Algoritmos de Búsqueda para el RECONOCIMIENTO DE VOCALES (septiembre de 2022)

Sebastián Rojas, Santiago A. Rocha, Introducción a Inteligencia Artificial

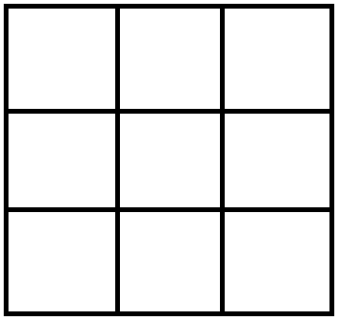
Abstracto - Existe en las personas la voluntad de, escribiendo a mano y con sus propias caligrafías, poder trasladar del papel a la máquina aquello que se escriba, con propósitos generales tales como guardar y/o visualizar información. Para esto, se indagará en el campo de la identificación de caracteres por medio de ciertos patrones que se pueden observar la información que sea adquirida. Con el propósito de cumplir este objetivo, se aplicarán los diferentes estándares que componen la inteligencia artificial queriendo diseñar un programa, que, acompañado de una investigación en el campo determinado, logre realizar buenos acercamientos a la correcta traducción de la escritura hecha a mano, a lenguaje entendible por un computador.

El proyecto está planteado de forma que la extensibilidad permita ampliar el campo de aplicación del programa que se quiere diseñar e implementar. Buscamos tener la capacidad de identificar vocales limitadas por una representación minimal de pixeles, para asentar las bases de futuras mejoras que se quieran diseñar.

I. INTRODUCCIÓN

En este artículo Nos centraremos principalmente en explorar varias ideas de un mismo problema para de esta manera llegar a la solución más eficiente, daremos un recorrido a través de los diferentes algoritmos aplicables entendiendo de esta manera cuales son extensiones más eficientes que otras entre las cuales las brechas de rendimiento son muy grandes.

Para aterrizar un poco más el tema el problema que queremos solucionar trata sobre el reconocimiento de patrones para de esta manera asociarlos con caracteres, en específico nos centraremos en una pequeña muestra de la gran cantidad de caracteres que podemos encontrar y para que se imaginen la complejidad en espacio de este problema aun una rejilla de libertad de 5x5 podría dar a lugar a una complejidad muy alta enfrentándonos a tiempos muy prolongados para desarrollar este problema, tiempo que las máquinas de ahora no gastan para realizar este mismo problema, de esta manera, trabajaremos únicamente en una rejilla de 3x3:



Esto nos lleva a dos posibles ideas al problema planteado que dada una rejilla encontremos su caracter correspondiente, la primera trata en que dado un estado inicial sin ninguna coloración sobre él se comenzarán a generar paso por paso los siguientes estado, con paso por paso nos referimos a comenzar a rellenar uno por uno las rejillas no pintadas dándonos así una ramificación inicial de 9, este problema se tiene pensado modelar y solucionar gracias a un diseño informado, en el cual se definirá una función heurística que nos facilite el recorrido en el árbol, esto con el propósito de alejarnos de la forma clásica de resolver un problema de este tipo y por medio de algoritmos con inteligencia mejorar la eficacia con la que nos enfrentamos realizando este problema con fuerza bruta.
Por último, definimos una serie de condiciones de "frontera" en las que tendríamos en cuenta casos particulares que se pueden llegar a presentar a la hora de hacer uso de la solución, siendo estas condiciones:

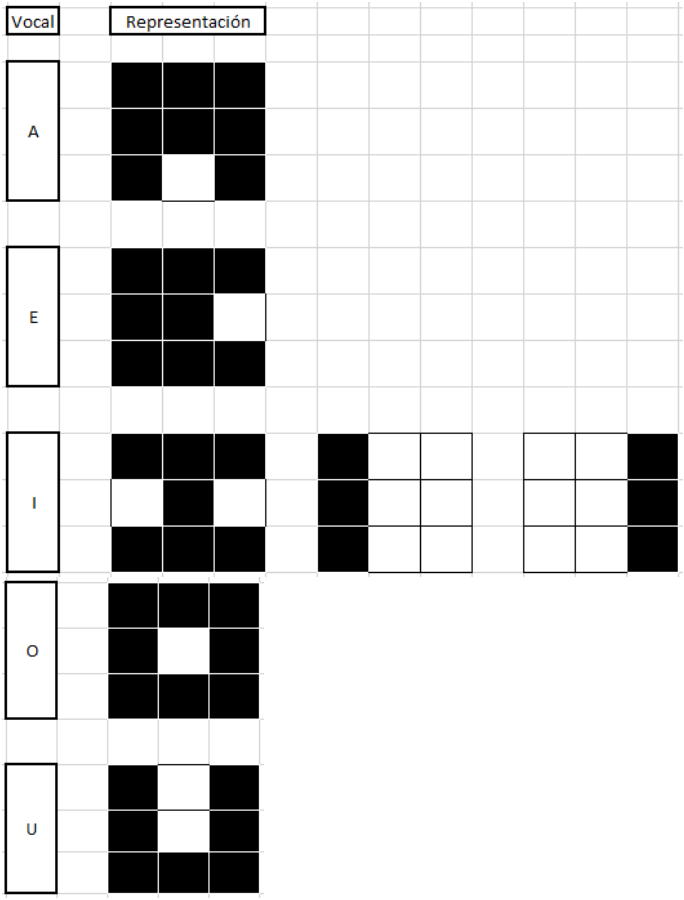
- Input vacío (Rejilla vacía): El input no es una vocal / Input invalido
- Input tiene solo un pixel coloreado: El input no es una vocal / Input invalido
- Si no encuentra similitudes: El input no es una vocal

II. PRIMER ACERCAMIENTO

Documento enviado el 26 de septiembre de 2022.

A. Etapa de Planeación

Para el primer acercamiento, en la matriz de 3x3 los usuarios que utilicen el programa para la identificación de vocales, ingresan algún tipo de trazo/patrón/diseño que consideren como una vocal del alfabeto español, para su posterior identificación.
Adicionalmente, se definió una serie de patrones predefinidos/preestablecidos, vocales, para poder planear cómo serían tenidas en cuenta las entradas que deseasen los usuarios.



Teniendo estos dos puntos fundamentales para la partida, se comenzó a experimentar sobre qué manera se podría llevar el desarrollo del proyecto. Fue necesario considerar cómo se plantearían los estados, sucesores y, sobre todo, lo más importante, la Heurística. Sobre la evaluación de los estados es donde reside la inteligencia del programa contra la fuerza bruta que sería empleada para la resolución de muchos de los problemas de búsqueda existentes.

Se definió como estado inicial el tablero / la matriz ingresada por el usuario, y a partir de esta, los sucesores fueron considerados como:

- La misma entrada del usuario, pero con cada uno de los pixeles ingresados desplazados a izquierda a derecha, cubriendo cada posible desplazamiento. EL resultado de este diseño fue menos que favorable, teniendo árboles de búsqueda demasiado anchos, innecesariamente complejos y con problemas graves de recálculo.
- La misma entrada del usuario, fusionada con cada una de las vocales definidas iniciales (i.e. la unión de los pixeles presentes en la entrada del usuario y en las vocales definidas). El problema con esta creación de sucesores es el nivel bajo de profundidad del árbol de búsqueda que se generaría (profundidad 2) sumado a las bajas capacidades de hacer comparaciones viables que desenvolvieran en la creación de una heurística aceptable, afectando directamente a los resultados que se obtendrían.
- Por último, y siendo la opción seleccionada, los sucesores se definieron como la calca entre la entrada del usuario y cada una de las vocales definidas (i.e. los pixeles exclusivamente en común entre la entrada del usuario y las vocales definidas). De esta forma, en papel y lápiz, resultó favorable la visualización de tableros sucesores, facilitando la posterior definición de la Heurística a asignar a cada uno de estos estados sucesores.

Ahora bien, para el planteamiento y evaluación de la Heurística que permitiera evaluar los tableros, tuvimos en cuenta dos factores de alta importancia: ¿Qué tanto se parecen los tableros a la entrada? Y ¿Qué tanto se parecen los estados a las vocales definidas? Así, tendríamos en cuanta ambos flancos que nos acercarían a mejores resultados.
Teniendo esto en cuenta, la Heurística se desarrolló de la siguiente manera:
Se compararía cada sucesor (calca) del estado inicial (input del usuario) con el estado inicial y con la vocal de la cual fue resultado de la calca, (ej: se compararía la calca de la entrada y la única representación de la vocal A, con la entrada y con la vocal A) y se obtendría un valor numérico correspondiente al número de diferencias presentes en cada comparación (i.e. cada pixel diferente entre la calca con la entrada y entre la calca y la vocal sería contado y almacenado para su posterior tratamiento). Ahora bien, cada uno de estos dos valores obtenidos representaba qué tan diferentes eran las entradas de las vocales, pero, ¿cómo relacionar ambos resultados?

- Inicialmente, se pensó en hacer el producto de los valores obtenidos, favoreciendo los bajos resultados (muy pocas diferencias presentes) y resaltando los resultados altos (demasiado numero de diferencias para ser considerados similares). El problema con este producto, resultó en que en caso que la calca no tuviera diferencias entre la entrada o las vocales, el resultado de la Heurística sería de 0 (un resultado muy favorable) aunque esto estuviera completamente equivocado. Por ejemplo, si el usuario ingresa dos pixeles que convenientemente son compartidos con alguna de las vocales definidas como la O, la calca sería la misma entrada, con un valor de diferencia entre entrada y calca de 0, haciendo que el cálculo de la Heurística fuera favorable cuando es un error grave que se interprete así.
- Es por esto que, en lugar de hacer el producto, se decidió finalmente hacer la suma de los valores obtenidos, evitando el problema de los resultados falsamente favorables.

Así, se diseñó el problema, se hicieron pruebas de escritorio y se determinó que los resultados eran lo suficientemente acertados para proceder con la etapa de desarrollo.

B. Etapa de Desarrollo

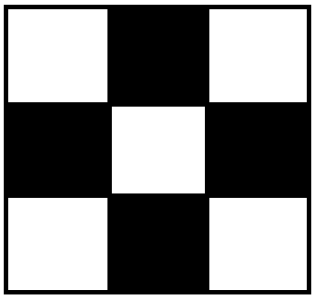
Para la solución de este problema nos enfocaremos inicialmente en el algoritmo Best-first Search esta es una clase de algoritmos de búsqueda que exploran un gráfico expandiendo el nodo más prometedor elegido de acuerdo con una regla específica.[3]

Se realizarán algunas pruebas de este algoritmo, pero como vamos a ver la mayoría de estás tendrán un resultado no muy favorable, asunto que se podría evitar si el campo de muestras fuera mucho más reducido en comparación a este, se da un valor inicial para que así se pueda compilar.

Se definieron las representaciones en matrices de las vocales finales, para posteriormente construir objetos de tipo Testado, conteniendo esta representación junto con un valor desconocido para la vocal que representa (Inicialmente desconocido, posteriormente, si la vocal es identificada, se actualizará este valor a la vocal correspondiente).
Se estructuró el programa, bajo el nombre de Identificador, cuyos parámetros son la vocal a ingresar por el usuario (Testado con representación matricial e identificador desconocido) y el conjunto de representaciones de las vocales previamente definidas (permitiendo en un futuro extender de vocales a más letras del abecedario, o incluso símbolos o caracteres), y se implementó el algoritmo de búsqueda Best-First, aquí nombrado bajo el método identifica().

C. Resultados Obtenidos

Una vez el programa completo, se realizaron test para probar la efectividad de la implementación. Estos consisten en entradas que se sepa están destinadas para coincidir con alguna vocal en específico, o para que no coincidan con ninguna de estas.
Por ejemplo: para la entrada



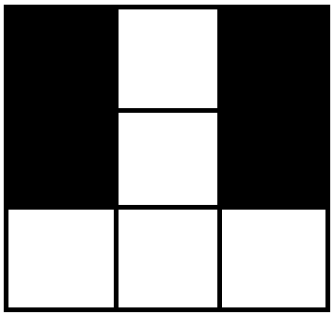
representada en código por:

```
boardO = {1: ' ', 2: 'x', 3: ' ',
          4: 'x', 5: ' ', 6: 'x',
          7: ' ', 8: 'x', 9: ' '}
vocalOident = TEstado(boardO, "?")
identificaO = Identificador(vocalOident, vocalesFinales)
identificaO.identificaVocal()
```

es un test cuya respuesta esperada es O

```
La vocal ingresada es: O
```

Mientras que el test con la entrada



Representado en código como

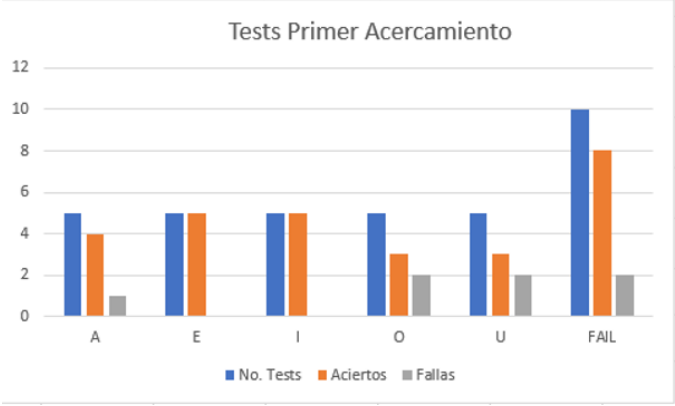
```
ejemploSant3 = {1: 'x', 2: ' ', 3: 'x',
                4: 'x', 5: ' ', 6: 'x',
                7: ' ', 8: ' ', 9: ' '}
vocalSantident3 = TEstado(ejemploSant3, "?")
identificaSant3 = Identificador(vocalSantident3, vocalesFinales)
identificaSant3.identificaVocal()
```

Es un test cuya respuesta esperada es Inválida

```
El Input recibido no es ninguna vocal conocida.
```

Así, 5 test fueron diseñados para cada vocal, y 10 test fueron diseñados para fallar, obteniendo los siguientes resultados:

Test	No. Tests	Aciertos	Fallas
A	5	4	1
E	5	5	0
I	5	5	0
O	5	3	2
U	5	3	2
FAIL	10	8	2



III. Segundo Acercamiento

A. *Etapa de Revisión*

Durante el proceso de desarrollo todo el trabajo tomó dos caminos esto gracias a la idea de que había dos posibles estados iniciales, por un lado, un estado inicial el cual se basaba en la entrada del usuario y por el otro, un estado inicial el cual partía de una rejilla completamente vacía y a medida que iba bajando en profundidad se iban formando sus sucesores pintando de a uno a uno los espacios de la rejilla.

B. *Etapa Final*

Se decidió comenzar a modelar los estados en base a fuentes de información dadas todo esto en pro de realizar un adecuado modelamiento de las funciones básicas para poder aplicar los algoritmos de búsqueda específicos para esta rama, se iniciará sin más ni menos con un input dado ya sea por el usuario o quién desee probar esta solución.

Procediendo así a la implantación según la estructura recomendad del modelo básico de un estado. Ahora procederemos a modelar todos los estados iniciales, punto importante para poder diferenciarlos como estados finales de entre toda la variedad presente.

C. *Estados Finales*

Los estados finales se definieron a partir de la idea de que un estado final debe tener ya sea una vocal asociada a el o una heurística de valor cero. Preguntando a la

```
def esFinal(self):
    value = False
    if  self.vocal is not None or self.costoEstimado() == 0:
        value = True
    return value
```

D. *Heurística.*

Una heurística es un atajo mental que usa nuestro cerebro que nos permite tomar decisiones rápidamente sin tener toda la información relevante. Se pueden considerar como reglas generales que nos permiten tomar una decisión que tiene una alta probabilidad de ser correcta sin tener que pensarlo todo.[1]

Durante esta segunda aproximación se realizará un diseño de heurística el cual consiste en encontrar las similitudes que puede llegar a tener el input de entrada con los inputs con los que se está comparando, este proceso se hace casilla por casilla comenzando desde un valor máximo de 9 que irá reduciendo a medida que se detecten más similitudes en comparación del input.

```
def setHs(self):
    hs = 9
    for board in self.boards:
        hsTemp = 9
        for i in board:
            if board[i] == user_input[i]:
                hsTemp = hsTemp -1
            if hsTemp < 0:
                return 0
        if hsTemp < hs:
            hs = hsTemp
    return hs
```

E. *Greddy Fast Search*

Un Greddy Fast Search es cualquier algoritmo que sigue la heurística de resolución de problemas de hacer la elección localmente óptima en cada etapa.[2]

Se indagó por algoritmos que tuvieran un mayor rendimiento, pero en esencia la misma idea encontrando algunos tales, como Beam, Hill Climbing, etc...

Para el desarrollo de esta parte optima en comparación y más efectiva uno de los que nos resulta útil por el tipo de respuesta que estamos buscando es Beam.

F. *Beam*

Este un algoritmo de búsqueda heurística que explora un gráfico al expandir el nodo más prometedor en un conjunto limitado. Beam search es una optimización de la mejor búsqueda primero que reduce sus requisitos de memoria. La búsqueda del mejor primero es una búsqueda gráfica que ordena todas las soluciones parciales (estados) de acuerdo con alguna heurística.

Pero en la búsqueda de haces, solo se mantienen como candidatos un número predeterminado de las mejores soluciones parciales.[4]

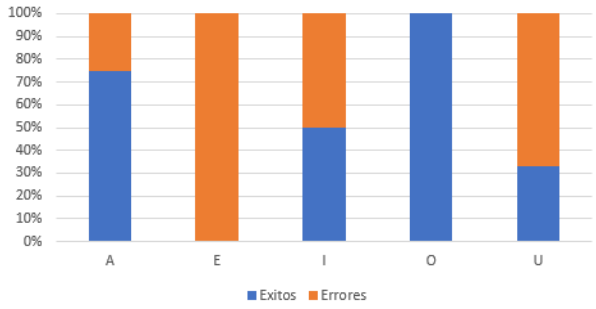
Esta versión de Best First en mucho más sencilla partiendo que no analiza cada una de las ramas desplegadas, sino que en vez de esto se encarga de expandir aquellas con un valor más favorable

Su error en resultados es mucho más bajo que las otras, como vamos a evidenciar. Pudiéndonos dar cuenta que al primer intento y con un estado inicial muy similar al anterior.

IV. Indicaciones útiles

A. *Figuras y tablas*

Profundidad 3		
Vocal	Exitos	Errores
A	3	1
E	0	1
I	2	2
O	5	0
U	1	2

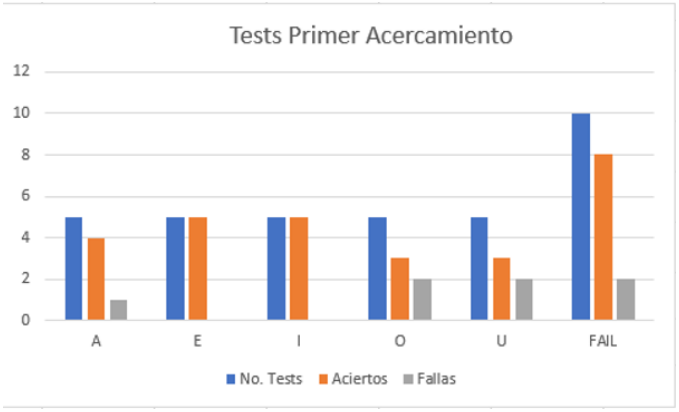


.

Profundidad 4		
Vocal	Exitos	Errores
A	4	0
E	1	0
I	3	1
O	5	0
U	1	2

V. Comparación de los resultados

Como podemos ver en las gráficas adjuntas, el primer acercamiento



Posee una presición del 80% teniendo en cuenta todas las pruebas realizadas. (28 / 35 pruebas acertadas)
Por otro lado, vemos que el segundo acercamiento tiene una precición del 64.7% para una profundidad de 3 (11 / 17 pruebas acertadas) y una precición del 82% (14 / 17 pruebas acertadas).

Con esto nos podemos dar cuenta que, la Heurística del primer acercamiento mantiene resultados constantes con una profundidad constante, mientras que con el segundo acercamiento, la heurística otorga mejores resultados conforme se maneje mayor profundidad.

VI. CONCLUSIONES

- La optimización es muy importante al momento de modelar algoritmos dirigidos a la IA. Ya que en la vida real nos enfrentaremos a problemas con un espacio de muestra tan amplio que el problema actual parecerá pequeño. Por lo que es fundamental centrarnos y algoritmos que nos ofrecen está posibilidad, claro está, que sin dejar los principales atrás.
- Conseguimos modelar e implementar 2 acercamientos para de Reconocimiento de Vocales, adaptando los algoritmos de Búsqueda por Anchura, Búsqueda Mejor Primero; definimos dos Heurísticas según el acercamiento y obtuvimos resultados favorecedores para ambos

acercamientos verificados por medio de Tests.

- Logramos brindar acercamientos válidos y coherentes a una abstracción del problema del reconocimiento de vocales diferente a los ya conocidos que implementan redes neuronales, y en otros casos, modelos de Markov.

REFERENCIAS

[1] Heuristics https://conceptually.org/concepts/heuristics .
[2] Black, Paul E. (2 February 2005). "greedy algorithm". Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology (NIST). Retrieved 17 August 2012.
[3] Mejor primera búsqueda https://en.wikipedia.org/wiki/Best-first_search
[4] "FOLDOC - Computing Dictionary". foldoc.org. Retrieved 2016-04-11..
[5] https://www.geeksforgeeks.org/introduction-to-beam-search-algorithm/
[6] https://www.geeksforgeeks.org/best-first-search-informed-search/C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.