Teoría computacional de la gestalt.

Curso de postgrado en informática.
Docente: Dr. Andrés Almansa

Proyecto: mejoras al detector de alineamientos.

Fernando Fernández
fefernan@adinet.com.uy

Febrero de 2006

Palabras clave: alignments recognition, meaningful segments, minimal description length, exclusion principle, ordered priority trees, data structures, gestalt theory, pattern recognition, image processing, computer vision

Resumen

Una imagen digital es un conjunto finito de puntos que toman valores discretos. La teoría de la gestalt considera que se puede percibir una forma si, al suponer que los puntos de la imagen fueron generados al azar, resulta improbable que esa forma aparezca. Eso determina la significatividad de la forma. En particular, que un segmento sea percibido depende de la cantidad de puntos que lo constituyen y de cuantos de ellos están correctamente alineados con la recta en la que yace. La orientación de un punto es la normal al gradiente en el punto. Como los valores son discretos, la orientación se obtiene con cierta precisión, que es la que da lugar a la probabilidad de existencia de ese segmento. Pero además resulta que un punto puede estar bien alineado con rectas de diferentes direcciones. De acuerdo al principio de exclusión, cada punto sólo puede pertenecer a un segmento, a aquel que sea más significativo. Es necesario removerlo de los otros segmentos y así modificar la significatividad de ellos. Una forma de hacer esto es la que estaba implementada. Pero el método hace que se excluyan segmentos que deberían ser percibidos. Lo que aquí se documenta es otro método, que permite resolver el problema. El programa implementado recupera los segmentos mencionados y además resultó ser más rápido.

Índice

Índice de algoritmos.

1. Introducción.

Este es un informe acerca del diseño e implementación de un programa para detectar alineamientos en imágenes digitales. Es continuación de *align.c* y *align_mdl.c* y pretende resolver dificultades que habían quedado pendientes.

En este capítulo presentamos los conceptos teóricos en que se basa el detector. Las definiciones y demostraciones rigurosas se pueden encontrar en [DMM04].

Un alineamiento es una cantidad suficiente de puntos en una recta cuya orientación es la misma que la de la recta. En lo que sigue explicaremos que se entiende por orientación de un punto y cual es la condición de suficiencia para la cantidad de puntos.

Aquí consideramos que una imagen es una grilla finita de niveles, cuantizados, de gris. Los puntos en que está definida la grilla son las coordenadas de los centros de cuadrados de lado 1 denominados pixels. La cantidad de información es finita y la resolución es acotada. Por lo tanto, toda la información geométrica tiene una precisión. Pero mediante la interpolación de Shannon se puede calcular el nivel de gris en todos los puntos y no sólo en los de la grilla. El nivel de gris en el punto de coordenadas (x, y) lo representamos con la función $u(x, y)$.

Se llaman líneas de nivel a los conjuntos de puntos de la imagen que tienen un mismo valor de gris. La orientación de un punto es la dirección de la línea de nivel que pasa por él. Es la rotación en $\pi/2$ del gradiente. El gradiente en el punto (x, y) es:

$$Du(x, y) = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right)(x, y).$$

y entonces la orientación es:

$$Orientacion(x, y) = \frac{1}{\|Du(x, y)\|}\left(-\frac{\partial u}{\partial y}, \frac{\partial u}{\partial x}\right)(x, y).$$

El gradiente en (m, n) lo calculamos como la interpolación en el centro de la ventana 2×2 constituida por los pixels (m, n), $(m+1, n)$, $(m, n+1)$ y $(m+1, n+1)$,o sea en $(m+1/2, n+1/2)$. Por lo tanto tenemos:

$$Orientacion(m,n) = \frac{\vec{O}}{\|\vec{O}\|}$$

donde

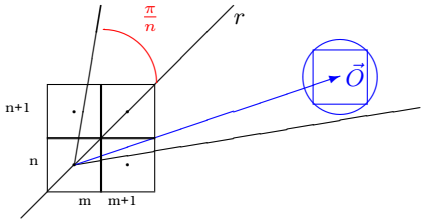
$$\vec{O} = \frac{1}{2}\begin{pmatrix}(u(m,n)+u(m+1,n)) - (u(m,n+1)+u(m+1,n+1))\\ (u(m+1,n)+u(m+1,n+1)) - (u(m,n)+u(m,n+1))\end{pmatrix}$$

De acuerdo a esto, dos puntos son independientes si están a distancia 2 o más.

Las línea y los segmentos que en ellas yacen los consideramos orientados. Un punto (m,n) y una línea r tienen la misma orientación con precisión $1/n$ si

$$|Angulo(Orientacion(m,n), Orientacion(r))| \leq \frac{\pi}{n}.$$

En lo que sigue, cuando esto se cumple diremos que el punto está alineado con la línea, o alineado con el segmento, o simplemente que el punto está alineado, cuando por el contexto estén implícitos el segmento y la precisión.



Debido a la cuantización sólo se considera definida la orientación en el punto si la norma de $\vec{O}$ es suficientemente grande. Esto es porque el error en cada una de las dos componentes de la orientación puede llegar a ser 1. Si la norma es chica el error en la precisión del ángulo puede ser mayor que π/n. Establecemos la condición

$$\|\vec{O}\| > \frac{\sqrt{2}}{\sin\frac{\pi}{n}}.$$

Como la geometría que usamos es discreta, cuando hablemos de segmentos nos referimos a un conjunto discreto de puntos consecutivos a lo largo de una recta, donde dos puntos son consecutivos si la distancia entre ellos es 1. Estos puntos en general no coinciden con las coordenadas del centro de un pixel. En las aplicaciones prácticas se lo aproximará al pixel en el que está contenido.

El fundamento teórico para decidir si un segmento se considera un alineamiento se basa en lo que se conoce como principio de Helmholtz. Lo que este dice es que se percibe lo que no puede ocurrir por azar. Concretando esto en el tema de las imágenes, se puede decir que ninguna estructura se percibe en una imagen de ruido o, dicho por la positiva, que una forma es percibida si es un conjunto de puntos cuya disposición no puede ocurrir en el ruido.

Una imagen es *ruido blanco* si los valores de cada pixel son variables aleatorias independientes e idénticamente distribuidas.

Supongamos que tenemos l objetos, k de los cuales tienen una propiedad común. Asumimos que la propiedad está distribuida independiente e idénticamente en los l objetos. Si la disposición no es probable es porque hay un proceso de agrupamiento, llamado *gestalt*

En una distribución así, si la probabilidad de que un objeto tenga la propiedad es p, entonces la probabilidad de que al menos k de esos l objetos la tengan es

$$B(l,k,p) = \sum_{i=k}^{l} \binom{l}{i} p^i (1-p)^{l-i}.$$

esto es la cola de la distribución binomial que llamaremos en adelante *tail-binomial*.

Introducimos el concepto de número de falsas alarmas(NFA) de un evento como el valor esperado del número de ocurrencias de ese evento.

Para determinar el NFA, la probabilidad anterior se debe multiplicar por el número de posiciones en los que se busca el agrupamiento, N_{conf}.

$$NFA = N_{conf} \cdot B(n,k,p). \tag{1}$$

La perceptibilidad, y la improbabilidad de un evento está relacionada con el concepto de significatividad.

Definición 1 *(**evento** $\epsilon-$**significativo**) Un evento de tipo "determinada configuración de objetos geométricos tiene tal propiedad" es $\epsilon-$significativo si el valor esperado del número de ocurrencias de ese evento es menor que ϵ bajo la suposición de distribución aleatoria uniforme.*

Entonces, un evento es $\epsilon-$significativo si su NFA es menor o igual que ϵ.

Cuando el objeto geométrico es un segmento y la propiedad es que los puntos estén alineados, hablaremos de segmentos $\epsilon-$ significativos.

Empezando a precisar la definición dada al principio, decimos que un alineamiento es un conjunto de puntos en una recta cuyas orientaciones son suficientemente cercanas a la de la recta y en número suficientemente grande para hacer el evento improbable en una imagen de ruido blanco. Como se estableció antes, las orientaciones son cercanas si difieren en menos de π/n, y en ese caso la probabilidad $p = 1/n$ es la que contribuye a decidir si el evento es improbable.

Sea A un segmento de longitud l y $x_1, \ldots, x_l$ los puntos, independientes, de A. Sea X_i la variable aleatoria cuyo valor es 1 si el punto está alineado con el segmento y 0 en otro caso. O sea que siguen la distribución de Bernoulli. S_l es la variable aleatoria que representa el número de puntos alineados, y es

$$S_l = X_1 + \ldots + X_l$$

y su ley está dada por la distribución binomial

$$\Pr[S_l = k] = \binom{l}{k} p^k (1-p)^{l-k}.$$

Si la imagen tiene M filas y N columnas tiene por tanto $M \times N$ puntos. Como los segmentos y las líneas los estamos considerando con orientación, un segmento puede empezar en cualquiera de esos puntos y terminar en cualquier otro. La cantidad de segmentos es entonces $(M \cdot N) \cdot (M \cdot N - 1)$ y esto que aproximamos como $(M \cdot N)^2$ es N_{conf} en la ecuación (1).

Llegamos a la definición de segmento significativo y NFA de un segmento.

Definición 2 *(**segmento** $\epsilon-$**significativo**) Un segmento de longitud l es $\epsilon-$significativo en una imagen $M \times N$ si contiene al menos $k(l)$ puntos que están alineados con la orientación del segmento, donde $k(l)$ está dado por*

$$k(l) = \min \left\{ k \in \mathbb{N}, \Pr[S_l \geq k] \leq \frac{\epsilon}{(M \cdot N)^2} \right\}$$

Tomaremos $\epsilon = 1$ y lo que hemos estado llamando alineamientos son los segmentos significativos.

Definición 3 *(**Número de falsas alarmas de un segmento**.) Sea A un segmento que tiene l puntos y k de ellos están alineados cuando la probabilidad es p. El número de falsas alarmas de A es NFA(l,k)= $(M \cdot N)^2 \cdot$ B(l,k,p)*

Un segmento es significativo si su NFA es menor que 1.

Se puede demostrar que el NFA de un segmento es el menor valor ϵ para el cual el segmento es $\epsilon-$significativo. Entonces, dados dos segmentos es más significativo aquel cuyo NFA es menor.

Señalamos dos propiedades intuitivamente evidentes y fáciles de demostrar que serán usadas repetidamente:

Dados dos segmentos de igual longitud es más significativo el que tiene más puntos alineados.

$$(M \cdot N)^2 B(l, k_1, p) < (M \cdot N)^2 B(l, k_2, p)$$

cuando $k_1 > k_2$

Dados dos segmentos que tiene la misma cantidad de puntos alineados es más significativo el más corto.

$$(M \cdot N)^2 B(l_1, k, p) < (M \cdot N)^2 B(l_2, k, p)$$

cuando $l_1 < l_2$

Hacemos una aclaración sobre la notación. Sean a y b dos segmentos disjuntos de una misma línea, concretamente a termina antes de que empiece b. Entonces ab representa el segmento que empieza donde empieza a y termina donde termina b, incluyendo todos los puntos que están entre ambos. Esto mismo se generaliza cuando los segmentos son más de dos. Hay que notar que el NFA del segmento así definido puede ser mayor, igual o menor que cualquiera de aquellos con los que se construyó.

La percepción debe obedecer a un principio de economía, aquí llamado **principio de exclusión**, de acuerdo al cual dos alineamientos no pueden solaparse. Ningún punto puede pertenecer a más de un segmento. Más en general, el principio establece que, dados dos grupos obtenidos por la misma ley gestáltica, ningún punto puede pertenecer a ambos.

Un punto alineado con una línea pertenece a todos los segmentos de ella que lo contengan y por lo tanto debería ser contado en la cantidad de puntos alineados de cada uno de ellos. Pero además puede pertenecer a los segmentos de otras líneas cuando la orientación de ellas es suficientemente cercanas a la considerada. El principio de exclusión provoca que deba removerse del conteo respectivo de todos los segmentos excepto de uno. El criterio seguido es que ese segmento que conserva el punto es el que sea más significativo.

Lo siguiente es el contenido del resto del informe. En el capítulo 2 se desarrolla el principio de exclusión, los programas que actualmente lo implementan y los problemas que no pueden resolverse. En el capítulo 3 se plantea una nueva versión del principio que resuelve esos problemas y se describe la implementación. En el 4 se describe una manera distinta de recorrer la imagen para mejorar el rendimiento. En el capítulo 5 se explica como aplicar la nueva versión del principio a los segmentos de una línea para disminuir el tiempo de ejecución. Los resultados experimentales y los requerimientos de memoria están en el capítulo 6. Y en el capítulo 7 se plantean algunas limitaciones de la propuesta desarrollada y posibles correcciones.

2. Antecedentes.

En este capítulo profundizamos en el principio de exclusión y vemos los programas *align* y *align_mdl* y como resuelven algunos problemas presentados por ese principio. También las limitaciones que tienen en ese tema.

2.1. align.c

En esta sección consideramos segmentos que yacen en una misma línea.

En [DMM04] se establece el denominado *Algorithm 4 Exclusion Principle(EP).* Aplicado al problema de los alineamientos, lo que implica es que, una vez calculado el NFA de todos los segmentos, cada punto se asigna a aquel segmento que lo contiene y que sea más significativo que cualquier otro que también lo contuviera. Ese segmento se considera el propietario de ese punto. Luego de haber procesado todos los puntos, se recalcula el NFA de cada segmento pero ahora no se cuenta como punto alineado aquellos que no le pertenecen.

Se trata de encontrar una forma de resolver esto eficientemente. Un segmento significativo enmascara otros. Es decir, algunos segmentos incluidos en él también pueden ser significativos. Asimismo, algunos que lo contienen es posible que también lo sean. Pero lo que intentamos es encontrar, entre todos ellos, cual es el que es percibido. Se introduce entonces el concepto de maximalidad. Se dice que un segmento es maximal si es más significativo que cualquier otro segmento que lo contenga y también que cualquiera que esté contenido en él (estrictamente, se puede aceptar que un segmento contenido en él tenga la misma significatividad). Se dice que es significativo maximal si además de maximal, es significativo.

Es intuitivo y fácil de demostrar que todo segmento maximal empieza y termina en puntos alineados y que los puntos inmediatamente anterior e inmediatamente posterior no están alineados.

El denominado *Algorithm 7 Exclusion Principle for Alignments (EP ter)* presentado en [DMM04] puede formularse de la siguiente manera: Se encuentran todos los segmentos que empiecen y terminen en puntos alineados y que sus puntos inmediatamente anterior e inmediatamente posterior no lo estén. Una vez hecho esto se consideran todos los pares posibles en los que uno de esos segmentos contiene otro y se descarta el menos significativo de ambos.

El resultado de esto son los segmentos maximales. Esto es así porque todos los segmentos que no son maximales son descartados en alguna de las comparaciones.

Mediante una conjetura acerca de *tail-binomial* se concluye que dos segmentos maximales no pueden solaparse. Como por definición no puede estar uno contenido en el otro, entonces son disjuntos.

Por lo tanto los segmentos maximales serían el resultado de aplicar el principio de exclusión antes descripto. Entonces, si en el primer paso del algoritmo anterior seleccionamos sólo segmentos que sean significativos, su salida es lo que buscamos: los segmentos significativos que se obtienen como resultado de la aplicación del principio de exclusión.

Lo anterior es lo que implementa *align.c*. El algoritmo 1 es una parte del programa. Previamente hubo que calcular la orientación de cada punto de la imagen para poder determinar los alineamientos y *tail-binomial* para determinar si un segmento es o no significativo. La maximalidad está desde la línea 12.

En la línea 7 se hace referencia a los bloques. Un bloque es una secuencia de puntos consecutivos todos ellos alineados precedidos por un punto no alineado y a los que sigue un punto no

Algoritmo 1 align

```
1: para cada borde de la figura
2:     para cada punto del borde
3:         para cada direccion
4:             para fase = 0 a 1
5:                 mientras el punto está dentro de los límites
6:                     comprobar si el punto está alineado
7:                     marcar inicio o fin de bloque
8:                     avanzar el punto
9:                 para cada  [inicio de bloque,fin de bloque]
10:                    si es un segmento significativo entonces
11:                        crear el segmento
12:                {maximalidad}
13:                para cada par de segmentos creados
14:                    si el primero está contenido en el segundo entonces
15:                        descartar el menos significativo
16:                para cada segmento no descartado
17:                    incluirlo en la salida
```

alineado. Cada una de estas últimas dos condiciones no se exigen si el bloque empieza o termina en un borde. Esto se hace para obtener los segmentos maximales.

Pero esta versión del principio de exclusión presenta un problema. Supongamos que en la siguiente figura los segmentos a, b y ab son significativos. Además suponemos $NFA(b) < NFA(ab) < NFA(a)$. Como a está contenido en ab que es más significativo será descartado. Pero ab también será descartado porque contiene a b que es el más significativo. Como resultado de esto en la salida del algoritmo sólo se incluye b. O sea que el algoritmo hace desaparecer al segmento a, que habíamos dicho que era perceptible. A este problema lo llamamos exclusión mutua y no es un problema generado por la implementación sino que intrínseco a la forma en que se ha enunciado el principio de exclusión. Este problema será resuelto con otra versión del principio.

Pero para este programa se presenta otro problema al dejar de considerar únicamente segmentos que están en una misma línea. En la figura 1 vemos la imagen ampliada de los segmentos que genera el programa cuando la imagen consiste en un trazo.

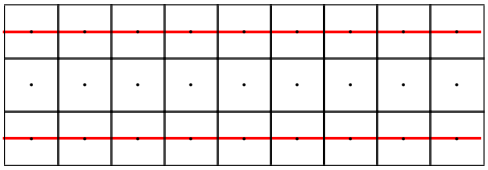
Figura 1: Segmentos de un trazo.

Debería haber sólo dos segmentos, uno por cada borde del trazo. Sin embargo además de esos dos, aparecen varios segmentos paralelos a ellos y otros oblicuos inclinados un ángulo relativamente pequeño respecto a los otros. Los primeros son causados por el dispositivo óptico. Los otros

se deben al error aceptado en la precisión de la orientación. Cuando se escanea una línea cuya orientación difiere de la del segmento perceptible un ángulo inferior al error aceptado, el punto se considera alineado también con esta dirección. Como esa línea coincide en muchos puntos con la del segmento perceptible y como además están los otros segmentos generados por el dispositivo, la línea oblicua tiene muchos puntos alineados y se generan segmentos significativos. En principio, podría pensarse que este segundo problema se resolverá al aplicar el principio de exclusión, ahora en dos dimensiones. Cada uno de esos puntos que es contenido por varios segmentos sólo debe pertenecer a uno y al aplicar el principio los segmentos oblicuos quedarían sin puntos. Pero de todas formas quedaría sin resolverse el problema de los segmentos paralelos.

2.2. align_mdl.c

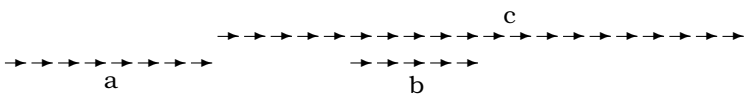
Para resolver el anterior problema se define el concepto de segmento dilatado. Esto consiste en asumir que los segmentos tienen un ancho de dos pixels. Entonces, si un segmento es el que debe ser incluido en la salida, los puntos que están a su lado no pueden pertenecer a algún otro segmento. Por cada punto del segmento existe lo que llamaremos punto dilatado, que está compuesto por tres puntos: el propio punto, y sus dos puntos adyacentes, uno en la dirección y sentido del gradiente, y otro en el sentido opuesto. Al aplicar el principio de exclusión, cuando se determina a que segmento pertenece cada punto de la imagen, cada segmento compite por los tres puntos del punto dilatado de cada uno de sus puntos. Después de hacer esto, para que un punto se considere alineado con el segmento, además de estarlo, el segmento debe ser el propietario de todo el punto dilatado. Esto es así porque, si no se cumple, significa que hay otro segmento muy cercano que es más significativo. De esta manera, los segmentos oblicuos y paralelos mencionados al final de la sección anterior reducen su conteo de puntos alineados.



Esto es lo que hace *align_mdl.c* y resuelve el problema. El seudo código se muestra en el algoritmo 2. El principio de exclusión se implementa en tres partes. En la línea 10 se calcula el NFA del segmento. En el ciclo de la línea 12 cada segmento compite por sus puntos. Y en el de la línea 18 se vuelve a calcular el NFA.

Esto es una generalización de la maximalidad usada en *align* cuando se tienen dos dimensiones y se considera al segmento dilatado. Si se aplica sobre una sola línea el resultado sería el mismo que la versión anterior. Esto es porque lo que aquí se hace es la implementación del principio de exclusión tal cual está formulado. Por lo tanto sigue presente el problema planteado en la sección anterior denominado exclusión mutua. Esto era que un segmento es descartado por uno que lo contiene y este último es a su vez descartado por otro segmento contenido en él. En la figura 2 se puede ver un ejemplo de esto. En la subfigura izquierda está un fragmento de la imagen *trazos.rim* y en la derecha los segmentos que *align_mdl* genera de ella. Para el trazo que está en la equina superior izquierda no se generan segmentos. Esto es porque ese trazo está en la misma línea que el que está en la equina inferior derecha y entonces sucede lo que se señaló antes.

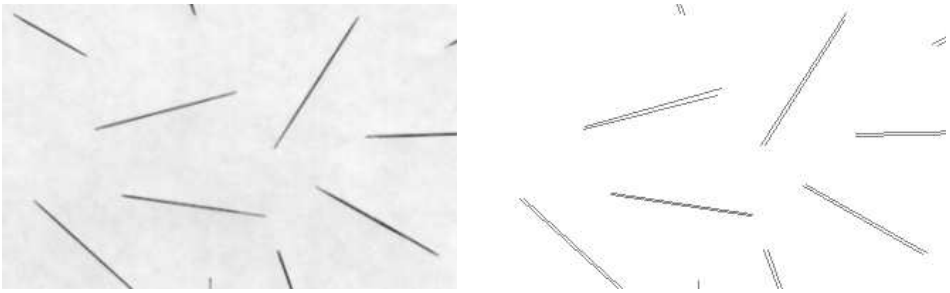
Al trabajar en dos dimensiones el problema se incrementa. En la siguiente figura suponemos que a es un segmento significativo pero ab es el más significativo de la línea. En este caso no habría exclusión mutua en la línea. Pero si el segmento c de la línea inmediatamente superior es más significativo, entonces ab pierde los puntos de b para su conteo de puntos alineados. Supongamos que después de esto ab deja de ser significativo. Entonces el bloque a no aparece en la salida. En este caso a fue excluido por ab y este por c. Si no hubiera estado el bloque b, c no habría podido excluir a a.



El problema no se da sólo en el caso genérico de segmentos compuestos por varios bloques, sino también cuando corresponde a un único bloque. En la nueva figura suponemos que el bloque ab es un segmento significativo, pero que el segmento a contenido en él también lo es. Una vez más, si c es más significativo que ab, y si cuando este pierde los puntos alineados correspondientes a b deja de ser significativo, entonces el resultado tampoco incluye el segmento a. Lo que sucede

Algoritmo 2 align_mdl

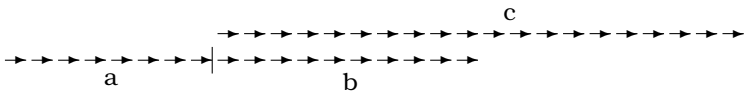
```
1: para cada precision
2:     para cada borde de la figura
3:         para cada punto del borde
4:             para cada direccion
5:                 mientras el punto está dentro de los límites
6:                     comprobar si el punto está alineado
7:                     marcar inicio o fin de bloque
8:                     avanzar el punto
9:                 para cada  [inicio de bloque,fin de bloque]
10:                    si es un segmento significativo entonces
11:                        almacenar el segmento
12:                        para cada punto del segmento
13:                            si el punto está alineado entonces
14:                                para cada punto del punto dilatado
15:                                    si este segmento es más significativo que el actual propietario del
                                       punto entonces
16:                                        asignar el punto a este segmento
17: para cada segmento almacenado
18:     para cada punto del segmento
19:         si el punto está asignado al segmento entonces
20:             contarlo como uno de sus puntos alineados
21:         recalcular la significatividad
22:         si es significativo entonces
23:             incluirlo en la salida
```



(a) Fondo (b) Segmentos

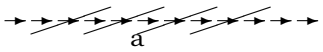
Figura 2: Trazos.

es que un bloque enmascara los que están contenidos en él. Pero es en principio correcto utilizar el bloque, porque él es el resultado de la aplicación del principio de exclusión a todos los bloques contenidos. Esto es evidente por maximalidad. Este problema no ocurría en *align* porque no existía el segmento dilatado.



Una sencilla modificación en el algoritmo puede resolver el problema planteado en los dos ejemplos anteriores. Cuando un segmento que era significativo va a descartarse porque, la cantidad de puntos alineados disminuyó como consecuencia de la asignación de puntos a otros segmentos, es posible que algunos de los puntos perdidos sean los de los extremos. En ese caso también se debería modificar la longitud del segmento y podría ocurrir que entonces el segmento siga siendo significativo. Lo que sucede es que es que al modificar los extremos del segmento en realidad lo que se está haciendo es volver a obtener un segmento que se había descartado antes. En el primero de los ejemplos anteriores el segmento a fue descartado explícitamente aplicando el principio de exclusión. Lo eliminó ab que a su vez fue eliminado. En el segundo ejemplo, a fue eliminado implícitamente porque al decidir usar bloques se está aplicando el principio de exclusión parcialmente al conjunto de segmentos contenidos en él. En ambos casos a es recuperado porque no intersecta al segmento dilatado de c.

Pero esto no resuelve el problema general. En un nuevo ejemplo supongamos que el segmento horizontal a es significativo y debería ser incluido en la salida. Las lineas oblicuas representan segmentos obtenidos al escanear y que resultan ser más significativos que a. Pero estos segmentos son generados porque, como se explicó al final de la sección anterior, la diferencia entre su orientación y la del segmento a es menor que la precisión con la que acepta el error en la orientación. Una explicación por la cual estos segmentos serían más significativos que a es que hay otros segmentos paralelos a a los cuales permiten que esos segmentos oblicuos sean suficientemente largos.



Al aplicar el principio de exclusión cada uno de los segmentos oblicuos le hace perder tres puntos al horizontal y este deja de ser significativo. Pero el concepto de segmento dilatado pretende, justamente, eliminar esos segmentos oblicuos. Y suponemos que eso ocurre. Entonces tenemos otra vez la exclusión mutua: a es excluido por los segmentos oblicuos y estos por algún otro.

Pero los segmentos oblicuos no sólo deben eliminarse de la salida. También debe eliminarse su influencia en la asignación del propietario de cada punto. No deberían competir por los puntos. Pero al principio no se sabe cuales son. El problema está en la enunciado de la actual versión del principio de exclusión. Se calcula el NFA de cada segmento independientemente de los otros, y es con ese NFA con el que compite por los puntos.

3. Principio de exclusión.

En este capítulo presentamos otra versión del principio de exclusión que resuelve los problemas señalados en la anterior. Y explicamos el algoritmo y la estructura que lo implementa.

En esta versión se selecciona el mejor segmento en cada paso y continúa mientras haya segmentos significativos. Cuando un segmento es seleccionado todos los puntos del segmento dilatado son asignados definitivamente a él, y por lo tanto deben ser removidos del conteo de puntos alineados de otros segmentos que los contuvieran. Entonces, para seleccionar el próximo mejor segmento se tiene en cuenta el NFA modificado por la interacción con los más significativos. Esta es la versión que en [DMM04] se enuncia como *Algorithm 5 Exclusion principle (EP-bis)*. Ahora cuando un segmento es seleccionado se hace propietario de los puntos del segmento dilatado y entonces los segmentos oblicuos y paralelos los pierden para su conteo y dejan de ser significativos. Entonces estos segmentos, que no son parte de la imagen sino que son introducidos por el dispositivo óptico o por el método de escaneo, y que contribuían a eliminar segmentos que sí pertenecen a la imagen, son descartados. Así es como se resuelve el problema planteado al final del capítulo anterior.

El problema con esta versión es que su implementación es más difícil y es aparentemente también más costosa en tiempo y en memoria. Un algoritmo de fuerza bruta en el que en cada paso se selecciona un segmento y luego se recalcula el NFA de los que quedan es $O(segmentos^2)$. Y el propio recálculo no es una operación elemental sino que implica recorrer todos los puntos del segmento haciendo varias comparaciones y otras operaciones en cada uno.

En lo que sigue se explica la solución aquí implementada.

3.1. Algoritmo.

Para la aplicación del principio de exclusión debe implementarse una cola de prioridad. Se extrae repetidamente el segmento con menor NFA mientras este sea menor que el máximo aceptado. Pero al extraer el segmento hay que actualizar el NFA de los segmentos que tengan puntos en común con el segmento dilatado extraído y después de eso reordenar la cola de prioridad. Para la primera de esas tareas, una implementación eficiente no puede evitar mantener, para cada uno de los puntos de la imagen, el conjunto de segmentos que pasa por él. El tamaño de esta estructura es igual a la cantidad de segmentos por el largo promedio de los segmentos. Si, por ejemplo, hay $1,000,000$ segmentos con longitud promedio 30, al necesitarse 4 bytes para representar un entero, la estructura ocupa $120,000,000$ bytes. Este tamaño es muy grande y es la causa de que este método no sea aplicable.

La solución aquí implementada no actualiza los segmentos que intersectan al extraído, y se describe en el algoritmo 3.

Algoritmo 3 Principio de exclusión

```
para cada segmento almacenado
    insertarlo en la cola de prioridad
mientras la cola de prioridad no está vacía
    extraer el segmento con menor NFA
    encontrar el nuevo inicio y el nuevo fin del segmento
    para cada punto del segmento
        si el punto está alineado y no fue asignado a otro segmento entonces
            contarlo como uno de sus puntos alineados
    recalcular la significatividad
    si sigue siendo significativo entonces
        si es menos significativo que el que quedó primero en la cola de prioridad entonces
            reinsertar el segmento con el NFA actualizado en la cola de prioridad
        en otro caso
            para cada punto del segmento dilatado
                marcarlo como asignado
            incluir el segmento en la salida
```

Lo que aquí se hace es extraer el segmento más significativo y recalcular su NFA teniendo en cuenta como le afectan los segmentos más significativos que ya se habían extraído en ciclos anteriores. El NFA podría aumentar. Esto hace que en principio no sepamos si este segmento es realmente el más significativo de todos los que quedan. Podría ocurrir que alguno de los segmentos que al comenzar el actual ciclo tenía un NFA mayor, tenga menos puntos en común con los segmentos extraídos en los ciclos anteriores, y por lo tanto, si se recalculara su NFA podría encontrarse que es menor que el del segmento considerado. Lo que se hace es tener en cuenta el nuevo NFA del segmento extraído. Hay tres rangos de valores que interesan. Primero, si este valor es mayor o igual al máximo aceptable entonces descartamos el segmento. Segundo, si es menor o igual que el NFA del segmento que lo seguía en la cola de prioridad, entonces sabemos que el segmento extraído es el más significativo. Esto es porque el NFA de los segmentos que quedan en la cola de prioridad, sólo puede aumentar o quedar igual cuando se actualice teniendo en cuenta los puntos asignados a segmentos en ciclos anteriores. Entonces el segmento extraído lo insertamos en la salida, y además hacemos que los puntos del segmento(dilatado) queden marcados como ya asignados. El tercer rango de valores para el NFA es cuando es menor que el máximo aceptable pero mayor que el NFA del segmento que lo seguía. En este caso no podemos saber cual es el más significativo de todos. Lo que se hace es reinsertar el segmento, con el NFA actualizado. Aunque no puede producirse un ciclo infinito porque las reinserciones son con un NFA necesariamente mayor, aquí es donde este método puede resultar ineficiente. Lo será si la cantidad de reinserciones es grande. Sin embargo podemos suponer que esto no ocurre. Esto es porque por cada

segmento que insertamos en la salida, hay una gran cantidad que están en torno a él, paralelos y oblicuos, los cuales tienen casi todos sus puntos en común con el segmento dilatado (esta fue la razón para crear el concepto de segmento dilatado). Entonces al recalcular el NFA tenemos que la cantidad de puntos alineados que se siguen contando es casi nula y dejan de ser significativos, por lo que se descartan sin reinsertar.

El costo del algoritmo del principio de exclusión tiene básicamente dos componentes. Uno de ellos es el recálculo del NFA. Para esto hay que recorrer el segmento y en cada uno de sus puntos ejecutar la operación de verificación. Esto se hace para cada segmento extraído, por lo que el costo es $O(\text{cantidad de segmentos} * \text{largo promedio})$. En realidad, al factor cantidad de segmentos habría que sumarle en esta implementación la cantidad de reinserciones. Pero en la primer versión considerada, aquella en la que cada vez que se extrae un segmento se actualiza el NFA de todos los que intersectan el segmento dilatado este costo es aún mayor, porque muchos segmentos podrían ser recorridos varias veces.

Una alternativa a esto sería mantener por cada segmento, el conjunto de los puntos que están alineados, que ya se vio que implica un tamaño muy grande. El otro componente está determinado por la estructura de la cola de prioridad, por el costo de las operaciones eliminar el mínimo, insertar y obtener el mínimo.

3.2. Estructura.

La implementación aquí usada para la cola de prioridad se conoce como montículo, montículo binario, heap o árbol de prioridad ordenado. Es un árbol binario completo, es decir, que todos los niveles están llenos, excepto tal vez el más bajo. Se puede implementar con un array y evitar el uso de punteros. En la posición 0 se pone un elemento centinela. Para el nodo en la posición i los hijos están en las posiciones $2i$ y $2i + 1$ y el padre en $\lfloor i/2 \rfloor$. Las operaciones para posicionarse en el padre o en los hijos son, entonces, muy rápidas.

La propiedad de orden del árbol es que la etiqueta de cada nodo es menor o igual que la de sus descendientes. Por supuesto esto implica que en la raíz está el elemento mínimo. Por lo tanto obtener el NFA del segmento más significativo requiere tiempo constante. Se ha demostrado [Wei95, capitulo 6] que en promedio las inserciones requieren $2{,}607$ comparaciones, por lo que se puede considerar que también requieren tiempo constante. Esto hace a esta estructura adecuada para tratar con la posibilidad de inserciones. La construcción del árbol es $O(n)$, siendo n la cantidad de nodos del árbol. Extraer el mínimo se realiza en tiempo $O(\log n)$, porque la altura de un árbol binario completo de n nodos es $\lfloor \log n \rfloor$. Entonces todo el proceso de vaciar el árbol es $O((n + inserciones) \log n)$.

Para construir el árbol es necesario saber cual es el tamaño máximo que puede alcanzar. Esto no genera problemas porque este dato se conoce y es el tamaño del conjunto de segmentos almacenados. En realidad no se construye un árbol de segmentos sino uno de referencias a segmentos. Esto es para que la operación de intercambio de nodos sea menos costosa. Las referencias se implementan con la posición en el array de segmentos almacenados.

4. Recorrido.

Se modifica la forma de recorrer la imagen usada anteriormente. Con los parámetros predeterminados, *'align_mdl'* utiliza 3 precisiones y entonces hace 6 recorridas por cada línea, 3 orientadas en un sentido y 3 en el opuesto. Posicionarse y consultar si el punto está alineado es la operación que más se realiza, por lo que se justifica mejorar su eficiencia. El costo de posicionarse implica multiplicaciones y sumas en punto flotante, redondeos y cuatro comparaciones para establecer si todavía está dentro de los límites. Consultar si el punto está alineado consiste en determinar si la norma del gradiente es suficientemente grande, y varias comparaciones y restas. Mucho de esto es redundante. Si la norma es demasiado chica para la precisión menor, podrían descartarse las consultas de las otras 5 pasadas. Además, si un punto no está correctamente alineado con una determinada precisión tampoco lo está para las más exactas y no hace falta consultar. Por otro lado, si está bien alineado con la primera orientación, entonces no puede estarlo con la segunda y se puede descartar las consultas correspondientes.

El nuevo algoritmo de recorrido toma en cuenta esto y cambia el orden de los ciclos. Además, para evitar las consultas acerca de si el punto está dentro de los límites, se calcula cual es la cantidad de puntos que tiene la línea a recorrer. Para esto, en lugar de obtener, para cada precisión, una matriz con las orientaciones de los puntos para aquellos cuya norma sea suficientemente grande, lo que se hace es obtener la matriz de las orientaciones para todos los puntos y otra

para los cuadrados de la norma. Esto significa que se necesita más memoria. También, como la precisión ya no es el ciclo más externo sino el más interno, hay que construir la matriz con la evaluación de *tail-binomial* para todas las precisiones y conservarlas en memoria.

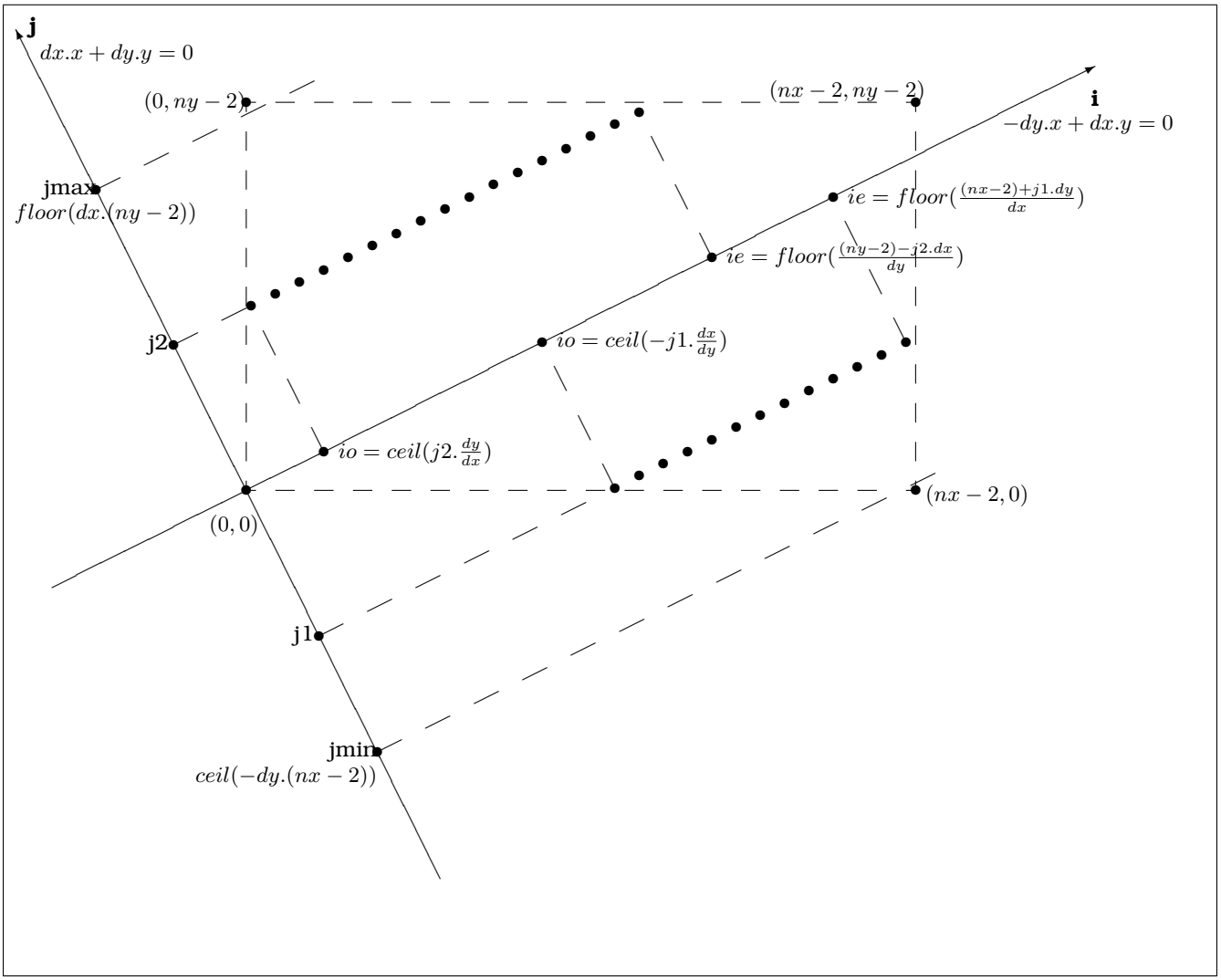
También se modifica la forma de construcción de los bloques. La mayoría de los puntos no están alineados y sólo interesa hacer algo en esos casos cuando ese punto es el primero después de un bloque. Entonces lo que se hace ahora es registrar en un array los puntos que están alineados y en los que no están no se hace nada. Posteriormente se recorre ese array construyendo los bloques con esa información. El tamaño máximo de este array es el de la diagonal más larga de la imagen (en realidad es eso multiplicado por 6, porque hay uno por cada una de las recorridas que se están haciendo de manera implícita).

Para cada dirección se define nuevos ejes de coordenadas. El eje i de acuerdo a la dirección y el eje j perpendicular a ella. Los puntos recorridos tienen coordenadas enteras en estos ejes. Para obtener las coordenadas x e y del siguiente punto sólo hay que sumar los ángulos directores de la dirección de la línea que se está recorriendo, que son $dx = \cos\theta$ y $dy = \sin\theta$, siendo θ el ángulo de la línea respecto al eje x. Es decir, si las coordenadas del punto actual son (i, j) en los ejes solidarios a la actual dirección y (x, y) en los ejes de la imagen, entonces el punto $(i + 1, j)$ tiene coordenadas $(x + dx, y + dy)$.

Tratamos de manera diferenciada las direcciones entre 0 y $\pi/2$ a las que están entre $\pi/2$ y π.

La siguiente figura ejemplifica el primero de estos dos casos: $0 < \theta \leq \pi/2$. Hay que notar que como la primera desigualdad es estricta siempre se cumple $dy > 0$. El origen de coordenadas se ubica en el punto $(0, 0)$. El tamaño de la imagen está representado por nx y ny. Como no calculamos la orientación del borde derecho ni del superior, el rango de filas que consideramos va desde 0 hasta $ny - 2$ y el de columnas desde 0 hasta $nx - 2$. Para ubicarnos en el primer punto de una línea necesitamos la transformación entre sistemas de coordenadas y ella es:

$$(x, y) = (i \cdot dx - j \cdot dy, i \cdot dy + j \cdot dx)$$



La ecuación del eje i es $-dy \cdot x + dx \cdot y = 0$. La ecuación del eje j es $dx \cdot x - dy \cdot y = 0$.

La distancia de $(nx-2, 0)$ hasta el eje i es $d = fabs(-dy \cdot (nx-2) + dx \cdot (0))$. Entonces $jmin = (int)ceil(d)$ es el mínimo valor entero de la ordenada de una recta paralela al eje i que intersecta la figura.

La distancia de $(0, ny-2)$ hasta el eje i es $d = fabs(-dy \cdot (0) + dx \cdot (ny-2))$. Entonces $jmax = (int)floor(d)$ es el máximo valor entero de la ordenada de una recta paralela al eje i que intersecta la figura.

Si $j \leq 0$ la linea escaneada entra a la figura después de intersectar la recta $y = 0$. Entonces el i mínimo es $iorg = ceil(-j.\frac{dx}{dy})$.

Si $j > 0$ la linea escaneada entra a la figura después de intersectar la recta $x = 0$. Entonces el i mínimo es $iorg = ceil(j.\frac{dy}{dx})$. Aquí hay una división por dx. Pero dx sólo vale 0 cuando $\theta = \pi/2$. Y en ese caso no se puede cumplir $j > 0$.

El i máximo es el menor valor de las intersecciones de la línea escaneada con las rectas $y = ny - 2$ y $x = nx - 2$. Entonces

$$iend = \text{mín}(floor(\frac{(ny-2) - j \cdot dx}{dy}), floor(\frac{(nx-2) + j \cdot dy}{dx})).$$

De la misma manera se pueden calcular los valores para $\pi/2 < \theta \leq \pi$. En este caso el origen de coordenadas se sitúa en $nx - 2, 0$. Teniendo en cuenta esto la transformación de coordenadas es:

$$(x, y) = (i \cdot dx - j \cdot dy + (nx-2), i \cdot dy + j \cdot dx)$$

La ordenada de las líneas varía entre $jmin = (int)ceil(dx \cdot (ny-2))$ y $jmax = (int)floor(dy \cdot (nx-2))$.

Si $j \leq 0$ la linea escaneada entra a la figura después de intersectar la recta $x = nx-2$. Entonces el i mínimo es $iorg = ceil(j.\frac{dy}{dx})$.

Si $j > 0$ la linea escaneada entra a la figura después de intersectar la recta $y = 0$. Entonces el i mínimo es $iorg = ceil(-j.\frac{dx}{dy})$.

Además se calcula:

$$iend = \text{mín}(floor(\frac{(ny-2) - j \cdot dx}{dy}), floor(\frac{-(nx-2) + j \cdot dy}{dx})).$$

Otro cambio es que como algún punto de los segmentos dilatados podría quedar fuera de la imagen, lo que se hace es rodear las matrices de la orientación, cuadrado de la norma y asignación de puntos con un borde. Por lo tanto las transformaciones de coordenadas anteriores deben modificarse sumándole 1 a cada coordenada.

Al crear y almacenar un segmento es necesario conocer en que sentido se está recorriendo la línea. Entonces la estructura para los segmentos tiene un atributo llamado *orientacion* cuyo valor es 0 si el sentido coincide con el ángulo de la línea o 1 si es el opuesto. Por lo tanto el ángulo del segmento se puede calcular como $\theta - orientacion \cdot \pi$.

Se puede señalar un par de ventajas de este recorrido. Una es que la distancia entre dos líneas consecutivas en la misma dirección es siempre igual a 1. En cambio en *'align'* esa distancia es $\sin\theta$ cuando las líneas comienzan en dos de los lados y es $\cos\theta$ cuando comienzan en los lados perpendiculares. O sea que hay una falta de uniformidad que depende de la dirección y también del punto de inicio. En la versión aquí presentada hay menos líneas, pero no se pierde información porque cada par de líneas consecutivas están separadas la mínima distancia a la cual se puede obtener resultados diferentes entre ambas. Además podría diseñarse alguna estructura que aproveche el hecho de que el segmento dilatado pertenece exclusivamente a la línea escaneada y sus dos adyacentes.

La otra ventaja es que no se pierde información sobre el ángulo del segmento. Como las coordenadas en los ejes x e y se tienen en punto flotante, los datos del origen y el fin se almacenan con la exactitud que esta aritmética permite y entonces el ángulo almacenado del segmento es igual al de la línea escaneada.

El algoritmo 4 muestra como es esta versión del detector.

5. Principio de exclusión en una línea.

Vimos que el costo del principio de exclusión es

$$c1 \cdot n \cdot largo + c2 \cdot n \cdot \log n \tag{2}$$

donde n es la cantidad de segmentos.

El primer término aparece por la necesidad de volver a recorrer los segmentos. Pero hay casos en los que esto no es necesario. Por ejemplo si el segmento a está contenido en el segmento b que

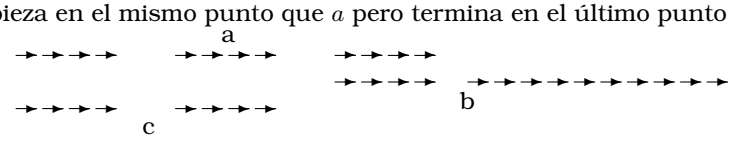
Algoritmo 4 Nueva versión de detector de alineamientos.

```
1: para cada direccion
2:     para cada línea
3:         para cada punto
4:             si el punto está alineado con alguna de las dos orientaciones entonces
5:                 mientras permanezca alineado con esa orientación
6:                     marcarlo como alineado para esa precisión
7:                     incrementar la precisión
8:         para orientacion=0 a 1
9:             para cada precisión
10:                {definir bloques}
11:                para cada punto marcado como alineado
12:                    si inicia un bloque entonces
13:                        marcar el inicio de un bloque y el fin de otro
14:                para cada  [inicio de bloque,fin de bloque]
15:                    si es un segmento significativo entonces
16:                        almacenar el segmento
17: aplicar el principio de exclusión (algoritmo 3)
```

es más significativo, sabemos que, si b es seleccionado mediante el principio de exclusión como uno de los segmentos más significativos, entonces el segmento a quedará con 0 puntos alineados y por lo tanto no es significativo. Y en este caso no es necesario volver a recorrerlo. Además al descartar el segmento a antes de incluirlo en el árbol, también disminuye el segundo término de la ecuación (2).

De lo que se trata esta sección es de como utilizar la información que se tiene al recorrer una línea para lograr mejoras en el rendimiento.

Para esto vamos a tener en cuenta que si un segmento es el más significativo, entonces todos los que tienen intersección con él pueden descartarse. En el párrafo anterior vimos que esto es cierto cuando el otro segmento está totalmente contenido en él. Ahora suponemos que el segmento a empieza antes y termina dentro del b, que es más significativo. Consideremos el segmento c, que empieza en el mismo punto que a pero termina en el último punto del último bloque anterior a b.

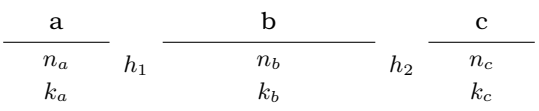


En la figura cada segmento aparece en una línea diferente pero recordemos que están en la misma línea.

Al aplicar el principio de exclusión y remover los puntos de b del conteo de los de a, la cantidad de puntos alineados de a es igual a la de c. Como este tiene una longitud menor, su NFA también es menor y es más significativo. Toda vez que por una posterior aplicación del principio de exclusión c pierda puntos, por ejemplo, el bloque que está más a la izquierda, también los perderá a y seguirá siendo c más significativo. En algún momento c tendrá un NFA mayor que el máximo aceptable y deberá ser descartado y con más razón debe ser descartado a, o c es el más significativo y después de aplicar el principio de exclusión a quedará con 0 puntos alineados y deberá ser descartado.

De la misma forma se descartan los segmentos que empiezan dentro de b y terminan a la derecha.

Queda por ver que pasa con los segmentos que contienen a b.



En la figura suponemos que a, b y c son segmentos que están formados por cualquier cantidad de bloques. Sus longitudes son n_a, n_b y n_c respectivamente. Asimismo, las cantidades de puntos alineados son k_a, k_b y k_c. La separación entre los dos primeros segmentos es h_1 y entre los dos últimos es h_2. Todas estas magnitudes son mayores que 0. Suponemos que b es el segmento más significativo. Quisiéramos probar que después de aplicar el principio de exclusión, el segmento abc es menos significativo que a o menos significativo que c. Sin pérdida de generalidad digamos

que es menos significativo que a. Si a fuera seleccionado y se aplicara el principio de exclusión, entonces abc pasaría a ser necesariamente menos significativo que c. Por lo tanto nunca sería seleccionado el segmento abc. Intuitivamente, suponer que esto sea cierto proviene de considerar que si b es más significativo que ab implica que el hueco h_1 es grande. Por lo mismo también debe ser grande h_2. Además el propio b debe ser suficientemente grande para ser el más significativo. Entonces, después de aplicar el principio de exclusión, abc tiene en relación a su tamaño pocos puntos alineados. Esto lo formalizamos con la siguiente conjetura:

Conjetura 1 *Si se cumple*

$$\begin{aligned} B(n_b, k_b, p) < \text{mín}(&B(n_a, k_a, p),\\ &B(n_c, k_c, p),\\ &B(n_a + n_b + h_1, k_a + k_b, p),\\ &B(n_c + n_b + h_2, k_c + k_b, p),\\ &B(n_a + n_b + n_c + h_1 + h_2, k_a + k_b + k_c, p)) \end{aligned}$$

entonces se debe cumplir

$$B(n_a + n_b + n_c + h_1 + h_2, k_a + k_c, p) > \text{mín}\left(B(n_a, k_a, p), B(n_c, k_c, p)\right)$$

Que esto se cumpla fue comprobado experimentalmente en una línea de 3000 puntos, para probabilidades $p = 1/d$ con $d \in \{4, 8, 16, 32, 64\}$ en el caso en que a,b y c sean bloques, es decir que $n_i = k_i$ para $i \in \{a, b, c\}$.

Si tomamos como cierta la conjetura anterior, concluimos que la aplicación del principio de exclusión a segmentos en una línea recta produce segmentos sin puntos de intersección. La importancia de esto es que una vez que se encuentra el segmento más significativo en una línea de longitud l, la búsqueda de los siguientes segmentos más significativos se restringe a búsquedas en dos líneas de longitudes menores que l, y además el NFA de los segmentos candidatos no cambia y no es necesario volver a recorrerlos.

Esto lo aprovechamos para implementar el algoritmo. Supongamos que en la línea se encontraron m bloques. Hay $m \cdot (m+1)/2$ posibles segmentos. Podríamos ordenarlos (en realidad sólo haría falta ordenar aquellos cuyo NFA sea menor que el máximo aceptable), y luego hacer una iteración seleccionando los de menor NFA siempre que no intersecten a alguno de los seleccionados antes. El problema con esto es el tiempo de la ordenación. En lugar de ordenarlos, sólo obtenemos el más significativo y aplicando la técnica 'dividir y conquistar' hacemos lo mismo recursivamente por un lado con los segmentos que están completamente a la izquierda del más significativo y por otro con los que están completamente a la derecha. Esto puede hacerse fácilmente si estructuramos los segmentos en una matriz triangular. Las filas representan los inicios de los bloques y las columnas los finales de ellos. Si el segmento más significativo está en la entrada (i, j), significa que el segmento empieza en el bloque i y termina en el j. La búsqueda debe seguir por un lado entre los segmentos que terminan a lo sumo en el bloque $i-1$ y por otro lado entre los que empiecen en el bloque $j+1$ o después.

	l		i-1				j				
	x	x	x	-	-	-	-	-	-	-	
		x	x	-	-	-	-	-	-	-	
			x	-	-	-	-	-	-	-	
i				-	-	-	min	-	-	-	
					-	-	-	-	-	-	
						-	-	-	-	-	
							-	-	-	-	
								x	x	x	j+1
									x	x	
										x	m

El rectángulo superior derecho corresponde a segmentos que contienen al más significativo y los descartamos por la conjetura 1 . El rectángulo central identifica los segmentos contenidos en el más significativo. Los otros dos rectángulos con entradas marcadas con $-$ son de los segmentos que lo solapan. La búsqueda continúa en los dos triángulos con entradas marcadas con x.

Los algoritmos 5 y 6 sustituyen el ciclo de la línea 14 del algoritmo 4.

Se mantiene una columna con el mínimo NFA de la fila y otra con la columna en la que está. Una vez establecidas estas columnas la búsqueda del mínimo es lineal en el tamaño de la matriz. La matriz inferior derecha no necesita ser actualizada en la llamada recursiva. La superior izquierda sí, porque el mínimo podría estar en una columna que queda fuera del rango.

Algoritmo 5 Principio de exclusión en una línea

```
1: para cada [inicio de bloque,fin de bloque]
2:     calcular el NFA del segmento
3:     almacenar el NFA en la matriz triangular
4:     si NFA es el mínimo de la fila entonces
5:         actualizar el valor del mínimo de la fila y la columna en la que está.
6:     principio de exclusión entre bloques 1 y m
```

Algoritmo 6 Principio de exclusión entre bloques

```
1: si bloque inicio <= bloque fin entonces
2:     encontrar la fila r, y la columna c, en la que está el mínimo
3:     si el mínimo es menor que el máximo aceptable entonces
4:         almacenar el segmento [r,c]
5:         para filas entre bloque inicio y (r-1)
6:             actualizar el mínimo de la fila entre las columnas inicio y (r-1)
7:         principio de exclusión entre bloques inicio y r − 1
8:         principio de exclusión entre bloques c + 1 y fin
```

Sólo los segmentos obtenidos con este algoritmo pasan a la estructura descrita en 3.2. Esta se hace más chica, es menos probable que necesite hacer accesos a memoria secundaria, y decrece el segundo término de la ecuación (2). Debe quedar claro que parte de este costo fue pagado en el tiempo que lleva aplicar este algoritmo en cada línea.

En el cuadro 1 se puede ver la cantidad de segmentos con los que se construye el árbol para aplicar el principio de exclusión. La primera columna es para el árbol en que se insertan todos los segmentos significativos que se detectan. En la segunda están los que quedan después de aplicar el algoritmo de esta sección a cada línea. En cada columna esta además la cantidad de reinserciones que deben realizarse de acuerdo a lo explicado en la sección 3.

	Todos		Por línea	
	Tamaño	Reinserciones	Tamaño	Reinserciones
route	9698	75	1224	21
uccello1a	30301	54	1492	11
cimage	25142	358	1117	25
fimage	84903	497	4481	86
peine1	1203	13	1157	13
peine2	1408	13	1268	13
trazos	30775	46	1915	10
aerial	338545	3861	12720	217
uccello	482174	4970	21200	593
appell	560138	3974	10288	244
building	286147	2160	13422	192
building2	382820	3974	9808	363
fdm1	645150	10041	12688	382
fdm2	359614	7282	8720	252
dellmousepad	331107	4755	2683	70

Cuadro 1: Cantidad de segmentos para el principio de exclusión.

6. Experimentos.

En lo que sigue los programas fueron ejecutados con los mismos parámetros predeterminados de *align_mdl*, que también son los de todos los otros programas.

Estos programas están escritos en *C* utilizando *Megawave2*. Están codificados con el objetivo de hacerlos eficientes. Esto implicó dejar de lado técnicas de ingeniería de software que permitirían un mejor mantenimiento. Un ejemplo de esto es que se trató de evitar la llamada a funciones.

En el cuadro 2 está la cantidad de segmentos que devuelven los algoritmos. Dos de ellos utilizan la primera versión del principio de exclusión dada en [DMM04, capítulo 6] y otros dos utilizan la otra.

La conocida como *EP, algorithm 4*, es la que implementa *'align_mdl'*. También es la del algoritmo de la columna *Modificado* pero este utiliza el recorrido descripto en este documento y además, cuando vuelve a recorrer los segmentos, si corrige la cantidad de puntos alineados, también puede modificar la longitud del segmento en el caso en que los puntos de alguno de los extremos hayan sido asignados a otro segmento. Esto permite recuperar algunos segmentos más. En particular con *trazos.rim* ahora se generan los segmentos correspondientes al trazo que antes desaparecía. Y tiene otra modificación al competir por los puntos del segmento dilatado. En *align_mdl* el segmento compite si el punto está alineado por los tres puntos. En esta versión compite por los dos puntos adyacentes sólo si ellos también están alineados. El motivo de esto es que el objetivo del segmento dilatado es eliminar los segmentos paralelos y oblicuos que tienen una diferencia en la orientación menor al error en la precisión. Al no tener en cuenta la orientación de los dos puntos laterales, un segmento muy significativo puede hacer que reduzca en dos el conteo de puntos alineados de un segmento que es perpendicular a él, lo cual no es lo que se pretende.

Las otras dos columnas corresponden a las versiones del algoritmo *EP-bis, algorithm 5*. La encabezada con *Todos* es en la que todos los segmentos significativos encontrados al escanear las líneas se incluyen en la estructura a la cual se le aplica el principio de exclusión. En la encabezada con *Por línea*, para mejorar el rendimiento en tiempo de ejecución y en memoria necesaria, se aplica el principio de exclusión por línea como se describe en el capítulo 5 y sólo los segmentos retornados por ese algoritmo se incluyen en la estructura.

Entre las dos versiones del algoritmo 4 de [DMM04] vemos que la nueva recupera algún segmento más. Pero al compararlo con las que implementan el algoritmo 5 de [DMM04] comprobamos que la modificación no era suficiente. Estas dos últimas versiones devuelven muchos más segmentos, en algunos casos alrededor del doble. Como se esperaba la versión que incluye todos los segmentos significativos en el árbol es la más exacta.

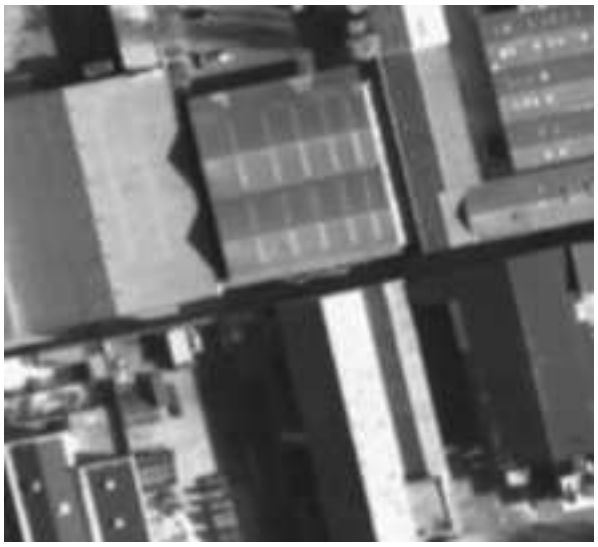
	EP, Alg. 4		EP bis, Alg. 5	
	align_mdl	Modificado	Todos	Por línea
route	18	16	20	20
uccello1a	55	56	70	71
cimage	27	29	39	39
fimage	45	52	87	87
peine1	22	22	22	22
peine2	19	19	19	19
trazos	43	45	44	44
aerial	174	202	319	310
uccello1	166	214	456	438
appell	278	417	512	492
building	202	231	346	339
building2	185	269	372	367
fdm1	320	482	658	640
fdm2	196	262	349	341
dellmousepad	41	36	71	69

Cuadro 2: Cantidad de segmentos producidos.

Tomemos un ejemplo de la mayor producción de segmentos. En la figura 3 vemos una sección de la imagen *aerial* y los correspondientes segmentos generados por las versiones *align_mdl* y *Todos*. En la imagen, en la parte superior y central hay un techo en el que se perciben segmentos claros casi verticales. Los segmentos correspondientes no son generados por *align_mdl*. Sin embargo utilizando la otra versión del principio de exclusión son producidos.

En el cuadro 3 tenemos el tiempo de ejecución de las cuatro versiones. Hay una gran diferencia de velocidad entre las dos versiones que usan el algoritmo 4 de [DMM04] en favor de la nueva. Esto sólo puede ser causado por la nueva forma de recorrer la imagen.

Las dos versiones que implementan el algoritmo 5 de [DMM04]son aún más rápidas. La que aplica el principio de exclusión en cada línea haciendo una selección previa, es la más rápida. Esto se debe a la estructura. Se comprueba con una imagen muy grande como *dellmousepad*, que además tiene pocos segmentos. En este caso el recorrido, que es igual en las dos versiones, tiene un peso relativo mayor. Como consecuencia la diferencia en el tiempo de ejecución es menor que



(a) Fondo

(b) align-mdl

(c) Todos

Figura 3: Aerial.

en las otras imágenes.

Sin considerar los mejores resultados en la producción de segmentos, la versión de la columna *Todos* es además más rápida que la de la columna *Modificado*. Y esto ocurre a pesar de que en ella hay que tratar con las inserciones y eliminaciones del árbol. La explicación es que en *Modificado* cada segmento almacenado para aplicarle el principio de exclusión, es recorrido dos veces más. La primera de ellas es recorrido como segmento dilatado, lo que implica casi el triple de tiempo. Entonces podemos decir que el tiempo es proporcional a $4 \cdot cantidad\ de\ segmentos$. En cambio en *Todos* todos los segmentos se recorren una vez, y sólo los que se incluyen en la salida son recorridos una segunda vez, ahora sí como segmentos dilatados. Y estos son relativamente pocos. Como también hay que volver a recorrer los que se reinsertan el tiempo es $cantidad\ de\ segmentos + reinserciones + 3 \cdot retornados$.

		EP, Alg. 4		EP bis, Alg. 5	
	Dimensiones	align_mdl	Modificado	Todos	Por línea
route	128 X 128	11.719	1.975	1.504	1.293
uccello1a	220 X 220	41.160	8.486	6.307	5.078
cimage	256 X 256	50.104	10.566	8.818	8.307
fimage	256 X 256	1m14.773	16.859	12.262	7.445
peine1	256 X 256	31.016	3.564	3.473	3.715
peine2	256 X 256	31.123	3.582	3.512	3.650
trazos	439 X 552	3m2.107	16.424	14.961	14.326
aerial	512 X 512	5m14.256	1m36.219	1m12.924	32.621
uccello1	640 X 477	8m53.844	2m8.971	1m39.971	40.350
appell	896 X 600	13m51.301	4m1.018	3m0.152	1m33.322
building	896 X 600	9m48.135	1m59.930	1m45.188	1m1.875
building2	600 X 896	10m35.180	2m50.529	2m25.373	1m15.141
fdm1	600 X 896	12m25.188	4m21.629	3m10.332	1m29.637
fdm2	600 X 896	11m28.285	2m50.119	2m20.115	1m26.102
dellmousepad	1600 X 1200	27m26.105	6m35.492	5m33.785	5m15.291

Cuadro 3: Tiempo de ejecución.

En el cuadro se incluye una columna con las dimensiones de la imagen. De lo que se trata es de verificar la hipótesis de que el tiempo de ejecución está en principio linealmente relacionado con el tamaño de la imagen. En la figura 4 se ven una curva para cada una de las versiones, en las que lo que se grafica es la división de la cantidad de puntos entre el tiempo de ejecución. Cada curva está multiplicada por un factor diferente teniendo en cuenta la diferencia de velocidades. Las imágenes que se apartan más del promedio son *peine1*, *peine2* y *trazos*. Una primera explicación es que en ellas muy pocos puntos tienen un gradiente con norma suficientemente grandes como para ser considerados definidos. Las dos primeras son imágenes digitales y la tercera corresponde a un dibujo hecho a mano con la intención de que haya segmentos.

En esta sección se incluye la utilización de memoria. Lo que vemos en el cuadro 4 es una columna con las distintas estructuras, otra con la expresión con la que se calcula su dimensión y otras dos con los datos correspondientes a la imagen *appell*. En ellas $diagonal$ es la longitud de la más grande diagonal de la imagen. La cantidad máxima de bloques que podría haber en una línea es $maxblocs = diagonal/2 + 1$. La estructura *one-segment* que se usa para almacenar los atributos de un segmento ocupa 19 bytes. $heap$ es el árbol para aplicar el principio de exclusión. seg es un array donde se almacenan los segmentos. $angle$ y $norm$ son el ángulo y la norma del gradiente(rotado $\pi/2$) de los puntos de la imagen. $test$ mantiene los resultados de *tail-binomial*. $countbloc, startbloc, endbloc, posk, calign$ son estructuras para la construcción de bloques. $line - exclusion$ es la estructura para aplicar el principio de exclusión en una línea. Esto no es utilizado por la versión *Todos*. Esta última estructura y las que sirven para construir los bloques son liberadas antes de construir $heap$, por lo que debe tenerse en cuenta para calcular la necesidad de memoria.

Podemos ver que la mayor velocidad de estos algoritmos tiene su costo en la necesidad de memoria para $test$ que es la estructura más grande. Como el ciclo donde se utiliza las diferentes precisiones es el más interno, deben estar en memoria todos los datos. Esto no pasa en *align_mdl*. Ahí el ciclo externo es el de precisión, por lo que al comenzar se calcula $test$ sólo para la precisión actual.

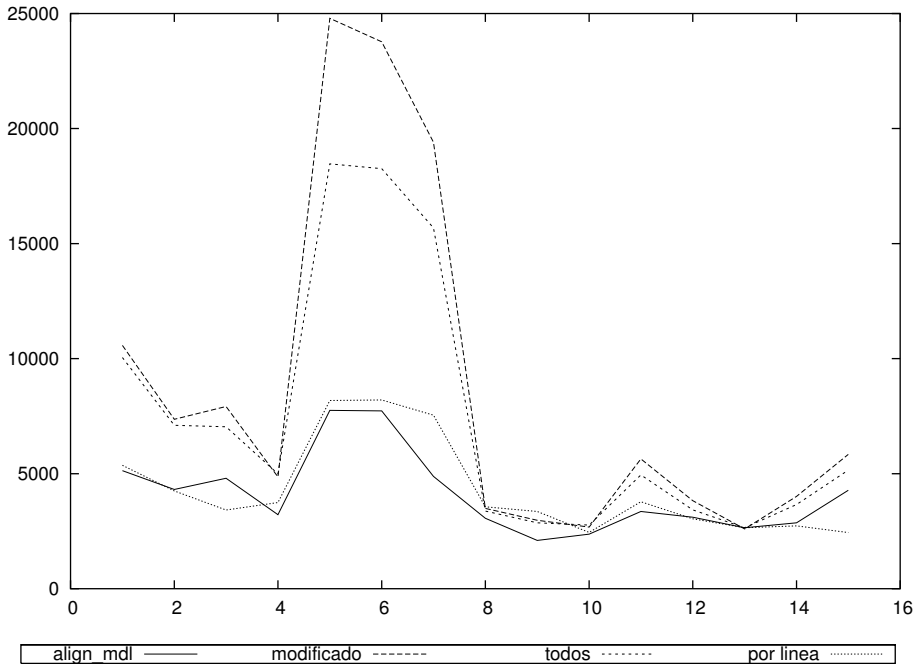


Figura 4: Dimensión/tiempo.

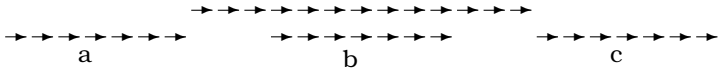
Estructura	Expresión	Todos	Por línea
heap	$segmentos \cdot 4$	2240552	41152
seg	$segmentos \cdot 19$	10642622	195472
angle	$(columnas + 2)(filas + 2) \cdot 4$	2162384	2162384
norm	$(columnas + 2)(filas + 2) \cdot 4$	2162384	2162384
test	$precisiones \cdot (diagonal + 1)^2 \cdot 8$	27941784	27941784
countbloc	$maxblocs \cdot 2$	1080	1080
startbloc	$maxblocs \cdot 2$	1080	1080
endbloc	$maxblocs \cdot 2$	1080	1080
posk	$2 \cdot precisiones \cdot diagonal \cdot 2$	6468	6468
calign	$2 \cdot precisiones \cdot 2$	12	12
ngrad	$precisiones \cdot 2$	6	6
gthreshold	$precisiones \cdot 4$	12	12
prec	$precisiones \cdot 4$	12	12
line-exclusion	$maxblocs^2 \cdot 8 + maxblocs \cdot 14$	0	2340360
Máximo		45149756	34812134

Cuadro 4: Utilización de memoria.

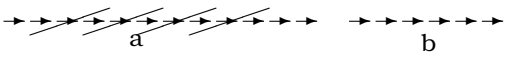
7. Conclusión.

Se implementó la segunda versión del principio de exclusión, la cual resuelve el problema de la exclusión mutua. La implementación consiste en una cola de prioridad. Se hizo la suposición de que pocas veces sería necesario reinsertar el segmento que encabeza la cola y empíricamente resultó correcta. Esto provoca una mejora en el rendimiento del algoritmo, contrariamente a lo que se podría pensar, que era que esta versión del principio de exclusión era más costosa que la anterior. Además de esto se implementó el recorrido de la imagen de una manera diferente para mejorar aún más el rendimiento. También hay otra versión en la que se aplica el principio de exclusión en cada línea. Tomando como válida una conjetura acerca de *tail-binomial*, que fue comprobada numéricamente para un rango de valores de sus parámetros suficiente para las aplicaciones prácticas, se encontró un algoritmo que lo resuelve eficientemente. Esta versión, que aquí denominamos *Por línea* podría preferirse a la más general debido a la mayor rapidez.

Sin embargo es posible que con ella se pierdan segmentos. En la siguiente figura suponemos que las dos líneas paralelas son consecutivas, que los segmentos a, c y abc son significativos, y que este último es el más significativo de los tres. También suponemos que el segmento de la línea superior es aún más significativo y que al tratarlo como segmento dilatado y aplicar el principio de exclusión, el segmento abc deja de ser significativo. Al usar *Todos*, el segmento se descarta, pero como también en el árbol fueron insertados a y c, estos últimos estarán en la salida. Pero con la versión *Por linea* esos segmentos habían sido descartados en el propio tratamiento de la línea.



Pero el problema de la siguiente figura afecta también a la versión *Todos*. Aquí suponemos que ab es significativo pero que b no lo es. Las líneas oblicuas representan segmentos más significativos que ab y que por lo tanto le hacen perder tres puntos alineados cada uno, provocando que deje de ser significativo. Pero podría ocurrir que la parte derecha de a, la no afectada, junto con b formen un segmento significativo. Esto no es detectado por el algoritmo, ni siquiera con la mejora de corregir la longitud, porque esta no cambia ya que los puntos extremos no son afectados. Lo mismo pasaría si b, en lugar de ser un bloque fuera parte del bloque a. El problema surge por la utilización de bloques. El bloque enmascara segmentos y también inicios y fines de segmentos.



Una solución al problema presentado en las dos figuras anteriores consistiría en volver a procesar los segmentos descartados. Se lo vuelve a recorrer, ahora teniendo en cuenta la posible asignación de los puntos a otros segmentos, reconstruyendo bloques y segmentos significativos. Y esos se insertan en el árbol. Esto presenta un par de problemas. Uno tiene que ver con la estructura del árbol, que como se construye del tamaño exacto dado por la cantidad de segmentos almacenados, podría llegar a salirse del rango. El otro problema es el del tiempo de ejecución. Casi todos los segmentos son descartados por lo que el tiempo se incrementaría mucho, y esto sería para resolver un problema que tal vez sea de escasa dimensión. Aunque en la versión *Por linea*, en la que hay relativamente pocos segmentos, podría resultar ventajoso.

Pero hay un problema que tiene que ver con el modelo. En la figura 5 vemos los segmentos obtenidos de la imagen *peine2*.

De los tres segmentos horizontales producidos, sólo es percibido el de más abajo. Los otros dos sustituyen a secuencias de segmentos que se perciben como los dientes del peine. Sin embargo el algoritmo encuentra, correctamente, que el segmento largo es más significativo que cada uno de los cortos. Pero si tuviéramos en cuenta que la probabilidad de que haya una secuencia de bloques, todos de la misma longitud y separados por la misma distancia, es menor que la del segmento largo, se debería considerar que lo percibido es la secuencia y incluir en la salida todos los segmentos cortos. El problema es que el modelo busca un solo segmento por vez.

Referencias

[DMM04] Agnès Desolneux, Lionel Moisan, and Jean-Michel Morel. *A theory of digital image analysis.* Julio 2004. http://www.cmla.ens-cachan.fr/Utilisateurs/morel/lecturenote.pdf.

[Wei95] Mark Allen Weiss. *Estructuras de datos y algoritmos*. Addison-Wesley Iberoamericana,S.A., 1995. ISBN:0-201-62571-7.

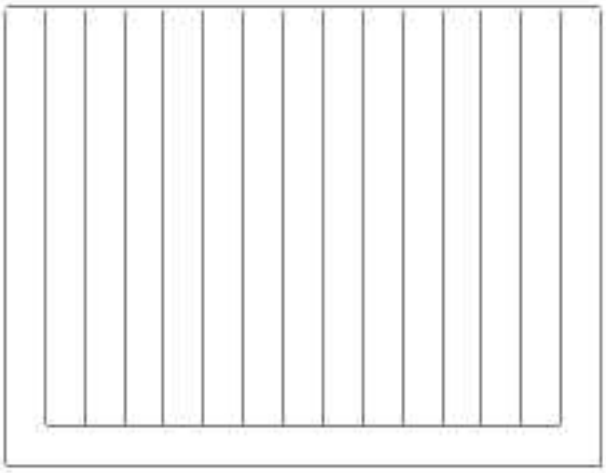


Figura 5: Peine.