \cdot n}$ , expresado en porcentaje, permite una comparación estandarizada entre instancias. Según el Cuadro 5.2, en diez instancias de Twitter con dimensionalidad variable, el valor de  $\frac{z^*}{m \cdot n}$  oscila entre 2.5 % y 3.5 %, indicando alta consistencia. Para los parámetros de holgura  $hd = 0.3$  y variabilidad  $vd = 0.3$ , este rango, inferior al umbral de 5 %, refleja soluciones de calidad óptima.



(a) Dimensión vs tiempo



(b) Rectángulos vs tiempo

Figura 5.6. Graficaciones tiempos de ejecución

Una cuestión interesante de abordar es el atributo de escasez de mosaicos en el problema, el cual se define 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, color blanco). Esto incluye, naturalmente, a los extremos inferior derecho y superior derecho. Sin embargo, al ver el fotomosaico en la 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 sería deseable que todos los contornos tuvieran el máximo de luminosidad, existe una disponibilidad limitada del mosaico más luminoso, así como de todos en general. Por lo tanto, se recurre a otros mosaicos, lo menos oscuros posibles, que minimicen la suma de diferencias cuadráticas de luminosidades para cubrir el contorno de la imagen sin exceder los recursos existentes. Haciendo uso de los parámetros introducidos y

Dimensión $n = m$	Total de 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

**Cuadro 5.1. Resultados de ejecución para diferentes tamaños de mosaicos.**

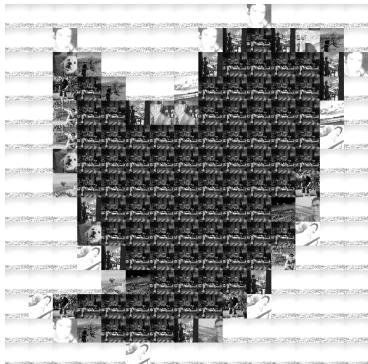
explicados en el presente escrito, podríamos hacer el ejercicio sin limitar estos recursos. Basta indicar que  $hd = m \cdot n$ , queriendo decir que, si tuviéramos dimensiones  $m \times n = 15 \times 15$ , entonces los mosaicos disponibles son  $15^2 = 225$  y, si asignamos  $hd = 225$ , entonces dispondremos de 22500% más de los mosaicos estrictamente necesarios. Además, si  $vd = 0$  las disponibilidades se distribuirán homogéneamente, con lo cual garantizamos que cada tipo de mosaico,  $f$ , tiene el potencial de llenar por sí mismo todo el fotomosaico. Cabe mencionar que esta reparametrización tiene una alternativa: eliminar las restricciones del tipo definido en las desigualdades (3.1b), que usan al vector  $u$ . 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

Dimensión $n = m$	Total de mosaicos	Valor óptimo, $z^*$	Valor $z^*$ 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 %

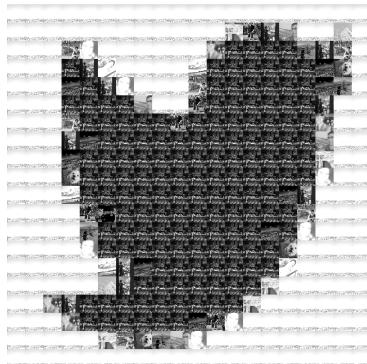
**Cuadro 5.2.** Valor óptimo  $z^*$  y su comparación porcentual con el valor máximo por varias dimensiones de rectángulos.

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 al describir casos extremos. En Figura 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 la “Matriz Perfecta”, Figura 5.5, puesto que el costo total de diferencias cuadradas de luminosidades (valor de la función objetivo) se reduce al limitar menos las disponibilidades. Para estas instancias tenemos resultados de distancias  $z^* = 0.30$  y  $z^* = 0.59$  para  $n = 15$  y  $n = 20$  respectivamente, representando en ambos casos 0.13 % y 0.15 % del supremo del rango de la función objetivo. Recordemos que en ejercicios previos con



(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  $dp = m \cdot n$  y  $vp = 0$ ; dimensiones  $15 \times 15$  y  $20 \times 20$**

holguras de disponibilidad más amplias estos porcentajes eran del orden del 3%; es decir, con niveles hasta 20 veces menores. 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 mosaicos disponibles se están empleando, lo cual es indicador de que la imagen carece notablemente de variedad en escalas de intensidad lumínica. Esto nos podría hacer perder interés en esta clase específica de instancias, no por estar mal ejecutadas (el resultado es muy similar en distancia de *greyscales* en cada rectángulo a la imagen de Twitter), sino por la falta de variedad de mosaicos ocupados.

Finalmente, vale la pena explorar la posibilidad de construir una

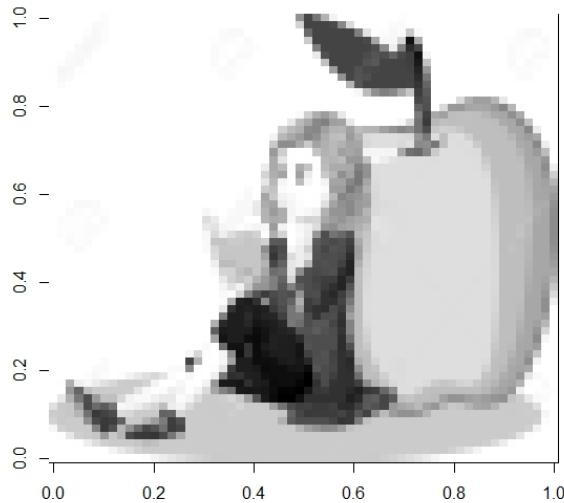
imagen más elaborada, con mayor relieve y color que el logo de Twitter. Para ello, se propuso reproducir el ejercicio de los fotomosaicos con una ilustración vectorial caricaturizada del connotado científico británico, Isaac Newton, disponible en la Figura 5.8 (blueringmedia, n.d., Depositphotos).



**Figura 5.8. Imagen objetivo: caricatura de Isaac Newton**

En ejercicios anteriores ya se demostró que el algoritmo computado para este trabajo soporta composiciones de imágenes de hasta  $60 \times 60$  rectángulos. Por ello, se optó por una asignación ligeramente inferior a ese límite, de  $50 \times 50$ . Reproduciendo de nueva cuenta la escala de grises del promedio en cada rectángulo, se genera el resultado de la matriz objetivo (o “Matriz Perfecta”), desplegada en Figura 5.9.

Con base en el planteamiento inicial que propuso reproducir una imagen más elaborada que el logo de Twitter mediante fotomosaicos, utilizando la ilustración caricaturizada de Isaac Newton (Figura 5.8, blueringmedia, n.d., Depositphotos), los resultados presentados en la Figura 5.10 confirman la viabilidad de este enfoque. La configuración

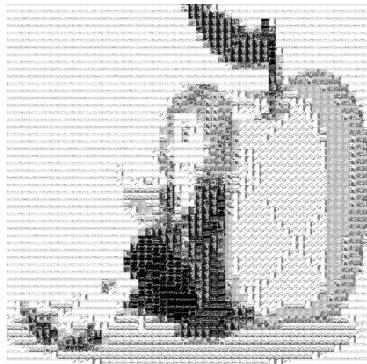


**Figura 5.9. Fotomosaico de Isaac Newton, Matriz Perfecta**  
 $50 \times 50$

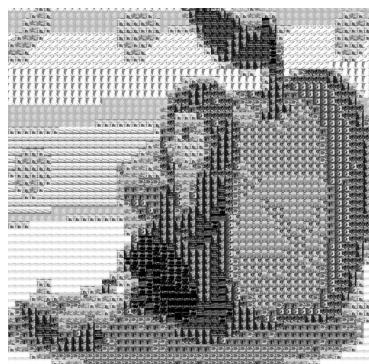
del vector  $u$  (construido a partir de  $hd$  y  $vd$ ) con holgura indefinida ( $(hd, vd) = (m \cdot n, 0)$ ) y la parametrización con variabilidad controlada ( $(hd, vd) = (0.3, 0.3)$ ) demuestran que se logró reproducir una imagen con mayor complejidad y detalle (la de Newton) de forma efectiva, destacando la robustez del modelo frente a distintas clases de instancias.

### 5.2.2. Fotomosaicos a color

En el Capítulo 3 que plantea el 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



(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  $dp = m \cdot n$  y  $vp = 0$ ; y sin holgura de  $dp = 0.3$  y  $vp = 0.3$**

elección de variedad de mosaicos a elegir, ya que éstos representan una dimensión superior al del escala de grises en proporciones cúbicas; la partición las  $m \cdot n$  submatrices-rectángulo en las tres dimensiones del espacio *RGB* (es decir, una adecuación a la ya referida función *matsplitter(.)* de *R*); y el posicionamiento del resultado final haciendo yuxtapuestos los arreglos matriciales de dimensión tres. Sobre todo esta primera consideración hace no tan trivial la labor de reproducir el ejercicio de los fotomosaicos con cualquier imagen.

Por consiguiente, se trabaja partiendo de un ejercicio con una imagen fuente relativamente limitada en cuanto a colores, dígase la bandera de México, y a raíz de eso, selecciónense los colores a ocupar. El concepto propuesto en la subsección 3.3.2 de la foto de una flor blanca con fondo de intensidad mesurada en la Figura 3.3 se extiende

a otros colores identificables en la bandera nacional: rojo, verde, café; complementariamente añádese un rosa y un anaranjado para verificar que el algoritmo acierte al minimizar su uso, ya que a simple vista no parecieran figurar como colores en el símbolo patrio. En Figura 5.11, está la bandera a reproducir.

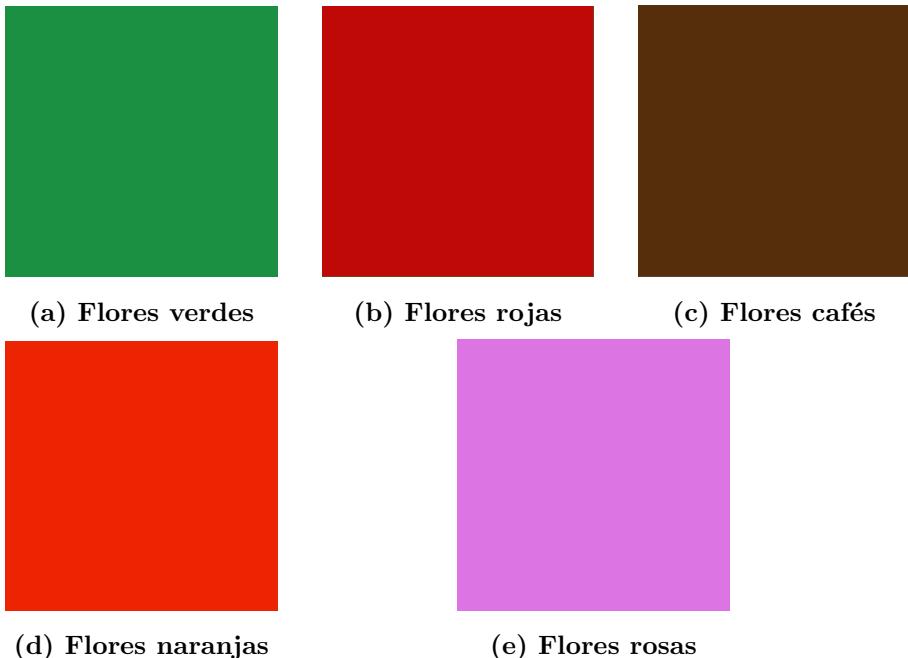


**Figura 5.11. Imagen objetivo: bandera de la República Mexicana**

En 5.12 se desglosan los respectivos colores dominantes generados al reproducir k-medias con  $k = 2$  sobre una selección manual de imágenes de flores realizada.

Habiendo seleccionado la imagen a computar y los mosaicos, se procede a generar el algoritmo de fotomosaico. Vale la pena sugerir empezar con dimensiones pequeñas,  $m$  y  $n$  con  $n > m$  ya que la bandera tiene una orientación horizontal. Así pues, sea  $m = 15$  y  $n = 20$ , y  $m = 20$  y  $n = 30$ , se generan los respectivos resultados de Figura 5.13.

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



**Figura 5.12.** Desglose de color promedio o predominante de 5 tipos de flores que, junto con las de tipo blanco, conforman los mosaicos a asignar de forma óptima

computar el caso de  $20 \times 30$  se observa que las flores naranjas están desplegadas en 3 ocasiones; esto se explica porque las fronteras entre dos objetos de distinto color tienen centroides que pueden estar cerca de un tercer color en el hiperplano RGB, tal es el caso de dos rectángulos ubicados entre el verde de la encina, el laurel y el nopal y el café del águila; así como un tercer rectángulo en el listón tricolor en la parte inferior del laurel. El rosa se aleja de las posibilidades de ser seleccionado por el modelo, 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



(a) Dimensiones  $15 \times 20$



(b) Dimensiones  $20 \times 30$

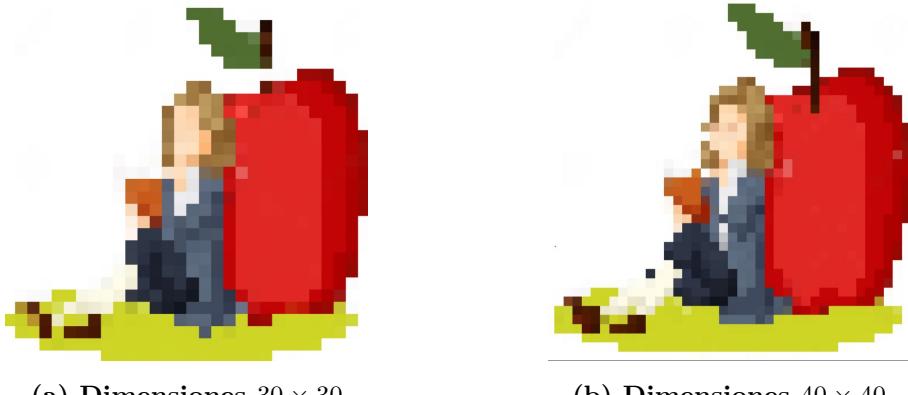
**Figura 5.13. Resultado de fotomosaicos a color: bandera de la república mexicana a base de flores**

versión a color. Las versiones degradadas de la imagen original (asociadas a la matriz  $MP$ ), en sus versiones de  $40 \times 40$  y  $30 \times 30$  se presentan en Figura 5.14, con dos despliegues que son fácilmente identificables.

El primer ejercicio se realiza con la instancia de  $30 \times 30$  empleando las mismas flores que en el ejercicio de la bandera nacional (ver Figura 5.13) y los resultados se muestran en Figura 5.15.

Con algo de imaginación se puede identificar que el desarrollo de esta instancia entrega un buen resultado. Ajusta el verde de la hoja de manzana con el mosaico 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 dos correspondientes ejecuciones manipulando los parámetros que definen al vector  $u$ , *i.e.*,  $vd$  y  $hd$ : la ejecución con disponibilidades de máxima holgura y la de variación forzada en el uso de las distintas fichas.

Se observa en los resultados en Figura 5.16 que ambos ejercicios, en los cuales se agregaron nuevos mosaicos, ocupan naturalmente una



**Figura 5.14. Resultado de arreglos objetivos, versión Newton**

mayor variedad de éstos que en las instancias anteriores de Newton. Resulta llamativo cómo en el primer ejercicio el algoritmo identificó el color del pétalo con las flores grises más que con las propias verdes claras con las que sí se asociaban en el ejercicio de Figura 5.15, con dimensiones  $30 \times 30$  y una variedad más limitada de mosaicos disponibles. Esto se debe a que el tipo de gris que representa al pixel “promedio” de la flor gris, se encuentra cercano a muchos colores, entre ellos este verde. El segundo ejercicio tuvo como resultado una seriación de colores variados plasmados en el fondo, originalmente blanco. La yuxtaposición de mosaicos bajo un patrón horizontal, particularmente de colores como rosa, azul y vino, que carecen de similitud razonable con el blanco, indica que su colocación responde a un objetivo de cumplir las restricciones del problema, asegurando una distribución equitativa de los veinte tipos de colores. Esto se alinea con la limitación de recursos definida por la variable  $u$ , que regula la asignación de mosaicos. Si bien el resultado es imperfecto, sí es visualmente atractivo e interesante desde el enfoque artístico y



**Figura 5.15. Imagen objetivo: Newton en dimensiones  $30 \times 30$**

aglorítmico. Además, analizando visualmente la imagen fuente y los resultados, sí es factible razonar que una imagen es una aproximación de la otra generada bajo un objetivo de asignación óptima. El resultado en general de las instancias de Newton a color puede considerarse exitoso, al haberse producido imágenes similares

### 5.2.3. Fotomosaicos con librería de R

En el Capítulo 3 se señala que existen librerías de código abierto, como *R*, que facilitan la generación de fotomosaicos, incluyendo la librería *RSimMosaic*. Ésta emplea la función *composeMosaicFromImageRandomOptim* para crear un mosaico a partir de tres insumos básicos: la imagen objetivo, una carpeta de



(a) Dimensiones  $40 \times 40$

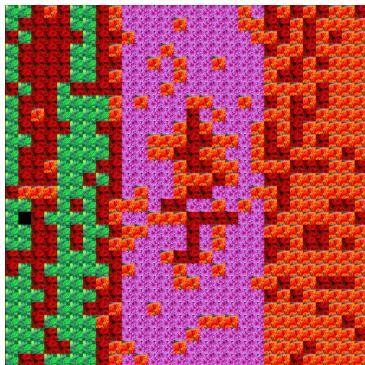


(b) Dimensiones  $40 \times 40$

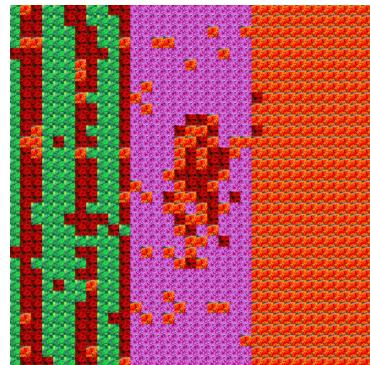
**Figura 5.16. Resultado de arreglos objetivos, versión Newton**

imágenes-mosaico y el nombre del archivo de salida en formato JPEG. A diferencia del algoritmo desarrollado en este trabajo, que optimiza la asignación de mosaicos mediante programación lineal, *RSimMosaic* prioriza la máxima utilización de los mosaicos disponibles, seleccionando aleatoriamente subrectángulos y asignándoles tiles basados en similitud de colores con los  $k = 20$  vecinos más cercanos (hiperparámetro ajustable pero por defecto asignado tal que  $neig = 20$ ). Esta heurística, que no garantiza un mínimo de distancia euclídea ni un resultado óptimo, explica la menor similitud observada en las instancias generadas, así como la mayor variedad de mosaicos en contraste con las soluciones controladas del enfoque propuesto en el presente proyecto.

Una de las ventajas de esto es que se puede desplegar una solución rápida, sin un costo computacional tan elevado. Una desventaja visualmente observable de la solución es que el resultado es menos bueno cualitativamente. De esto podría derivarse otro reto: el de identificar qué hiper-parámetros pueden ajustarse para hacer una



(a) Dimensiones  $30 \times 30$



(b) Dimensiones  $35 \times 35$

**Figura 5.17. Resultado de fotomosaicos a color: bandera de la república mexicana a base de flores**

mejor reproducción de la imagen de la bandera (número de vecinos, reemplazar mosaicos en el proceso de selección, etc). Vale la pena rescatar que este algoritmo procesa cada pixel de la imagen tal como se lo entrega el archivo JPEG, siendo esta una tarea compleja en imágenes de dimensión  $400 \times 400$  o mayor. De manera que tuvo que hacerse el ejercicio de reducción de dimensiones desarrollado anteriormente de agrupar un conjunto de pixeles, asignando manualmente dimensionalidades  $m \times n$  y asignarles su color promedio (ver sección 5.1 donde se define la construcción del arreglo *matsplitter()* y los cálculos derivados). Haciendo esto, el algoritmo logra computarse rápidamente, generando imágenes tales como se muestra en Figura 5.17.

Vale la pena notar que este algoritmo se fija como objetivo desplegar la mayor variedad posible de mosaicos 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,

implicando que 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, se procede a convertir la escala del conjunto  $\{0, \dots, 255\}$  a la escala  $\{0, \dots, \gamma\}$ , empleando la ecuación (4.3) desglosada en la sección 4.2 de la presente tesis. Así pues, obtenemos un vector de dimensión  $m \cdot n$ , llámese  $g$ . Para el primer ejercicio, asígnese  $\gamma = 7$ . Bajo esta  $\gamma$ , las escalas primera a treintaiseisava de luminosidad coinciden con la escala 7 en oscuridad (o siete “ciudades”). Las siguientes luminosidades (por orden) corresponden a seis ciudades; sucesivamente, hasta llegar a la escala 255, que corresponde a cero ciudades. Naturalmente, 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 consiste en ubicar las ciudades en el plano  $x - y$ , haciendo un loop a través de los  $m \cdot n$  rectángulos, representados como matrices.

1. La asignación del índice  $r \in \{1, \dots, m \cdot n\}$  a una coordenada  $(i, j)$  de una matriz  $m \times n$  se define mediante el mapeo  $\phi : r \rightarrow (i, j)$ , dado por:

$$j = ((r - 1) \bmod n) + 1,$$

$$i = \left\lfloor \frac{r - 1}{n} \right\rfloor + 1.$$

Este mapeo, recorre la matriz por filas, asociando cada  $r$  a una única coordenada  $(i, j)$ .

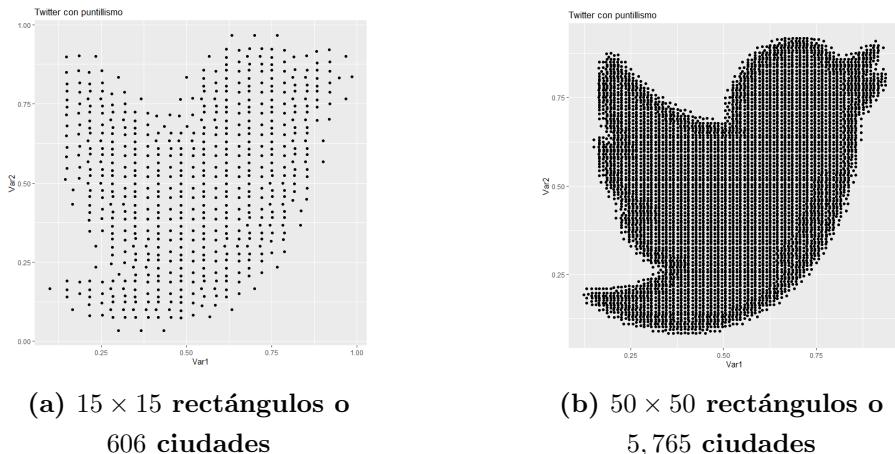
2. Una vez que tenemos la asignación  $\phi : r \mapsto (i, j)$ , el siguiente paso es plasmar los puntos que asemejarán a la imagen objetivo sobre el rectángulo  $[0, 1] \times [0, 1]$  en el plano  $x - y$ . Para ello, vale la pena considerar que el extremo superior izquierdo de la imagen corresponde 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  $\frac{1}{m}$ . En este esquema se puede demostrar que para todo par de coordenadas de matriz de tipo  $\{(i, j), (i + 1, j)\}$  o  $\{(i, j), (i, j + 1)\}$  existirá una relación de adyacencia entre sus correspondientes rectángulos. Además, las esquinas de la figura que envuelve a este conjunto de  $m \cdot n$  rectángulos serán las coordenadas  $\{(0, 1), (1, 1), (0, 0), (1, 0)\}$ . Con esto, garantizamos que los rectángulos mencionados conforman una partición del cuadrado  $[0, 1] \times [0, 1]$ .
  
3. El siguiente paso es usar el valor entero  $g_r = g[r]$  para colocar esa cantidad de ciudades en el rectángulo asociado a la coordenda  $(i(r), j(r))$ , tomando la especificación del número de ciudades conforme a la expresión (4.3). 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 bajo esta perspectiva, se busca 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 respectivamente y  $x_{l+1} - x_l = q \quad \forall l \in \{0, \dots, 9\}$  ( $q = \frac{1}{10 \cdot n}$ ); otro grid para las coordendas  $y$ :  $\{y_0, y_1, \dots, y_{10}\}$ , con  $y_{p+1} - y_p = r \quad \forall p \in \{0, \dots, 9\}$  ( $r = \frac{1}{10 \cdot m}$ ) con  $y_0$  y  $y_{10}$  extremos superior e inferior. Con esto, ya habremos construido el *grid* 2-dimensional  $G = \{(x_i, y_j) | (i, j) \in \{0, \dots, 10\}^2\}$  que consiste en 121 coordenadas. La propuesta principal es iterativa sobre el total de rectángulos. Para cada uno de éstos, sobre el conjunto de coordenadas generadas por el *grid*, aplicar un k-medias con  $g_r \in \{1, \dots, \gamma\}$  centroides - *clusters*. Por lo tanto en una “buena” asignación de ciudades en función de los criterios mencionados anteriormente, se pueden guardar las coordenadas de estos centroides e incorporarse los mismos a la selección de ciudades correspondiente al algoritmo k-medias aplicado sobre los rectángulos anteriores (si los hay).

Al iterar este proceso las  $m \cdot n$  veces, se obtienen las  $\sum_{k=1}^{m \cdot n} g_k = \tau$ <sup>5</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 (606 ciudades) y con  $50 \times 50$  (5,765 ciudades) se desglosan en

<sup>5</sup>esta es una expresión equivalente a la ecuación ec.(4.4), pero en la versión que mapea  $(i, j) \rightarrow r(i, j)$

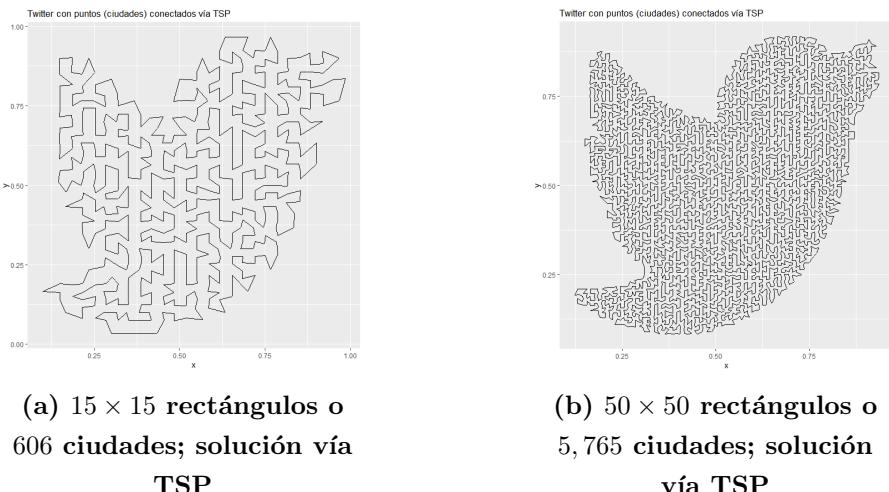
la imagen que se nombra “puntillismo”, la Figura 5.18, por su similitud con la técnica artística que despuntó de manera importante en Francia a inicios del siglo XX y consiste en generar una obra con puntos diminutos (el tradicional puntillismo se trabaja sobre lienzo y con óleos de colores).



**Figura 5.18. Gráfica de los puntos sobre el cuadrante  $x - y$**

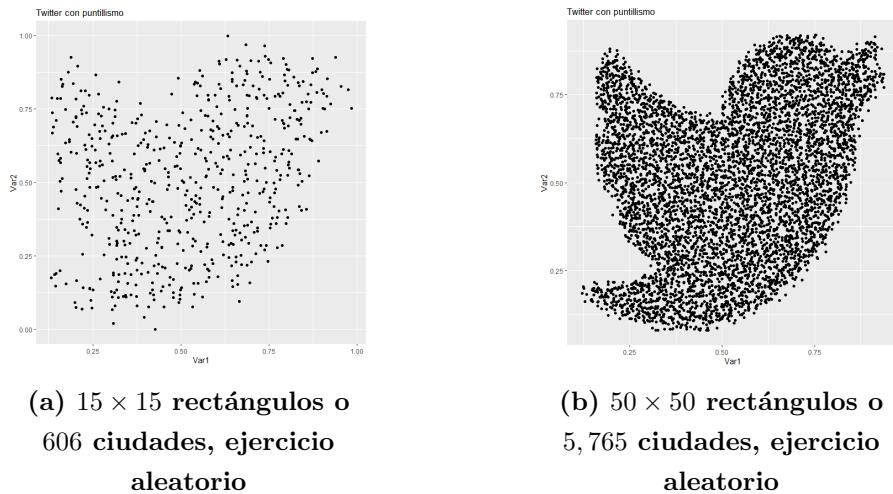
Existe un tipo de archivo cuya extensión es “.tsp”. Éste se basa en el conjunto de ciudades sobre el que se ha de resolver una instancia del problema lineal. Dicha solución puede procederse mediante distintos métodos heurísticos y es ejecutable por diversos *softwares*, tales como *R*, Python, Java y Concorde. Por motivos de practicidad, y dado que la construcción de los rectángulos y las ciudades se ha realizado en R hasta ahora, se utilizará 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 ciudad) 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, sobre coordenadas terrestres. La función encargada de resolver un TSP euclidiano (ETSP) es `solveTSP(.)`. Aunque permite especificar el método o algoritmo como entrada, el valor por defecto emplea el algoritmo de inserción arbitraria (que construye una solución inicial seleccionando nodos de manera secuencial y añadiéndolos al tour en la posición menos costosa) con refinamiento mediante el algoritmo *2-opt*. Como indica el término “heurística”, la inserción arbitraria y el *2-opt* no garantiza un óptimo global, pero sí una ejecución rápida y una solución localmente óptima. El tipo de datos del *output* es doble: uno es de la clase *tour* y otro es entero. La salida entera 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 fotomosaicos resultantes con dimensiones  $15 \times 15$  y  $50 \times 50$  se despliegan en Figura 5.19.

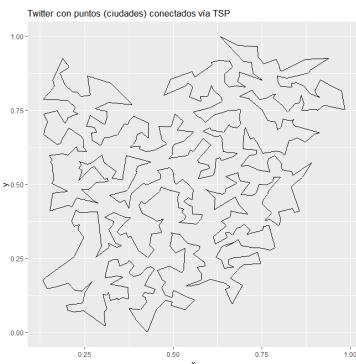


**Figura 5.19.** Aristas unen ciudades eficientemente, heurística *2-opt*, cuadrante  $x - y$ . Ciudades colocadas con k-medias

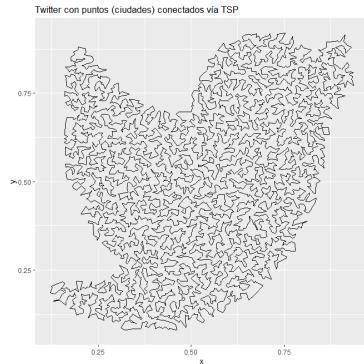
En el punto 3 de los pasos del instructivo en la subsección 5.3.1 se describe la lógica de la disposición de ciudades determinística y regular, garantizando una cobertura equitativa del espacio rectangular definido en el punto 2. Aquí hay una buena dispersión en el sentido de que cada región del rectángulo tiene una representación proporcional. Sin embargo, alternativamente se puede contemplar el enfoque de aleatorizar las coordenadas de puntos del rectángulo, respetando las escalas de color definidas por rectángulo (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 vez para la dimensión  $x$  y otra para la dimensión  $y$ . Los resultados aleatorios 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 indexados como Figura 5.20 y Figura 5.21.



**Figura 5.20. Gráfica de los puntos sobre el cuadrante  $x - y$ , ejercicio aleatorio**



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



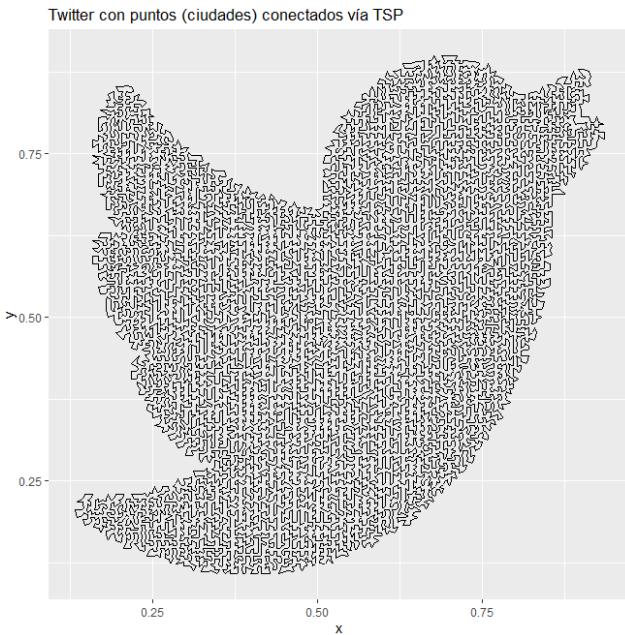
(b)  $50 \times 50$  rectángulos o  
5,765 ciudades, ejercicio  
aleatorio; solución vía TSP

**Figura 5.21.** Aristas unen ciudades eficientemente, heurística 2-opt, cuadrante  $x - y$ . Ciudades colocadas aleatoriamente.

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 dispersión regular regida por k-medias frente al de distribución aleatoria (contrastar subfiguras 5.19b y 5.21b), esto se debe a que la concentración de ciudades en la extensión de la silueta del ave es tan grande en ambos planteamientos  $50 \times 50$ , que las diferencias resultan menos perceptibles. La distinción entre el planteamiento regular y el aleatorio se detecta al comparar sus correspondientes instancias 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 de dimensión 50 sí se identifica de forma más evidente (contrastar imagen 5.19a contra 5.21a).

Finalmente, como se ve en Figura 5.22, el ejercicio se hace extensivo a la versión más grande que se logró correr del TSP con 85 rectángulos

y 15,311 ciudades y donde el logo de Twitter es distinguible con muy alta precisión.



**Figura 5.22.  $85 \times 85$  rectángulos o 15,311 ciudades; solución vía TSP**

### 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 *solve\_TSP(.)*) al registrar el tiempo que lleva resolver una determinada imagen conforme se agregan más dimensiones y, naturalmente, más ciudades.

El Cuadro 5.3 asocia la dimensión única ( $m = n$ ) con el número de

ciudades en la imagen de Twitter; asimismo describe la relación del número de 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. Si bien las restricciones de MTZ no se implementan en la heurística *2-opt*, se mencionan para dimensionar el tamaño del reto al que habría que enfrentarse en un escenario de búsqueda de soluciones exactas y no subóptimas.

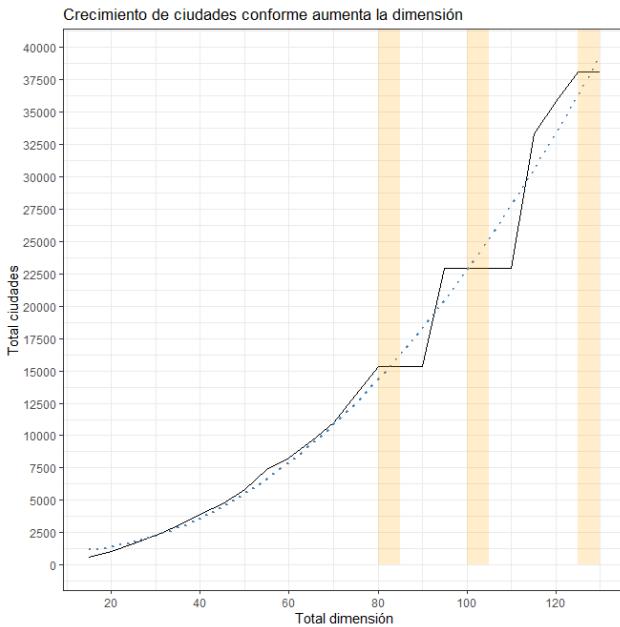
El Cuadro 5.3 asocia la dimensión única ( $m = n$ ) con el número de ciudades en la imagen de Twitter, ilustrando la relación entre este número y el tiempo de solución del TSP. Esta relación exhibe una complejidad superior a la lineal, reflejando el incremento creciente del esfuerzo computacional. La columna de restricciones teóricas, basada en el modelo de Miller-Tucker-Zemlin (MTZ) con  $n'^2$  restricciones, sirve como referencia para dimensionar el desafío de un enfoque exacto, a diferencia de la heurística empleada. Dicha heurística, que combina inserción arbitraria con refinamiento *2-opt*, no implementa las restricciones MTZ, sino que optimiza localmente el tour, logrando tiempos de ejecución rápidos. Esta comparación ilustra el contraste entre métodos heurísticos y enfoques de optimización lineal, como los utilizados en la construcción de fotomosaicos con *lpSolve* en la sección 5.2.1

Dimensión $m = n$	Total de ciudades $n'$	Tiempo de ejecución (s)	Variables	Restricciones teóricas (MTZ)
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	2,788,900	2,792,240
30	2,284	2.46	5,216,656	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	221.77	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
75	13,149	1,096.58	172,896,201	172,922,499
80	15,327	2,268.22	234,916,929	234,947,583
85	15,311	1,922.58	234,426,721	234,457,343

**Cuadro 5.3.** Tiempos de ejecución de la función `solve_TSP()` para instancias definidas por la dimensión, con restricciones teóricas basadas en el modelo MTZ como referencia.

Algo que resalta al observar el Cuadro 5.3 es que el número de ciudades ( $n'$ ) crece respecto a la dimensión ( $n$ ) a tasas crecientes en una curva “suave” hasta antes del incremento de dimensiones en el intervalo de 80 a 85; basta evaluar la gráfica de Figura 5.23 para concluir que en este intervalo se empiezan a manifestar quiebres. Éstos podrían deberse a que, en la medida de que el largo y ancho en rectángulos de la imagen se aproximan a los píxeles reales de la misma van a existir transiciones de dimensiones donde algunos píxeles reciben asignaciones de escalas de grises 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 variaciones de dimensión disminuye 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 rectángulos, en sección 5.1, la dimensión original se va recortando a lo largo y a lo ancho, de tal manera que ambas ( $\mu'$  y  $\nu'$  respectivamente) sean divisibles entre el *input* entregado. Esto da lugar a que las matrices objetivo difieran en dimensión. Ejemplo, 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 transicionó a  $400 \times 480$  y  $425 \times 510$  respectivamente. Por consiguiente los rectángulos tienen formas, oscuridades y estructuras distintas contribuyendo a conjuntos de ciudades con cardinalidades que no se estiman muy facilmente.

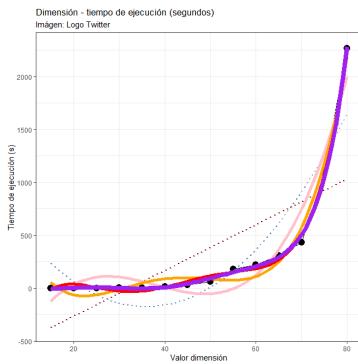
Similar a lo realizado en la subsección 5.2.1 de implementación para fotomosaicos, se reproducen las gráficas asociadas al Cuadro 5.3 anterior en Figura 5.24. Se añaden los correspondientes ajustes polinomiales. A diferencia del caso de los fotomosaicos donde bastaban los ajustes de orden 3 (cúbicos), en este ejercicio fue hasta llegar a los ajustes de orden



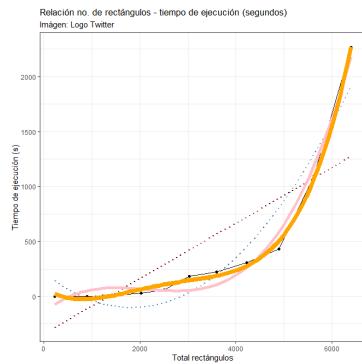
**Figura 5.23. Ciudades generadas vs dimensión (con  $m = n$ ).**  
**Rangos donde se reduce la cantidad de ciudades se resalta en amarillo**

6 y 4 para dimensión ( $n$ ) y rectángulos ( $m \cdot n = n^2$ ) 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. Se puede decir que las ciudades crecen polinomialmente en función de la dimensión de rectángulos, tal como se observa en la gráfica 5.23. Interesa por lo tanto conocer el



(a) Tiempo vs dimensión

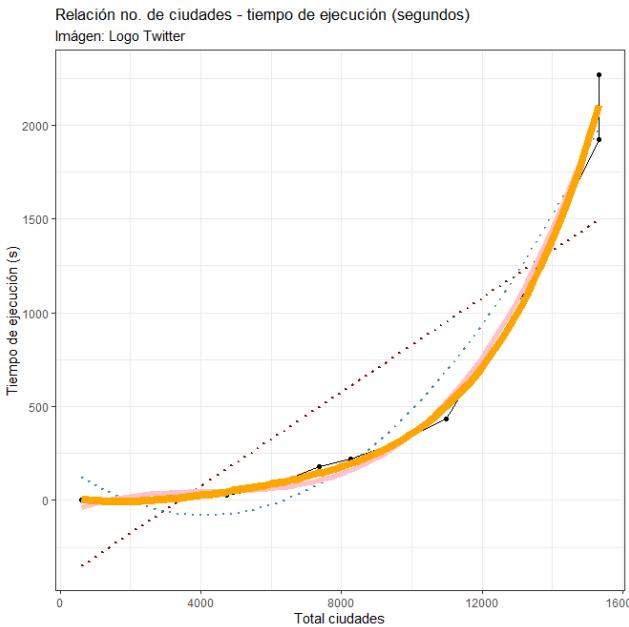


(b) Tiempo vs rectángulos

**Figura 5.24. Graficaciones tiempos de ejecución**

comportamiento del tiempo de ejecución conforme crecen las ciudades. La gráfica 5.25 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. En síntesis, en notaciones *Big O*, hablamos de un algoritmo de orden  $O(n^3)$  para fotomosaicos frente a uno  $O(n^p)$  con  $p \geq 4$  para el TSP; esto aún cuando el primero opera un problema de optimización exacta, mientras que el segundo opera heurísticas, inserción arbitraria reforzada con *2-opt* (¡una de las más sencillas heurísticas de mejora entre las documentadas en este escrito!). Esta confronta basta para dimensionar la diferencia en complejidades de un enfoque TSP, mucho más grande versus su contraparte, que son los fotomosaicos.

La cuestión que vale la pena abordar es relativa a si el algoritmo programado es capaz de reproducir imágenes de un grado más elevado de detallado visual, o se limita a ser aplicable únicamente a imágenes sin relieve, ni dimensión, ni variedad de colores. Para este efecto se buscará reproducir una foto personal del autor de este escrito en su escritorio



**Figura 5.25. Relación Tiempo tsp vs ciudades**

de casa, la foto tendrá por nombre “*home office*”; se aprecia en Figura 5.26.

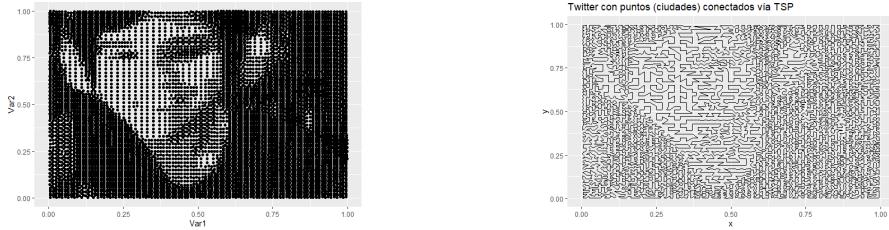
Lo que sigue es reproducir la imagen. Para esto, se hizo uso de un truco: atenuar los colores más claros. Es decir, a partir de un parámetro determinado de escala de luminosidad en adelante se aclarará la escala al máximo valor posible, a saber, 1. Para esta imagen el parámetro elegido es 0.70. Esto permitió que la imagen resultara más distinguible, sobre todo en lo que concierne a las facciones faciales: cara, ojos, boca y al artículo electrónico sobre las orejas del personaje (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



**Figura 5.26. Selfie en pandemia**

imágenes indexados como Figura 5.27 y Figura 5.28, respectivamente.

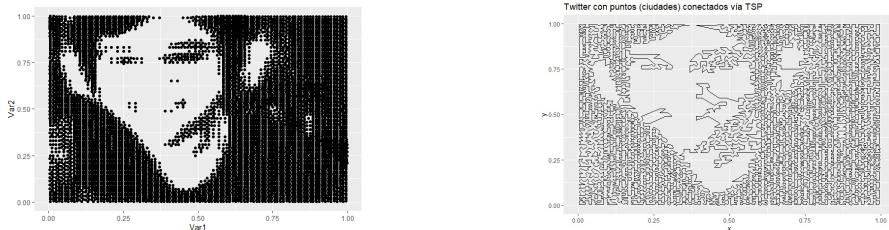
Finalmente se trabajó la instancia de Newton que se mencionó en el ejercicio de los fotomosaicos descrito en la sección 5.2.1. Se reproduce procesando TSP en dimensiones de rectángulos  $50 \times 50$ . El resultado genera una imagen (Figura 5.29) que se distingue con claridad como una manzana tanto por la forma de la cáscara como por la hoja, acompañada de una silueta. Dicho contorno complementario a la fruta (el hombre, Newton) se identifica según el observador, cuando el escritor preguntó, las respuestas fueron variadas. Algunos empezaron por identificar primero 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 completa, ya concibiendo el libro que sostiene. Esta equiparación mejoró tras sugerirse al espectador entrecerrar los ojos.



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

(b) TSP con puntos (ciudades) conectados vía TSP  
Twitter con puntos (ciudades) conectados vía TSP  
ciudades

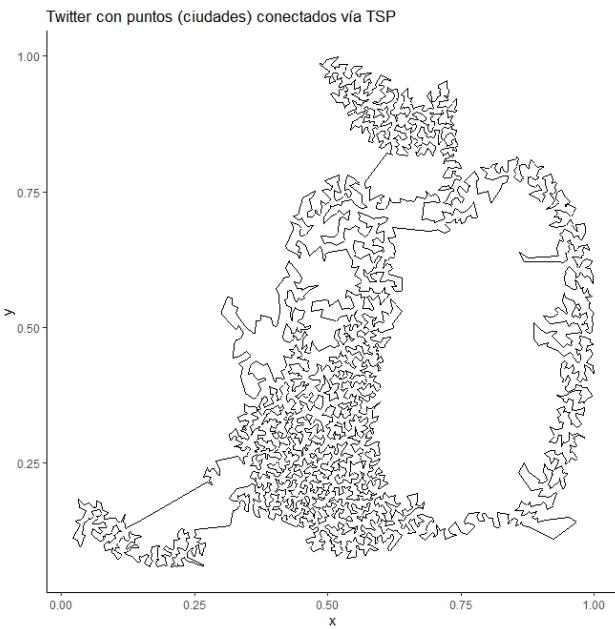
**Figura 5.27. Puntillismo y TSP para autorretrato (sin agudizar brillos)**



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

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

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



**Figura 5.29. Ejercicio de Newton con el problema del agente viajero, versión  $50 \times 50$**

# Capítulo 6

## Conclusiones

En el presente trabajo se presentó una aplicación de la optimización en la generación de fotomosaicos. Primero, se revisaron modelos de programación lineal para figuras en escalas de grises, posteriormente con color y por último se describió cómo generar una plantilla para hacer un fotomosaico al resolver un problema del agente viajero que nos dé un tour hamiltoneano que se asemeje a la figura subyacente. Es interesante resaltar que este conjunto de aplicaciones de la optimización ocurre en el ámbito del diseño o arte más que en el de ingeniería, logística, gestión empresarial y operaciones industriales, que es donde tradicionalmente se ha centrado esta disciplina.

Analizamos la manipulación de las escalas de grises, y del modelo aditivo de color de tipo RGB para poder darle un valor a los mosaicos que se van a usar y a los rectángulos a los cuales van a suplir; asimismo, se aplicó una variante de este modelo de escalas de grises por rectángulo para definir la dimensionalidad del conjunto de ciudades a visitar para el agente viajero. Se determinó que la interpretación en arreglos de pixeles de las imágenes dan pie a plantear una variedad de

problemas de programación lineal, cuyos resultados se pueden evaluar visualmente. Esto permite plantear nuevos fotomosaicos con base en la creatividad del proponente (un ejemplo de estas variantes poco profundizado en este texto y propuesto por Bosch (2004) es el planteamiento con fichas de dominó, el cual incorpora un componente de orientación y deriva en buenos resultados visuales, como fue mostrado en la sección 3.2). Tanto los fotomosaicos como el TSP para replicar la imagen objetivo son formas creativas de interpretar el arreglo numérico de pixeles partiendo de las dimensiones de renglones y columnas que se entreguen exógenamente; asimismo, la segmentación en k-medias de una imagen resultó ser una buena aplicación del modelo RGB para sintetizar un color principal, o promedio. En particular la aplicación de fotomosaicos es interesante en el rubro de la mercadotecnia para demostrar cómo se altera la percepción del observador en función de la lejanía con que se contempla una imagen (a mayor lejanía, el observador logra dilucidar mejor la foto a replicar); el nexo fundamental radica en el concepto de percepción, ya que la mercadotecnia se centra en la manipulación de la percepción. De manera complementaria, la concepción y entendimiento de los pixeles permite procesar diversos algoritmos con aplicaciones fundamentales en la era tecnológica actual, tales como la identificación de formas / siluetas de objetos, detección de colores, análisis de archivos de video, identificación de patrones de movimiento, etc.

Para dar un marco teórico de los dos tipos de problemas desarrollados en este trabajo fue necesario hacer una revisión general de la programación lineal, de la complejidad que implica pasar de un problema en un espacio continuo de soluciones a uno entero ocupando el algoritmo de ramificación y acotamiento (uno de sus principales métodos); finalmente, se identificó una aplicación de un problema

lineal entero donde, aún con la ramificación y acotamiento resulta infactible resolver instancias en escalas muy grandes de variables, tal es el caso del problema TSP que requiere de heurísticas para su solución: algunas heurísticas conocidas de construcción y mejora de tours, que fueron mencionados y discutidos en la bibliografía de Nilsson (2003).

Los programas de R para particionar las imágenes objetivo que se codificaron para este proyecto, *matsplitter()*, resultaron útiles y robustos a las dos clases de problemas abordados: fotomosaicos y TSP. Fungieron como punto de partida para obtener la escala de gris promedio ayudando a hacer más eficiente y modular el código ya que ambos los planteamientos compartían esta lógica. Adicional, una versión más compleja de *matsplitter()* ejecuta la misma tarea sobre el espacio de la triada RGB.

Las instancias trabajadas de los fotomosaicos en *grayscale* 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 escalas de grises usando 20 mosaicos (en el rango de 0 a 1 con *grid* de 0.05); 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 el conjunto de imágenes objetivo que se establecieron, la bandera de México y la representación de Newton y su manzana, ya que se eligieron mosaicos manualmente, pensando pragmáticamente en cumplir un objetivo, conciliando visualmente colores de flores versus colores en los segmentos de la imagen. En el espacio RGB 3-dimensional, significa un reto computacional relevante el crear un *grid* con respectivas imágenes asociadas que satisfaga la variedad exhaustiva que se construía con simplicidad en el ejercicio con escalas

de grises. Esto, ya que el salto de una a tres dimensiones es cúbico, haciendo muy difícil pasar de hacer  $10$  a  $10^3 = 1,000$  mosaicos, o de  $20$  a  $20^3 = 8,000$ . En contraste con la librería de *R* especializada en fotomosaicos (*RSimMosaic* y su función, *composeMosaicFromImageRandomOptim*), los resultados del problema de fotomosaicos desarrollado aquí fueron visiblemente más estéticos, ya que la librería no trabaja un problema de minimización de distancias sobre cada rectángulo, sino de asignación de mosaicos mediante heurísticas de rápida —no necesariamente eficiente— solución. De manera que, en el planteamiento aquí desarrollado, la similaridad, establecida por un buen criterio matemático, se ve bien reflejada. Frente a esta observación, podemos establecer un posible alcance futuro para este proyecto: optimizar y documentar el código para poder construir la correspondiente librería para uso común en *R*, haciendo accesible este algoritmo para todo el público.

Para el enfoque del TSP se hizo una introducción teórica, pasando por restricciones factibles que mejoran el cómputo en términos de complejidad, así como algoritmos de construcción y mejora de tours, donde existen heurísticas de diseños muy variados y con distintos grados de complejidad y cercanía con el óptimo, explorando la cota de *Held-Karp* que aproxima los niveles óptimos. En lo referente a estas heurísticas del TSP, exhaustivamente estudiadas en el campo de la investigación de operaciones, la reflexión final conduce a balancear el *trade-off* entre economizar tiempo de cómputo por un *software* (hacer que éste sea factible) y obtener una buena cercanía de la solución al óptimo global.

Para definir las posiciones de las ciudades se inspeccionaron dos principales propuestas de disposición en el cuadrado  $[0, 1] \times [0, 1]$ : sobre cada rectángulo, una disposición regular basada en k-medias y

otra basada en dispersión aleatoria. Estéticamente, las evidencias se inclinaron por el primer enfoque, pues los ejercicios con disposición regular fueron más claros y asociables a la imagen subyacente.

Sobre el cómputo del TSP, la metodología de construcción y mejora de tours elegida para ejecutar las instancias definidas en este texto fue la heurística de inserción arbitraria, aunada a la heurística *2-opt*. Respecto a los tiempos de ejecución con una computadora Intel Core i7 con 8 GB de memoria RAM instalada, estos fueron relativamente razonables para las proporciones del problema, llegando al nivel de 2,000 segundos (media hora aproximadamente) para el procesamiento de 15,000 ciudades, como se mostró para el logo de Twitter con las dimensiones de  $85 \times 85$  rectángulos. Los despliegues con TSP tuvieron buenas aproximaciones a las imágenes objetivo, incluso variando sus grados de elaboración: pasando por la imagen del logo de Twitter, el Newton y el autorretrato. La complejidad de este modelo, no obstante, se demostró en las gráficas correspondientes (Figura 5.23), con un tiempo de procesamiento que es notablemente más sensible ante cambios en dimensionalidad de rectángulos elegidos que lo que resultó ser en la solución de fotomosaicos (ambos enfoques mostraron contrastes de complejidades con cuando menos un grado polinomial de diferencia). Igualmente, si observamos la respectiva tabla de complejidad correspondiente al ejercicio de Twitter para ambos planteamientos (ver Cuadro 5.1 y Cuadro 5.3), la máxima dimensionalidad en común de rectángulos procesados fue de  $60 \times 60$ , la cual con el TSP implicó una resolución del problema de 221 segundos versus los 64 para fotomosaicos, esta referencia en tiempos puede dar noción de la significativa diferencia entre la complejidad computacional de uno y otro problema. Esto tiene que ver, primero con la restricción de integralidad que sí se tiene en TSP y en los

fotomosaicos no (ya que, como se menciona, la unimodularidad total de este problema permite que no sea necesario agregar integralidad sobre las variables de decisión para que ésta se cumpla en el óptimo global); y segundo, con la creciente complejidad inherente al problema de TSP comentada en este escrito, cuestión que lo caracteriza como un problema NP-Duro. Esta complejidad es principalmente atribuible a las restricciones de no-subtours. Aún con heurísticas o relajación de restricciones para resolverse de manera computacionalmente viable, no se logró ejecutar el ejercicio de manera más económica a lo expuesto en las tablas de complejidad.

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, esta ciencia computacional también puede aplicarse en la producción de arreglos visualmente estéticos. Como indica Robert Silvers (1997) 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, E. (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] Bosch, R. (2003). Continuous line drawings via the traveling salesman problem. Ohio: Oberlin College.
- [6] Bosch, R. (2006). Opt Art. Ohio: Oberlin College.
- [7] Castañeda, W. (2005). Color. Caldas: Universidad de Caldas.
- [8] Aqil Burney, SM. (2014). K-Means Cluster Analysis for Image Segmentation. Sind: University of Karachi.
- [9] Goic, M. (2009). Análisis Post Optimal y Algoritmo de Ramificación y Acotamiento. Santiago: Universidad de Chile.

- [10] Nilsson, C. (2003). Heuristics for the Traveling Salesman Problem. Suecia: Linkoping University.
- [11] Chong, B. (2021). K-means clustering algorithm: a brief review. Shanxi: Academic Journal of Computing & Information Science, Vol. 4.
- [12] González-Santander, G. (2020). Tres métodos diferentes para resolver el problema del viajante. Baobab Soluciones. [https://baobabsoluciones.es/blog/2020/10/01/problema-del-viajante/#:~:text=El%20problema%20del%20viajante%20\(por,Su%20origen%20no%20est%C3%A1%20claro.](https://baobabsoluciones.es/blog/2020/10/01/problema-del-viajante/#:~:text=El%20problema%20del%20viajante%20(por,Su%20origen%20no%20est%C3%A1%20claro.)
- [13] Fernandez, V; Zelaia, A. (2011). Investigación operativa. Programación lineal. Vizcaya: Universidad del País Vasco.
- [14] Fraga, A. (2024). Uncalibrated: Drawing, Photography and the Raw Images of Astronomy. Yorkshire del Oeste: The University of Leeds.
- [15] Desrochers, M; Laporte, G. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. Quebec: Operations Research Letters, Volumen 10.
- [16] Hillier, F. S., & Lieberman, G. J. (2015). Introduction to Operations Research (10ma ed.). McGraw-Hill.
- [17] blueringmedia. (n.d.). Sir Isaac Newton with big apple illustration [Ilustración]. Depositphotos. <https://depositphotos.com/mx/vector/sir-isaac-newton-big-apple-illustration-549925186.html>

## Apéndice A

# Apéndice

### A.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  $n$  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  $\mu \times \nu$ .

$$\begin{aligned} DIM = & \{(\mu', \nu') = (\max(a), \max(b)) | 0 = a \text{mod}(m), \\ & 0 = b \text{mod}(n), a \in [0, \mu] \cap \mathbb{N}, b \in [0, \nu] \cap \mathbb{N}\} \end{aligned} \quad (\text{A.1})$$

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).

## A.2. Reduciendo dimensionalidad del vector de costos

La forma genérica que ayuda a convertir el arreglo tridimensional de costos en los fotomosaicos en uno unidimensional se muestra a continuación. El índice  $t$  se asocia a las combinaciones  $(i, j, f)$  mediante la función  $\eta$ , donde  $t = \eta(i, j, f)$  genera un índice único basado en el patrón descrito en la tabla. Además, se establece que el costo original  $c_{fij}$  en el arreglo tridimensional equivale al costo  $c'_t$  en la representación unidimensional, es decir,  $c_{fij} = c'_t$ . En el entorno de  $R$ , el arreglo inicial se define como  $A$ , donde  $A[i, j, f] = c_{fij}$  representa los costos asociados a cada combinación.

**Cuadro A.1. Tabla de costos para combinaciones de  $f, i, j$**

$t$	$f$	$i$	$j$	costo ( $c$ )
1		1	1	$c[1,1,1]$
2		1	1	$c[1,1,2]$
$\vdots$		$\vdots$	$\vdots$	$\vdots$
$n$		1	1	$c[1,1,n]$
$n + 1$		1	2	$c[1,2,1]$
$n + 2$		1	2	$c[1,2,2]$
$\vdots$		$\vdots$	$\vdots$	$\vdots$
$2n$		1	2	$c[1,2,n]$

Continúa en la siguiente página

<i>t</i>	<i>f</i>	<i>i</i>	<i>j</i>	costo ( <i>c</i> )
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(m - 1) \cdot n + 1$	1	$m$	1	$c[1,m,1]$
$(m - 1) \cdot n + 2$	1	$m$	2	$c[1,m,2]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m \cdot n$	1	$m$	$n$	$c[1,m,n]$
$m \cdot n + 1$	2	1	1	$c[2,1,1]$
$m \cdot n + 2$	2	1	2	$c[2,1,2]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m \cdot n + n$	2	1	$n$	$c[2,1,n]$
$m \cdot n + n + 1$	2	2	1	$c[2,2,1]$
$m \cdot n + n + 2$	2	2	2	$c[2,2,2]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m \cdot n + 2n$	2	2	$n$	$c[2,2,n]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m \cdot n + (m - 1)n + 1$	2	$m$	1	$c[2,m,1]$
$m \cdot n + (m - 1) \cdot n + 2$	2	$m$	2	$c[2,m,2]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2 \cdot m \cdot n$	2	$m$	$n$	$c[2,m,n]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(cf - 1) \cdot m \cdot n + 1$	$cf$	1	1	$c[cf,1,1]$
$(cf - 1) \cdot m \cdot n + 2$	$cf$	1	2	$c[cf,1,2]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(cf - 1) \cdot m \cdot n + n$	$cf$	1	$n$	$c[cf,1,n]$
$(cf - 1) \cdot m \cdot n + n + 1$	$cf$	2	1	$c[cf,2,1]$
$(cf - 1) \cdot m \cdot n + n + 2$	$cf$	2	2	$c[cf,2,2]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(cf - 1) \cdot m \cdot n + 2 \cdot n$	$cf$	2	$n$	$c[cf,2,n]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(cf - 1) \cdot m \cdot n + (m - 1) \cdot n + 1$	$cf$	$m$	1	$c[cf,m,1]$
$(cf - 1) \cdot m \cdot n + (m - 1) \cdot n + 2$	$cf$	$m$	2	$c[cf,m,2]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m \cdot n \cdot cf$	$cf$	$m$	$n$	$c[cf,m,n]$

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 definida por  $t = (f - 1) \cdot (m \cdot n) + m \cdot (i - 1) + j$  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  $RM \in \mathbb{R}^{re \times m \cdot n \cdot cf}$  con  $re = cf + m \cdot n$ , los primeros  $cf$  renglones representando a las restricciones de disponibilidad de los mosaicos que hay, y las restricciones restantes representando el correcto llenado de cada celda genérica  $(i, j)$ . La matriz consta únicamente de valores 0 y 1. Los 1's se asignan fácilmente conociendo, con la ayuda de la asignación  $t(.)$  qué posiciones corresponden a cada  $f$  para el primer conjunto de restricciones; similarmente, para el segundo conjunto de restricciones basta reconocer qué posiciones le corresponden a cada celda  $(i, j)$ .
4.  $rh_f = u_f \quad \forall f \in \{1, \dots, cf\}$   

$$rh_t = 1 \quad \forall f \in \{cf + 1, \dots, cf + m \cdot n\}$$
5.  $sign_f = "\leq" \quad \forall f \in \{1, \dots, cf\}$   

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

*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.