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

Julio César Espinosa

Verano 2021

1. Introducción

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

2. TSP: Agente viajero

La teoría de grafos estudia las propiedades y relaciones de un grafo que es un par (V, E) con V un conjunto de vértices y E conjunto de aristas que los une. Siendo la arista modelada, ya sea como un par dirigido, donde el orden de los vértices importa (digrafo), o bien, no dirigido. Existen problemas de optimización planteados sobre esquemas de grafos que pueden ser resueltos por la vía de la programación lineal, siendo el problema del agente viajero (*Traveling Salesman Problem*, TSP) uno de los más interesantes y populares. Este problema se esboza partiendo de un individuo (agente) que desea visitar una cantidad n_* de ciudades partiendo de una ciudad origen y finalizando el recorrido en dicha ciudad con el objetivo de minimizar la distancia total recorrida. Existe un único conjunto de parámetros necesario para definir el problema: la distancia que hay entre cada par de ciudades existente; para ello defínase $c_{i,j}$ como el valor cuantitativo de dicha la distancia que hay entre las ciudades (o vértices) (i,j) para cada par $(i,j) \in \{1,...,n_*\}^n_*$. Explicado este problema, podemos desarrollar una intuitiva formulación matemática mediante la programación lineal entera siguiente.

$$\min_{x_{i,j}} \sum_{i=1}^{n_*} \sum_{j=1}^{n_*} c_{i,j} x_{i,j} \tag{1a}$$

$$\text{s.a.} \sum_{i=1}^{n_*} x_{i,j} = 1 \qquad \forall j \in \{1,...,n_*\} \tag{1b}$$

$$\sum_{j=1}^{n_*} x_{i,j} = 1 \qquad \forall i \in \{1,...,n_*\} \tag{1c}$$

$$\sum_{i \in S} \sum_{j \in S} x_{i,j} \leq |S| - 1 \qquad \forall S \subset \{1,...,n_*\} \tag{1d}$$

Donde S es el conjunto potencia generado con $\{1,...,n_*\}$. El problema lineal esbozado arriba se conforma por componentes. Para entenderlo, defínase antes el conjunto de variables de decisión como $\{x_{i,j}|(i,j) \in \{1,...,n_*\}^2\}$ con dos asignaciones de valor posibles: 0 y 1. 1 representa que el agente realizó un recorrido directo partiendo de la ciudad i hacia la ciudad j con una distancia correspondiente de $c_{i,j}$; 0, por el contrario, representa el caso opuesto. En términos técnicos, el agente viajero ocupó el dígrafo dirigido (i,j).

Así, la primera expresión ec. (1a) es la función objetivo: el recorrido total para arribar a las n_* ciudades. (1b) exige que de cada ciudad se parta hacia una única ciudad; (1c), análogo a (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 (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 $\{1,...,n_*\}$ generen cada un recorrido o tour cerrado. Supongamos que se generó una solución optimal con las restricciones (1b) y (1c) cumplidas pero que constara de tres subtours: T_1, T_2, T_3 con $T_1 \cup T_2 \cup T_3 = \{1,...,n_*\}$. Poniéndonos imaginativos, esto querría decir que el agente terminó de recorrer T_1, al terminar 'se teletransportó' a T_2 sin recorrer distancia alguna, finalmente siguió haciendo lo propio de T_2 a T_3. Claramente esto no tiene sentido en el plano real; además, se viola el supuesto de que el agente parte de un punto y termina en el mismo. La restricción (4) elimina la posibilidad de subtours. Sin embargo, al tratarse de una restricción que explora a cada uno de los 2^{n_*} posibles subconjuntos generables con $\{1,...,n_*\}$, es una restricción con una elevada complejidad computacional asociada que debe simplificarse con adecuadas restricciones alternativas (véanse restricciones de Miller, Tucker &

Zemlin, 1960). Si a eso le sumamos que el programa lineal tiene como conjunto factible los $n_*!$ posibles ordenamientos de $\{1, ..., n_*\}$, podemos entender que se diga que los programas que lo resuelven trabajan bajo los llamados 'algoritmos de optimización combinatoria'.

2.1. Arte con el agente viajero

En la introducción se hace referencia a dos maneras de replicar imágenes con optimización lineal que se desarrollarán en el presente estudio. La primer versión es vía el TSP: parte del supuesto de que se cuenta con una imagen en blanco y negro con una gama de *greyscales* (o escala de grises). Asociamos una mayor *greyscale* a una mayor luminosidad o blancura. Así pues, procedemos de la siguiente manera

- Particionamos la imagen en grupos de pixeles. Cada grupo constará de rectángulos con k pixeles a lo largo por h pixeles a lo ancho. Derivamos de la imagen m submatrices-renglón de grupos o rectángulos y n submatrices-columna de éstos, de tal manera que la matriz grande cuenta con dimensiones $\mu' \times \nu'$ con $\mu' = k\ m$ y $\nu' = h\ n$; asímismo, obtenemos una escala de grises para cada pixel en el rango discreto de 0 a 255.

- Obtenemos la *greyscale* promedio de cada rectángulo y fijamos un parámetro que ponderará el nivel de detalle que deseamos para la imagen a reproducir: $\gamma \in [4, 9] \cap \mathbb{N}$. De tal manera, para el rectángulo $(i, j) \in \{1, ..., m\} \times \{1, ..., n\}$ la *greyscale* del pixel promedio será de ψ_{ij}, con valores entre 0 y 255. De tal forma constrúyase

$$g_{ij} = \lfloor \gamma - \gamma\psi_{ij}/255 \rfloor \in [0, \gamma] \tag{2}$$

  Esta será una escala de oscuridad del rectángulo.

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

$$\tau = \sum_{i=1}^{m} \sum_{j=1}^{n} g_{ij} \tag{3}$$

  Computamos las distancias (o 'costos') entre las τ distintas 'coordenadas' en una matriz $C = [c]_{ij}$ de $\tau \times \tau$; resolvemos el TSP sobre estas distancias. Siendo que ahora el planteamiento se esboza sobre τ y relacionándolo con la expresión para el agente viajero explicada arriba, podemos entender que ahora $n_* = \tau$.

0. En adelante se emplearan indistintamente '*greyscales*' o 'escala de grises'

3. Foto mosaicos

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

$$\min_{x_{fij}} \sum_{f \in F} \sum_{i=1}^{n} \sum_{j=1}^{n} (b_f - \beta_{ij})^2 x_{fij} \tag{4a}$$

$$\text{s.a.} \sum_{i=1}^{n} \sum_{j=1}^{n} x_{fij} \leq u_f \qquad \forall f \in F \tag{4b}$$

$$\sum_{f \in F} x_{fij} = 1 \qquad \forall (i,j) \in \{1, ..., m\} \times \{1, ..., n\} \tag{4c}$$

La expresión ec. (4a) minimiza la suma de diferencias cuadráticas entre las luminosidades del área subyacente sobre la cual se coloca la ficha f en la posición (i,j) (Possani). La función de costos $c_{i,j}$ del problema viajero ahora tiene un análogo tridimensional: $c_{fij} = (b_f - \beta_{ij})^2$. (4b) garantiza que se no se excedan las dotaciones disponibles de cada figura. La restricción (4c) se asegura que haya exactamente una foto en la posición (i,j). Cabe destacar que, si bien este problema es de programción lineal entera (los valores factibles son solo 0 y 1), el mismo se puede plantear como un problema con variables de decisión continuas, dando lugar a un resultado optimal entero y factible. Esto es una noticia buena en la medida de que añadir la propiedad de 'entero' implica en la solución restricciones asociadas con una complejidad computacional y de planteamiento no siempre tan triviales. En otras palabras, la restricción de enterez dificulta el cálculo de soluciones.

4. Implementación

4.1. Partición de la matriz en submatrices

Una imagen en escala de grises se entiende como un despliegue visual en el que cada unidad básica mostrada en la pantalla —mejor conocida como pixel— se ve representada por una cifra que cuantifica la cantidad de luz (o información de intensidad lumínica) con que cuenta. Representando la mínima unidad al negro y, la máxima, al blanco. La escala tiende a volverse discreta, ya que contar con un continuo estándar de 0 a 1 (o 0 % a 100 %) representaría un reto computacional y de almacenaje considerable. Algunas imágenes en escala de grises tempranas cuantificaban hasta 16 valores, lo cual implicaba un almacenaje de 4 bits por pixel; en la medida de que la rama de la fotografía digital evolucionó, aumentó hasta 256 intensidades, implicando un peso de 8 bits por pixel. Si bien actualmente hay adicionales variaciones con mayor o menor número de valores (también discretos), los cuales se eligen en función de la aplicación de la imagen, un estándar recurrentemente usado es el de 256 escalas. Por eso, y en aras de seguir convenciones, se usará esta escala discreta, o bien, la escala continua de 0 a 1. Para pasar de la escala $[0, 1]$ a la escala $\{0, ..., 255\}$ hacemos $e_{255} = round(255e_1)$; para pasar en sentido inverso se usará $e_1 = \frac{1}{255}e_{255}$.

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

Para solventar este problema se usará una versión auxiliar de la matriz original, rotándola 90º

0. mostrar número de sección

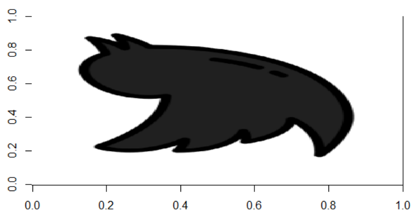

Figura 1: "Despliegue image(.), matriz original"

en sentido de las manecillas del reloj. El resultado se despliega a continuación.

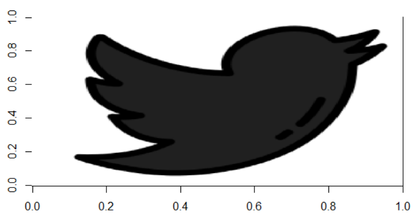

Figura 2: "Despliegue image(.), matriz rotada 90 a la derecha"

El siguiente paso consiste en usar las dimensiones m y n para delimitar el tamaño de la matriz tal que se permita generar una partición de la misma en submatrices que sea factible ante los *inputs* enviados a la función de partición. 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, se escoge una nueva dimensión $\mu' \times \nu'$ que corresponda al máximo valor factible que no exceda $\mu \times \nu$.

$$DIM = \{(\mu', \nu') | 0 = \mu' mod(m), 0 = \nu' mod(n), \mu' \in [0, \mu] \cap \mathbb{N}, \nu' \in [0, \nu] \cap \mathbb{N}\} \tag{5}$$

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 la diferencia de, a lo mucho, uno). El siguiente paso consiste en convertir la matriz de la escala $[0, 1]$ a la escala $\{0, ..., 255\}$. Luego, se procede a fraccionar la matriz en un conjunto de m n submatrices mediante la función $matsplitter(.)$ de R, la cual recibe como entradas la matriz a fraccionar, la cantidad de renglones por submatriz, $k = \mu'/m$ y la cantidad de columnas, $h = \nu'/n$. $matsplitter(.)$ entregará una lista de $m \times n$ elementos. Los elementos 1 a n de la lista consistirán en las primeras n submatrices superiores de la matriz, los elementos $n+1$ a n consistirán en las segundas n submatrices superiores. Así sucesivamente, hasta llegar a las n submatrices inferiores, ennumeradas $n\ m-n+1$ a $m\ n$. En el supuesto de que cada submatriz representara un único valor numérico (o arreglo 1×1) dentro de una matriz $m \times n$; entendemos que este tipo de ordenamiento es de clase *rowwise* (o por renglón), ya que se va llenando renglón por renglón, empezando por los renglones de arriba y continuando hacia abajo. Cada renglón se llena de izquierda a derecha. El otro tipo común de ordenamiento es *columnwise*, o por columna. Conociendo esto, podemos calcular la luminosidad promedio de cada elemento del output de $matsplitter(.)$. Dicho conjunto de luminosidades puede sintetizarse en un vector de dimensión $m\ n$.

4.2. Arte con el agente viajero

Tras haber seguido los pasos de la sub-sección 4 lo siguiente consiste en convertir la escala del conjunto $\{0, ..., 255\}$ a la escala $\{0, ..., \gamma\}$, empleando la ecuación, ec. (2) desglosada en la sub-sección 2.1 de la presente tesina. Así pues, obtenemos un vector de dimensión $m\ n$, llámese g. Para el primer ejercicio, asígnese $\gamma = 7$. Bajo esta γ, las escalas primera a treintaiseisava de luminosidad coinciden con la escala 7 en oscuridad (o siete 'ciudades'). Las siguientes luminosidades (por orden) corresponden a seis ciudades; sucesivamente, hasta llegar a la escala 255, que corresponde a cero ciudades. Es obvio que situar más puntos, o ubicar más aristas que conecten éstos en un rectángulo generará una mayor oscuridad en el mismo, de ahí que se considere a g como una escala de oscuridad. Lo siguiente consta en ubicar las ciudades en el plano $x-y$, haciendo un loop a través de las $m\ n$ matrices.

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

   $mod2 = kmod(n)$

   $si\ mod2 = 0$

   $entonces\ i = mod2$

   $e.o.c.$

   $i = mod2$

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

2. Una vez que tenemos la asignación $k \to (i, j) = (i(k), j(k))$, el siguiente paso es plasmar los puntos que asemejarán a la imagen objetivo sobre el rectángulo $[0, 1] \times [0, 1]$ en el plano $x-y$. Para ello, vale la pena considerar que el extremo superior izquierdo de la imagen corresponde al elemento $(1, 1)$ de la matriz y a la coordenada $(0, 1)$ del plano $x-y$. Partiendo

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

de aquí, deducimos la posición del extremo superior izquierdo de la coordenada genérica (i, j) de la matriz. La coordenada x corresponde al pixel con la columna j, representando un paso de $\frac{(j-1)}{n}$ a la derecha de $x = 0$. La coordenada y, con el renglón i, corresponde su ubicación en el eje y a un paso de $\frac{(i-1)}{m}$ unidades abajo de $y = 1$. Bajo esta formulación, derivamos que cada rectángulo (asociado a una submatriz) tendrá un ancho $\frac{1}{n}$ y un largo de $\frac{1}{m}$. Bajo este planteamiento se puede demostrar que para todo par de coordenadas de matriz de tipo $\{(i, j), (i + 1, j)\}$ o $\{(i, j), (i, j + 1)\}$ existirá una relación de adyacencia entre sus correspondientes rectángulos. Además, las esquinas de la figura que envuelve a este conjunto de $m\ n$ rectángulos serán las coordenadas $\{(0, 1), (1, 1), (0, 0), (1, 0)\}$. Con esto, garantizamos que los rectángulos mencionados conforman una partición del cuadrado $[0, 1] \times [0, 1]$

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

Al iterar este proceso las $m\ n$ veces, se obtienen las $\sum_{k=1}^{m\ n} g_k = \tau$ ciudades objetivo y podemos resolver el TSP sobre éstas. Los resultados de escribir todos las ciudades como puntos bajo los ejercicios con 15×15 rectángulos y (608 ciudades) y con 50×50 (5807 ciudades) se desglosan a continuación.

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. Dicha solución puede proceder mediante distintos métodos o heurísticas y puede realizarse mediante distintos *softwares*, algunos de los cuales son R, Python, Java y Concorde. En aras de practicidad, y en vista de que hasta el momento la construcción de

0. esta es una expresión equivalente a la ecuación ec.(3) , pero en la versión que mapea $(i, j) \to k(i, j)$

(1) 15×15 rectángulos o 608 ciudades

(2) 50×50 rectángulos o 5807 ciudades

Figura 3: Gráfica de los puntos sobre el cuadrante $x - y$

los rectángulos y las ciudades se ha llevado a cabo con R, se usará este lenguaje para manipular tanto objetos como archivos de tipo TSP. Para este efecto se usará la librería homónima cuya función, $ETSP(.)$ debe su nombre a *Euclidean TSP* y convierte el *data frame* de 2 columnas (una por coordenada) y τ renglones (uno por ciudades) en un problema del agente viajero. El *Euclidean* obedece a que las coordenadas podrías ser terrestres. La función encargada de resolver un etsp es $solve_TSP(.)$. Si bien se le puede ingresar el método a emplear, el que se usa por *default* es el algoritmo de inserción arbitraria con refinamiento de tipo two opt. El tipo de datos del *output* es doble: tour y entero. La versión íntegra consta del orden de las ciudades que se anidan para construir el tour. Con ello se puede graficar la solución con ayuda de $ggplot(.)$. Los resultados con 15×15 y 50×50 son los siguientes.

(1) 15×15 rectángulos o 608 ciudades; solución vía TSP

(2) 50×50 rectángulos o 5807 ciudades; solución vía TSP

Figura 4: Gráfica de las líneas uniendo a los puntos de manera 'eficiente' sobre el cuadrante $x - y$

4.3. Fotomosaicos

Posteriormente a reproducir los pasos descritos en la subsección 4.1, se procede a convertir la escala del conjunto $\{0, ..., 255\}$ a $[0, 1]$ dividiendo entre 255 (ver primer párrafo de la subsección referida). Llámese h al vector resultante de extraer estas escalas promedio de las submatrices ordenadas con base en el esquema *rowwise*. Supóngase que las dimensiones de las imágenes que compondrán el fotomosaico son cuadradas, de valor $pi \times pi$. Entonces, generaremos tres arreglos importantes: uno de parámetros de costos, dos de escalas de grises. El primero será uno numérico tres dimensional de tipo $|F| \times m \times n$, llámese $Arr3$, con $Arr3[f, i, j] = c_{fij}$, donde el lado derecho es el mismo ponderador de costos de la expresión (4a) . Uno de los arreglos con *greyscales* será de dimensión $m \times n$, tendrá por nombre mp, por 'matriz perfecta' el cual es un eufemismo por tratarse de la mejor representación factible de una imagen con las dimensiones que se le entregan; el otro, será un arreglo de $(m\ pi) \times (n\ pi)$ y será MF, por 'matriz final'. Además, se contará con una dotación de $\sum_{f=1}^{|F|} u_f$ imágenes de solo $|F|$ tipos, cada uno denotado mediante Mg_f por las iniciales de 'matriz de *grayscales*' de la imagen tipo f. A continuación se describe el bucle a través de $m\ n$ iteraciones que es la base para llenar los arreglos mencionados.

1. Tomamos el número de iteración, k, y le asignamos su correspondiente coordenada matricial, $(i(k), j(k))$ deducida por el esquema *rowwise* (previamente descrito, ver pseudo-algoritmo del primer paso del *loop* en la sección 4.2)

2. Obtener la escala $[0, 1]$ de luminosidad del elemento (i, j), h_k; es decir, el valor promedio de la correspondiente submatriz.

3. Obtener la diferencia cuadrática de las luminosidades; es decir, la que hay entre h_k y cada elemento del vector de luminosidades promedio de las $|F|$ figuras disponibles, llámese $lumP$. Esto es hacer $c_{ij} = [(h_k - lumP_f)^2]_{f \in \{1,...,|F|\}} \in [0, 1]^{|F|}$. Asignar el valor de este vector a los elementos $Arr3[i, j,]$ del arreglo $Arr3$.

4. Asignar el valor h_k al elemento $mp[i, j]$ de la matriz perfecta.

Posteriormente hay que construir un vector que cuantifique las dotaciones disponibles de las $|F|$ figuras. Para ello, nos valemos de dos parámetros: uno, hd, de 'holgura', que nos dirá qué tantas más de las fichas estrictamente necesarias tendremos. Si $hd = 0{,}1$, tendremos aproximadamente 10 % más de las fichas exactamente suficientes.

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

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

t	f	i	j	costo (ce)
1	1	1	1	A[1,1,1]
2	1	1	2	A[1,1,2]
⋮	⋮	⋮	⋮	⋮
n	1	1	n	A[1,1,n]
n+1	1	2	1	A[1,2,1]
n+2	1	2	2	A[1,2,2]
⋮	⋮	⋮	⋮	⋮
2n	1	2	n	A[1,2,n]
⋮	⋮	⋮	⋮	⋮
(m-1)n+1	1	m	1	A[1,m,1]
(m-1)n+2	1	m	2	A[1,m,2]
⋮	⋮	⋮	⋮	⋮
mn	1	m	n	A[1,m,n]
mn+1	2	1	1	A[2,1,1]
mn+2	2	1	2	A[2,1,2]
⋮	⋮	⋮	⋮	⋮
mn+n	2	1	n	A[2,1,n]
mn+n+1	2	2	1	A[2,2,1]
mn+n+2	2	2	2	A[2,2,2]
⋮	⋮	⋮	⋮	⋮
mn+2n	2	2	n	A[2,2,n]
⋮	⋮	⋮	⋮	⋮
mn+(m-1)n+1	2	m	1	A[2,m,1]
mn+(m-1)n+2	2	m	2	A[2,m,2]
⋮	⋮	⋮	⋮	⋮
2mn	2	m	n	A[2,m,n]
⋮	⋮	⋮	⋮	⋮
(|F|-1)mn+1	|F|	1	1	A[|F|,1,1]
(|F|-1)mn+2	|F|	1	2	A[|F|,1,2]
⋮	⋮	⋮	⋮	⋮
(|F|-1)mn+n	|F|	1	n	A[|F|,1,n]
(|F|-1)mn+n+1	|F|	2	1	A[|F|,2,1]
(|F|-1)mn+n+2	|F|	2	2	A[|F|,2,2]
⋮	⋮	⋮	⋮	⋮
(|F|-1)mn+2n	|F|	2	n	A[|F|,2,n|]
⋮	⋮	⋮	⋮	⋮
(|F|-1)mn+(m-1)n+1	|F|	m	1	A[|F|,m,1]
(|F|-1)mn+(m-1)n+2	|F|	m	2	A[|F|,m,2]
⋮	⋮	⋮	⋮	⋮
m n |F|	|F|	m	n	A[|F|,m,n]

Como se muestra, el ordenamiento de $df3$ sigue la jerarquía f, i, j. Con ayuda de esta tabla, podemos definir el modelo en un objeto de clase 'lp', auxiliándonos en el módulo *lpSolve* de R. Dicho objeto se basa en (a) un vector de pesos, ce, (b) una instrucción de optimización, io: 'minimiza' o 'maximiza', (c) una matriz RM de restricciones con dimensiones $re \times (m\ n\ f)$ que representa a las re restricciones existentes sobre las variables (d) un vector de lados derechos, $rh \in \mathbb{R}^{re}$ con el valor contra el cual será contrastada la combinación lineal de x, $RM[T,]x$, (e) un vector de carácteres,

$sign$, con el cual la restricción t-ésima será $x\ RM[T,]x\ 'sign[t]'\ rh_t$, con $sign \in \{'<', '\leq', '>', '\geq'\}$.

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

(a) $ce = df3['costo']$

(b) $io =' min'$

(c) Restricciones $RM \in \mathbb{R}^{re \times m\ n\ f}$ con $re = |F| + m\ n$, los primeros $|F|$ renglones representando a las restricciones de disponibilidad de las fichas que hay, y las restricciones restantes representando el correcto llenado de cada celda genérica (i,j). La matriz consta únicamente de valores 0 y 1. Los 1's se asignan facilmente conociendo, con la ayuda de la asginación $t(.)$ 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 (ij).

(d) $rh_f = u_f\ \forall f \in \{1, ..., |F|\}$

$rh_t = 1\ \forall f \in \{|F|, ..., |F| + m\ n\}$

(e) $sign_f =' \leq'\ \ \forall f \in \{1, ..., |F|\}$

$sign_t =' ='\ \ \forall f \in \{|F|, ..., |F| + m\ n\}$

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

1. Extraemos el k-ésimo vector-renglón.$S_k = dfSel[k,]$. Así, obtenemos $S_k['t'] = t$, $S_k['i'] = i(t)$, $S_k['j'] = j(t)$, $S_k['f'] = f(t)$, con el renglón, la columna y la figura inferidas de la función inversa de $t(.)$

2. Definimos el extremo superior izquierdo de la imagen $f(t)$ sobre la imagen final, $MF[rI, cI]$. Sígase que $rI(t) = (i(t)-1)pi + 1$ y $cI(t) = (j(t)-1)pi + 1$

3. Luego, se asigna el valor de $MF_{f(t)}$ a la submatriz con el extremos superior izquierdo en (rI, cI); esto es, $MF[\{rI, ..., rI + pi - 1\}, \{cI, ..., cI + pi - 1\}] = MG_f$

Se puede validar que este procedimiento llena por completo y con valores factibles a la matriz MF. Habiendo terminado las iteraciones y probado el ejercicio con $dp = 0{,}3$ y $vp = 0{,}3$ podemos desplegar la imagen resultante de los fotomosaicos con dimensiones 15×15 y 20×20

(1) Fotomosaico: 20 tipos de figuras, 15×15 mosaicos

(2) Fotomosaico: 20 tipos de figuras, 20×20 mosaicos

Figura 5: Soluciones gráficas al problema de los fotomosaicos con $dp = 0{,}3$ y $vp = 0{,}3$; dimensiones 15×15 y 20×20

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

(1) Matriz perfecta, dimensiones 15×15

(2) Matriz perfecta, dimensiones 20×20

Figura 6: Matrices perfectas basadas en luminosidades promedio de submatrices

5. Referencias

[1]