Implementación y contrastación empírica de una técnica de reconocimiento de imágenes

Sergio Sebastián Giro
Universidad Nacional de Rosario

Director: Doctor Gabriel Valiente
Universidad Politécnica de Cataluña

10 de abril de 2004

Índice General

Capítulo 1

Introducción

El propósito de esta tesina es estudiar en profundidad una técnica para el cálculo de similaridad entre imágenes utilizando grafos. La idea está presentada en [3]. A los fines de realizar experimentos con imágenes para evaluar qué tan apropiados son los modelos y las aproximaciones allí expuestos extenderemos los algoritmos para manejar imágenes a color, estudiaremos en profundidad los algoritmos en busca de alguna posible mejora, propondremos procesamientos de las imágenes y parámetros adecuados para mejorar la calidad con la que la similaridad es calculada, y probaremos una implementación concreta con diferentes tipos de imágenes.

Los métodos para comparación de imágenes pueden dividirse, a grandes rasgos, en reconocimiento de imágenes de alto nivel y análisis de imágenes de bajo nivel. El primer método intenta extraer objetos de interés con técnicas como uso de umbrales, segmentación, contornos, detección de texturas, y luego comparar los objetos. El último método, por otro lado, realiza una comparación directa de las imágenes como entidades completas calculando alguna medida de similaridad de imágenes. Nuestra aproximación toma elementos de ambos métodos ya que, por un lado, la imagen es dividida en regiones y, por el otro, la forma de los contornos, las texturas, etc. no son tenidos en cuenta.

Las medidas más ampliamente usadas para calcular la similaridad entre imágenes basadas en la distancia son las métricas basadas en la distancia de Hausdorff. Para un resumen sobre las carácterísticas esenciales de las métricas de Hausdorff y la bibliografía disponible véase [3]. La distancia normalizada de Hausdorff tiene la ventaja de que puede ser calculada en tiempo $O(n)$, donde n es el número de píxeles de la imagen. Sin embargo, la distancia de Hausdorff no es invariante ante rotación ni *scaling* (ampliación o reducción): si rotamos la imagen o la ampliamos, la distancia de Hausdorff no puede darse cuenta de que la imagen es la misma.

Las propiedades de invariancia ante traslación y rotación se obtienen automáticamente cuando usamos una representación basada en grafos junto con técnicas de matching, como se ha demostrado en el campo de reconocimiento de patrones estructurales. Además, la invariancia ante *scaling* es considerada en nuestra propuesta dado que la información que calculamos es el tamaño relativo de una región respecto del tamaño total de la imagen.

La idea de utilizar grafos etiquetados para representar objetos con estructura no es nueva. Esta idea ya fue usada en campos como la biología molecular, la química y la farmacología, entre otros. A partir de esta motivación se introduce en [2] la distancia de edición entre grafos, definida como la secuencia de operaciones para transformar uno en el otro con menor costo. También se demuestra la NP-completitud del problema del cálculo de la distancia y se proponen algoritmos heurísticos basados en encontrar secuencias de costo pequeño, a través de la búsqueda en un espacio de estados. Nuestro enfoque es diferente, ya que el cálculo que realizaremos es exacto (un matching de costo mínimo) sólo que éste no es necesariamente la secuencia de operaciones para transformar un grafo en el otro, sino que es una asignación de lados en lados con el menor costo posible.

En [1] se comparan la búsqueda heurística y los algoritmos genéticos para resolver el problema de encontrar el grafo con menor distancia promedio respecto de todos los grafos de un conjunto dado. El método de búsqueda es A* y la heurística no está especificada. Los algoritmos genéticos obtienen mejores resultados, tanto en tiempo de ejecución como en la cantidad de memoria necesaria, dando en general el resultado exacto.

El matching ya fue utilizado en reconocimiento de patrones. En [4] se describe una técnica que consiste en descomponer los grafos originales (*attributed relational graphs*, ARGs) en subgrafos más pequeños (*basic attributed relational graphs*, BARGs) y realizar un matching de menor costo que asigne los BARGs de uno a los del otro. A su vez, también se usa el matching para calcular el costo de asignar un BARG a otro.

Capítulo 2

Grafos de adyacencia de regiones

Con el propósito de comparar dos imágenes, recurriremos a los grafos de adyacencia de regiones. Cada imagen será trasladada a uno de estos grafos, quienes serán comparados posteriormente para medir el grado de similaridad de las imágenes en base a las que fueron construidos. Las siguientes definiciones serán útiles en este capítulo y los siguientes.

2.1 Definiciones preliminares

Definición 2.1 (Grafo dirigido). Un par de conjuntos $G = (V, L)$ es un grafo dirigido si $L \subseteq V \times V$, donde $X \times Y$ denota el producto cartesiano entre X e Y. El conjunto V es llamado el *conjunto de vértices o conjunto de nodos de* G y L es llamado el *conjunto de lados o conjunto de arcos de* L. Si existe un lado $l = (v_1, v_2)$ se dice que v_1 y v_2 son *adyacentes*, que l va de v_1 a v_2, que v_1 es la *cola* de l y v_2 es la *cabeza* de l y que l es *incidente* en v_1 y en v_2.

Definición 2.2 (Grafo dirigido etiquetado). Decimos que un grafo dirigido (V, L) está etiquetado con la función e, si existen dos conjuntos Σ_V y Σ_L (correspondientes a las etiquetas de los vértices y a las etiquetas de los lados, respectivamente) tales que $e : V \cup L \to \Sigma_V \cup \Sigma_L$ y además $\forall v \in V \bullet e(v) \in \Sigma_V \wedge \forall l \in L \bullet e(l) \in \Sigma_L$.

Con el fin de poder trabajar algorítmicamente con una imagen, debemos formalizar este concepto. La formalización siguiente refleja evidentemente una imagen digitalizada.

Definición 2.3 (Imagen). Dado un conjunto C de colores posibles, dos números naturales m (la altura de la imagen) y n (el ancho de la imagen), una imagen es una función $i : [0..m-1]_{N_0} \times [0..n-1]_{N_0} \to C$.

Definición 2.4 (Píxel). Dada una imagen I de altura m y ancho n, un píxel en la imagen I es un par ordenado (i, j), donde $i \in [0..m-1]_{N_0}$ y $j \in [0..n-1]_{N_0}$.

Definición 2.5 (Color de un píxel). Dada una imagen I y un píxel p en I, el color de p (en símbolos $C(p)$), se define como $i(p)$, donde i es la función correspondiente a la imagen I.

Definición 2.6 (4-adyacencia). Se dice que dos píxeles $p_1 = (i_1, j_1)$ y $p_2 = (i_2, j_2)$ son 4-adyacentes si $|i_1 - i_2| + |j_1 - j_2| = 1$. Podemos denotar la 4-adyacencia como una relación $p_1 \operatorname{Ady}_4 p_2$.

Definición 2.7 (8-adyacencia). Se dice que dos píxeles distintos $p_1 = (i_1, j_1)$ y $p_2 = (i_2, j2)$ son 8-adyacentes si $\max(|i_1 - i_2|, |j_1 - j_2|) = 1$. Podemos denotar la 8-adyacencia como una relación $p_1 \operatorname{Ady}_8 p_2$.

En general, se fija un tipo de adyacencia y se utiliza el mismo con todos los píxeles de una imagen y todas las imágenes a comparar. Cuando esté claro el tipo de adyacencia, podemos denotar a ésta como $p_1 \operatorname{Ady} p_2$. Las relaciones de adyacencia son obviamente simétricas. (Lo cual puede probarse sencillamente a partir de sus definiciones y de las propiedades del valor absoluto.)

Definición 2.8 (Similaridad). Sea una imagen I con conjunto de colores C y una función de distancia entre colores $\operatorname{distcol} : C \times C \to N$ (donde se requiere que $\operatorname{distcol}(c, c) = 0$). Se dice que dos píxeles p_1 y p_2 en I son similares bajo un umbral de tolerancia θ, si $\operatorname{distcol}(C(p_1), C(p_2)) \leqslant \theta$. Podemos denotar la similaridad como una relación $p_1 \operatorname{Sim} p_2$.

Definición 2.9 (Corregionalidad). Fijado un tipo de adyacencia (4-adyacencia u 8-adyacencia) la relación CoReg se define como la clausura reflexivo-transitiva de la relación $\operatorname{Ady} \wedge \operatorname{Sim}$.

2.2 Explicaciones coloquiales de las definiciones del apartado anterior

Figura 2.1: 4-adyacencia (izquierda) y 8-adyacencia (derecha) en una imagen.

Las imágenes que trabajaremos son grillas, en las que cada uno de los píxeles tiene un color bien definido. Los tipos de adyacencia que podemos llegar a considerar son dos: la 4-adyacencia permite que sólo una de las coordenadas de un píxel diste a lo sumo 1 de la misma coordenada del otro píxel; la 8-adyacencia permite que ambas coordenadas de un píxel disten a lo sumo 1 de la misma coordenada en el otro píxel. La figura 2.1 ilustra ambos tipos de adyacencia.

La similaridad entre dos píxeles se da cuando éstos tienen colores que están dentro de un determinado umbral: es frecuente que en una imagen una superficie homogénea esté compuesta de píxeles de diferentes colores, pero parecidos entre sí.

Para entender mejor la definición de corregionalidad, es mejor introducir antes la noción de región.

2.3 Regiones

La definición de regiones de una imagen parte del siguiente resultado.

Teorema 2.1. *La relación* CoReg *es una relación de equivalencia.*

Demostración. Las relaciones Sim y Ady son simétricas. Por tanto, la relación $\mathrm{Sim} \wedge \mathrm{Ady}$ es simétrica también. La clausura reflexivo-transitiva de $\mathrm{Sim} \wedge \mathrm{Ady}$ (es decir, CoReg) es entonces reflexiva, simétrica y transitiva.

Entonces, CoReg es una relación de equivalencia. □

Corolario 2.1. *La relación* CoReg *particiona el conjunto de píxeles.*

Demostración. Trivial a partir del teorema anterior. □

Definimos ahora las regiones de una imagen.

Definición 2.10 (Regiones). Una región es cada uno de los conjuntos de píxeles en que es particionado el conjunto P de píxeles de una imagen por la relación CoReg.

Definición 2.11 (Tamaño relativo de una región). El tamaño relativo de una región R_i es igual a $\operatorname{card} R_i / \operatorname{card} P$.

Definición 2.12 (Adyacencia de regiones). Dos regiones R_1 y R_2 distintas son adyacentes si existen $p \in R_1$ y $q \in R_2$ tales que $p \,\mathrm{Ady}\, q$.

Definición 2.13 (Color de una región). Sea una función $\mathrm{cr} : \mathcal{P} \to C$, donde P es el conjunto de píxeles y C el conjunto de colores posibles. (cr es una función que, dado un conjunto de píxeles, trata de "promediar" su color). Entonces, se define el color de una región R como $\mathrm{cr}(R)$.

Con estas definiciones estamos listos para la introducción del algoritmo para la construcción del grafo de adyacencia.

2.4 Algoritmo para la obtención del grafo de adyacencia (Parte 1)

Bástenos con saber, por el momento, que para la construcción del grafo de adyacencia de una imagen necesitamos conocer el tamaño relativo de cada región, su color y cuáles regiones son adyacentes.

Dividiremos el algoritmo en dos etapas. La primera se encargará de obtener el tamaño relativo y el color de cada región; la segunda se encargará de establecer cuáles son los lados que los unen.

2.4.1 El algoritmo

La presentación siguiente es un algoritmo conceptual. Se ilustran sólo los parámetros relevantes y no se tienen en cuenta restricciones de alcance de variables.

La etapa de obtención de lados puede ejecutarse más eficientemente si, para cada región, disponemos de una serie de píxeles candidatos en los cuales buscar regiones adyacentes. La A en el algoritmo es una función que para cada región devuelve un conjunto de píxeles posiblemente pertenecientes a regiones adyacentes. No se asegura que todo píxel perteneciente al conjunto pertenezca a otra región (puede que pertenezca a la misma), pero sí que las regiones de todos los vértices en el conjunto cubren todas las regiones adyacentes a la región en cuestión.

En lo que sigue, utilizaremos r para denotar una región determinada por el algoritmo y R para denotar una región en la imagen. (La idea de esta sección es argumentar que estos dos conjuntos se corresponden entre sí).

Algoritmo 2.1 *(ObtenerGrafoAdyacencia).* Dados un conjunto P de píxeles, una imagen I, un umbral de similaridad y una función que para cada píxel devuelve los píxeles adyacentes a éste,

  $A = \mathit{ObtenerRegiones}$
  $\mathit{ObtenerArcos}$

retorna un grafo dirigido etiquetado que representa la imagen I.

El algoritmo para obtener regiones tomará un píxel y buscará todos los píxeles que pertenezcan a la misma región. Puede que un píxel ya haya sido asignado a una región anteriormente, lo cual se revisa antes para no repetir el proceso. De no estar asignado, se crea una nueva región, se marca el píxel como perteneciente a ésta y se intenta extenderla.

Algoritmo 2.2 *(ObtenerRegiones).*

```
marcar cada píxel en P como no asignado a ninguna región
para todo p ∈ P hacer
   si p no está asignado a ninguna región entonces
      HacerRegion(p)
   fin si
fin para
```

Algoritmo 2.3 *(HacerRegion(p)).*

crear una nueva región r
marcar a p como perteneciente a r
ExtenderRegion(r, p)

El algoritmo para extender una región trata de extender la región a sus píxeles vecinos. Si de alguno de éstos no puede decirse que pertenezca a la región, es marcado como candidato a ser adyacente.

Algoritmo 2.4 *(ExtenderRegion(r,p)).*

para todo p_i adyacente a p **hacer**
  si p es similar a p_i y p no está asignado a ninguna región **entonces**
    marcar a p_i como perteneciente a r
    ExtenderRegion(r, p_i)
  fin si
  si p no es similar a p_i **entonces**
    marcar a p_i como candidato a adyacente de r
  fin si
fin para

2.4.2 Pruebas de corrección

Conforme los algoritmos presentados hasta el momento, valen los siguientes resultados.

Teorema 2.2. *Nunca se marca el mismo píxel más de una vez (ni siquiera para la misma región).*

Demostración. Nunca se marca un píxel sin haberse verificado que no haya sido asignado anteriormente. Esto es obvio salvo en el caso de la marca en *HacerRegion*, pero *HacerRegion* se llama solamente desde *ObtenerRegiones* y luego de verificar si el píxel ya fue asignado. □

Teorema 2.3. *Nunca se llama dos veces a ExtenderRegion(r,p) para el mismo píxel p.*

Demostración. Se llama a *ExtenderRegion* para un píxel cada vez que se lo marca como perteneciente a una región. Por el teorema anterior, se llama una vez exactamente para cada píxel. □

Teorema 2.4. *Todo píxel es asignado a una región.*

Demostración. Nunca se marca como no asignado un píxel salvo en la inicialación, ni se marca el mismo píxel más de una vez. Por tanto, un píxel que fue asignado a una región sigue así hasta la finalización del algoritmo.

Para cada píxel no asignado a ninguna región, *ObtenerRegiones* llama a *HacerRegion*, el cual lo marca como perteneciente a una región.

Luego, al finalizar todo píxel está asignado. □

Teorema 2.5. *Al finalizar ObtenerRegiones, dos píxeles p y q están asignados a la misma región si y sólo si* $p \operatorname{CoReg} q$.

Demostración. Después de la inicialación, ningún píxel está asignado a ninguna región. Por lo tanto, es trivialmente verdadero que el algoritmo funciona consistentemente hasta ese punto.

Trataremos ahora de demostrar que la propiedad se sigue cumpliendo a través de las sucesivas llamadas a *HacerRegion*, y lo haremos por inducción.

Caso base: la primera vez que se llama a *HacerRegion* ningún píxel está asignado. Por tanto, todos los píxeles que se encuentren marcados durante su ejecución (y la de los algoritmos que invoca) han sido marcados en la misma región r del píxel inicial.

Puede probarse por inducción sobre las veces en que se marcan píxeles que todo píxel q que se marca en la región r de p es tal que $p \operatorname{CoReg} q$, basándose en los hechos de que $p \operatorname{CoReg} p$ (caso base, marca en *HacerRegion*) y de que si $p \operatorname{CoReg} p_1$ y p_1 $(\mathrm{Ady} \wedge \mathrm{Sim})$ p_2, entonces $p \operatorname{CoReg} p_2$ (por propiedad transitiva de CoReg, paso inductivo, correspondiente a la marca en *ExtenderRegion*).

Supongamos que un píxel q fuera tal que $p \operatorname{CoReg} q$. Esto quiere decir que existe una sucesión (finita) de píxeles distintos $p_0, p_1, \ldots, p_j$ tales que $p_0 = p$, $p_j = q$ y $\forall i : 1..j.p_{i-1}(\mathrm{Sim} \wedge \mathrm{Ady})p_i$.

Puede probarse por inducción que cada uno de ellos es asignado a la región r, basándonos en la argumentación del primer párrafo que establece que si un píxel ha sido asignado, lo ha sido a r, y en el hecho de que cada vez que se marca un píxel, se llama a *ExtenderRegion* para éste. Luego, p y q aparecen marcados en la misma región.

Podemos concluir que, después de la primera llamada a *HacerRegion*, dos píxeles p y q están asignados a la misma región si y sólo si $p \operatorname{CoReg} q$.

Paso inductivo: supongamos ahora que el algoritmo funciona consistentemente hasta la enésima llamada a *HacerRegion*. Verificaremos ahora que sigue haciéndolo después de la siguiente llamada.

Por idéntico argumento al del caso base obtenemos que todos los píxeles q marcados en la nueva región r son tales que $p \operatorname{CoReg} q$.

También existe la sucesión p_i para todo píxel q tal que $p \operatorname{CoReg} q$. Sólo que ninguno de éstos p_i está asignado a ninguna región al comienzo del algoritmo (ya que $p_i \operatorname{CoReg} p$ y, por hipótesis inductiva, debería estar marcado p). Por inducción puede probarse que cada uno de ellos es asignado a la región r, basándonos en que, si han sido asignados, lo han sido a r, y en el hecho de que cada vez que se marca un píxel, se llama a *ExtenderRegion* para éste. Luego, p y q aparecen marcados en la misma región.

Podemos concluir que después de la $(n+1)$-ésima llamada a *HacerRegion*, se cumple la propiedad deseada.

A través del caso base y del paso inductivo, concluímos el teorema. $\square$

Teorema 2.6. *Las regiones asignadas por el algoritmo se corresponden bi - unívocamente con las regiones en la imagen.*

Demostración. Encontraremos una función f biyectiva que vaya del conjunto de regiones de la imagen al conjunto de regiones asignadas por el algoritmo.

Sea una región R en la imagen. Todos los píxeles correspondientes a esa región han sido asignados a una región r (por el teorema anterior). Entonces, podemos definir $f(R) = r$.

Para ver que la función es inyectiva, supongamos que $f(R_1) = r$ y $f(R_2) = r$ entonces, todos los píxeles de R_1 y todos los de R_2 han sido asignados a r. Por el teorema anterior, todos los píxeles de R_1 están en la misma región que todos los de R_2. Entonces, $R_1 = R_2$.

Para probar que la función es sobreyectiva, notemos que al estar hablando de regiones asignadas por el algoritmo, cada una de las r tiene un píxel al que fue asignada. Como cada píxel está en una región, una de las R es tal que $f(R) = r$ para cada r.

□

Teorema 2.7. *Si una región R_1 es adyacente a otra (distinta) R_2, entonces al menos existe un píxel en R_2 que está marcado como posible adyacente de r_1 (la región asignada por el algoritmo correspondiente a R_1).*

Demostración. Existen, por definición de adyacencia de regiones, dos píxeles adyacentes $p_1 \in R_1$ y $p_2 \in R_2$ tales que $p_1 \,\mathrm{Ady}\, p_2$ pero $(p_1, p_2) \notin \mathrm{Sim}$. Cuando se asignó p_1 a r_1, se llamó a *ExtenderRegion* para el píxel. Como los dos píxeles no son similares, *ExtenderRegion* agregó el píxel p_2 a la lista de posibles adyacentes.

□

También podemos decir que, si un píxel está en la lista de candidatos pero no es adyacente, entonces pertenece a la región (lo cual puede ser verificado fácilmente).

A partir de los resultados anteriores, podemos dejar de distinguir entre regiones en la imagen y regiones asignadas por el algoritmo.

La prueba que sigue será necesaria en el algoritmo de obtención de arcos.

Teorema 2.8. *Si un píxel está marcado como posible adyacente de una región, es adyacente a un píxel de la región.*

Demostración. Sólo se marcan píxeles como posibles adyacentes en *ExtenderRegion*, y esta función se llama sólo para píxeles adyacentes a un píxel de la región.

□

2.4.3 Obtención de los datos pertenecientes a las regiones

Cada vez que se marca un píxel como perteneciente a una región, podemos ir calculando los datos que nos propusimos al comienzo, es decir: el tamaño relativo de una región y su color.

Aprovechando que cada píxel se marca una sola vez, podemos computar el tamaño relativo de una región sumando uno a un contador cada vez que se marca un píxel como perteneciente a ella. Al final del algoritmo, sólo deberíamos hacer una división entre el contador y el tamaño total.

Si la función para el color de una región cr puede calcularse diferencialmente y en tiempo lineal (esto es, si $\mathrm{cr}(S \cup p) = f(p, \mathrm{cr}(S))$ con $p \notin S$ para alguna f que puede ser calculada en tiempo $O(1)$), podemos calcular el color de una región incrementalmente, a medida que agregamos sus píxeles, sin agregar complejidad.

Dado que cada vez que asignamos un píxel a una región sólo realizaremos estos cálculos y marcaremos (por ejemplo, en un arreglo, en tiempo constante) el píxel como perteneciente a la región, vale el siguiente teorema.

Teorema 2.9. *Asignar un píxel a una región consume tiempo constante.*

(Bajo la hipótesis de que el color de una región puede calcularse diferencial y linealmente)

2.4.4 Complejidad del algoritmo presentado

El siguiente teorema parece precisar demasiadas hipótesis. Lo cierto es que estas hipótesis valen para los diferentes parámetros con los que utilizaremos el algoritmo (definiciones de similaridad, adyacencia, etc.) y es muy probable que valgan para otros.

Teorema 2.10. *Bajo las hipótesis de que:*

- *la similaridad puede ser calculada en tiempo constante.*
- *el color de una región puede calcularse diferencial y linealmente.*
- *la estructura de datos que guarda los píxeles candidatos a adyacentes de una región permite inserción en tiempo constante*

El algoritmo tarda tiempo $O(n)$ (donde n es la cantidad de píxeles de la imagen) en asignar cada píxel a una región, y en calcular el tamaño promedio y el color de cada una de las regiones.

Demostración. Para esta demostración, agruparemos las diferentes acciones que realiza el algoritmo. Veremos que ninguna de estas particiones supera la complejidad $O(n)$.

El algoritmo realiza estas acciones:

- Marcar un píxel como no asignado a región alguna.
- Estimar si un píxel es similar a otro.
- Revisar si un píxel está asignado.
- Marcar un píxel como asignado a una región.
- Marcar un píxel como candidato a adyacente a una región.

El primer grupo ocurre una vez para cada píxel y, obviamente, cada asignación puede realizarse en tiempo constante (de hecho, puede utilizarse un "valor mágico" que indique que el píxel aún no está asignado, en un arreglo que ligue los píxeles a las regiones). Por tanto, el costo de este grupo de operaciones es $O(n)$.

Se pregunta si un píxel p es similar a otro para cada píxel adyacente a p. En nuestras definiciones de adyacencia, cada píxel es adyacente a un número constante de píxeles. Entonces, para cada píxel p se realizan $O(1)$ comparaciones de similaridad. De aquí deducimos que para los n píxeles se necesita tiempo $O(n)$, siempre y cuando el test de similaridad pueda realizarse en tiempo constante.

Por idéntico argumento obtenemos que el tercer grupo también puede llevarse a cabo en tiempo lineal (pues un píxel se revisa para ver si está asignado conjuntamente cuando se revisa su similaridad con otro píxel).

Hemos demostrado que cada píxel se asigna a una región exactamente una vez, y también que (bajo la hipótesis sobre el cálculo del color) esta asignación puede realizarse en tiempo constante. Basándonos en esto, podemos afirmar que el cuarto grupo de operaciones se realiza en $O(n)$.

El último grupo de operaciones puede realizarse colocando en listas (o alguna estructura de datos similar) los píxeles que pueden ser adyacentes a una determinada región. Cada píxel puede ser agregado a una lista a lo sumo un número constante de veces (una vez por cada píxel al cual es adyacente). Como se tarda un tiempo constante en almacenar el píxel (por hipótesis), concluimos que para todos los píxeles el tiempo total es $O(n)$.

Como una ejecución del algoritmo consiste en una secuencia de estos pasos en algún orden, y la complejidad temporal de todos ellos es $O(n)$, concluimos que la complejidad del algoritmo es $O(n)$.

$\square$

Teorema 2.11. *El algoritmo anterior consume $O(n)$ espacio.*

Demostración. Las estructuras necesarias son:

- Una estructura que ligue cada píxel a un valor de verdad que indique si ya ha sido asignado
- Una estructura que ligue cada píxel a la región a la que está asociado (en el caso de que esté asociado a alguna región)
- Una estructura que, para cada región, indique el cómputo (posiblemente parcial) de su color y su tamaño promedio, y la lista de píxeles posiblemente adyacentes.

La primera estructura puede implementarse utilizando $O(n)$ espacio asignando a cada píxel un número y utilizando un arreglo de bits.

La segunda estructura puede implementarse de manera análoga, asignando a cada región un número.

(En la práctica, estas dos estructuras pueden unirse en una sola, que utilice un "número mágico" para indicar que un píxel no ha sido asignado aún a ninguna región).

La tercera estructura puede implementarse con un arreglo que para cada píxel devuelva una tripleta con los dos cómputos parciales y una lista enlazada con los píxeles posiblemente adyacentes. Es claro que, de no existir las listas enlazadas la estructura puede almacenarse en O(n). Pero las listas enlazadas también ocupan O(n) espacio, ya que cada píxel se introduce en una lista, en el peor caso, por todos sus vecinos.

□

2.5 Algoritmo para la obtención del grafo de adyacencia (Parte 2)

Las regiones, calculadas a partir de la imagen como se explica en la región precedente, constituirán los vértices del grafo de adyacencia. Estos vértices estarán etiquetados por el tamaño relativo de la región.

Los lados del grafo, en cambio, estarán relacionados con la adyacencia de regiones, y sus etiquetas con las diferencias de colores entre las diferentes regiones. No utilizaremos nuevamente a este propósito la función distcol que utilizamos anteriormente en el cálculo de la similaridad, ya que ésta sólo intenta especificar qué *tan* diferentes son los colores y no *cómo* lo son. Dada esta nueva necesidad, utilizaremos una función difcol : $C \times C \to C^*$ que trata de capturar la diferencia entre dos colores, donde C^* es un nuevo conjunto de colores "diferencia". Este conjunto deberá tener un elemento identidad 0_C. Requeriremos que $\forall c \in C \bullet \text{difcol}(c,c) = 0_C$ y que $\text{difcol}(c_1,c_2) = \text{difcol}(c_2,c_1) \iff c_2 = c_1$.

A los fines de que el grafo sea más simple, estableceremos un criterio para elegir entre $\text{difcol}(c_1,c_2)$ y $\text{difcol}(c_2,c_1)$ (y utilizarlo consistentemente). Dadas dos regiones r_1 y r_2 adyacentes, con colores c_1 y c_2, respectivamente, en el grafo de adyacencia existirá un lado desde el vértice correspondiente a r_1 hasta el correspondiente a r_2 con etiqueta $\text{difcol}(c_1,c_2)$ si el criterio elige esta diferencia, o uno desde r_2 a r_1 si el criterio determina la diferencia $\text{difcol}(c_2,c_1)$, con este valor como etiqueta. Si la diferencia es 0_C, trazamos lados entre ambas regiones (la situación parece difícil de hallar en la práctica, ya que es bastante poco probable que dos conjuntos de píxeles tengan el mismo color, sean adyacentes y no sean parte de la misma región). Esta convención evita trazar dos lados por cada par de regiones adyacentes (salvo en el caso de lados con etiqueta 0_C) sin perder información de ningún tipo (siempre y cuando se utilice el mismo criterio para la elección de la diferencia en todas las imágenes). Podemos implementar el criterio como una función criterio : $C^* \times C^* \to \{\text{verdadero}, \text{falso}\}$ tal que criterio(a,b) es verdadero si la diferencia que debe escogerse es a, y falso si es b.

Para ejemplificar, supongamos que los colores son números naturales, y que la función diferencia es $f(a,b) = b - a$ (en cuyo caso, el conjunto C^* es el conjunto de los números enteros). Podemos tomar como criterio, para elegir una de las dos diferencias, escoger la que sea positiva o cero. En este caso, el arco indicaría un "aumento en el color" de una región a otra.

2.5.1 El algoritmo

Gracias al trabajo de guardar las píxeles posiblemente adyacentes podemos ahorrarnos el costo de recorrer todos los píxeles.

El algoritmo para obtener los arcos queda ahora de la siguiente manera:

Algoritmo 2.5 *(ObtenerArcos).*

para todo r region **hacer**
  para todo p pixel en la lista de posibles adyacentes a r **hacer**
    si region(p) $\neq r$ **entonces**
      si cr(region(p)) $\neq$ cr(r) **entonces**
        $dif1 = \text{difcol}(\text{cr}(\text{region}(p)), \text{cr}(r))$
        $dif2 = \text{difcol}(\text{cr}(r), \text{cr}(\text{region}(p)))$
        si criterio(*dif1*, *dif2*) **entonces**
          marcar un arco desde region(p) a r con etiqueta *dif1*
        sino
          marcar un arco desde r a region(p) con etiqueta *dif2*
        fin si
      sino
        marcar arcos entre ambas regiones con etiqueta O_C
      fin si
    fin si
  fin para
fin para

2.5.2 Pruebas de corrección

Probaremos un resultado en cuanto a la existencia de los arcos.

Teorema 2.12. *Existe un arco entre dos regiones si y sólo si las dos regiones son adyacentes.*

Demostración. Probaremos primero que si existe un arco entre dos regiones, éstas son adyacentes. Para eso, observemos que en la sección anterior probamos que todos los píxeles marcados como candidatos a adyacentes eran adyacentes a un píxel de la región. Como, dada una región, dentro del bucle principal sólo se marcan arcos desde o hacia píxeles que estén marcados como posibles adyacentes y que no pertenecen a la región en cuestión, entonces las dos regiones son adyacentes.

Para ver que si dos regiones son adyacentes se crea un arco que las une, recordemos el teorema que indica que toda región r_1 adyacente a otra r_2 tiene un píxel p que pertenece a ella en la lista de posibles adyacentes de r_2 (y viceversa). En el caso de que se revise primero la lista de posibles adyacentes de r_2, cuando llegue el turno de p se marcará un arco con la dirección adecuada. Cuando se revise r_1, el algoritmo se encontrará con las diferencias intercambiadas y marcará el mismo arco (en la práctica, esta repetición indeseable puede eliminarse si se establece un orden entre las regiones, y se examina un píxel sólo si pertenece a una región "mayor"). El caso en el que se revisa primero la lista de adyacentes de r_1 es análogo.

$\square$

2.5.3 Complejidad del algoritmo

El algoritmo itera sobre todos los nodos de las listas de posibles adyacentes de todas las regiones. Como la cantidad de estos nodos es $O(n)$ (dado que cada nodo se introduce en una lista, a lo sumo, un número constante de veces - una por cada uno de su vecinos).

La única estructura necesaria en esta parte del algoritmo es el grafo. Está claro que, como hay $O(n)$ píxeles en los que buscar adyacentes, hay a lo sumo $O(n)$ lados.

Si se utiliza una lista para cada nodo, indicando cuáles son los arcos que salen desde él, y con qué etiqueta, el algoritmo consumirá $O(n)$ espacio. (Esta representación es conocida como *representación a través de listas de adyacencia.*) La inserción puede realizarse insertando los lados sin verificar repeticiones y eliminando los lados repetidos al final, utilizando, por ejemplo, un algoritmo de ordenamiento. Esto lleva la complejidad del algoritmo a $O(n \log(n))$

2.6 Definición de grafo de adyacencia de dos imágenes

Definición 2.14 (Grafo de adyacencia de una imagen). Dada una función I, una definición de adyacencia y un umbral de similaridad, el grafo de adyacencia de I es el grafo dirigido etiquetado que el algoritmo *ObtenerGrafoAdyacencia* tiene como salida.

Capítulo 3

Comparación de imágenes digitales

En el capítulo anterior, nos ocupamos de construir un grafo a partir de una imagen; ahora, nos dedicaremos a ver qué tan similares son esos grafos, basados en el postulado de que la similaridad de los grafos refleja la similaridad de las imágenes originales. La contrastación de los postulados que realizamos será realizada en el capítulo siguiente, a través de pruebas experimentales realizadas con los algoritmos.

3.1 Distancia entre grafos

La distancia entre grafos se define informalmente como el "esfuerzo" necesario para transformar uno de los grafos en el otro. Está claro que este esfuerzo estará relacionado con cuánto cuesta transformar un nodo en otro o un lado en otro.

Las operaciones permitidas son:

- Borrar uno de los nodos que no tenga arcos incidentes
- Agregar un nodo con una determinada etiqueta
- Borrar un lado
- Agregar un lado cuya cabeza y su cola están en el grafo
- Reemplazar un lado entre dos nodos dados por otro con diferente etiqueta
- Reemplazar un nodo por otro

Las operaciones tienen diferente costo dependiendo de las etiquetas.

El costo de convertir un grafo en otro es el costo de una secuencia de costo *mínimo* de estas acciones. El fácil ver que, para asignaciones de costos razonables, cada nodo o cada arco participa una vez de cada operación en una secuencia de costo mínima (no podría ocurrir, por ejemplo, que se agregara un nodo con una determinada etiqueta, y luego ésta se cambiara, ya que seguramente es menos costoso insertarlo con la etiqueta adecuada). Podemos entonces restringirnos a los casos en que para cada nodo o lado se realiza una sola operación.

Para alcanzar este objetivo, observemos que seguramente habrá un conjunto de nodos eliminados y un conjunto de nodos agregados. Las operaciones que los eliminan y los agregan deberán realizarse. Los nodos del grafo de partida que no sean eliminados serán reemplazados por nodos en el grafo de llegada. Llamemos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ a los grafos a comparar. Si queremos transformar G_1 en G_2, y llamamos W_1 a los vértices en V_1 que son reemplazados, entonces el conjunto de los de los vértices eliminados es $V_1 \setminus W_1$. De igual manera, si llamamos W_2 al conjunto de vértices en el grafo de llegada que se obtienen por sustituciones de vértices del grafo de partida, el conjunto de nodos que se agrega es $V_2 \setminus W_2$.

Para borrar los vértices en V es necesario que eliminemos todos los lados incidentes en algún vertice de $V \setminus V_1$. Entonces, el grafo que nos queda consiste sólo de V_1 y de los lados que unen los vértices en él. Ahora, debemos cambiar los nodos en V_1 por nodos en V_2 y los lados en E_1 por lados en E_2

Esto motiva la definición de los matchings de aproximación de grafos.

Definición 3.1 (Matching de aproximación de grafos). Un matching de aproximación de grafos de un grafo etiquetado $G_1 = (V_1, E_1, e_1)$ a un grafo etiquetado $G_2 = (V_2, E_2, e_2)$ es una función biyectiva $f : W_1 \to W_2$ donde $W_1 \subset V_1$ y $W_2 \subset V_2$.

Un matching de aproximación de grafos f puede verse como una secuencia de operaciones de edición que transforman un grafo de edición en el otro (el grafo de *partida* en el de *llegada*). Todos los vértices en $V_1 \setminus W_1$ son borrados de G_1, todos los vértices en W_1 son sustituidos por vértices en W_2 y todos los en $V_2 \setminus W_2$ son insertados en G_2. Haciendo $X_1 = E_1 \cup (W_1 \times W_1)$ y $X_2 = E_2 \cup (W_2 \times W_2)$. Entonces, todos los arcos en $E_1 \setminus X_1$ son borrados de G_1, todos los arcos en X_1 son sustituidos por arcos en X_2 y todos los arcos en $E_2 \setminus X_2$ son insertados en G_2.

Asignando costos a cada una de las operaciones de edición, puede definirse entonces el costo de un matching de aproximación de grafos (siendo este último la suma de los costos de todas las operaciones elementales correspondientes). Podemos hacer entonces la siguiente definición:

Definición 3.2 (Distancia de edición entre grafos). La distancia de edición entre dos grafos G_1 y G_2 es el costo del mínimo matching de aproximación de grafos.

3.2 Nuestra aproximación

El cálculo de la distancia de edición es un problema NP-completo. A los fines de resolver el problema en un tiempo razonable, intentaremos una aproximación.

La definición de distancia de edición se basa en una correspondencia entre nodos. La idea de nuestra aproximación se basa en establecer una correspondencia entre lados. La inexactitud de la aproximación se debe a que al realizar diferentes asignaciones puede que aparezcan inconsistencias. Por ejemplo, un lado (a, b) puede ser asignado al lado (a', b') del otro grafo, pero el lado (a, c) puede ser asignado al lado (c', d') del otro grafo, con lo cual la región a queda asignada a dos regiones: a' y c'.

Cada lado en un grafo podrá ser asignado a un lado en el otro. Los lados en el grafo de pueden ser borrados y los lados en el grafo de llegada pueden ser agregados.

Para realizar esta asignación de lados en lados, de forma que también puedan borrarse y agregarse lados, necesitamos definir qué es el costo extendido de un matching.

3.2.1 Matching en grafos

Definición 3.3 (Grafo dirigido bipartito). Diremos que un grafo (V, L) es bipartito si existen $V_1 \subseteq V$ y $V_2 \subseteq V$ (las *clases* de V) tales que

$$\forall e \in L \bullet \mathrm{cola}(e) \in V_1 \wedge \mathrm{cabeza}(e) \in V_2$$

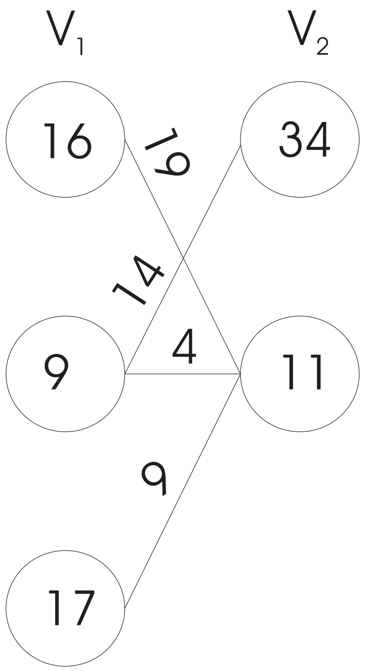

Figura 3.1: Un grafo bipartito

A menudo se suele dibujar a los grafos bipartitos colocando los conjuntos V_1 y V_2 en dos columnas. El grafo de la figura 3.1 muestra un grafo bipartito etiquetado con valores numéricos.

Definición 3.4 (Matching). Un matching en un grafo (V, L) es un subconjunto M de L tal que

$$\forall (v_1, v_2) \in L \bullet ((v_1, v_2') \in L \Rightarrow v_2 = v_2') \wedge ((v_1', v_2) \in L \Rightarrow v_1 = v_1')$$

El matching se utiliza a menudo para modelar asociaciones biunívocas: si hay un lado entre dos vértices en el matching, éstos se encuentran mutuamente asignados.

Definición 3.5 (Pertenencia de un vértice a un matching). Se dice que un vértice v pertenece a un matching M ($v \in M$) si

$$\exists v_2 \bullet (v, v_2) \in M \vee \exists v_1 \bullet (v_1, v) \in M$$

Definición 3.6 (Costo extendido de un matching). Sea (V, L) un grafo etiquetado con valores númericos por medio de la función f. El costo extendido de un matching M es

$$\sum_{e \in L \bullet e \in M} f(e) + \sum_{v \in V \bullet v \notin M} f(v)$$

Llamamos a este costo *extendido* ya que difiere del costo utilizado tradicionalmente (que no considera el segundo sumando). Este costo modela situaciones en las que hay una cierta pérdida cuando se dejan nodos sin asignar.

Para calcular el costo extendido de un matching llevaremos este problema a uno de transporte.

Definición 3.7 (Red de transporte). Sea $G = (V, L)$ un grafo dirigido, y sean dos funciones $p : L \to N_1$ (el costo o peso de un lado) y $c : L \to N_2$ (la capacidad de un lado), donde N_1 y N_2 son valores numéricos no negativos. Se dice que el grafo G es una red de transporte con costos p y capacidad c si existen un vértice f (llamado *fuente*) que no es cabeza de ningún lado y un vértice s (llamado *sumidero*) que no es cola de ningún lado.

Definición 3.8 (Flujo en una red y su costo). Sea $G = (V, L)$ una red con capacidad c, costo p, sumidero s y fuente f.

Un flujo en G es una función flujo : $L \to N_2$ tal que

$$\forall l \in L \bullet f(l) \leqslant c(l)$$

y

$$\forall v \in V \setminus \{f, s\} \bullet \sum_{\{e |\ \mathrm{cabeza}(e) = v\}} \mathrm{flujo}(e) = \sum_{\{e |\ \mathrm{cola}(e) = v\}} \mathrm{flujo}(e)$$

.

El costo del flujo es igual a

$$\sum_{e \in L} \text{flujo}(e) * c(e)$$

El tamaño del flujo en una red es igual al flujo que entra en el sumidero, es decir, a:

$$\sum_{(v,s) \in G} \text{flujo}((v,s))$$

(Puede probarse que esta cantidad es igual al flujo que sale de la fuente).

La idea de estos tipos de redes es que se debe transportar una cierta carga desde la fuente al sumidero en forma constante. Transportar una unidad de carga entre vértices tiene un costo, y por cada lado puede transportarse una cantidad máxima de carga. Además, a cada vértice debe llegar la misma cantidad de carga que la que sale (el segundo requisito de la definición anterior).

Definición 3.9 (Red asociada a un grafo bipartito). Dado un grafo $G = (V, L)$ etiquetado con la función l con clases V_1 y V_2. Definimos la red $R = (V', L')$ asociada a él como sigue:

$$\begin{array}{rcll} V' & = & V & \cup\{f, s, \Lambda_1, \Lambda_2\} \\ L' & = & L & \cup\{(f, v) \bullet v \in V_1\} \\ & & & \cup\{(v, s) \bullet v \in V_2\} \\ & & & \cup\{(\Lambda_1, \Lambda_2)\} \\ & & & \cup\{(\Lambda_1, v) \bullet v \in V_2\} \\ & & & \cup\{(v, \Lambda_2) \bullet v \in V_1\}\,. \end{array}$$

Su función de costo p se define como

$$\begin{array}{lcll} p(e) & = & l(e) & \text{si e} \in \text{L} \\ p((\Lambda_1, v)) & = & l(v) & \text{si } v \in V_2 \\ p((v, \Lambda_2)) & = & l(v) & \text{si } v \in V_1 \\ p(e) & = & 0 & \text{para los demás lados}\,. \end{array}$$

y su función de capacidad c como

$$\begin{array}{lcll} c((f, \Lambda_1)) & = & \text{card}(V_2) & \\ c((\Lambda_1, \Lambda_2)) & = & \min(\text{card}(V_1), \text{card}(V_2)) & \\ c((\Lambda_2, s)) & = & \text{card}(V_1) & \\ c(e) & = & 1 & \text{para los demás lados}\,. \end{array}$$

Teorema 3.1. *A cada matching de un grafo bipartito $G = (V, L)$ con clases V_1 y V_2 le corresponde un flujo en la red $R = (V', L')$ asociada a G cuyo tamaño es máximo y cuyo costo es igual al costo extendido del matching. Además a cada flujo de tamaño máximo en R le corresponde un matching en G con el mismo costo.*

Demostración. Dado que la suma de las capacidades de todos los lados que llegan hasta el sumidero es igual a $\mathrm{card}(V)$ un flujo no podrá tener mayor tamaño que éste. Por otra parte, el flujo dado por

$$\begin{array}{lcll}
\text{flujo}((f,\Lambda_1)) & = & \mathrm{card}(V_2) & \\
\text{flujo}((f,v)) & = & c((f,v)) & \text{si } v \in V_1 \cup \{\Lambda_1\} \\
\text{flujo}((\Lambda_2,s)) & = & \mathrm{card}(V_1) & \\
\text{flujo}((v,s)) & = & c((v,s)) & \text{si } v \in V_2 \cup \{\Lambda_2\} \\
\text{flujo}((v,\Lambda_2)) & = & 1 & \text{si } v \in V_1 \\
\text{flujo}((\Lambda_1,v)) & = & 1 & \text{si } v \in V_2 \\
\text{flujo}(e) & = & 0 & \text{para los demás lados}\,.
\end{array}$$

es de tamaño $\mathrm{card}(V)$, de donde se sigue que el mayor tamaño de un flujo es $\mathrm{card}(V)$.

Demostraremos ahora que a todo matching le corresponde un flujo en R de tamaño $\mathrm{card}(V)$ del mismo costo.

Sea el matching $M = \{(v_{1,1}, v_{2,1}), \ldots, (v_{1,n}, v_{2,n})\}$. Le asignaremos el flujo

$$\begin{array}{lcll}
\text{flujo}(e) & = & 1 & \text{si } e \in M \\
\text{flujo}(\Lambda_1,v) & = & 1 & \text{si } v \in V_2 \text{ y no pertenece a } M \\
\text{flujo}(v,\Lambda_2) & = & 1 & \text{si } v \in V_1 \text{ y no pertenece a } M \\
\text{flujo}(\Lambda_1,\Lambda_2) & = & \mathrm{card}(M) & \\
\text{flujo}(f,v) & = & 1 & \text{si } v \in V_1 \text{ y pertenece a } M \\
\text{flujo}(v,s) & = & 1 & \text{si } v \in V_2 \text{ y pertenece a } M \\
\text{flujo}(e) & = & 0 & \text{para los demás lados}\,.
\end{array}$$

Dado que los únicos lados que tienen costo distinto de cero son los de las tres primeras ecuaciones, el flujo que tienen asignado es uno y los costos de los lados son las etiquetas relevantes al costo del matching, el costo del flujo es igual al costo del matching.

A un flujo de tamaño máximo, le asignaremos un matching M tal que

$$(v_1, v_2) \in M \iff v_1 \in V_1 \land v_2 \in V_2 \land f((v_1, v_2)) \neq 0$$

(Es obvio que, dadas las restricciones de flujo, M es efectivamente un matching).

Dado que el flujo tiene tamaño $\mathrm{card}(V)$ deben existir lados de cada nodo no perteneciente a M hacia Λ_2 o desde Λ_1, según corresponda, con flujo igual a uno. Se sigue entonces, por un argumento similar al aplicado anteriormente, que el costo del flujo es igual al costo del matching.

$\square$

Para ilustrar el teorema, las figuras 3.2 y 3.3 muestran un matching particular y el flujo correspondiente en la red asociada.

Esto permite calcular un matching de costo mínimo extendido en un grafo de n vértices como un problema de máximo flujo de menor costo en una red de $n+2$ vértices. Dado que existen algoritmos para calcular un flujo máximo

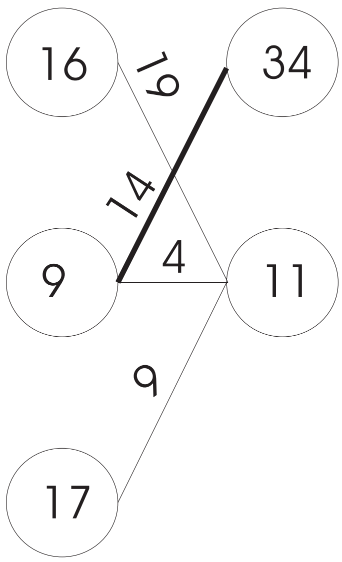

Figura 3.2: Un matching particular

de menor costo en tiempo $O(n^2 * F)$, donde F es el tamaño del flujo máximo, podemos calcular el matching de costo mínimo extendido en tiempo $O(n^3)$. Dado que el flujo queda determinado por el matching, puede usarse el matching como representación del flujo en el algoritmo de flujo máximo de menor costo, indicando, con dos arreglos, a qué nodo está asignado cada uno, si lo está. Esto hace que el algoritmo, dada una función que devuelva el costo entre dos lados, pueda implementarse en espacio $O(n)$.

3.2.2 Construyendo el grafo para calcular el matching de costo extendido mínimo

El grafo bipartito se construye a partir de dos grafos dirigidos etiquetados $G_1 = (n_1, l_1, e_1)$ y $G_2 = (n_2, l_2, e_2)$ de la siguiente manera:

- Cada clase corresponde a un grafo, y cada nodo en esa clase a un lado en el grafo correspondiente
- Cada nodo (que corresponde a un lado (a, b)) está etiquetado por por $(1 - k) * e_i(a) + (1 - k) * e_i(b) + k * e_i((a, b))$ con $i \in \{1, 2\}$. Recordemos que este es el costo de borrar o insertar un lado. Notemos que se trata de mantener los lados correspondientes a regiones de tamaño mayor y con mayores cambios de color (dado que éstos tienen mayor costo)
- Hay lados de cada nodo de cada clase a todos los nodos de la otra clase. Del nodo correspondiente al lado (a, b) al nodo correspondiente al lado

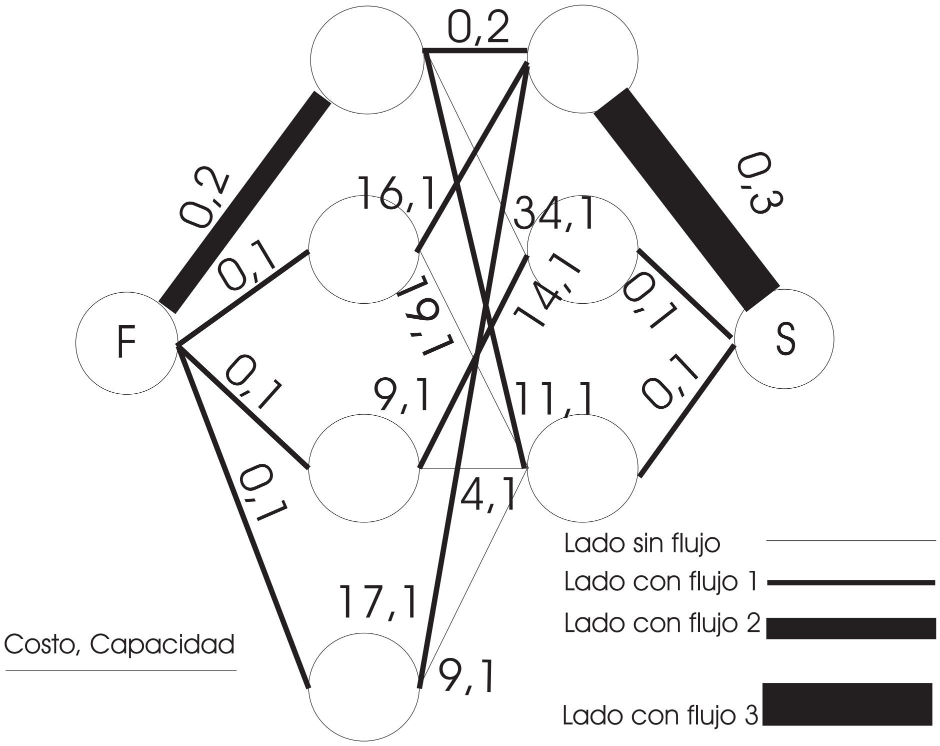

Figura 3.3: Flujo correspondiente al matching de la figura 3.2

(a', b') el lado está etiquetado con $(1-k) * |e_1(a) - e_2(a')| + (1-k) * |e_1(b) - e_2(b')| + k * |e_1((a,b)) - e_2((a',b'))|$.

donde k es un parámetro que mide el peso que tiene el valor de las etiquetas de los nodos respecto del valor de las etiquetas de los lados.

Definición 3.10. Similaridad aproximada entre imágenes Definimos la similaridad aproximada entre imágenes como el costo del matching extendido de costo mínimo de sus grafos de adyacencia de regiones.

Capítulo 4

Resultados Experimentales

Hasta aquí la formalización de la técnica de reconocimiento. En este capítulo intentaremos, a través de la utilización de diferentes parámetros, ver qué tan satisfactorio resulta su funcionamiento.

Al final del documento hay un índice de figuras que puede resultar particularmente útil en este capítulo.

4.1 Detalles particulares

A los fines de probar la técnica de reconocimiento de imágenes se implementaron dos programas: `grafoady` y `apares` que implementan los algoritmos de obtención del grafo de adyacencia y calculan las distancias de edición aproximadas entre todos los pares de una lista, respectivamente.

Las imágenes estaban en formato `ppm`, descriptas a través de tres componentes RGB, que variaban entre 0 y 255. Se consideraron entonces dos tipos de similaridades:

- por luminosidad: en la que se obtiene la luminosidad de cada píxel como un número real consistente en una combinación convexa de las tres componentes provista por la librería de manejo de imágenes ($0.299r + 0.587g + 0.114b$). El promedio de color de una región se obtiene promediando las luminosidades. La distancia entre colores se define como la diferencia en valor absoluto de los mismos y la diferencia se define como la resta. El criterio para la dirección de los arcos es que los arcos van de la región menos luminosa a la más luminosa

- por color: el promedio de color de una región consta de tres componentes, cada una de las cuales es el promedio de todas las componentes respectivas pertecientes a cada uno de los píxeles. La distancia entre dos colores se calcula como $\max\{|r_1 - r_2|, |g_1 - g_2|, |b_1 - b_2|\}$. La diferencia entre colores de regiones se calcula componente a componente y un arco dirigido de una región R_1 a otra R_2 indica que el color de R_1 es lexicográficamente menor que el de R_2. En el grafo para el matching la etiqueta que indica la sustitución de un lado por otro se calcula en forma análoga a la distancia (sólo que aquí el cálculo es entre diferencias, que pueden tener valores negativos, por lo que los valores pueden variar entre 0 y 510).

Se implementó también la posibilidad de eliminar regiones de un tamaño menor a cierto número dado (regiones superfluas), dado que es muy poco probable que una región con pocos píxeles sea importante para la estructura de la imagen.

Se agregó la posibilidad de difuminar los píxeles en cuadrados de lado n (n parámetro). Esta posibilidad estuvo motivada por el hecho de que, muchas veces, las imágenes intercalan píxeles de colores muy disímiles a los efectos de crear el efecto visual de un color homogéneo e intermedio entre esos colores. Al bajar la tolerancia, esto puede llevar a que regiones importantes se sobresegmenten o se eliminen (a través de la eliminación de regiones superfluas). La difuminación divide la imagen en pequeños cuadrados, y cambia el color de cada píxel en cada cuadrado por el color promedio del cuadrado en cuestión.

Los resultados de utilizar los diferentes parámetros son el tema de la siguiente sección.

4.2 Experimentos realizados

Se trabajó sobre una colección de 199 imágenes escogidas al azar (de hecho, fue el programa de reconocimiento quien fue indicando cuáles eran los grupos de imágenes similares). Se calcularon todas las distancias entre las imágenes, probando diferentes parámetros a los efectos de ver con cuáles de ellos se realizaba mejor el reconocimiento de las imágenes.

Dado que el rango de valor de las diferencias entre etiquetas de lados está entre 0 y 255 (usando similaridad por luminosidad), y el rango de valor de las diferencias entre etiquetas de nodos está entre 0 y 1, las pruebas con similaridad por luminosidad se realizaron con el valor $k = 0.0035$ que es cercano a $1/255$, para que estén los valores correspondientes a los arcos y a los vértices en la misma magnitud. El valor es menor que $1/255$ para que las regiones cobren algo más de peso (no hay que olvidar que la aproximación que estamos realizando hace posible que las regiones se confundan). En el caso de similaridad por color, por las mismas razones, se utilizó $k = 0.001$ o $k = 0.0015$, dado que las diferencias entre etiquetas de lados pueden variar entre 0 y 510.

La eliminación de regiones superfluas permitió mantener la cantidad de lados manejable (en ningún experimento superó los 1000 lados) considerando porciones importantes de las imágenes (nunca menos del 70%, y generalmente más del 90% de las mismas).

Los primeros experimentos se limitaron a utilizar el algoritmo considerando luminosidad, 4-adyacencia, eliminando las regiones con menos de 3 ó 4 píxeles y probando valores de θ cada vez más pequeños (25, 20 y 15). La figura 4.1 muestra la distribución de distintas imágenes sobre las distancias para $\theta = 25$, eliminando las regiones con menos de tres píxeles (línea fina); $\theta = 20$, eliminando las regiones con menos de cuatro píxeles (línea algo más gruesa) y $\theta = 15$, eliminando las regiones con menos de cuatro píxeles (línea más gruesa).

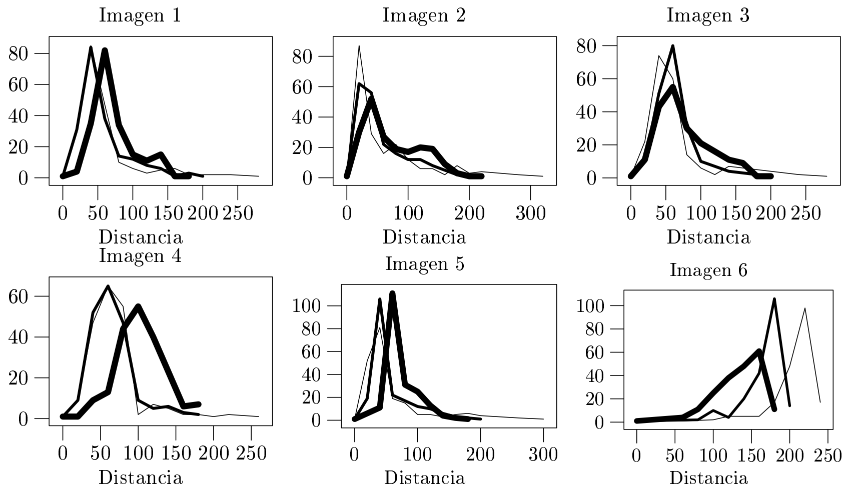

Figura 4.1: Distribución de imágenes para 6 imágenes diferentes y tres valores distintos de θ

Como podemos ver, en general las distribuciones parecen ser normales, con el eje de simetría corriéndose a la derecha a medida que disminuimos (salvo en el caso de la imagen 6) el valor de θ. Esto es bastante gráfico en el caso de la imagen 2, en el que para $\theta = 25$ el eje de simetría está muy cercano a 0, lo cual indica que para ese valor de θ todas las imágenes son muy similares a ella, y a medida que el valor de θ disminuye el programa empieza a distinguir. El caso de la imagen 6 se debe a que ésta es una imagen de cuyo tipo no hay otras en la colección. A medida que aumentamos θ, aumenta la cantidad de lados, y por lo tanto la posibilidad de confusión con otras imágenes (notemos que, a pesar de todo, el eje de simetría sigue lejos del cero). Para verificar esto podemos ver que, en el caso de $\theta = 15$, el matching con la imagen a menor distancia llegó a asignar a más de una región el 75% de las regiones de una imagen y el 70% de la otra, mientras que con $\theta = 20$ y la misma imagen los porcentajes bajan a 69% y 37%. Lo mismo ocurre con $\theta = 20$ y $\theta = 25$: los porcentajes bajan de 79% y 60% a 65% y 47%.

La figura 4.2 muestra la cantidad de imágenes a distancia menor o igual que un número dado para las imágenes 1, 2, 3, 4 y 5.

Un problema es determinar qué tan baja debe ser la distancia entre dos imágenes para considerarlas similares. Los aumentos bruscos en la pendiente de las curvas indican que, al tolerar una distancia algo mayor, la cantidad de imágenes consideradas similares se vuelve mucho mayor. En base a eso, podríamos pensar que nuestro algoritmo funciona correctamente si las imágenes

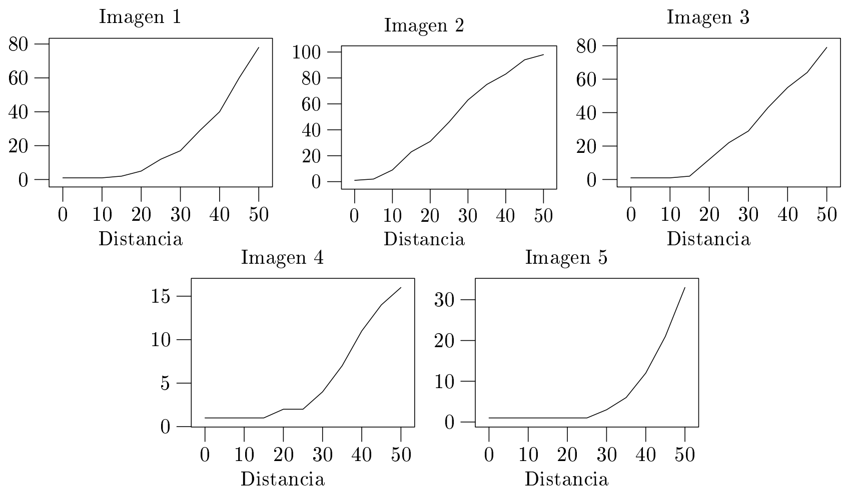

Figura 4.2: Cantidad de imágenes a menor o igual distancia para 5 imágenes

realmente similares son las que se encuentran a una distancia menor de la distancia en la que comienza el primer cambio brusco en la pendiente. En el caso de las 4 imágenes en la figura, las distancias correspondientes serían 20, 10, 15, 25 y 35. En el caso de la imagen 4, el cambio importante en la pendiente comienza en 10, pero no hay imágenes a menor distancia que ésa; por lo tanto, en ese caso consideramos el 25[1] (distancia que nos da como similar una sola imagen).

Para la imagen 1, la imagen a menor distancia no está relacionada. Posiblemente el programa confunda los tramos de la figura en la imagen 1 con las ventanas en la imagen a menor distancia. Además, los contornos claros de la figura en la imagen 1 son adyacentes a la figura, así como la base clara y el techo claros son adyacentes a las ventanas. El valor de θ que estamos considerando hace que el algoritmo no considere todas las regiones que ve un observador humano: quizás la consideración de más regiones permitiera notar al algoritmo que las regiones son diferentes. En la figura 4.3 se muestran la imagen 1, la imagen a menor distancia, y las imágenes coloreadas por regiones (las regiones en negro fueron ignoradas por tener menos de 4 píxeles).

Para la imagen 2 la cantidad de imágenes a distancia menor que 10 (exceptuándose a ella misma, que está a distancia 0) es de 8. Existe una imagen a distancia 4.2 y las demás están a distancia entre 7.3 y 9.3. La imagen a distancia 4.2 es bastante similar a la 2, como puede apreciarse en la figura 4.4, habiendo en ambos casos una figura luminosa en la superficie, con cielo en el horizonte.

[1] Podríamos reformular el criterio esbozado arriba diciendo que nuestro algoritmo funciona correctamente si las imágenes realmente similares son las imágenes que se encuentran a una distancia menor de la distancia en la que comienza el primer cambio brusco de la pendiente en el que la cantidad de imágenes es distinta de 0.

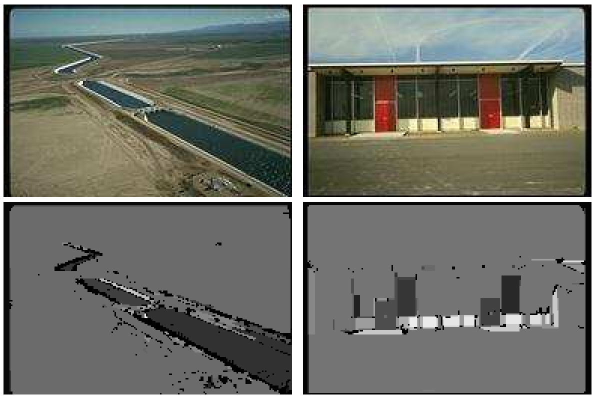

Figura 4.3: La imagen 1 (arriba, izq.), la más cercana a ella (arriba, der.) y ambas coloreadas por regiones (abajo)

Recordemos que nuestro algoritmo no considera la forma de las regiones, por lo cual era de esperar que confundiera regiones rectas con regiones curvas.

Figura 4.4: La imagen 2 (izq.) y la más cercana a ella (der.)

Para la imagen 3 la cantidad de imágenes a distancia menor que 15 es de sólo una imagen, a distancia 10.3. La imagen, como puede verse en la figura 4.5 es una vista del mismo lugar tomada desde otro punto. El resultado en este caso es, entonces, aún más satisfactorio.

También hay sólo una imagen a distancia menor que 25 para la imagen 4, a distancia 17.5. Las imágenes (en la figura 4.6) son de nuevo dos tomas del mismo lugar, por lo cual vemos que el algoritmo funciona correctamente también en esta ocasión.

En el caso de la imagen 5, la imagen a menor distancia está a 26.8. Las imágenes no son similares entre sí. La calificación del programa puede deberse a que en la imagen 5 hay un área de gran tamaño bordeada por un área de menor luminosidad, y que el patrón se repite en la imagen a menor distancia (sólo que aquí es mas angosto). Entre las figuras a distancia menor que 35 se encuentra una bastante similar, a distancia 32.6. Las tres imagénes se muestran

Figura 4.5: La imagen 3 (izq.) y la más cercana a ella (der.)

Figura 4.6: La imagen 4 (izq.) y la más cercana a ella (der.)

en la figura 4.7.

Figura 4.7: Imágenes similares a la imagen 5 (izquierda), imagen similar a menor distancia (centro), imagen similar a distancia pequeña (derecha).

Los resultados con $\theta = 15$, 4-adyacencia y eliminación de las regiones con menos de cuatro píxeles fueron muy promisorios. Un análisis de la lista de imágenes más cercanas a las 199 estudiadas reveló que gran cantidad de imágenes tenían como imagen más cercana a la que tiene la imagen 4. (Véase figura 4.6). La simple inspección de las imágenes mostró que todas eran imágenes del mismo lugar tomadas desde diferentes ángulos y distancias, mostrando que, en la práctica, el algoritmo cumple correctamente con la invariancia ante rotación y escalamiento. Lo mismo ocurrió con otra imagen cuya imagen a menor distancia era la misma que para la imagen 3. Esta imagen también resultó ser una vista del mismo lugar. Llamaremos, por brevedad, parámetros *satisfactorios* a estos parámetros.

Con diferentes parámetros durante la fase de creación del grafo de adyacencia la calidad de la imagen puede mejorarse, a costo de segmentar más regiones (a veces más de lo deseado). En la figura 4.8 pueden verse las imágenes que - procesadas con $\theta = 15$ y eliminando las regiones con menos de 4 píxeles - aparecen en la figura 4.3, procesadas ahora con $\theta = 5$, difuminando la imagen en cuadrados de dos píxeles de lado y eliminando las regiones con menos de diez píxeles. Puede verse que en la misma imagen conviven regiones que aún son confundidas con regiones que están sobresegmentadas. Parece que el aumento en la cantidad de regiones que se consideran perjudica el funcionamiento del algoritmo, ya que para estos parámetros el funcionamiento no resultó satisfactorio. La similaridad por color tampoco resultó mejor que los parámetros satisfactorios.

Diferentes valores de k fueron probados en los distintos experimentos, pero, en general, los resultados fueron similares. También se intentó cambiar el costo de crear o agregar un lado de $(1-k)t_1 + (1-k)t_2 + kc$ a $(1-k)\min\{t_1, t_2\} + kc$ (donde t_1y t_2 son los tamaños de las regiones y c es el cambio de color de una región a otra). Este cambio obedece a que un lado debería ser considerado importante si *ambas* regiones son importantes: si sólo una de ellas lo es, el costo de borrar el lado puede resultar excesivo (esto puede resultar decisivo en el caso de una región grande adyacente a muchas regiones pequeñas, y esta situación es frecuente en la práctica - por ejemplo, en el caso del cielo). Pese a este argumento, los resultados no cambiaron demasiado (hasta puede decirse que fueron peores: los grupos detectados utilizando los parámetros satisfactorios no estaban tan claros luego de haber realizado el cambio).

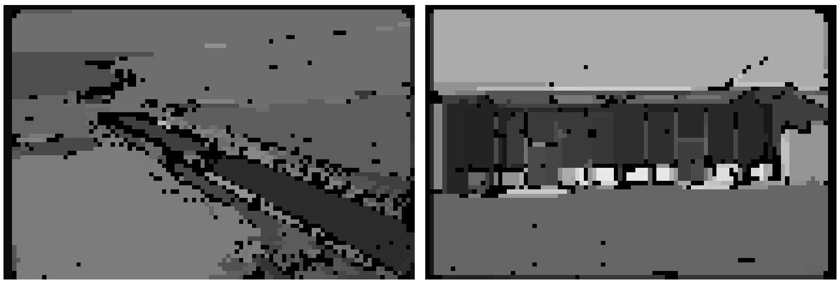

Figura 4.8: Imágenes de la figura 4.3 procesadas con diferentes parámetros

4.3 Una nota sobre la confusión de regiones

Como sabemos, el hecho de que nuestra aproximación trabaje solamente con los lados del algoritmo hace que puedan confundirse regiones. En la figura 4.9 se muestran dos modificaciones de la imagen 4. En estas modificaciones se agregó un cuadrado blanco a la imagen en dos zonas diferentes. Para la primera imagen, el porcentaje de regiones confundidas (comparándola con la imagen original y los parámetros satisfactorios) estuvo cerca del 50% para ambas imágenes (aunque ninguna región fue asignada a más de dos regiones), mientras que para la segunda no hubo regiones confundidas (el algoritmo simplemente borró la región nueva). La comparación entre las dos imágenes modificadas confundió también el 50% de las imágenes en ambas regiones.

Figura 4.9: Modificaciones de la imagen 4

Es digno de mención que, pese a la confusión, las distancias entre las imágenes se mantuvieron todas mucho más pequeñas que la distancia de la imagen a menor distancia en el conjunto de las 199 imágenes (fueron de 7.7,2 y 6.6 respectivamente), indicando que el algoritmo reconoció que las imágenes son muy similares.

Capítulo 5

Conclusiones

El funcionamiento observado pudo explicarse a partir de la medida de similaridad utilizada. Como era de esperarse, el algoritmo confundió regiones con tamaños similares, pero distintas en su forma (la cual no es considerada en nuestro modelo).

El hecho de que la asignación sea por lados y no por regiones dificulta la indagatoria sobre el funcionamiento del algoritmo, ya que es difícil saber cuáles fueron las correspondencias. De todas maneras, pudo obtenerse información a partir de otras medidas como el porcentaje de regiones confundidas o la distribución de las imágenes respecto de la cercanía a otra imagen dada.

La confusión de regiones elevada en imágenes similares se debe a que el algoritmo encuentra la mejor asignación entre lados, aunque ésta no sea la que hace un observador humano. Esto dificultaría el uso del algoritmo para "entender" imágenes, ya que no podría determinar qué región de una imagen corresponde a una región de la otra.

El tiempo de ejecución del algoritmo no es despreciable. La observación de su funcionamiento indica que el algoritmo de matching debe deshacer demasiadas asignaciones durante el transcurso de la ejecución, lo cual aumenta la longitud de los caminos aumentadores, disminuyendo la velocidad de ejecución. Esto puede deberse a que los lados no sean demasiado diferenciables entre sí, y que este aspecto haga que se realicen asignaciones que deben descartarse a futuro. El tiempo de obtención del grafo de adyacencia de regiones es despreciable: quizás una inversión en complejidad en esta etapa podría acelerar la siguiente.

El reconocimiento de imágenes de un mismo lugar tomadas desde diferentes ángulos demuestra que el algoritmo verifica en la práctica tolerancia ante rotación y traslación. Por los resultados obtenidos, el algoritmo parece ser útil en la clasificación de imágenes dentro de grupos.

Bibliografía

[1] Xiaoji Jiang Horst Bunke, Andreas Münger. Combinatorial search versus genetic algorithms: A case study based on the generalized median graph problem. *Pattern Recognition Letters*, 1999.

[2] Gung-Wei Chirn Jason Wang, Kaizhong Zhang. Algorithms for approximate graph matching. *Information Sciences 82*, 1995.

[3] Gabriel Valiente Ricardo Baeza-Yates. An image similarity measure based on graph matching. *0-7695-0746-8/00 IEEE*, 2000.

[4] M. A. Ismail Yasser El-Sonbaty. A new algorithm for subgraph optimal isomorphism. *Pattern Recognition*, 1997.