



Trabajo Práctico 2: Segmentación de imágenes

Algoritmos y estructuras de datos III

Integrante	LU	Correo electrónico
Blufstein, Marcos	300/17	marcos@blufstein.com
Martino, Maximiliano	123/17	maxii.martino@gmail.com
Peretti, Olivia	359/17	operetti@dc.uba.ar
Pironio, Nicolás	37/17	npironio@dc.uba.com

Resumen: se implementa el algoritmo presentado por Felzenszwalb y Huttenlocher para segmentación eficiente de imágenes, buscando una representación de los datos que resulte eficiente. Luego se hace un análisis cualitativo sobre los resultados generados.

Palabras clave: segmentación, estructuras.



Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

<http://www.exactas.uba.ar>

Índice

1. Introducción	2
2. Desarrollo	3
3. Implementación	5
3.1. Ordenamiento de las aristas	5
3.2. Representación de componentes	6
3.2.1. Arreglo de representación	6
3.2.2. Árbol de representantes	6
3.2.3. Árbol de representantes con <i>path compression</i>	7
3.2.4. Algoritmo de segmentación	7
4. Experimentación	9
4.1. Comparación de las estructuras	9
4.1.1. Análisis temporal del algoritmo	10
4.2. Influencia de k sobre la granularidad del resultado	11
4.3. Calidad de la segmentación	12
4.4. Distintas aplicaciones y análisis cualitativo	14
4.4.1. Reconocimiento de transeuntes	14
4.4.2. División de rasgos faciales	15
4.4.3. Identificación de nucleos celulares	15
5. Conclusión y trabajos futuros	17
Referencias	18

1. Introducción

En el presente trabajo se aborda el problema de segmentación de imágenes: dada una imagen buscamos obtener una partición de esta donde cada región delimitada represente un objeto identificable y distinguible. Claramente este no es un problema bien definido, ya que depende de la percepción subjetiva de cada individuo sobre qué es lo que se puede observar en una imagen específica. Debido a esto es común que distintos trabajos busquen dar soluciones a conjuntos de fotos sobre un área determinada, como por ejemplo imágenes médicas para el estudio de tumores en el cerebro, aprovechando características especiales del objeto de estudio.

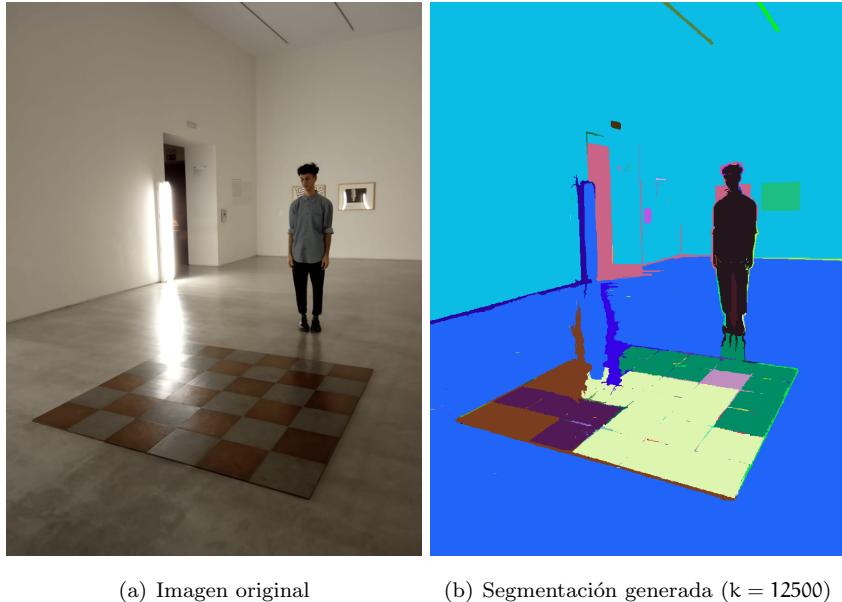


Figura 1: Ejemplo de una segmentación

Se puede observar en la figura 1 que si bien se logran distinguir las partes centrales de la imagen (persona, cuadrados, cuadros, pared, piso) en la imagen original pueden haber muchos factores que influencian de manera negativa en la separación. Un claro ejemplo es la luz de tubo junto a la puerta que impacta notablemente sobre el piso, o la sombra del sujeto afectando sobre los cuadrados. Al mismo tiempo, como se explicara más adelante, buscar una segmentación muy general puede perder detalles que podrían ser interesantes, como las distinciones entre cuadrados dentro del cuadrado principal en el piso.

Para la resolución del problema, el trabajo se basa en el método expuesto por Pedro F. Felzenszwalb y Daniel P. Huttenlocher [1] que, como describiremos en las secciones siguientes, proponen un algoritmo eficiente basado en grafos para conseguir una segmentación “ni muy fina ni muy gruesa” en su terminología. A lo largo del trabajo se estudia este método en profundidad, considerando las distintas implementaciones posibles, midiendo la eficiencia de estas e intentando responder a la pregunta de si el predicado propuesto por Felzenszwalb y Huttenlocher verdaderamente se refleja en los resultados obtenidos.

2. Desarrollo

A continuación presentaremos las ideas generales e intuiciones que se desarrollan en [1] y discutiremos sobre las implementaciones posibles.

Como se mencionó, el método se basa en modelar el problema con un grafo. Este se deduce de la imagen de la siguiente manera: $G = (V, E)$ donde

V : píxeles de la imagen.

$(v_i, v_j) \in E$: representa la disimilitud entre el pixel v_i y v_j .

De esta forma podemos definir que una segmentación de la imagen es tomar un *subgrafo inducido* de G o dicho de otra forma una segmentación S es una partición en componentes conexas de V . Esta definición se justifica con la idea que los vértices de una componente deben ser similares entre sí, y no así con los de una componente distinta. Esto se termina traduciendo en que los ejes de vértices en la misma componente deben tener un peso “chico” mientras que entre vértices de componentes distintas deben tener un peso “grande”.

A partir de este modelo se construye un predicado que busca responder a la pregunta de si hay evidencia de que existe un límite entre dos componentes distintas. Este se basa en comparar la diferencia entre dos vértices vecinos de dos componentes distintas, teniendo en cuenta la diferencia inter-componente de cada uno para reflejar el valor local de cada componente.

Se trabajan con dos conceptos para poder definir este predicado. Por una parte se define para una componente C su *internal difference* (diferencia interna) $\text{Int}(C)$ como el mayor de los pesos de las aristas presentes en el *árbol generador mínimo* (AGM) de C . La intuición que refleja esta medida es que la componente admite que los píxeles tengan a lo sumo una disimilaridad de $\text{Int}(C)$.

Luego para componentes C_1, C_2 definimos la diferencia entre ellos $\text{Diff}(C_1, C_2)$ como la arista de menor peso entre $v_1 \in C_1$ y $v_2 \in C_2$. En un principio esta última no refleja substancialmente la relación entre dos componentes, ya que tiene en cuenta una única “conexión”, pero se observa empíricamente que los resultados obtenidos son razonables y que de esta medida se desprende un algoritmo considerablemente eficiente.

Dichas estas definiciones se explica el predicado D como:

$$D(C_1, C_2) = \begin{cases} \text{True} & \text{si } \text{Diff}(C_1, C_2) > M\text{Int}(C_1, C_2) \\ \text{False} & \text{Caso contrario} \end{cases}$$

con $M\text{Int}(C_1, C_2) = \text{Min}(\text{Int}(C_1) + \tau(C_1), \text{Int}(C_2) + \tau(C_2))$ con τ una función para regular el grado en el cual la diferencia entre componentes tiene que ser mayor que la diferencia interna para afirmar que existe un límite entre componentes. Intuitivamente si solo se tomase el mínimo entre diferencias internas no reflejaría la mayor parte de la información interna de la componente. Con esto en mente, podemos tomar $\tau(C) = \frac{k}{\#C}$ con k un hiperparámetro. De esta forma la elección de k influye sobre cuándo se decide que hay evidencia para definir que hay un límite entre componentes, siendo que un k mayor es propenso a resultar en delimitar menos componentes y un k más chico implicaría que para afirmar que hay un límite para una componente pequeña se requiere más evidencia.

Considerando estas nociones se desprende el siguiente algoritmo para computar la segmentación:

1. Ordenar no decrecientemente las aristas: $E = (e_1 \dots e_m)$.
2. Empezar con la partición S_0 con cada vértice en una componente separada.
3. Para $i = 0 \dots m$ hacer 4.

4. Sea C_1 la componente de la cola de e_i y C_2 la componente de la cabeza de e_i . Si $C_1 \neq C_2$ y $\text{peso}(e_i) < M\text{Int}(C_1, C_2)$ entonces S_i se construye uniendo las componentes C_1 y C_2 de S_{i-1} .
5. Devolver $S = S_m$.

Notar que el algoritmo que se obtiene de las definiciones es en esencia el algoritmo de Kruskal para obtener un AGM. Recordemos que el invariante de Kruskal es mantener un bosque generador mínimo y en cada iteración agregar una arista segura y candidata. La diferencia que podemos encontrar en el algoritmo de segmentación propuesto es lo que definimos como arista candidata: en nuestro caso una arista es candidata cuando es la mínima que une dos componentes distintos y además su peso no supera $M\text{Int}$. Esto claramente no tiene porqué resultar en un AGM pero sí en un bosque generador mínimo donde sus componentes sí son AGM, dándole una nueva interpretación a lo previamente desarrollado.

Si bien no es el enfoque de este trabajo demostrar las propiedades que posee el algoritmo si vamos a mencionarlas:

Se define que una segmentación es *muy fina* si existen regiones C_1, C_2 para las cuales no existe evidencia de que haya un límite entre ellas según el predicado D .

Se dice que T es un *refinamiento propio* de una segmentación S si $\forall C' \in T, \exists C \in S$ tal que $C' \subsetneq C$

Una segmentación S es *muy gruesa* si existe un refinamiento propio de S que no es muy fino.

Con estas nociones establecidas se prueba que el algoritmo propuesto provee una segmentación que no es ni muy fina ni muy gruesa.

3. Implementación

En esta sección discutiremos las particularidades de cómo se llevó a cabo la implementación del método propuesto en [1]. Lo primero a mencionar es que para segmentar una imagen el primer proceso es obtener una versión en escala de grises de esta donde ademas se le aplica un filtro Gaussiano con parametro $\sigma = 0,8$. Esto último se hace con el fin de disminuir el ruido de *artifacts* que pueden llegar a estar presentes en la imagen digital.

Como se mencionó en la sección anterior el algoritmo requiere intrinsecamente dos funcionalidades:

- Ordenar las aristas de forma no decreciente.
- Poder determinar si dos vértices pertenecen a la misma componente y en caso de ser necesario juntarlas.

Como estas dos partes no influyen sobre la otra las analizaremos por separado y tendremos como definición general que $\#V(G) = n$ = “Cantidad de píxeles en la imagen”.

3.1. Ordenamiento de las aristas

Lo primero que hay que definir es qué conjunto de aristas se toman, ya que la definición del conjunto E en la sección anterior admite tomar cualquier subconjunto de aristas posibles. En lo desarrollado en [1] (y en este trabajo) se considera que $(v_i, v_j) \in E$ si y solo si v_j está entre los 8 píxeles más cercanos de v_i , así $\#E = m < 8n = O(n)$.

Además se utilizó como estructura de representación de grafos una *lista de aristas*: guardando en una lista doblemente enlazada las aristas en forma de triples (peso, cola, cabeza) y el valor n de la cantidad de vértices totales. Luego se plantea el siguiente procedimiento para obtener el grafo G de la imagen a segmentar:

Algorithm 1 Imagen a lista de aristas ordenadas por peso

```
1: procedure IMG2SORTEDGRAPH(Img)
2:    $G \leftarrow$  nuevoGrafo( $\{\}$ , Img.height * Img.width)
3:   aristas  $\leftarrow$  vector(lista(arista)) (256,  $\{\}$ )
4:   Para pix en Img:
5:     Para vec en 8masCercanos(pix), si  $(vec, pix) \notin E$ :
6:       nueva_arista  $\leftarrow$  (pix - vec, pix, vec)
7:       insertar nueva_arista en la lista de aristas[pix-vec]
8:   G.aristas  $\leftarrow$  juntarListasEnOrden(aristas)
9: end procedure
```

Este procedimiento genera las aristas según el criterio planteado y las guarda de forma ordenada con una complejidad $O(n)$.

Respecto a la correctitud generamos las aristas sin repetirlas y las guardamos en el vector **aristas** en función de su peso. Luego encadenando las listas de la posición 0 a la 255 del vector obteniendo una única lista en donde los pesos se encuentran ordenados de manera no decreciente. En esencia: se implementó como un *counting sort*. Para deducir la complejidad podemos ver que las líneas 1 y 2 toman tiempo constante, luego el ciclo principal revisa una cantidad de $8n$ píxeles y sobre ellos realiza operaciones de tiempo constante. Por último encadenar las listas en la línea 8 consiste en reencadenar 256 punteros, tomando tiempo constante para cada uno de ellos. Por lo tanto la complejidad del procedimiento resulta $O(n)$.

3.2. Representación de componentes

En el algoritmo propuesto en la sección desarrollo se vio que era necesario poder mantener en simultáneo para cada píxel a qué componente pertenecía, saber el tamaño de una componente y eventualmente unir dos componentes. Notar que como se mencionó previamente la segmentación es una partición de V y por lo tanto cada píxel pertenece a un solo conjunto dentro de la partición. A raíz de esto se puede modelar este problema como mantener la partición en conjuntos disjuntos entre sí. A esta estructura se la llama *Disjoint set* y provee la siguiente interfaz:

- `crear(n)`: crea n conjuntos disjuntos.
- `find(i)`: devuelve el identificador del conjunto al que pertenece el elemento i .
- `unite(i,j)`: junta los conjuntos a los que pertenecen i y j .
- `tam(i)`: devuelve el cardinal del conjunto al que pertenece el elemento i .

A continuación discutiremos las posibles implementaciones de este tipo de datos. Todas las implementaciones tienen en común que se utiliza un arreglo `tams` en su representación interna tal que `tams[i]` = “tamaño del conjunto identificable por i ” para i un identificador de conjunto. Además todas las implementaciones utilizan la heurística de *union by rank* que consiste en unir el conjunto de menor cardinal al de mayor cardinal al ejecutar el método `unite`.

3.2.1. Arreglo de representación

La estructura interna que utiliza es de un arreglo de n posiciones donde $\text{arr}[i] = \text{Id}_i$ con Id_i el identificador del conjunto al que pertenece el elemento i . Con esta representación la interfaz se implementa de la siguiente manera:

- `crear(n)`: inicializar un arreglo `arr` de n posiciones tal que `arr[i] = i` $\rightarrow \mathcal{O}(n)$.
- `find(i)`: devolver `arr[i]` $\rightarrow \mathcal{O}(1)$.
- `unite(i,j)`: para toda posición k que cumple `arr[k] = Idi` cambiar por `arr[k] = j`, actualizar `tams[Idj] += tams[Idi]` $\rightarrow \Theta(n)$.

3.2.2. Árbol de representantes

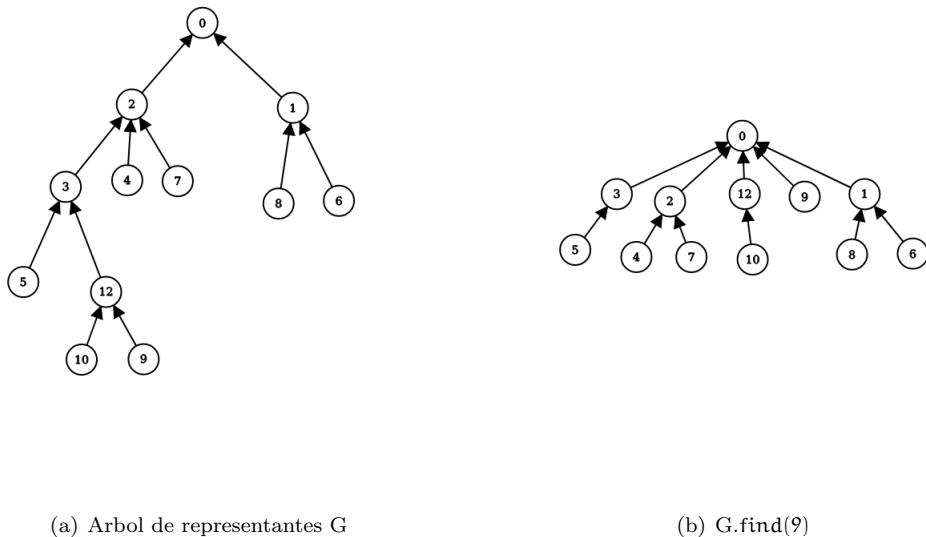
De forma parecida a la estructura anterior se utiliza un arreglo de n posiciones pero con la diferencia que `arr[i]` ahora representa una *relación de padre*. Esto nos define un árbol para cada conjunto y de esta forma podemos identificar al conjunto por la *raíz* del arbol.

- `crear(n)`: inicializar un arreglo `arr` de n posiciones tal que `arr[i] = i` $\rightarrow \mathcal{O}(n)$.
- `find(i)`: Si `find[i] ≠ i`: devolver `find[arr[i]]` sino devolver $i \rightarrow \mathcal{O}(n)$.
- `unite(i,j)`: Obtener los representantes de i y de j , `arr[Idi] = Idj`, actualizar `tams[Idj] += tams[Idi]` $\rightarrow \mathcal{O}(n)$.

Notar que en peor caso esta estructura es teóricamente menos eficiente, ya que ahora obtener el identificador de un conjunto puede costar a lo sumo n pasos.

3.2.3. Árbol de representantes con *path compression*

Su estructura interna es igual a la del *arbol de representantes*, pero realiza la operación de *find* con una diferencia: al buscar el identificador en la estructura anterior era necesario “subir” por la cadena de padres de un elemento para encontrar la raíz. A su vez esta raíz también es el identificador de todos los padres por los que se pasó para responder a la query de *find*, por lo tanto técnicamente al buscar al identificador de un elemento efectivamente se está resolviendo la query de buscar el identificador para toda la cadena de padres. Por lo tanto podemos aprovechar esto para cambiar el padre de todos estos elementos directamente por la raíz.



Con esta representación Tarjan [2] probó que la operación *find* (y por lo tanto *unite*) toman tiempo $\mathcal{O}(\alpha(n))$ amortizado, con $\alpha(n)$ la función inversa de Ackerman. En la práctica esto significa que el tiempo que toma hacer n operaciones de *find* es de $\mathcal{O}(n)$ ya que la inversa de Ackerman tiene un ritmo de crecimiento considerablemente chico.

3.2.4. Algoritmo de segmentación

Habiendo discutido las posibles implementaciones para el tipo de datos en lo siguiente se muestra un pseudocódigo para el algoritmo.

Algorithm 2 Segmentación de imagen

```

1: procedure SEGMENTAR(G)                                ▷ G grafo con representación de lista de aristas ordenadas
2:   componentes ← crear(G.tam)
3:   internal_diffs ← [0 ... 0]
4:   for arista a en G.ejes do
5:     c1 ← find(a.cola), c2 ← find(a.cabeza)
6:     if c1 ≠ c2 y a.peso < MInt(c1, c2) then
7:       unite(c1, c2)
8:       actualizar internal_diffs
9:     end if
10:   end for
11:   construirImagen(componentes)
12: end procedure
  
```

Teniendo esta versión más cercana a la implementación real se puede hacer un análisis del costo teórico que tomará el algoritmo para las distintas representaciones. Las dos primeras líneas no tienen ninguna diferencia para las distintas versiones y toman $\mathcal{O}(n)$ de la misma forma la linea 6 toma tiempo constante para todas las opciones, separemos en casos para el resto:

- Arreglo de representantes: La línea 5 toma $\mathcal{O}(n)$ para cada *find*, luego la línea 7 toma $\Theta(n)$. Como esto se hace para $\mathcal{O}(n)$ aristas $\rightarrow T(n) = \mathcal{O}(n^2)$.
- Árbol de representantes: La línea 5 toma $\mathcal{O}(n)$ para cada *find*, pero una vez hecho esto los *unite* de la línea 7 llamados con los representantes toman $\mathcal{O}(1)$. Para las $\mathcal{O}(n)$ aristas $\rightarrow T(n) = \mathcal{O}(n^2)$. Notar que de todas formas esta cota es muy gruesa, dado que las componentes se esperan que sean de un tamaño mucho menor a la cantidad de píxeles totales de la imagen, los *find* no deberían ser siempre de valores muy grandes.
- Árbol con *path relinking*: Como se realizan $\mathcal{O}(n)$ *finds* y *unite* (con los representantes) $\rightarrow T(n) = \mathcal{O}(n)$

Por último, y compartido para todas las estructuras, es necesario reconstruir la imagen segmentada a partir de cada píxel. Hacer esto toma tiempo lineal sobre n dado que hay que recorrer cada píxel para saber a qué componente pertenece.

4. Experimentación

A continuación analizaremos los distintos aspectos de la resolución implementada para la segmentación. Debido a las características del problema, la discusión se centrará en dos ejes principales: como esta es una implementación particular debemos considerar su eficiencia y las propiedades que posee midiendo las métricas correspondientes. Por otro lado, como se presentó en la introducción, este problema carece de una respuesta objetiva y absoluta, por lo tanto se buscará dar cierto parámetro del objetivo buscado y responder si el algoritmo lo satisface.

Específicamente los experimentos realizados consisten en comparar el efecto que tienen las distintas estructuras planteadas en la sección de desarrollo y justificar la complejidad temporal propuesta. Luego analizaremos si la función de *threshold* planteada en [1] cumple la propiedad propuesta. Hechas estas consideraciones buscaremos dar una respuesta a cuál es la calidad de la segmentación utilizando un *dataset* con su correspondiente *groundtruth*. Por último veremos cómo se comporta el algoritmo frente a distintas posibles aplicaciones y se intentara determinar si es apto para alguna de ellas.

4.1. Comparación de las estructuras

Como se explicó en la sección de implementación, para realizar ciertas operaciones en el algoritmo2, es necesario utilizar un tipo abstracto de datos *Disjoint Set*. Se mostró que este podía ser implementado de al menos tres formas distintas y se justificaron las complejidades. Ahora vamos a evaluar cuál es el rendimiento del algoritmo utilizando las distintas estructuras posibles.

Para este experimento se tomó una imagen original de dimensiones de 1240×960 píxeles a la cual se la redimensionó a distintos tamaños. Luego se ejecutó el algoritmo para cada imagen y se midió el tiempo en realizar únicamente la segmentación, ya que se consideró que la lectura de la imagen y la generación de las aristas era constante para las tres versiones. Además para este experimento se considera que tanto el contenido de la imagen original como el valor de k es irrelevante al tiempo que toma la ejecución.

Basado en lo discutido sobre el análisis teórico del costo, se esperó que la estructura que tarde más tiempo sea el arreglo de representantes, seguido del arbol de representantes y luego la versión con *path relinking*. Esto se justifica en el hecho de que los *unite* del arreglo tardan estrictamente n pasos, mientras que en este contexto para los arboles de representantes esta operación se realiza en tiempo constante. Además el uso de *path relinking* tiene un costo teórico menor al de las otras dos opciones, por lo tanto explicaría que sea la versión optima respecto al resto.

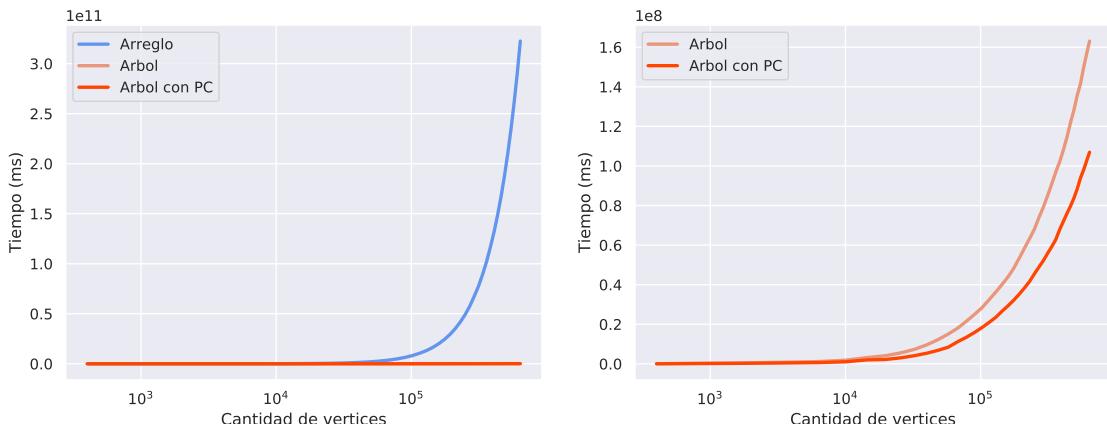


Figura 2: Comparación de tiempos para las distintas estructuras. La cantidad de vértices es de $(20p)^2$ con $p \in [1, 40]$ y $k = 1000$.

Efectivamente se cumplió experimentalmente lo que se esperaba en base al costo teórico. Podemos ver como el ritmo que conlleva utilizar el arreglo de representantes supera vastamente al de los árboles, debido a esto es necesario visualizar por separado el rendimiento de las estructuras de árboles. En el segundo gráfico se ve como para instancias de tamaño mayor sí aparece haber una notable diferencia entre las opciones, sin embargo no parecería que utilizar árbol de representación tenga un ritmo de crecimiento muy distinto a la versión con *path relinking*, sino más bien que hay una diferencia en la constante de multiplicación. Un experimento posible que no se realizó es comprobar si en la práctica y para esta aplicación usar un árbol de representantes implica un costo lineal del algoritmo. Esto último no se comprobó ya que, aunque llegue a ser cierto, sigue siendo conveniente utilizar *path relinking*.

4.1.1. Análisis temporal del algoritmo

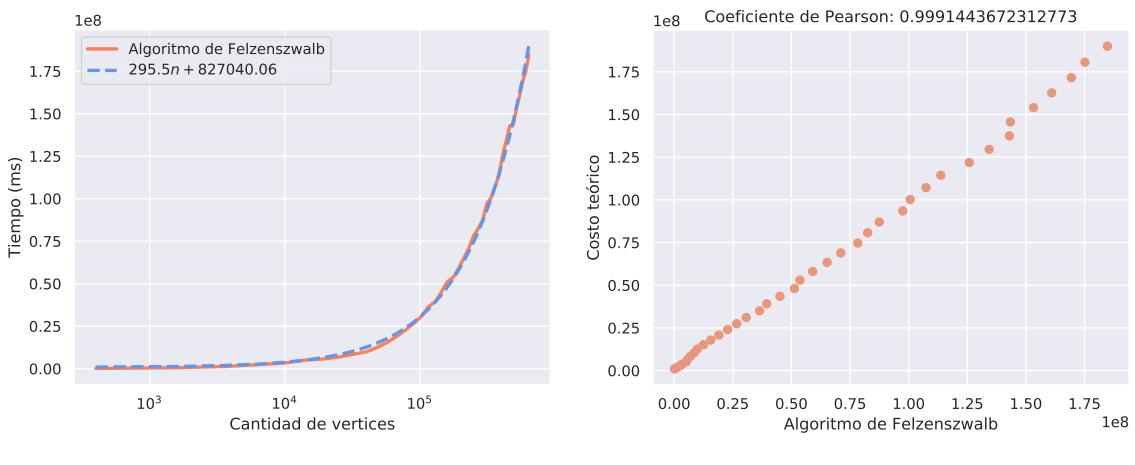
Habiendo hecho la comparación en el experimento anterior y habiendo decidido que utilizar *path relinking* era ventajoso, se propuso medir el tiempo de la totalidad del procedimiento esta vez teniendo en cuenta el costo de conseguir el grafo a partir de la imagen. Esto resulta de interés ya que al fin y al cabo el procedimiento en su totalidad es lo que verdaderamente se quiere saber si es eficiente o no. Para esto se tomó el mismo conjunto de imágenes propuesto para el experimento anterior y el mismo valor de k . El objetivo es mostrar empíricamente que el costo total de todo el algoritmo es de $\mathcal{O}(n)$.

Con fines de justificar esta afirmación se utilizó la teoría de *cuadrados mínimos lineales*:

Poseemos una muestra de valores de tiempo en función del tamaño de entrada $(n_1, t_1) \dots (n_k, t_k)$ y queremos aproximarlos con una función $f \in F$ familia de funciones, de forma que f minimice la expresión:

$$\sum_{i=1}^k (f(n_i) - t_i)^2$$

Esta teoría nos es útil pues podemos tomar la familia de funciones $F : an + b$. Luego la teoría nos asegura que siempre podemos determinar los coeficientes a y b que minimizan la expresión. Esto nos es ventajoso pues de esta forma encontramos una función que pertenece al supuesto orden del peor caso de nuestro algoritmo y que además minimiza el criterio de cuadrados mínimos. Luego de haber conseguido estos coeficientes (y por lo tanto la función) podemos utilizar la *fórmula de Pearson* sobre las muestras $(n_1, t_1) \dots (n_k, t_k)$ y $(n_1, f(t_1)) \dots (n_k, f(t_k))$ y así obtener una métrica sobre el grado de correlación entre los datos y la función.



(a) Comparación de tiempo de ejecución y costo teórico

(b) Correlación entre el algoritmo y el costo teórico

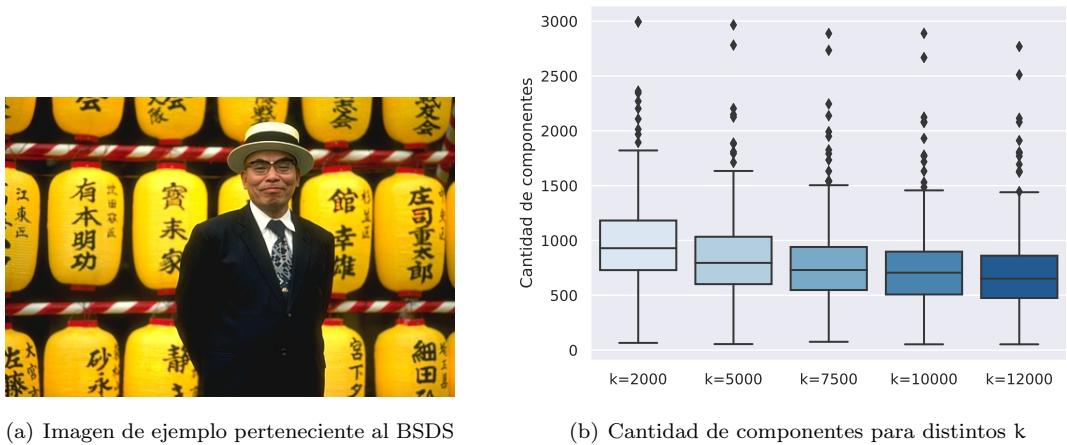
Figura 3

Como era esperado los datos obtenidos soportan la complejidad teórica propuesta, dejando en claro que el algoritmo implementado es eficiente para el problema a resolver.

4.2. Influencia de k sobre la granularidad del resultado

En el trabajo [1] se toma la función de *threshold* $\tau(C) = \frac{k}{\#C}$ con k un hiperparámetro. Intuitivamente podemos ver que un k menor da lugar a componentes más chicas y análogamente un k mayor tiende a unir componentes, ya que afecta directamente sobre el predicado que da evidencia si hay un límite entre componentes. El objetivo del siguiente experimento es buscar una correlación empírica entre esta noción y las segmentaciones resultantes.

Para poder dar una afirmación estadísticamente robusta se utilizó el *Berkeley Segmentation Dataset* (BSDS) [3] que consiste en 300 imágenes variadas. Luego se computó la segmentación para cada valor de k y para cada imagen, midiéndose la cantidad de componentes distintas.



(a) Imagen de ejemplo perteneciente al BSDS

(b) Cantidad de componentes para distintos k

Figura 4

Como podemos observar en la figura 4 (b), a medida que se aumenta el k la media de componentes disminuye. Además las relaciones de los cuartiles no son afectadas, lo que lleva a pensar que para todas las imágenes la influencia del valor de k afecta de manera similar.

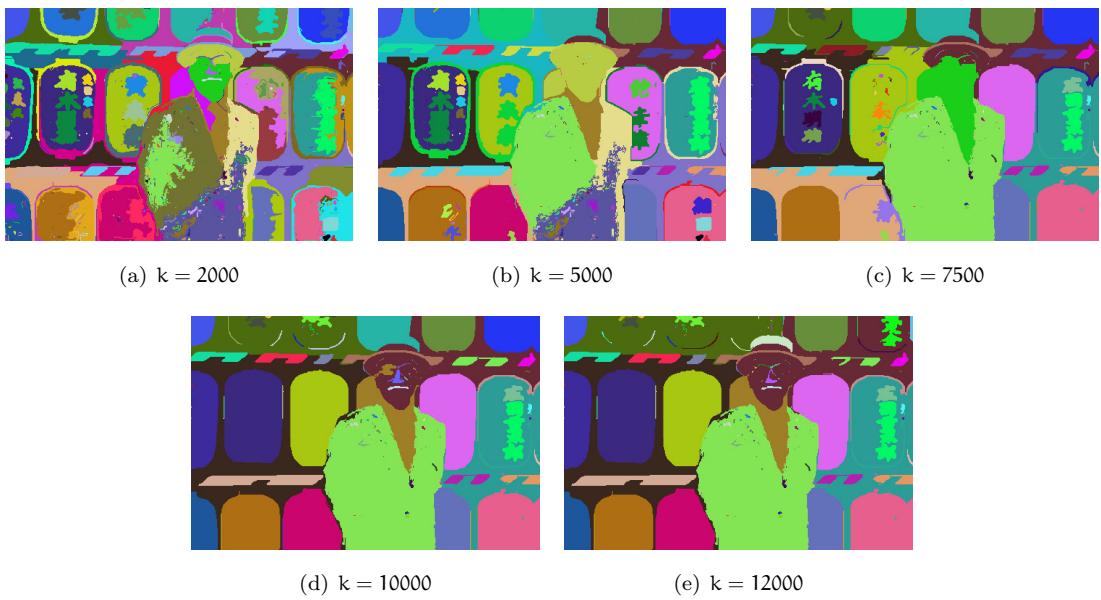


Figura 5

En la foto usada de ejemplo podemos ver efectivamente el efecto de incrementar al hiperparámetro. En un principio los rasgos faciales son más fáciles de distinguir y los kanjis en las linternas no fueron todavía

considerados como el interior del objeto. Al mismo tiempo se ve mucha variabilidad en zonas que se esperarían uniformes como el saco. A medida que se incrementa el valor vemos como se alisan ciertas secciones, ganando mayor definición y separación pero algunos detalles como los ideogramas se pierden. Ya para los últimos casos los contornos generales quedan mayormente separados con muy pocos detalles pequeños distinguibles. Notar que estas últimas generalizaciones a componentes más grandes puede traer uniones no deseadas, como es el caso del cuello de la camisa en este ejemplo: para $k = 5000$ hay una distinción con la lámpara que interseca con la camisa, pero para todos los valores de k siguientes estas dos regiones se unen.

Este experimento confirma la intuición original sobre los efectos del hiperparámetro k y se tendrán en cuenta para la experimentación siguiente.

4.3. Calidad de la segmentación

Si bien el objetivo de resolver el problema es dar una segmentación que refleje los objetos de interés en una imagen, como ya se mencionó, esto es completamente subjetivo. Principalmente hay dos partes fundamentales a esta subjetividad, una siendo que es lo que se quiere estudiar de un conjunto de imágenes (cuál es el objeto de interés) y la otra respecta al observador de la imagen segmentada (quien aprueba el resultado). Aunque este sea el caso no quita que sea de interés poder dar una respuesta a la pregunta de cuán buena es una segmentación.

Con este objetivo se consideró utilizar un *data set* que posea la *ground truth* de las segmentaciones: segmentaciones fabricadas por humanos que se consideraran como la respuesta objetivamente correcta al problema. Se utilizó un subconjunto del BSDS presentado en [4] en el cual se propone una construcción del *ground truth* basada en niveles de percepción de la imagen y consideran que su objeto de estudio son las figuras contrastantes con los distintos fondos de la imagen original. A grandes rasgos consideran que la segmentación debe ser gruesa y bien definida.

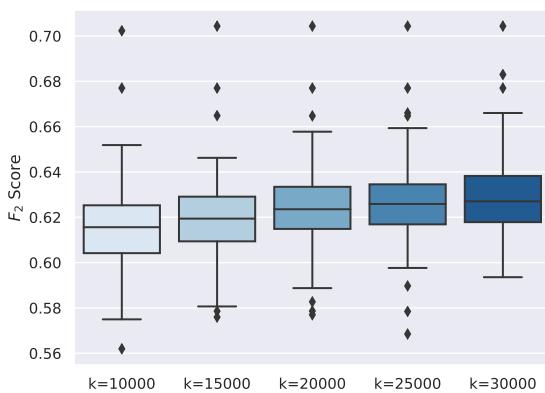
A partir de este *data set* se implementó una comparación entre las imágenes *ground truth* y la segmentación obtenida. La métrica considerada fue el F_2 score:

$$F_2 = (1 + 2^2) \frac{\text{Precision} \times \text{Recall}}{2^2 \times \text{Precision} + \text{Recall}}$$

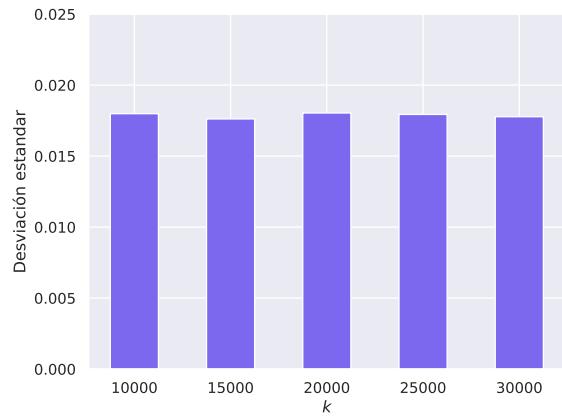
En este contexto *Precision* significa saber de todos los píxeles que la segmentación clasificó en la misma componente, cuantos verdaderamente lo estaban. *Recall* representa cuántas asociaciones correctas se hicieron. Por el problema a resolver y las consideraciones hechas en [1] se consideró darle mayor importancia al valor de *Recall* para reforzar el peso de los *falsos negativos*: poner un límite entre dos componentes cuando no lo hay. Con esto se justifica usar $\beta = 2$ para el F_β score.

Para cada píxel P_i se consideró su *k-ring* de píxeles P_j , comparando para cada par si pertenecían o no a la misma componente en la segmentación obtenida y en la *ground truth*. Con esto se obtienen los valores de *true positives*, *false positives*, *true negatives* y *false negatives*. En la experimentación se consideró el *5-ring*, con la justificación que este radio proporcionaría suficiente información local (contexto del píxel i).

Como las componentes que se generan son de gran tamaño, hacer el procedimiento para todo pixel desbalancearía la cantidad de *true positives*. Esto se debe a que se tendrían en cuenta grandes porciones lisas de la imagen y estas no proporcionan gran información sobre los puntos de interés (los límites). Por lo tanto se tuvo en cuenta únicamente los píxeles “borde” de la segmentación obtenida. Esto mide mejor lo propuesto ya que si un píxel es de “borde” en la imagen obtenida pero no lo es en la *ground truth* aumentarán los *false positives* que es justamente lo que se quiere medir.



(a) F_2 scores para el data set

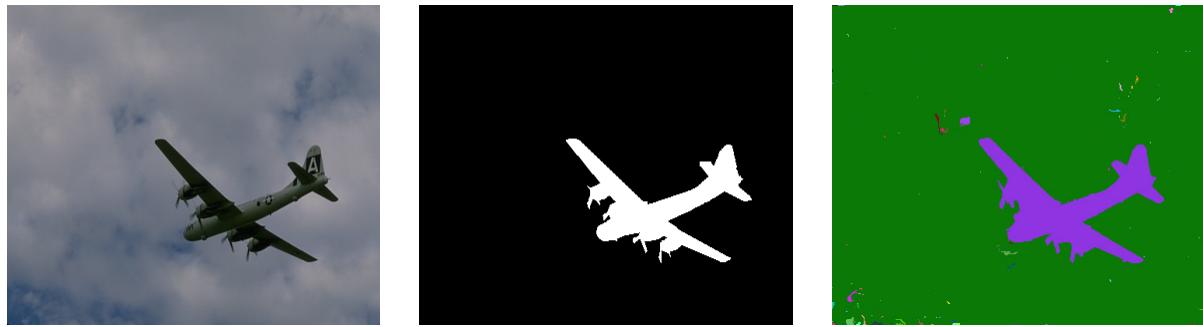


(b) Desviación estandar de la métrica

Figura 6

En primer lugar se observa que si bien hay una mejora en la calidad de segmentación a medida que se toma un k mayor, este incremento es relativamente pequeño ya que las medias de $k = 10000$ y $k = 30000$ difieren en no más de 0,04. A su vez, la variabilidad que presenta el data set es muy pequeña, como se puede observar en el gráfico de barras. Intuitivamente esto sugiere que la segmentación funciona con la misma calidad sin importar la imagen sobre la que se utilize.

Se puede observar que para todos los valores de k utilizados hay un *outlier* con un valor de aproximadamente 0,7, se comprobó que este siempre se corresponde a la misma imagen que se expone a continuación:



(a) Imagen original

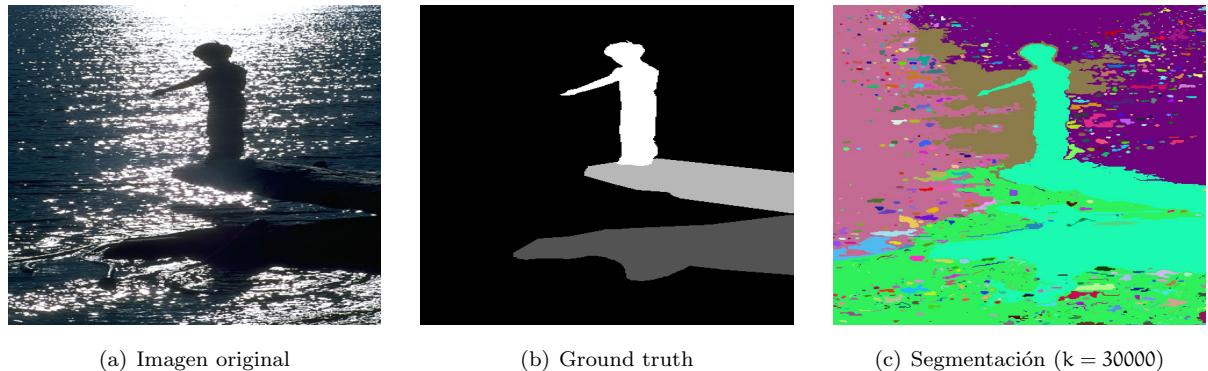
(b) Ground truth

(c) Segmentación ($k = 30000$)

Figura 7

No es sorprendente que una imagen de esta índole, cuyos contornos son fáciles de distinguir y con iluminación uniforme, sea la que produzca un mejor resultado. De todas formas el algoritmo sigue siendo susceptible a ciertos patrones como el cambio entre cielo y nubes, que le agregan ruido indeseado. Queda como trabajo futuro analizar si con esta implementación es posible modificar la función τ para evitar este comportamiento. Notar que si no se tomase como restricción considerar únicamente los píxeles “borde” imágenes como esta tendrían valores de $Precision \approx 1$ debido a la gran componente del fondo.

A su vez podemos observar la imagen que para la mayoría de los valores de k obtuvo el peor valor de F_2



(a) Imagen original

(b) Ground truth

(c) Segmentación ($k = 30000$)

Figura 8

Esta presenta muchos problemas para nuestro algoritmo por la alta variabilidad de colores que genera la luz. Es argumentable que el objeto de interés de esta imagen es el sujeto erguido, y en segundo plano las rocas y el agua. El algoritmo juntó al sujeto y las rocas mientras que el agua la separó basado en la iluminación que esta recibiendo. Se podría decir que para este tipo de fotos la segmentación no es la esperada.

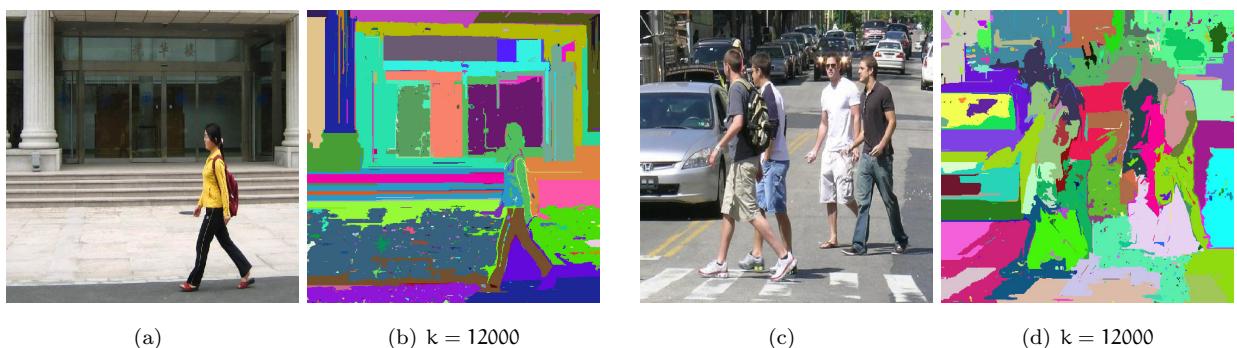
4.4. Distintas aplicaciones y análisis cualitativo

Si bien el algoritmo es correcto respecto a su especificación y cumple las propiedades propuestas, el objetivo de la segmentación es poder diferenciar un objeto de estudio para un conjunto de imágenes que cumplen alguna particularidad. A continuación se tuvieron en cuenta distintos problemas que se pueden resolver utilizando segmentación de imágenes y se evaluó si el algoritmo implementado pudiera ser de utilidad para estos casos.

4.4.1. Reconocimiento de transeúntes

Una de las nuevas tecnologías en desarrollo es la de automóviles autónomos. Uno de los desafíos que presenta es el reconocimiento de sujetos en la calle, a fin de evitar accidentes. En parte este problema se podría resolver utilizando una segmentación en tiempo real, junto con un clasificador del objeto segmentado.

Tomando imágenes de la *Penn-Fudan Database for Pedestrian Detection and Segmentation*¹ podemos evaluar el comportamiento del algoritmo sobre instancias reales.



(a)

(b) $k = 12000$

(c)

(d) $k = 12000$

Figura 9

Evidentemente el comportamiento no es el requerido, mucha información de los fondos afecta negativamente sobre el objeto de estudio y no es distinguible fácilmente. Se podría considerar tomar un valor de k mayor,

¹https://www.cis.upenn.edu/~jshi/ped_html/#pub1

pero esto tampoco funcionaría ya que en la subfigura (b) ya hay componentes que son la junta de dos partes distintas, como es el caso de la mochila y las escaleras del fondo. Mismo en la subfigura (d) la mayor parte de las cosas es irreconocible en la segmentación.

4.4.2. División de rasgos faciales

En varias aplicaciones puede ser de interés reconocer rasgos faciales de un individuo, como por ejemplo en filtros de imágenes hasta reconocimiento de gente desaparecida. Tomando algunas fotos de un *dataset*² podemos analizar el comportamiento de las segmentaciones.

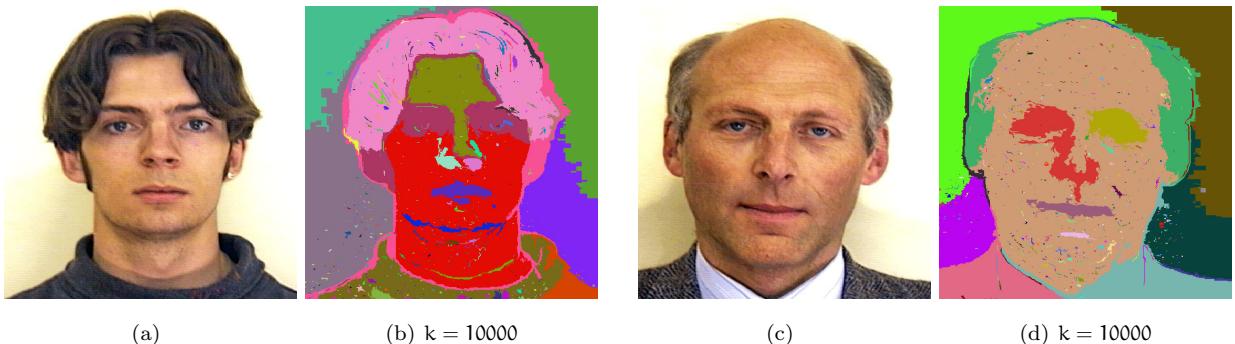


Figura 10

En comparación' al problema de reconocimiento de transeúntes, el algoritmo parece funcionar mejor para esta tarea. Vemos como ciertos aspectos como el pelo de las personas se encuentra bien delimitado, las bocas se pueden distinguir aunque con algo de ruido y partes como los ojos quedan sumidos en otras componentes. Si bien no parecería ser aplicable al problema propuesto, el rendimiento parece mejorar para este problema que se podría decir más simple.

4.4.3. Identificación de núcleos celulares

Uno de los ambientes en donde más se trabaja con procesamiento de imágenes digitales es en el área de la medicina. Reconocimiento de tejidos, tumores y separación de células de imágenes microscópicas son algunas de las tareas que se resuelven al día de hoy, aunque generalmente con un acercamiento desde lo que se conoce como *machine learning*.

De un *dataset* compuesto de imágenes microscópicas de células³, nos interesa ver si el algoritmo implementado es capaz de separar los límites entre las células y sus núcleos. Esta tarea requiere de una precisión mayor a las anteriores debido a que no hay una sola aparición del objeto de estudio en las imágenes, y con esto en mente se tomaron valores de k mucho más chicos que los que se utilizaron para el resto de las experimentaciones.

²http://pics.psych.stir.ac.uk/2D_face_sets.htm

³<https://nucleisegmentationbenchmark.weebly.com/dataset.html>

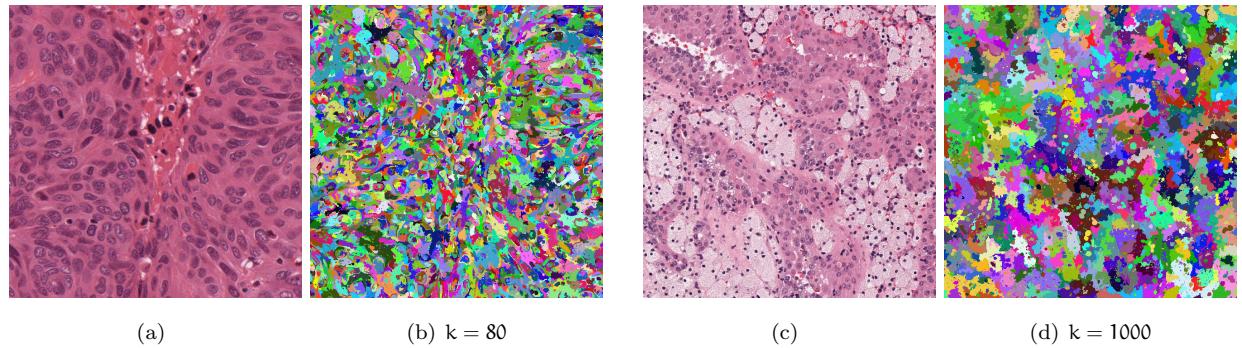


Figura 11

Luego de ver las segmentaciones generadas, no hay duda que esta no es una aplicación viable para el algoritmo. Ambos resultados son incomparables con las imágenes originales, haciendo imposible cualquier reconocimiento de secciones relevantes. En definitiva: el algoritmo no es apto para resolver problemas que requieran un alto grado de detalle.

5. Conclusión y trabajos futuros

A lo largo del trabajo se observó que el método propuesto es considerablemente eficiente en lo que respecta al tiempo de cómputo, pero no se pudo encontrar una aplicación específica en la que su desempeño como separador de objetos resulte útil. De todas formas se pudo comprobar experimentalmente que lo propuesto en [1] se cumplía y que la métrica propuesta en este trabajo reflejaba suficientemente bien lo planteado en [4].

Algunas de las cosas sobre las que se podría trabajar a futuro son considerar una nueva función de *threshold* que proporcione mejores resultados en cuanto a la segmentación y evaluar si el tiempo de cómputo resultante sigue siendo bueno o por lo menos aceptable. También resta encontrar alguna aplicación en donde la segmentación sea lo suficientemente buena para resolver un problema más complejo.

Referencias

- [1] Felzenszwalb, Pedro F., Huttenlocher, Daniel P., *Efficient Graph-Based Image Segmentation*, International Journal of Computer Vision, 01/09/2004
- [2] Robert Endre Tarjan., *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comput. Mach., 22:215–225, 1975.
- [3] D. Martin and C. Fowlkes and D. Tal and J. Malik, A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics Proc. 8th Int'l Conf. Computer Vision, 07/2001
- [4] Li, Hui and Cai, Jianfei and Nguyen, Thi Nhat Anh and Zheng, Jianmin, *A benchmark for semantic image segmentation*, 2013 IEEE International Conference on Multimedia and Expo (ICME) paginas 1 a 6, 2013