Sistema de Reconocimiento de Caracteres Orientado a Sistemas Android

Oriol Segura Morales

Resumen—Este proyecto trata sobre un sistema de reconocimiento de caracteres que se implementa para sistemas Android en forma de aplicación para móvil y Tablet. La aplicación está orientada para que los niños aprendan a escribir de una forma más amena. El sistema utiliza el patrón de diseño Modelo-Vista-Controlador para separar la lógica del sistema de la interfaz visual. En la interfaz, el sistema pide que se escriba un carácter utilizando la pantalla táctil y una vez el usuario lo ha escrito el sistema hace el reconocimiento y dice si es o no el carácter correcto. Para el reconocimiento del carácter se pasa por tres fases. La vectorización, que filtra los puntos encontrados dejándolos de manera que sea más fácil reconocer los elementos a los que pertenecen los puntos. El constructor del grafo, que es el que pasa los puntos filtrados y construye un grafo que será el que posteriormente será reconocido. Y el algoritmo de correspondencia de grafos, que se encarga de encontrar la correspondencia entre el grafo de entrada y el de las letras.

Palabras clave—Graph-Matching, String-Matching, AC4, A*, Android, reconocedor de caracteres.

Abstract— This Project proposes in a character recognition system which is aimed for Android systems as an application for both mobile phones and tablets. This application is educational-oriented with the pursuit of teaching children to write in an entertaining way. The system uses the MVC (Model View Controller) pattern to separate the logic of the system from the visual interface. In this interface, the system asks for the typing of a character using the touchscreen and once the user has typed the character asked, the system recognises and says if the character that has been typed is the correct one or not. To allow the system to recognise the character, we have to explain three different steps. Vectorisation, which filters the given points to make easier the recognition of all the elements to which those points belong to. The graph constructor, which examines all the filtered points and built a graph that will be the one recognised later and, finally, the algorithm, which is the responsible for finding out the correspondence between the first graph and the one related to the letters.

Index Terms— Graph-Matching, String-Matching, AC4, A*, Android, character recognizer.

1 INTRODUCCIÓN

ÉSTE Trabajo de final de grado consiste en el desarrollo de un reconocedor de caracteres que funcione como una aplicación para los sistemas Android, tanto para Smartphone como para Tablet. Puesto que, en la actualidad, en la mayoría de hogares hay un Smartphone o una Tablet al que puede acceder cualquier miembro de la familia, se quiere implementar una aplicación que sirva para enseñar a escribir de manera interactiva. Ésta aplicación está pensada para que la utilicen niños pequeños que están aprendiendo a escribir, de manera que escribirían las letras que la aplicación les pidiera como parte de un juego. Así, los niños no se cansarían tan rápido de escribir una y otra vez las letras y mejorarían el rendimiento académico. La complejidad del sistema nos ha hecho enfocar en los algoritmos de correspondencia de grafos y quedan por mejorarla interfaces a fin de hacerlas más atractivas para los niños.

En este documento se explicará la funcionalidad del sistema. Primero se mostrará una visión general de todos los diferentes módulos que contiene. También se hablará de las consideraciones previas que tiene este proyecto. Después se irán explicando las características de cada módulo por separado. Luego se detallarán los experimentos a los que ha sido sometida la aplicación y por último se explicarán posibles mejoras que se podrían llevar a cabo en el futuro.

2 ESQUEMA DEL SISTEMA

Para la implementación del sistema, se ha seguido el patrón MVC (Modelo-Vista-Controlador) que separa la parte lógica del sistema de la parte visual como se puede ver en la figura 1. Se ha decidido seguir este patrón por las siguientes razones:

- El modelo, que es el que contiene el reconocedor de caracteres, se puede reutilizar para otras aplicaciones u otros juegos en la misma aplicación sin tener que modificarlo. Solo se es necesario modificar la interfaz.
- La posibilidad de que el modelo no esté dentro de la aplicación sino en un servidor externo. Esto se puede conseguir simplemente modificando el controlador.

- *E-mail de contacto: oriol.segura@e-campus.uab.cat*
- *Mención realizada: Computación*
- *Trabajo tutorizado por: Maria Vanrell Martorell (Ciencias de Computación)*
- *Curso 2013/14*

- La facilidad de hacer pruebas unitarias de los componentes.
- No tener que preocuparse de las otras partes cuando modificas una.

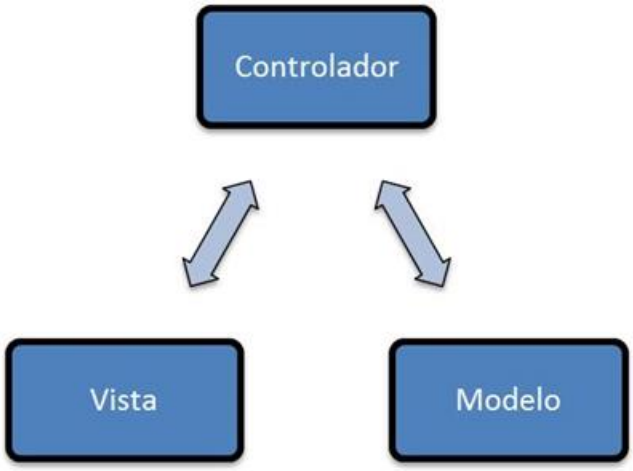


Fig. 1. Representación del patrón de diseño MVC

Como consideraciones previas cabe destacar que esta aplicación se ha hecho para Android, enfocada para la versión 4.4 (Kit Kat). El lenguaje utilizado es Java por su afinidad con Android y como librerías importantes destacan Math, InputStream y Scanner. Un detalle importante de esta aplicación es que solo reconoce letras en mayúscula.

2.1 Vista

La vista es la parte con la que interactúa el usuario con la aplicación, es la que recibe la respuesta del usuario y muestra por pantalla el resultado. Tiene dos partes diferenciadas, una es la interfaz y la otra la recogida de puntos.

La interfaz, es por donde se inicia la aplicación se ha diseñado como si fuera una pizarra de la escuela, también se ha intentado hacer muy sencilla para que cualquiera pueda usarla. Tiene una modalidad de juego que se trata de supervivencia. El jugador tiene que ir escribiendo las letras que la aplicación le pide, por cada letra que acierte se incrementa la puntuación pero si falla se guarda la puntuación del jugador y se reinicia el juego.

Para que la aplicación pueda pedir las letras sin mostrarlas de ninguna manera, ya que si las muestra le estaría enseñando el resultado al jugador, se utilizan grabaciones de voz.

La recogida de puntos, que es una de las partes que se nos proporcionó hechas se encuentra dentro de la interfaz.
Cuando se pasa el dedo o el lápiz por encima de la pantalla táctil, el sistema de recogida de puntos va guardando los puntos cada un pequeño intervalo de tiempo. Todos los puntos que se generan sin dejar de tocar la pantalla se consideran del mismo trazo.
Una vez se ha escrito una letra y recogido los puntos, se obtiene el valor máximo y mínimo global de todos los valores de la posición (x, y), se envía la información al controlador y se espera una respuesta.

2.2 Controlador

El controlador es la parte que comunica la vista con el modelo, recibe los puntos recogidos por la interfaz y se los envía a la lógica de sistema, pero antes de enviárselos cambia el formato de los puntos al que el modelo utiliza.
También, el controlador se encarga de cargar las letras desde un fichero al diccionario con las que el reconocedor de caracteres tendrá que comparar.
Una vez se ha enviado la información al modelo, el controlador se queda esperando la respuesta del modelo, y cuando ya la tenga se la envía a la interfaz para que muestre el resultado.

El diccionario es el nombre que se le ha puesto a todo el conjunto de letras que la aplicación puede reconocer. Estas letras están guardadas en forma de grafo en un fichero "letras.txt". La aplicación lee el fichero, monta el grafo de cada letra y lo almacena en una lista. El proceso de leer el fichero y montar los grafos solo se ejecuta una vez. El formato del fichero es el siguiente:

```
{}
Letra: V
Numero_de_Nodos: 3
Nodo: 1
Tipo: 1
Conectados: 2
Nodo: 2
Tipo: 5
Conectados: 1 3
Nodo: 3
Tipo: 1
Conectados: 2
```

Como se puede ver el formato es muy simple, solo se tiene que escribir la letra, el número de nodos, y por cada nodo un identificador, el tipo y los nodos con los que está conectado. Esta manera de representar los grafos, aunque no sea la más óptima, se ha elegido por la gran simplicidad al hacer cambios en un grafo.

En la versión inicial la intención era que el diccionario se cargara desde el modelo, pero en las aplicaciones Android, se necesita el contexto de la aplicación (Context) para poder utilizar algún recurso, que en nuestro caso es el fichero "letras.txt".

2.3 Modelo

El modelo es la parte más importante de la aplicación, es donde se construye el grafo a partir de los puntos y luego se compara utilizando algoritmos de Graph-Matching y una vez encontrada la letra, se envía la respuesta al controlador.

Fig. 2. Flujo de datos dentro del modelo

El modelo está formado por tres bloques tal y como se muestra en la fig. 2. El primer bloque es la vectorización, que es el que se encarga de filtrar los puntos de entrada y dejarlos como vectores.
El segundo bloque es el de construcción del grafo, que recoge los puntos filtrados de la vectorización y monta un grafo que simula la letra que se ha escrito.
Por último el algoritmo utiliza el grafo construido por el bloque anterior y lo compara con los grafos que ya tiene insertados de las letras utilizando algoritmos de Graph-Matching. Una vez se ha decidido de qué letra se trata, envía la respuesta al controlador.

3 VECTORIZACION

El código de este módulo se nos fue entregado como parte de otro proyecto, la idea era reutilizar el código y así centrar más los esfuerzos en la implementación de los algoritmos, pero sólo se pudo reutilizar una parte pequeña y por tanto se tuvo que dedicar más tiempo del esperado.

El módulo de vectorización recibe un ArrayList de puntos de la clase Punto, ésta clase contiene las coordenadas x e y que nos da la posición del punto, la velocidad y el tiempo en el que se ha dibujado desde el inicio. Sobre esta lista de puntos se aplicarán dos filtros.
El primer filtro consiste en calcular la tendencia de un punto con el anterior y el siguiente y determinar que si la diferencia máxima permitida de ángulo respecto a los 180^{o} es menor a un umbral, se considera que esos puntos están formando una recta.
El resultado de este filtro es un punto inicial y final de cada recta y la eliminación de los puntos intermedios.

El segundo filtro que se aplica es el de proximidad, se utiliza una ventana que va recorriendo la matriz de la imagen y si encuentra más de un punto dentro de la ventana, calcula la distancia entre los dos puntos y si es menor a un umbral, los une creando un nuevo punto en la posición intermedia y luego eliminando los dos puntos.

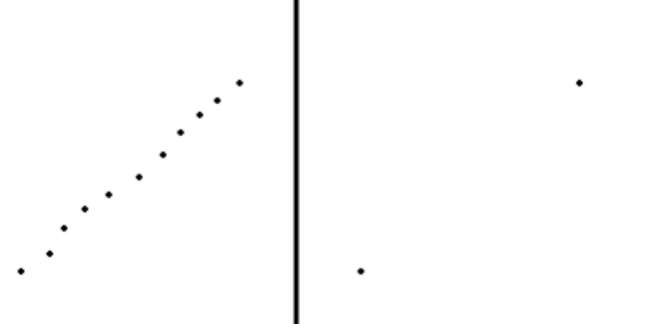


Fig. 3. Ejemplo de la aplicación del filtro 1. A la izquierda lista de puntos alineados puntos antes del filtro y a la derecha la recta despues del filtro solo con los puntos inicial y final.

4 CONSTRUCCIÓN DEL GRAFO

En esta sección se explica cómo funciona el constructor del grafo que queremos construir. En este grafo, los enlaces no están dirigidos de manera que un nodo conectado a otro representa que están dibujados juntos en la letra escrita por el usuario. Los nodos representan un elemento de la letra escrita, podríamos decir que es una pequeña parte del dibujo tal y como se puede ver en la figura 4.
Las letras se representan en forma de grafo porque mantiene la integridad estructural del dibujo.

4.1 Formas de reconocimiento

Cuando se decidió la manera en como reconocer los elementos a partir del ArrayList de puntos que posteriormente serían añadidos como nodos, se pensaron dos soluciones.
Solución 1. Guardar el ArrayList de puntos en una matriz para formar la imagen y así poder aplicar algoritmos que detectaran las rectas, las esquinas o los cruces, directamente sobre la imagen.
Solución 2. Mantener la lista de puntos y aprovechar el orden del ArrayList ya que de esta manera se puede saber cómo se ha ido dibujando el carácter punto por punto. También se pueden mantener los trazos separados y ordenados.

Al final se decidió tomar la segunda opción ya que poder reproducir como se ha ido dibujando el carácter da una idea de que es lo que se quería dibujar y es más fácil poder corregir los casos en que la vectorización no puede filtrar.

4.2 Tipos de nodos

Se han definido ocho tipos de nodos definidos en la Fig. 4. para poder reproducir las letras en mayúsculas del abecedario (Ver Apéndice 1). Todos los grafos estarán construidos con estos nodos.

Se ha de tener en cuenta, que en el momento de construir el grafo pueden haber errores que son debidos al gran parecido entre los tipos de nodos.
El nodo de tipo 1 es igual al de tipo 2 con la única diferencia que uno es más largo que el otro. El valor que define si es de tipo 1 o de tipo 2 viene marcado por un umbral. Una similitud muy grande también la tienen los nodos de tipo 4 y 5 ya que se determina de qué tipo de nodo se trata según el ángulo. Otro error viene dado por el parecido entre los nodos de tipo 6 y 8, ya que si una

recta sobrepasa a la otra más de lo que debe, siempre se considera como un nodo de tipo 8 cuando podría ser que fuera uno de tipo 6.

Fig. 4. Tipos de Nodos con su representación gráfica.

4.3 Funcionamiento

El módulo de construcción del grafo trabaja a partir de los puntos ya filtrados por la vectorización. Pasa por todos los puntos del ArrayList y según donde está situado el punto respecto a sus vecinos y el elemento definido anteriormente hace una acción u otra.

Para el funcionamiento del constructor del grafo se utiliza el nodo tipo 1 (Terminal largo) para simular que es una recta que conecta con otros nodos de intersección. Por tanto el nodo de tipo 1 no se utiliza como terminal durante la construcción del grafo sino como una recta.

Cuando el constructor detecta que un punto está a una distancia del siguiente punto mayor que un umbral, lo identifica como un nodo de tipo 1 y se lo guarda. Cada vez que se añade un nodo nuevo al grafo, se tiene que mirar como interactúa con el resto de nodos del grafo. Si el siguiente nodo que detecta es otro de tipo 1, se tiene que comprobar si las rectas se cruzan o si están cerca una de la otra. Como resultado se puede dar uno de los siguientes casos que se verán en los siguientes apartados.

4.3.1 Nodo tipo 1

Si dos rectas que han sido dibujadas una después de la otra de forma seguida, tienen la misma dirección y un extremo del trazo dibujado está muy cerca del otro, quiere decir que se trata de la misma recta, pero ha tenido una variación en la dirección en el momento de ser dibujada y esto ha hecho que el módulo de vectorización la separara en dos. Por tanto lo que hace el constructor del grafo es unir las dos rectas quedándose con los puntos que no están en contacto de cada una.

4.3.2 Nodo tipo 2

Este tipo de nodo se identifica una vez que ya se ha determinado que el elemento que se está tratando es una recta. Lo único que se ha de hacer el calcular la distancia que tiene la recta y si es inferior a un umbral, se considera la recta como nodo de tipo 2. El valor del umbral no puede ser fijo, sino que tiene que ser relativo al tamaño de la letra, y esto se consigue con el valor máximo y mínimo global de todos los puntos.

4.3.3 Nodo tipo 4 y 5

En el caso en que dos rectas tengan un punto muy cercano al de otra recta, si el ángulo que forman estas rectas es menor a un umbral quiere decir que es un nodo de tipo 5, en cambio si es mayor es de tipo 4.

Una vez identificado el nodo, e tiene que conectar con los dos nodos de tipo 1 o 2.

4.3.4 Nodo tipo 6 y 8

Para detectar estos tipos de nodos, primero se tiene que tener en cuenta que las rectas que se calculan a partir de los puntos son rectas que dibujadas en un plano llegarían hasta infinito. Por eso se define como recta imaginaria a la recta que llega hasta infinito y como recta real al segmento de recta que ha sido dibujado. Estas dos rectas son la misma pero la diferencia está en si han sido dibujadas o no. Dicho esto, el primer paso que se hace es calcular la intersección de la recta imaginaria, luego se calcula la distancia del punto de intersección a los puntos de los extremos de las rectas reales y se compara con un umbral. Si solo un punto de los cuatro es menor al umbral, quiere decir que estamos en una intersección de tipo “T” y por tanto el nodo debe de ser de tipo 6. En cambio sí dos puntos son menores al umbral, significa probablemente no se trate de una intersección si no que sea un nodo de tipo 4 o 5 que se determinan como se ha explicado en el apartado anterior. Si ninguna distancia es menor al umbral, se tiene que mirar si el punto de la intersección pertenece a las rectas reales o no, ya que si no pertenece quiere decir que las rectas reales están lejos una de la otra, mientras que si pertenece significa que el elemento a insertar es un cruce y por tanto un nodo de tipo 8.

Una vez se ha identificado el nodo, si es de tipo 4 se tendrá que separar un nodo de tipo recta en dos, entre medio de estos nodos irá conectado el de tipo 4 y la tercera recta también se tendrá que conectar. Si el nodo es de tipo 8, se tiene que dividir las dos rectas dejando así cuatro rectas unidas al nodo de tipo 8.

4.3.4 Nodo tipo 3 y 7

En la versión actual del constructor de grafo, no se pueden detectar estos dos tipos de nodos. El problema que aparece con el nodo de tipo 3 es intentar diferenciar entre dibujado un contorno redondeado y un pequeño giro del dedo o del lápiz. Cuando se va dibujando la letra, a veces se van haciendo pequeños movimientos curvos involuntarios lo suficientemente grandes como para que el constructor de grafo considere como contornos redondeados.
Otro problema con los nodos de tipo 3 es el tamaño de la curva, ya que si la curva es muy grande, los puntos del trazo se dibujan más rápido y quedan más separados mientras que si es una curva pequeña, los puntos están mucho más juntos. Estas dos situaciones tan diferentes dificultan mucho poder reconocer este tipo de nodo y por

tanto no se ha podido implementar esta parte.

El problema con el nodo de tipo 7 es que en todo el abecedario solo se utiliza para la letra "Y", y se ha considerado que el tiempo que se debería invertir para poder detectar este elemento es demasiado para solo una letra.

4.3.5 Reducción del grafo

Una vez se ha construido el grafo, se aplica un proceso de reducción de nodos. El objetivo es depurar la construcción del grafo eliminando aquellos nodos que no aportan información en la representación.

Los nodos que no nos aportan información son los de rectas. Al construir el grafo, son necesarios para saber si una recta nueva tiene algún tipo de interacción con otra ya insertada en el grafo. Pero una vez construido el grafo, los nodos de tipo recta que hay entre dos nodos que sean

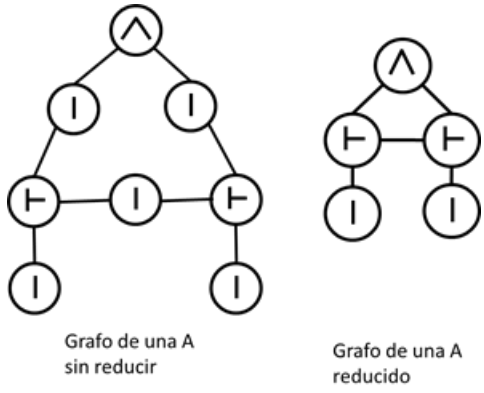


Fig. 5. Ejemplo de reducción de un grafo

de conexión (tipos 4, 5, 6, 7 y 8) se pueden eliminar ya que se intuye que entre dos nodos de tipo conexión habrá alguna recta. Lo único que se ha de hacer es conectar los dos extremos del nodo recta entre sí. Se puede ver un ejemplo de reducción en la figura 5.

5 GRAPH-MATCHING

Para hacer la parte del Graph-Matching se han ido probando diferentes algoritmos y diferentes variantes de un mismo algoritmo. En esta sección se explicarán todos los que se han implementado o estudiado.

Una vez se ha construido el grafo d la letra escrita por el usuario, entonces debemos pasar a identificar el grafo correspondiente del diccionario. Para ello deberemos tener en cuenta dos factores.

El primer punto a tener en cuenta es que no se puede considerar el orden de entrada de los nodos, puesto que cada persona puede escribir la letra en un orden diferente, por ejemplo al escribir la letra "A", se puede escribir en dos trazos o en tres, empezando desde arriba o desde abajo. Todas estas maneras de escribirla son válidas y dan una ordenación diferente de los nodos aun siendo el mismo grafo.

Un segundo factor a tener en cuenta es el error de los grafos. Un grafo de entrada puede haber sido entrado de manera erronea, por ejemplo un nodo puede ser de tipo terminal largo cuando debería ser de tipo terminal corto, o un nodo de ángulo grande cuando debería ser uno de ángulo pequeño. Estos errores son muy frecuentes, y por tanto hace que dos grafos que representan la misma letra no sean exactamente iguales.

Un tercer factor más a tener en cuenta es el tamaño de los grafos. Si al escribir una letra, dos rectas que tendrían que unirse, están más separadas que lo que permite el umbral, se crearán dos nodos cuando solo debería haber uno. Esto provoca que un grafo tenga más nodos que el otro o al revés.

Dicho esto, se puede ver que comparar grafos no es tan simple como parece y que por eso es necesario un algoritmo que tenga en cuenta estos factores en el momento de comparar dos grafos.

5.2 Algoritmo Graph-Matching básico

El algoritmo al que se hace referencia aquí, es un algoritmo que se basa en la fuerza bruta, calculando todas las posibles permutaciones que pueden realizarse entre la pareja de grafos que quieren compararse.

5.2.1 Funcionamiento

Este algoritmo funciona de la siguiente manera:
Dados dos grafos G1 y G2 con el mismo número de nodos y con todos los nodos etiquetados, se busca la substitución de los nodos de G1 en G2 que hace que los dos grafos sean iguales. Es decir, si G1 es un grafo de cuatro nodos n1, n2, n3 y n4 y G2 otro grafo de tres nodos a, b, c y d, se busca una combinación del tipo n1=a, n2=b, n3=c, n4=d que haga que los dos grafos sean iguales. Se puede ver más claro en la figura 6.
Para comparar los grafos, se obtiene la matriz de adyacencia de cada grafo, de manera que si las dos matrices son iguales, los grafos también lo serán.

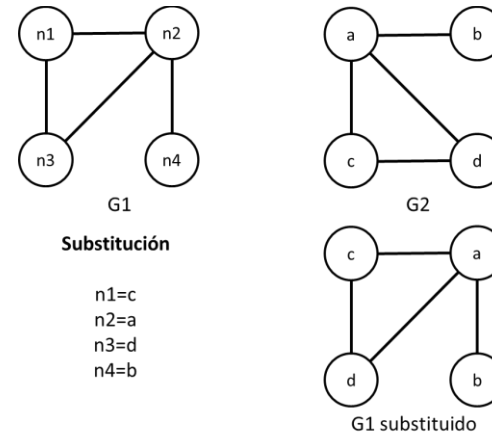


Fig. 6. Ejemplo de correspondencia de nodos

El algoritmo implementado, tiene tres pasos:

1. Obtener todas las permutaciones (o combinaciones) posibles en las que se puedan substituir los nodos, si el grafo tiene n número de nodos, hay n! combinaciones.
2. Por cada permutación, aplicar el cambio en la matriz de adyacencia reordenando las filas y las columnas.
3. Comparar valor por valor si las dos matrices son iguales. Si son iguales, fin del algoritmo, si no, se vuelve al paso 2 hasta que no haya más permutaciones.

5.2.2 Ventajas e inconvenientes

Como ventajas, la principal ventaja del algoritmo es que es fácil de implementar y rápido cuando hay pocos nodos ya que lo único que hace es mover los valores de una matriz y compararlos.

El principal inconveniente, viene dado por la complejidad, ya que es O(n!), y esto implica que cuando se trabaja con grafos grandes, la cantidad de permutaciones a probar es demasiado grande.
Otro inconveniente es que los grafos tienen que ser exactos, y si el grafo entrado por el usuario no coincide con ninguno del diccionario, ya sea porque el constructor de grafo ha confundido un terminal largo con uno corto, o porque el usuario no ha insertado bien la letra, el algoritmo será incapaz de corregir este fallo y encontrar un resultado.

5.3 Algoritmo AC4

En el AC4 se añade un paso previo al algoritmo que busca la correspondencia de los grafos y que mejora la rapidez con la que se encuentra el resultado. Esta mejora consiste en reducir el número de combinaciones posibles dividiendo el grafo en subconjuntos de nodos según las características del nodo. De manera que si por ejemplo un grafo tiene 6 nodos, las permutaciones que habría de forma normal sería de 6! = 702 permutaciones, en cambio si por las características de los nodos se puede separar el grafo en dos grupos de tres nodos cada uno, las permutaciones pasarían a ser 3! x 3! = 36 permutaciones, número que es mucho más bajo que las 720 anteriores.

Para hacer estas agrupaciones de nodos según las características, se utiliza el número de conexiones que tiene cada nodo y el tipo de nodos. Es decir, un nodo de tipo 1 (terminal largo) que solo tendrá una conexión con otro nodo, de tipo 1, no hace falta que estudiemos su substitución por otro nodo que tenga más conexiones, porque ya sabemos de antemano que no será un resultado bueno. De manera, que un nodo con una sola conexión solo se substituirá por otro con una sola conexión.

La función del AC4 se puede considerar como la de eliminar las permutaciones redundantes que nunca podrán tener solución. Reduciendo así significativamente la complejidad del algoritmo.

El uso del AC4 soluciona uno de los problemas del Graph-Matching simple, que era el del gran número de permutaciones, pero sigue existiendo el problema de que sólo es capaz de encontrar soluciones exactas.

5.4 String-Matching Adaptado

El String-Matching es un tipo de Graph-Matching que se utiliza para grafos de una sola dimensión calculando la distancia de edición entre los dos grafos.

La distancia de edición, es el coste de realizar ciertas acciones a un grafo para que sea igual que otro. En este caso, los tipos de acciones que se utilizan son la substitución, la inserción y la eliminación de un nodo.

Aquí la pregunta que nos podemos hacer es: ¿Si nuestra aplicación tiene grafos de más de una dimensión, de qué nos sirve el String-Matching?, y nuestra respuesta es que proponemos un nuevo algoritmo: El string-matching adaptado, que permite utilizar la eficiencia en el cálculo de la distancia de edición, sobre la matriz de adyacencia de nuestro grafo multidimensional.

Para poder hacer uso del String-Matching, se convierte la matriz de adyacencia, que representa el grafo totalmente, a una representación como si fuera una cadena. Se juntan las filas de la matriz una detrás de otra como si se tratara de una cadena. De esta manera ya se puede utilizar el String-Matching para calcular la distancia de edición, aunque se tengan que ir calculando para todas las posibles permutaciones.

5.4.1 Funcionamiento

El funcionamiento de este algoritmo consiste en ir nodo por nodo calculando el coste de cada posible acción. Este coste se guarda en una matriz sumándole el coste acumulado hasta el momento.

En cada casilla de la matriz, el coste que se ha de sumar es el mínimo entre la casilla de la izquierda, la superior, y la diagonal izquierda superior más el coste de la acción. (Ver figura 7)

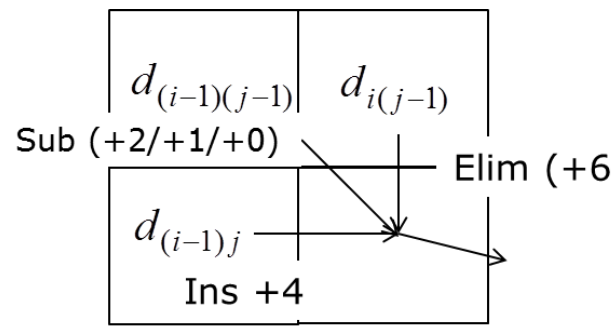


Fig. 7. Ejemplo de costes de una casilla del algortimo String-Matching.

Una vez rellenada la matriz, el coste total será el que haya en la parte inferior derecha de la matriz. Cuando ya esté calculado el coste de todas las letras, se tiene que

elegir el menor de todos, y esa será la letra reconocida.

Puede ser que aunque el coste sea el más bajo, el resultado sea erróneo, pero esto no quiere decir que el algoritmo funcione mal, sino que no hay una correspondencia exacta entre los grafos y el algoritmo devuelve la letra cuyo grafo es el más parecido al del entrado por el usuario.

Para que el algoritmo funcione correctamente, se tiene que poner un coste a las acciones de insertar, eliminar y substituir. Los costes que se han planteado para este proyecto son los siguientes:

- Inserción: coste 4
- Eliminación: coste 6
- Substitución:
    - 0 si se substituye por el mismo tipo de nodo.
    - 1 si la substitución es entre nodos de tipo 1 y 2 (terminal largo o corto), o nodos de tipo 4 (ángulo grande o pequeño) y 5, o también entre nodos de tipo 6 y 8 (T o X).
    - 2 si la substitución no es entre ningún caso de los anteriores.

Como se puede ver, en este reparto de costes, el de la eliminación es el más alto para que como mucho solo se pueda eliminar un nodo, ya que si se tiene que eliminar más de uno, el grafo seguramente esté mal escrito.
El coste de substitución penaliza mucho menos cuando se trata de nodos que son prácticamente iguales pero se diferencian por el valor de un umbral.

5.4.3 Ventajas e inconvenientes

Como ventajas de este algoritmo, tenemos que acepta grafos con diferente número de nodos y da la posibilidad de explorar soluciones eliminando e insertando nodos. Esto nos soluciona el problema de encontrarnos con nodos de más o de menos a la salida del constructor del grafo.

Como inconveniente tenemos que, el String-Matching adaptado, al usar grafos de una dimensión, los nodos siempre siguen el mismo orden estructural, y por tanto no acepta permutaciones. De manera, que si una letra se escribe en un orden diferente al esperado, el algoritmo no será capaz de reconocerla.

5.5 Algoritmo A*

Este algoritmo ha sido planificado pero no se ha podido terminar su implementación, pero de todas formas se explicará en que consiste y cómo se iba a usar.

El Algoritmo de A* es un algoritmo de búsqueda informada, se basa en el coste de los caminos recorridos y en una heurística que estima el coste de ese camino hasta la solución.

En este caso, la heurística se basa en dos diferencias entre los grafos de entrada y el de la letra del diccionario. La primera diferencia es entre el número de nodos, si por ejemplo el grafo de entrada tiene más nodos que el grafo de la letra, por cada nodo de más, se tiene que sumar una acción de eliminación. Los costes que se usan son los mismos que en el String-Matching.
La otra diferencia en la que se basa la heurística depende del tipo de nodo total del grafo. Se cuenta cuantos nodos de cada tipo hay en el grafo, y por cada nodo diferente, se suma el valor de una substitución.

5.5.1 Funcionamiento

Este algoritmo consiste en ir almacenando las soluciones que se van desarrollando en una lista ordenada por el coste más la heurística. Y en cada iteración desarrollar la solución con menor coste hasta encontrar la solución.

Cuando se habla de una solución que se está desarrollando se le llama nodo, pero para evitar confusiones con los nodos del grafo, se le llamara nodoA*.

Los pasos son los siguientes:

1. Inicializar la lista, poniéndole el nodoA* raíz.
2. Comprobar si el primer nodo de la lista es igual al nodo objetivo o si la lista está vacía. Si es que sí, se va al paso 7.
3. Extraer el primer elemento y eliminarlo de la lista.
4. Expandir el nodoA*.
5. Insertar los nuevos nodoA*s en la lista ordenados por el coste acumulado más la heurística.
6. Volver al paso 2.
7. Retornar el primer elemento de la lista, que puede ser la solución o nulo.

El paso de expandir el nodoA*, consiste en crear un nodoA* nuevo por cada posible acción, por tanto serían 3, uno donde la acción es la substitución, otro para eliminación, y otro para inserción.
Pero para que el algoritmo se adapte mejor a la aplicación, en vez de añadir un nodoA* de substitución, se añade uno por cada nodo del grafo que aún no se ha asignado. Para entenderlo mejor se puede ver la imagen de la Figura 8.

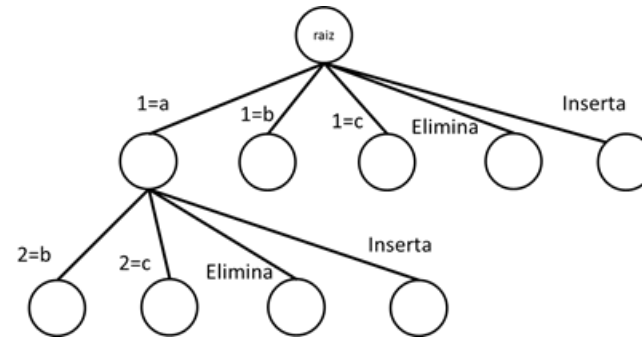


Fig. 8. Árbol de Búsqueda de A*

En esta imagen se puede ver cómo quedaría el árbol de búsqueda de un grafo de 3 nodos. Cada nivel del árbol

corresponde a un nodo del grafo de entrada, y cada nodoA* del árbol corresponde a un tipo de acción que se le haría al nodo del nivel del árbol, substituirlo por uno de los nodos del grafo de la letra, eliminarlo o insertarlo.
La profundidad del árbol siempre debe ser igual al número máximo de entre los grafos de entrada y la letra, de manera que si el grafo de entrada es mayor que el de la letra, al final las últimas opciones que tendrá serán las de eliminación. Y Si el grafo de entrada es menor, las últimas opciones serán las de inserción.

5.5.1 Ventajas e inconvenientes

Una ventaja de este algoritmo es que soluciona el problema de la diferencia del número de nodos de los grafos y también el de las permutaciones.
Otra ventaja es que al desarrollar primero las opciones con menor coste, lo más probable es que encuentre la solución sin tener que desarrollar todas las posibles soluciones.

Como inconveniente y a la vez uno de los problemas que impidieron que se llevara a cabo la implementación del algoritmo es la inserción. Cuando se inserta un nodo en un grafo no solo se debe decir el tipo de nodo que es, sino también a que otros nodos del grafo está conectado, ya que si las conexiones no son las mismas, la matriz de adyacencia tampoco. Para solucionar esto, se deberían explorar todas las posibles conexiones del grafo, no una conexión por cada nodo sino también contemplando que un nodo puede tener hasta un máximo de cuatro conexiones.

Dicho esto, si se probaran todas las posibles conexiones el árbol crecería mucho haciendo inviable esta opción.

6 EVALUACIÓN DEL RENDIMIENTO

En esta sección, se mostrarán los resultados que se han obtenido en las pruebas de rendimiento y los parámetros que se han encontrado que dan los mejores resultados.

Actualmente, el sistema es capaz de reconocer 12 letras distintas, que son las que se pueden escribir sin dibujar contornos redondeados ni uniones especiales.
Estas letras son: A, E, F, H, I, L, M, N, T, V, W, X.

6.1 Experimento

Para preparar las pruebas, se habilitó una opción del sistema (en el código) que guarda en un fichero dentro del móvil o Tablet todo los puntos que se recogen de la letra entrada por el usuario. Esta opción también guarda una imagen de la letra. Luego, se le pidió a diferentes usuario que escribieran las letras que el sistema es capaz de reconocer.

Una vez se obtuvo una cantidad de entradas de cada letra significativa, se cogieron 50 de cada letra para hacer un se renombró cada fichero con el carácter de la letra y un número para identificarlo del 1 al 50 para hacer un ground truth.

Una vez recogidas las 50 muestras, se creó un proyecto de Java aparte para trabajar en un PC, ya que si se ejecuta en un móvil o Tablet la el propio sistema operativo acaba interrumpiendo la ejecución si tarda mucho. Este nuevo proyecto contiene todo el código de la aplicación excepto la interfaz.

En el experimento se leen todos los ficheros de las letras y se aplica todo el reconocimiento de carácter, y se recoge el resultado. A partir de estos resultados se pueden calcular el número de aciertos y errores.

Tambien se ha realizado un experimento para obtener una parametrización del programa que consiga los mejores resultados. Se han ajustado los umbrales de la aplicación de manera que se maximicen el número de aciertos del ground truth.

Estos umbrales son:
th1, es la distancia mínima de un punto al punto de intersección para determinar si está cerca o no.
th2, ángulo mínimo para ser considerado nodo de tipo ángulo grande.
th3, distancia mínima entre dos puntos para ser considerado una recta.
th4, ángulo máximo que puede ser desviada una recta para no ser dividida.
th5, distancia mínima para ser considerado como nodo terminal corto.
th6, distancia mínima entre dos puntos para ser considerados cercanos. Se usa para saber cómo están posicionadas las rectas.

6.2 Resultados

En el Apéndice 2 se pueden ver las gráficas con la cantidad de letras reconocidas correctamente por cada valor del umbral.

Como se puede ver, las gráficas de los umbrales 4, 5 y 6 no varían. En el caso del th4, se debe a que el constructor del grafo corrige esta división uniendo las dos rectas.
En el caso del th5, son muy pocas letras las que lo usan, de manera que el algoritmo es capaz de corregirlo. En el caso del th6, no varía porque ese umbral se usa para saber la posición de una recta respecto a otra y esta información no es esencial para estas letras, solo orientativa.

Según las pruebas, la aplicación tiene una media de aciertos de 464 sobre 600, que esto equivale a un 77,3%. Se considera un buen resultado pero siempre teniendo en cuenta que solo reconoce 12 letras.

La mayor cantidad de aciertos es de 475 con los valores de umbral: th1=61, th2=61, th3=41, th4=21, th5=31, th6=31. Valores obtenidos según el resultado de la prueba.

7 POSIBLE MEJORAS

En esta sección, se explican las mejoras que se le harían a

la aplicación para una segunda versión.

La primera cosa que se debería mejorar es la capacidad de poder reconocer contornos redondeados, ya que ese factor es el que más ha limitado el número de letras que la aplicación puede leer.

La segunda mejora estaría relacionada con el algoritmo, ya que dependiendo del grafo, puede llegar a ir muy lento. Para mejorarlo, se podría añadir la mejora del AC4 al algoritmo actual o cambiarlo entero por otro que se ajustara bien al problema y pudiera dar mejores resultados.

Otra mejora que ya no tiene que ver con la lógica de la aplicación seria cambiar la forma de jugar. Hacer un juego que haga más amena la forma de introducir las letras, como por ejemplo un ahorcado, o un crucigrama donde el usuario tenga que ir poniendo las letras una a una para formar una palabra. Tampoco hace falta cambiar el juego entero sino que aprovechando que la lógica está separada de la interfaz, se pueden añadir juegos que usarán el mismo modelo.

Una mejora más sería la introducción de multiples opciones, como hacer las grabaciones en diferentes idiomas, traducir los textos de los mensajes y permitir escoger las letras que se quieren aprender.

8 CONCLUSION

En este apartado se irán explicando las conclusiones a las que se ha llegado en este proyecto.

Primero. Se ha usado un patrón de diseño de la aplicación para separar la lógica del sistema de la interfaz ha resultado ser un punto muy ventajoso, en el momento de hacer las pruebas unitarias de cada parte por separado. También ha sido de gran utilidad para obtener los resultados de rendimiento y para poder incluir cambios sin tener que tocar mucho código.

Segundo. Se ha usado una vectorización limitada al reconocimiento de rectas, se debería haber usado también para encontrar puntos de intersección y definir mejor los contornos redondeados.

Tercero. Se ha definido un constructor del grafo parte de una gran ventaja al tener los puntos ordenados según el orden en el que se han ido escribiendo, gracias a eso se han podido corregir los defectos de la vectorización.

Cuarto. Se ha implementado el algoritmo de Graph-Matching simple que es un algoritmo de fuera bruta que no da buenos resultados cuando hay muchos nodos (6 o 7) pero sí cuando el grafo tiene pocos nodos (3). El problema de este algoritmo es la limitación al comparar grafos con diferente número de nodos.

Quinto. Se ha implementado el algoritmo AC4 que es una mejora muy sencilla que da muy buenos resultados y soluciona el problema de tener algoritmos con muchos nodos reduciendo la cantidad de permutaciones.

Sexto. Se ha propuesto un algoritmo nuevo, El String-Matching adaptado, aunque no es un algoritmo pensado para grafos de más de una dimensión, ha dado muy buenos resultados al convertir la matriz de adyacencia del grafo a un vector de enteros para poder utilizarlo. Soluciona el problema de tener grafos con diferente número de nodos ya que contempla las operaciones de inserción y eliminación.
El inconveniente de esta algoritmo es que sigue teniendo demasiadas permutaciones cuando trata grafos con muchos nodos. Incluir la mejora del AC4 mejoraría mucho el rendimiento.

Séptimo. El rendimiento de la aplicación según las pruebas realizadas es bastante bueno, pero se tiene que tener en cuenta que solo reconoce 12 letras y si se añade el resto, seguramente el porcentaje de aciertos disminuirá.

AGRADECIMIENTOS

Como es costumbre en estos proyectos, un rinconcito para agradecer no está de más. Primero agradecer a Maria Vanrell por sus buenos consejos, el apoyo y sobre todo por saber cuándo ha llegado el momento de cerrar un proyecto. A Pau Riba por el código que me ahorró bastante tiempo. A Joan Mas por su proyecto y sus filtros. También agradecer en especial a Laia Aguilar por su increíble voz de GPS que le da color a este proyecto y por escucharme cada vez que le hablaba sobre filtros, grafos y algoritmos. Y ya viendo que esto es el fin de toda una carrera, a Sergio D. Fernández, por ese tercero de carrera del que salimos curtidos. Y por último, a mis amigos y amigas por seguir siéndolo después de tanto tiempo de escuchar frases como "No puedo, estoy de exámenes" o "No puedo, tengo que hacer proyecto"... Ahora sí que puedo.

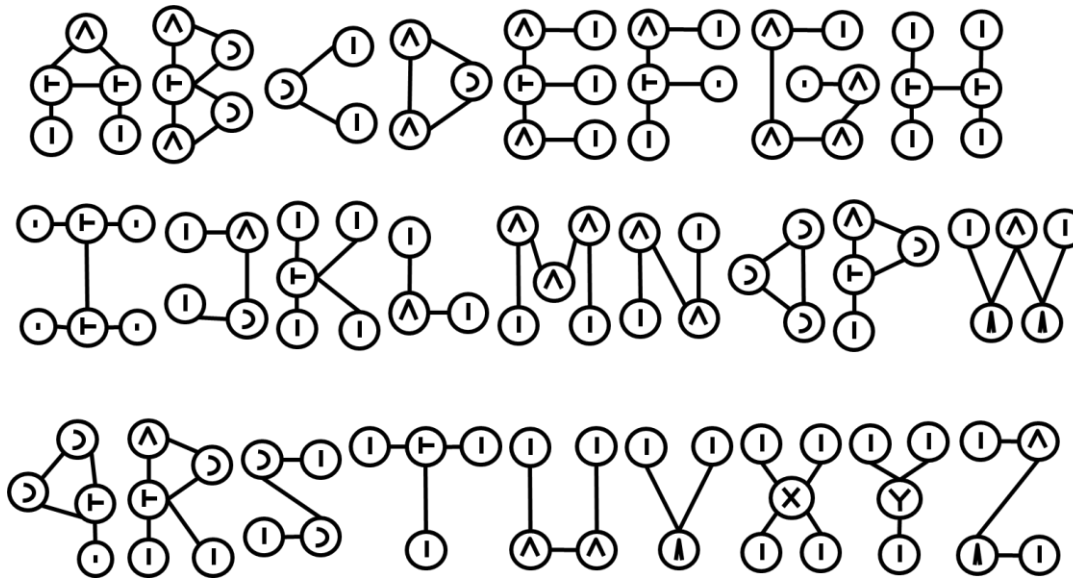
BIBLIOGRAFÍA

[1] Maria Vanrell Martorell. *Resolució de problemes de reconeixament de patrons*. Escuela de Ingeniería UAB.
[2] Maria Vanrell Martorell. *Construcció dels grafs de lletres*. Escuela de Ingeniería UAB.
[3] http://developer.android.com/reference/android/graphics/Bitmap.html (online) última visita 13/03/2014.
[4] http://developer.android.com/reference/android/view/View.html (online) última visita 13/03/2014.

[5] http://developer.samsung.com/s-pen-sdk/technical-docs/S-Pen-SDK-2-3-Tutorial (online) última visita 13/03/2014.
[6] M. Friedman and A. Kandel. *Introduction to Pattern Recognition.* World Scientific, 1999.
[7] R.O. Duda, P.E. Hart, D.G. Stork. *Pattern Classification*, 2nd Edition. Wiley, 2001.
[8] Joan Mas Romeu. A syntactic Pattern REcognition Approach based on a Distortion Tolerant Adjacency Grammar and a Sapatial Indexed parser. Application to Sketched Document Recognition. Tesis Doctoral, UAB 2010.

APÉNDICE

A1. ALFABETO EN FORMA DE GRAFO



A2. GRÁFICAS DE LA EJECUCIÓN.