ISSN 1870-4069

El método de anagramas: un rápido y novedoso algoritmo para generar jugadas de Scrabble

Alejandro González Romero[1], René Alquézar Mancho[1], Arturo Ramírez Flores[2], Francisco González Acuña[2,3], Ian García Olmedo[4]

[1] Universidad Politécnica de Cataluña, Departamento de Ciencias de la Computación, Barcelona, España

[2] Centro de Investigación en Matemáticas, Guanajuato, México

[3] Universidad Nacional Autónoma de México, Instituto de Matemáticas, México

[4] Universidad Nacional Autónoma de México, Dirección General de Servicios de Cómputo Académico, México

yarnalito@gmail.com, alquezar@cs.upc.edu, ramirez@cimat.mx, ficomx@yahoo.com.mx, ian.garcia@gmail.com

Resumen. Todos los motores que juegan Scrabble necesitan un algoritmo para generar todas las jugadas legales; después, de alguna manera, se necesita seleccionar una jugada. Para escoger la mejor jugada es conveniente simular un cierto número de jugadas. Debido a las limitaciones de tiempo en un juego de torneo de Scrabble (30 minutos para todo un juego por jugador), no es práctico simular todas las jugadas legales; en nuestros experimentos el promedio de jugadas legales por turno fue de 971. Llamemos *candidatos* a las jugadas que serán simuladas en el futuro; si tenemos más candidatos tendremos una mayor probabilidad de escoger la mejor jugada. Este artículo presenta un rápido y novedoso algoritmo para generar todas las jugadas legales, el cual es usado por nuestro motor de Scrabble llamado *Heuri* [2,6]. El método está basado fuertemente en anagramas; este método tiene la belleza de imitar la manera en que los humanos buscan una jugada válida. Definiremos un *anagrama* de una cadena de caracteres como una palabra contenida en el lexicón que es obtenida por una permutación de los caracteres que forman la cadena. Ya que los anagramas son el alma del algoritmo lo llamaremos *El Método de Anagramas*. Además de presentar el método de anagramas, este artículo da una breve descripción de cómo Heuri juega y sobre los algoritmos usuales utilizados para generar jugadas legales. Estos algoritmos son utilizados por otros motores de Scrabble como *Quackle* [4]. Finalmente se dan tiempos de desempeño y comparaciones entre los algoritmos que

generan jugadas de Quackle y Heuri.

Palabras clave: anagramas, el método de anagramas, lexicón, lexicón de computadora, generación de jugadas válidas, Scrabble, motores de Scrabble.

The Anagram Method: A Fast and Novel Scrabble Move Generator Algorithm

Abstract. All Scrabble engines need an algorithm to generate all legal moves, then somehow they have to select one move to be played. In order to select the best move it is convenient to simulate a certain number of candidate moves. Due to time limitations in a game of Scrabble it is unpractical to simulate all legal moves; in our experiments the average number of legal moves per turn was 971 moves. Let us call *candidates* the moves that will be simulated; the more candidates we have the more likely to choose the best move. This paper presents a fast and novel algorithm to generate all legal moves; the algorithm is used by our Scrabble engine named *Heuri* [2,6]. The method is strongly based on anagrams; the beauty of this method is that it mimics the way humans search for a valid move. An *anagram* of a string here is a word contained in the lexicon that is obtained by a permutation of the tiles of the string. Since the anagrams are the soul of the algorithm we will call this method the *Anagram Method*. Besides presenting the Anagram Method, the paper gives a brief description of how Heuri plays and gives some information about the usual algorithms to generate legal moves employed by other Scrabble engines like *Quackle* [4]. Finally time performances and comparisons between the algorithms to generate moves used by Quackle and Heuri are given.

Keywords: anagrams, the anagram method, lexicon, computer lexicon, generation of valid moves, Scrabble, Scrabble engines.

1. Introducción

Estudiamos el Scrabble, un juego de palabras en el cual los jugadores hacen puntos colocando fichas con letras sobre un tablero dividido en 15×15 cuadros. Las fichas deben formar palabras como en un crucigrama; para más información acerca del juego véase [8, 9].

Sobre la generación de jugadas válidas, Appel y Jacobson [1] introdujeron un algoritmo, el cual fue el más rápido y eficiente en su tiempo. Está basado en la estructura de datos DAWG (Directed Acyclic Word Graph, es decir, Gráfica Áciclica Dirigida de Palabras) construída a partir de las entradas de un lexicón. Después, Steve Gordon [3] introdujo una variante de esta estructura de datos GADDAG, la cual requiere 5 veces más espacio que el DAWG, pero duplica la velocidad de generación de jugadas.

En los ochentas, cuando las computadoras no tenían mucha memoria, el uso de DAWG para guardar el lexicón fue ingenioso y ahorró memoria. Sin embargo la memoria de las computadoras de hoy en día nos permite guardar los lexicones en listas que usan mucho más memoria que los DAWG y los GADDAG, pero este tipo de almacenaje, hecho de forma apropiada, aumentará la velocidad de generación de jugadas.

Hemos desarrollado Heuri [2, 6], un motor para jugar Scrabble que usa el método de anagramas para generar todas las jugadas legales. Heuri derrotó a un campeón mundial 6-0. Expliquemos cómo Heuri juega Scrabble:

Dados un tablero y un atril Heuri produce, para cada subatril del atril, todas las jugadas legales usando el método de anagramas (ver 3.2); además de generar las jugadas legales, Heuri también evalúa la puntuación de cada jugada (para una evaluación rápida cada cuadro guarda información como el puntaje hacia el norte, sur, oeste y este y qué tipo de premio tiene el cuadro).

Una parte importante de un programa que juegue Scrabble es la decisión de qué fichas dejar en el atril. En una jugada se juegan o cambian t fichas y las restantes $n - t$ fichas conforman el *residuo* (*leave* or *residue*) ($n = 7$ salvo quizá en el final).

Se da a todo residuo un valor como se describe en [6] o [2]. Es importante recordar que Heuri utiliza la posición actual del tablero para evaluar todo residuo. Heuri también calcula el puntaje de la jugada sobre el tablero. El valor del residuo y el puntaje de la jugada sirven para evaluar la jugada (véase [2, 6]).

Una vez que todas las jugadas son evaluadas y ordenadas, Heuri podría escoger, digamos, los primeros 40 candidatos. Entonces podríamos recalcular el valor de estos candidatos simulando cada uno de ellos; para tal efecto se encontrarían todas las posibles respuestas del oponente, usando 100 atriles aleatorios del adversario. Un algoritmo rápido para generar jugadas, como el método de anagramas, resulta esencial para esta fase. Esto mejoraría la defensa de Heuri tratando de evitar jugadas de alta puntuación del oponente.

Quackle también evalúa los residuos de cada posible subatril, pero a diferencia de Heuri, Quackle no utiliza el tablero real para estas evaluaciones; en vez de ello, Quackle precalcula cada posible residuo jugando cientos de miles de partidas. Por ello, cuando se juega en un nuevo lenguaje, para emplear toda su fuerza, Quackle necesita mucho tiempo para precalcular los nuevos residuos y Heuri solo necesita alrededor de 3 minutos para tener todo el lexicón de computadora listo y así poder jugar con toda su fuerza. Además, debido a que Heuri utiliza el tablero real para evaluar un residuo y Quackle no, los valores de los residuos de Heuri son más robustos que los de Quackle.

2. Algunas reglas de Scrabble y definiciones

2.1. El tablero de Scrabble

Un tablero estándar de Scrabble consiste en un conjunto de celdas que forman una cuadrícula. El tablero es de 15×15 celdas o cuadros. Las fichas usadas en

el juego encajan en cada celda del tablero. El tablero de Scrabble denota sus columnas con etiquetas de la A hasta la O y sus renglones con etiquetas del 1 al 15. Este etiquetado nos permite referirnos a cuadros y palabras en el tablero. Hay cuadros con premio, usualmente denotados con diferentes colores de acuerdo a su tipo. Ellos se muestran en la Fig. 1.

Más formalmente, el tablero vacío es $\beta = \{1, 2, \ldots, 15\} \times \{1, 2, \ldots, 15\}$, que también denotaremos por $\{1, 2, \ldots, 15\} \times \{\text{A,B,}\ldots\text{,O}\}$ (véase Fig. 1). Los elementos de β se llaman celdas o cuadros. A veces denotamos un cuadro por un número precedido de una letra o una letra precedida de un número. Por ejemplo, (8, 8)=8H=H8.

Un *tablero jugado* es un subconjunto T de β junto con una función que asigna a cada elemento de T una letra del alfabeto usado $\{A, B, \ldots, Z\}$; al subconjunto T solo también se le llamará tablero jugado. Cuando se juega Scrabble un *lexicón* es utilizado, el cual es la colección de palabras válidas.

Para cada carácter λ la bolsa inicial (que consiste de 100 fichas) contiene exactamente $b(\lambda)$ λ's (ver [7] para más información acerca de las distribuciones de letras en Scrabble).

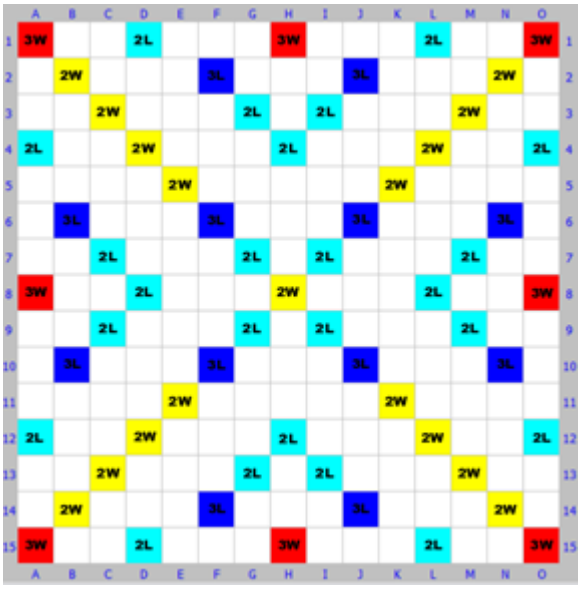

Fig. 1. Tablero de Scrabble.

En cada momento el jugador en turno tiene una colección no vacía de fichas (usualmente 7) que representan caracteres de {A,B,. . . , Z,#} (# denota un comodín y debe ser sustituido por una letra al jugarse en el tablero); estos caracteres forman un multiconjunto (un conjunto con posibles repeticiones; véase Knuth p. 694 [5]) al cual llamaremos un *atril* (*rack*).

2.2. Notación de palabras y opciones en un turno

Definamos una *cadena situada* como una palabra horizontal o vertical de longitud mayor que uno junto con las coordenadas de su carácter inicial. Una cadena situada no es necesariamente una palabra válida. Por ejemplo 8D FORMAL es una palabra horizontal que empieza en el cuadro 8D; H4 FORMAL es una palabra vertical que empieza en el cuadro H4. Ambas son cadenas situadas; también lo es H4 FRMLOA aunque FRMLOA no es una palabra legal.

Al principio cada jugador saca 7 fichas de la bolsa. Al terminar de hacer una jugada en el tablero (o cambiar), el jugador saca fichas de la bolsa de tal manera que el atril tenga 7 fichas (o menos si la bolsa no tiene suficientes fichas).

Hay tres opciones en cualquier turno. El jugador puede colocar una palabra, cambiar fichas por nuevas fichas o pasar.

Un *bingo* (o *scrabble*) es una jugada en la cual un jugador pone todas sus 7 fichas en el tablero haciendo una jugada válida; un *bingo* es premiado con un bonus adicional de 50 puntos. Ver [8, 9] para más información sobre el juego de Scrabble.

2.3. Algunos conceptos útiles

Definamos algunos conceptos que resultarán útiles para la explicación y entendimiento de cómo funciona el generador de jugadas de Heuri.

Sea S un subconjunto de β. Un cuadro σ de β es *adyacente* a S si no pertenece a S y un lado de σ es un lado de un cuadro σ' de S, es decir, $|i-i'|+|j-j'|=1$, donde $\sigma=(i,j)$ y $\sigma'=(i',j')$.

Sea T un tablero jugado. Si $T \neq \emptyset$, el *halo* (de T) es el conjunto de cuadros adyacentes a T; si no el *halo* es $\{8H\}$. Ver Fig. 2 para observar geometricamente el halo de cierto tablero jugado. El halo consiste en el conjunto de cuadros vacíos en la región encerrada por el polígono.

Sean L un renglón o columna de β, T un tablero jugado y r la cardinalidad del atril en juego. Un *intervalo* en L es un subconjunto I de L, que interseca el halo, formado al menos por dos cuadros consecutivos de L, tal que ningún cuadro de $T \cap L$ es adyacente a I y el número de cuadros vacíos de I, $|I-T|$, es menor igual que r.

Intuitivamente un *intervalo* es una región en el tablero donde una palabra podría encajar o entrar. Una jugada válida hecha sobre el tablero debe ocupar un intervalo, pero existen intervalos donde no es posible hacer una jugada.

Damos algunos ejemplos, denotando un conjunto de cuadros consecutivos $\{\sigma_1,\ldots,\sigma_k\}$ por $[\sigma_1,\sigma_k]$. Supongamos que el tablero jugado T es el mostrado en la Fig. 2 y la cardinalidad r del atril en juego es 7. Tenemos que [10A,10K], [10D,10L], [10I,10O], [D4,D5], [N8,N10] son intervals pero los siguientes no lo son: [11L,11N], [5E,5K] (no intersecan el halo); [D5,D9], [B10,B15] (son adyacentes a D10 y B9 respectivamente); [D6,D15] (contiene más de r cuadros vacíos).

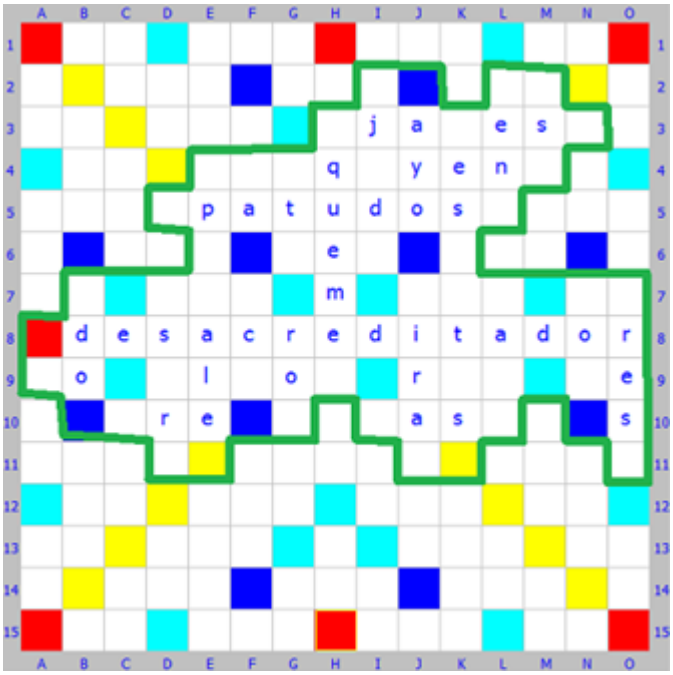

Fig. 2. Halo del Tablero Jugado.

2.4. Cálculo de palabras válidas

De las reglas de Scrabble, si el tablero no está vacío, para poder colocar una palabra, esta tiene que estar conectada al menos con una letra del tablero. Pero esto no es suficiente; para que una jugada sea válida, todas las palabras que se formen tienen que ser palabras válidas contenidas en el lexicón (desde el punto de vista de Heuri, contenidas en el lexicón de la computadora). Recuérdese que la dirección de juego en Scrabble es de norte a sur y de oeste a este.

Sean m y n cadenas; entonces mn indicará la concatenación de m y n. Para cada cuadro vacío del tablero se colecciona información acerca de las fichas jugadas arriba, abajo, a la izquierda y derecha del cuadro.

Sean T un tablero jugado y $\sigma \in \beta$ un cuadro vacío ($\sigma = (i, j) \notin T$). Si $(i-1, j) \in T$, defínase n_σ (el norte de σ) como la cadena situada máxima que ocupa cuadros consecutivos (k, j) de T con $k < i$, e $(i-1, j) \in n_\sigma$; de lo contrario n_σ es la cadena vacía. En una forma similar se definen s_σ, w_σ, y e_σ, el sur, oeste y este de σ.

Ahora, si n_σ o s_σ es no vacía, defínase H_σ como el conjunto de letras λ tales que la concatenación $n_\sigma \lambda s_\sigma$ es una palabra del lexicón; si no H_σ es todo el alfabeto; en este artículo los alfabetos utilizados consisten en el conjunto de letras {A,B,...,Z}. De forma similar, se define V_σ si w_σ o e_σ es no vacío; si $w_\sigma = e_\sigma = \emptyset$, V_σ es todo el alfabeto.

Nótese que si σ es un cuadro vacío que no está en el halo de T entonces H_σ y V_σ es todo el alfabeto.

Si $\sigma \in T$ entonces se define H_σ y V_σ como el conjunto de una sola letra, a saber, aquella representada por la ficha en σ.

Una palabra en el lexicón que ocupa las celdas consecutivas $\sigma_1, \ldots, \sigma_r$ de un renglón (resp. una columna) es una jugada válida si y solo si la letra representada por la ficha en σ_i ($i \in$ [1,r]) pertenece a H_{σ_i} (resp. V_{σ_i}) y alguna σ_i pertenece al halo.

Si T es el tablero vacío el halo es la celda o cuadro 8H. Por tanto, cualquier palabra válida horizontal o vertical que contenga a la celda o cuadro central es una jugada válida.

Llamaremos a H_σ (resp. V_σ) *el conjunto de letras admisibles horizontalmente* (resp. *verticalmente*) en σ , o brevemente, *el conjunto horizontal* (resp. *vertical*) de σ.

Para producir jugadas horizontales (resp. verticales) se usa H_σ (resp. V_σ). Menos intuitivas son las jugadas de una dirección de una ficha, ya que H_σ (resp. V_σ) corresponde a la construcción vertical (resp. horizontal) de una palabra. Las jugadas de dos direcciones de una ficha forman dos palabras (una horizontal y otra vertical).

Los siguientes ejemplos muestran como se usan los intervalos y los conjuntos horizontales y verticales de σ (H_σ y V_σ), para colocar palabras válidas en el tablero.

Supongamos que un jugador tiene el atril { D E E I M N T } y el tablero mostrado en la Fig. 3 de una partida jugada en español. Las letras en cuadros ocupados se escribirán dentro de paréntesis. Si σ es un cuadro desocupado del renglón 10 entonces H_σ es el conjunto de todas las letras {A,B,...,Z} con las siguientes excepciones:

Si σ = 10B (el segundo cuadro del renglón 10) entonces H_σ={M,N,S,Y} ya que B8 (DO)λ es una palabra válida si y solo si λ pertenece a {M,N,S,Y}, y si σ = 10G (el séptimo cuadro) entonces H_σ = { A, B, E, I, L, N, O, S } ya que G8 (RO)A, (RO)B, (RO)E, (RO)I, (RO)L, (RO)N, (RO)O, (RO)S son palabras válidas y (RO)λ no lo es si $\lambda \notin$ { A,B,E,I,L,N,O,S }.

Algunas palabras que se pueden jugar en el renglón 10 son: 10C I(RE) (una jugada de una letra que contiene RE), 10D (RE)MEND(AS)TEI(S) (un bingo que contiene RE, AS y S), 10A ENT(RE)MEDI(AS) (un bingo que contiene RE y AS), 10I M(AS)EEI(S) (que contiene AS y S), 10B MI(RE)N (contiene RE), 10H ME(AS)EN (contiene AS), 10M DE(S) (que contiene S).

También 10A EN (en el intervalo que precede a RE) y 10G ET (en el intervalo entre RE y AS) son jugadas válidas.

En la columna E, si σ es un cuadro sin ocupar entonces V_σ es el conjunto de todas las letras {A,B,...,Z}, sin excepciones, ya que, cuando se juegan fichas solo en la columna E, no se forman palabras horizontales.

Si σ es un cuadro sin ocupar de la columna K, entonces V_σ es el conjunto de todas las letras {A,B,...,Z} con las siguientes excepciones:

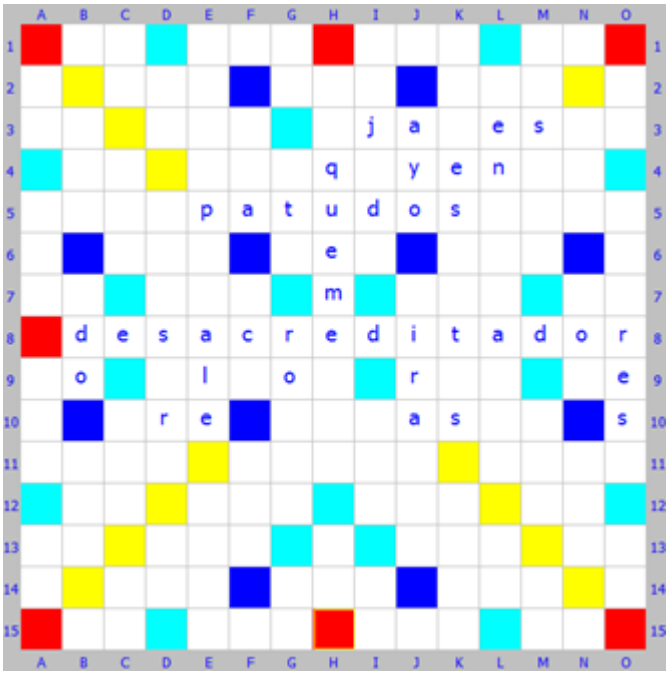

Fig. 3. Una posición de Scrabble.

Si σ = K3 entonces V_σ = { D,L,M,S,T } ya que 3I (JA)λ(ES) es una palabra válida si y solo si $\lambda \in$ {D,L,M,S,T}; y si σ = K9 entonces V_σ = { E,O } ya que (9J (R)E, 9J (R)O) son las únicas palabras válidas de la forma (R)λ.

Algunas palabras verticales que se pueden jugar son: F4 M(A)TI(C)E sobre el intervalo [F4,F9] (9E (L)E(O) es una palabra válida), K3 D(ES)TE(T)E(S) en [K3,K10] (3J (JA)D(ES) y 9J (R)E son válidas).

Un par de palabras horizontales que usan intervalos que tienen intersección vacía con el tablero jugado son: 2J MIDEN en [2J, 2N] (una jugada válida ya que 2J MIDEN, J2 M(AYO), L2 D(EN), M2 E(S) son válidas) y 7J MEDI en [7J,7M] (válida ya que 7J MEDI, J7 M(IRA), K7 E(T), L7 D(A), M7 I(D) son todas válidas). Véase la Fig. 3.

3. El generador de jugadas de Heuri: Estrategia anagramática

3.1. El lexicón de la computadora

Una vez que se tiene un lexicón en un idioma dado, las palabras del lexicón se arreglan en distintos archivos: Directos, Inversos, Indices y Anagramas. Algunos de estos archivos serán usados por el generador de jugadas de Heuri para producir

jugadas que pueden hacerse, y otros se usarán para calcular los valores de las jugadas.

Archivos directos e inversos. Los archivos $Directos_n$ e $Inversos_n$ consisten en listas que tienen todas las palabras válidas de longitud n ($2 \leq n \leq 15$), en orden directo o inverso. Nótese que, en Scrabble, no hay palabras de longitud uno. Se usan $Directos_n$ e $Inversos_n$ para calcular más eficientemente H_σ y V_σ (al buscar palabras, $Directos_n$ es un poco más rápido que $Inversos_n$ si el prefijo de la palabra buscada es más largo que el inverso del sufijo) y también se usan para evaluar las palabras perpendiculares.

Para calcular H_σ (resp. V_σ) cuando la longitud de s_σ es mayor que la de n_σ (resp. la longitud de e_σ es mayor que la w_σ), se usa $Inversos_n$ donde $n-1$ es la suma de las longitudes de n_σ y s_σ (resp. e_σ y w_σ). Si no es así, se usa $Directos_n$.

Archivos de índices y anagramas. Los archivos de índices contienen colecciones de cadenas que, después de substituir #'s por letras adecuadas, forman, aplicando una permutación, al menos una palabra válida. Estas cadenas también contienen un apuntador al final, que indica el principio de todos los anagramas que corresponden a cada cadena.

Los archivos de anagramas contienen todos los anagramas de una cadena dada. Un anagrama de una cadena w es una palabra válida obtenida reemplazando #'s en w (si existen) por letras y volviendo a arreglar la cadena resultante. Todo miembro (cadena) de $Indices_n$ está convenientemente ligado a un conjunto de anagramas de $Anagramas_n$ (n es la longitud de la cadena o palabra).

Los archivos de índices y anagramas son el alma del generador de jugadas. Véase la siguiente sección para mayor información.

Al principio de todo juego los archivos $Directos_n$, $Inversos_n$, $Indices_n$, y $Anagramas_n$ ($2 \leq n \leq 15$) se leen una vez de disco y se guardan en RAM, usando arreglos y tablas hash donde los $Indices_n$ son las claves y los $Anagramas_n$ son los valores.

Construcción de archivos de índices y anagramas. Los archivos de índices se obtienen usando todas las palabras válidas que están contenidas en los archivos directos, reemplazando 0,1 o 2 letras por comodines (#) en cada miembro de los archivos directos y finalmente ordenando lexicográficamente cada miembro y la colección de todos los miembros.

Describamos, de manera más precisa, la construcción de los archivos.

1. Si λ es una letra que aparece exactamente $n(\lambda)$ veces en una cadena u y $b(\lambda)$ es el número de λ's en la bolsa inicial, definimos $e(u)$, el *exceso* de u, por $e(u) = \Sigma \max\{0, n(\lambda) - b(\lambda)\}$ donde λ recorre el conjunto de letras que aparecen en u. Por ejemplo, e(safaris)=0 , e(maximum)=1 y e(ñiquiñaque)=2.

Para una palabra $w \in Direct_n$ sea $\{w'_1, \ldots, w'_{m(w)}\}$ el conjunto de cadenas distintas obtenidas como sigue: sustitúyanse 0, 1 o 2 letras en w por #'s, y ordénese lexicográficamenteca cada cadena resultante (el # viene después de las

letras); se requiere que el número de #'s en cada w'_i sea menor o igual que el número no negativo $2 - e(w'_i)$. Se puede calcular $m(w)$ fácilmente.

Consideraremos concatenaciones $w'_i w$.

2. El conjunto $\{w'_i w : w \in Directos_n,\ 1 \leq i \leq m(w)\}$ se ordena lexicográficamente. Llámese L_n a la lista resultante y denótese el renglón r de L_n por $\iota_r \tau_r$ donde ι_r y τ_r consisten de n caracteres. La Tabla 1 muestra un ejemplo.

3. De L_n constrúyanse dos archivos, $Indices_n$ y $Anagramas_n$, como sigue. El renglón inicial ($r = 0$) de $Indices_n$ es $\iota_0 0$, la concatenación de ι_0 con el entero 0. Si el renglón k de $Indices_n$ se ha definido como $\iota_r r$, la concatenación de la cadena ι_r con el entero r, definimos el renglón $(k+1)$ de $Indices_n$ como $\iota_s s$ donde s es el mínimo entero mayor que r tal que $\iota_s \neq \iota_{s-1}$. Si tal s no existe, no hay renglón $(k+1)$ en $Indices_n$. Para toda r el renglón r de $Anagramas_n$ es τ_r.

Tabla 1. Ejemplo que usa $Directos_3$ = ARE, ERA, ERE .

Renglón (r)	w'_i (casi $Indices_3$)	w ($Anagramas_3$)
0	AER	ARE
1	AER	ERA
2	AE#	ARE
3	AE#	ERA
4	AR#	ARE
5	AR#	ERA
6	A##	ARE
7	A##	ERA
8	EER	ERE
9	EE#	ERE
10	ER#	ARE
11	ER#	ERA
12	ER#	ERE
13	E##	ARE
14	E##	ERA
15	E##	ERE
16	R##	ARE
17	R##	ERA
18	R##	ERE

En la Tabla 1 se puede ver un ejemplo, para un lexicón de 3 palabras, de la construcción de $Indices_3$ y $Anagramas_3$, justo después de la fase 2. Las cadenas pequeñas de la Tabla 1 son repetitivas y desaparecen en la fase 3.

3.2. El generador de jugadas (el método de anagramas)

Describimos aquí el método de Heuri para generar todas las jugadas posibles.

La eficiencia del novedoso generador de jugadas se debe principalmente al uso adecuado de anagramas. Este generador es muy diferente de los utilizados por los motores más conocidos en la literatura sobre Scrabble; la mayoría usa DAWG o GADDAG para describir palabras y entonces seguir una trayectoria en la gráfica para verificar la existencia de palabras válidas.

En vez de ello, el generador de Heuri se aprovecha de la fuerza de los Anagramas. Las dos ideas claves son:

1) El almacenamiento de palabras en dos grupos de archivos ($Indices_n$ y $Anagramas_n$), donde n es el número de caracteres (incluyendo #) de las cadenas de $Indices_n$, y es también el número de letras de las palabras $Anagramas_n$ ($2 \leq n \leq 15$).

2) El enlace conveniente entre $Indices_n$ y $Anagramas_n$ ($Indices_n$ es el grupo de cadenas de longitud n que tienen al menos un anagrama; $Anagramas_n$ es el grupo de diferentes anagramas de longitud n). Este enlace ayuda a viajar por el lexicón de la computadora para generar jugadas legales.

Defínase *segmento* como un conjunto de, al menos dos, cuadros consecutivos contenidos en un renglón o columna.

Los atriles y subatriles son multiconjuntos de los caracteres de {A,B, . . . ,Z,#}, los cuales, a su vez, pueden identificarse con cadenas escritas lexicográficamente. El símbolo $\uplus$ denota multisuma (unión con multiplicidades; ver Knuth p.694 [5]).

Para explicar el método de anagramas, además de la descripción esencial que damos a continuación, se muestra en el Algoritmo 1, un pseudocódigo. Este código genera todas las jugadas horizontales; un código análogo se usa para las jugadas verticales.

Supongamos dados un tablero jugado T y un atril. Queremos coleccionar todas las jugadas válidas.

Primeramente, usando T, se calculan el halo y los conjuntos H_σ y V_σ para cualquier cuadro σ del tablero. Ver Algoritmo 1

Después buscamos intervalos en un renglón dado usando T, el halo, los conjuntos H_σ y V_σ y el tamaño del atril dado. Cuando se encuentra un intervalo I se procede de la siguiente manera:

Sea *string* el multiconjunto formado por las letras de $I \cap T$. Sea I_Rack la multisuma de *string* con las letras del atril. Sean $I\_Subracks$ los multisubconjuntos de I_Rack. Para cada $I\_Subrack$ de $I\_Subracks$ obténganse todos los anagramas y trátense de colocar en el intervalo I (véase el Algoritmo 1).

El $I\_Subrack$ se ordena lexicográficamente para encontrar todos sus anagramas, usando tablas hash donde los $Indices_n$ son las claves y $Anagramas_n$ son los valores. Entonces se trata de colocar cada anagrama en el intervalo usando las coordenadas iniciales de I y los conjuntos H_σ y V_σ. Finalmente, si el anagrama puede colocarse, una jugada válida se ha encontrado, la cual se guarda (ver la última parte del Algoritmo 1).

Algoritmo 1 El Método de Anagramas

```
halo=CalcularHalo(T=TableroJugado)
(H, V)=CalcularConjuntosH-y-V(T=TableroJugado,Atril, Directos_n,Inversos_n)
// Generación de jugadas horizontales
for renglón = 1 to 15 do
   for For icolumna=1 to 14 do
      for For tcolumna= icolumna+1 to 15 do
         segmento= [ (renglón, icolumna), (renglón, tcolumna) ]
         if (segmento ∩ halo) ≠ ∅ then
            if (renglón,icolumna-1) ∉ T and (renglón,tcolumna+1) ∉ T then
               if |segmento-T| ≤ TamañoAtril then
                  // Se encontró un Intervalo
                  string = multiconjunto de letras en el segmento
                  I_Rack = Rack ⊎ string
                  n=tcolumna-icolumna+1
                  I_Subracks = GeneraI_RackSubsets_deLongitud_n_Dada(I_Rack, n)
                  for all I_Subrack in I_Subracks do
                     I_Subrack = OrdenaLex(I_Subrack)
                     Anagramas= EncuentraAnagramas(I_Subrack,Indices_n,Anagramas_n)
                     // Hay jugadas válidas si Anagramas encajan en Intervalos
                     for all Anagramas do
                        if PonAnagramas(Anagrama, renglón, icolumna, H, V) then
                           JugadaVálida=(renglón,icolumna,Anagrama)
                        end if
                     end for
                  end for
               end if
            end if
         end if
      end for
   end for
end for
```

4. Conclusiones

Para mostrar la velocidad del generador de jugadas de Heuri usando el método de anagramas lo comparamos con el de Quackle 2015 jugando 1000

partidas Quackle vs. Heuri, en inglés, usando el lexicón twl06 y usando Quackle GADDAG para almacenar el lexicón; se obtuvieron los siguientes resultados:

El promedio de turnos fue 24.25 . El número máximo de turnos fue 37 . El promedio de jugadas legales analizadas por juego fue 23541.95 . El factor de ramificación (promedio de jugadas legales por turno) fue 970.88 . Tiempo promedio de Quackle por juego: 331.26 ms. Tiempo promedio de Heuri por juego: 126.69 ms. Razón de tiempos Quackle/Heuri (Q/H) por juego: 2.61 . Razón de tiempos Q/H en las primeras 12 jugadas: 5.14 .

Las comparaciones de tiempos por jugada entre Quackle y Heuri, después de 1000 partidas, se muestran en la Tabla 2.

Tabla 2. Comparaciones de tiempos por jugada entre Quackle y Heuri.

N. de Jugada	**T. de Quackle**	**T. de Heuri**	**Razón de Tiempos Q/H**
1	5.93 ms.	0.3 ms.	19.58
2	9.59 ms.	0.95 ms.	10.07
3	13.14 ms.	1.46 ms.	8.99
4	13.71 ms.	1.87 ms.	7.33
5	14.49 ms.	2.28 ms.	6.37
6	13.03 ms.	2.64 ms.	4.93
7	14.93 ms.	3.13 ms.	4.77
8	17.08 ms.	3.65 ms.	4.68
9	15.7 ms.	4.02 ms.	3.91
10	17.15 ms.	4.59 ms.	3.74
11	14.45 ms.	4.91 ms.	2.94
12	19.43 ms.	5.61 ms.	3.46
13	13.97 ms.	5.81 ms.	2.4
14	16.55 ms.	6.42 ms.	2.58
15	15.63 ms.	6.82 ms.	2.29
16	16.67 ms.	7.47 ms.	2.23
17	15.18 ms.	7.66 ms.	1.98
18	15.16 ms.	8.42 ms.	1.8
19	13.14 ms.	8.47 ms.	1.55
20	13.83 ms.	9.14 ms.	1.51
21	9.02 ms.	7.83 ms.	1.15
22	8.33 ms.	7.3 ms.	1.14
23	4.06 ms.	4.53 ms.	0.9
24	3.23 ms.	3.33 ms.	0.97

En promedio al comparar el método de anagramas de Heuri con el método GADDAG usado por Quackle, el método de anagramas resulta ser 2.61 veces más rápido que el método GADDAG cuando se juega una partida completa.

La razón de tiempos Q/H disminuye cuando el número de jugadas aumenta (ver Tabla 2); siendo las 12 primeras jugadas en las que el método de anagramas de Heuri utiliza menos tiempo en comparación con el método GADDAG utilizado por Quackle. Las primeras 12 jugadas son más significativas que las siguientes;

esto se debe a una característica que se da en los juegos de Scrabble: a menudo un jugador obtiene ventaja en las primeras jugadas y es difícil para el contrincante recuperarse de esta desventaja. Después de jugar 1000 partidas, el 75 % de las veces la partida fue ganada por quien iba ganando en la jugada 12.

En las primeras 12 jugadas la razón de tiempos Q/H es 5.14. Por tanto, el método de anagramas permite más tiempo que el GADDAG para analizar las 12 primeras jugadas lo cual, por lo explicado anteriormente, podría resultar en un mejor desempeño.

Para ilustrar las comparaciones de tiempo entre los generadores de jugadas cuando aparecen comodines (#), se hicieron tres experimentos usando el lexicón en inglés con una sola palabra en el tablero (8F ANEROID). Los experimentos consistían en producir todas las jugadas posibles para 3 atriles diferentes dados. En el primer experimento el atril tenía 2 comodines (#); en este experimento Heuri resultó ser 31 veces más rápido que Quackle (que usaba GADDAG). En los otros dos experimentos el atril tenía un comodín en uno y cero en el otro; véase la Tabla 3. Estos experimentos ilustran como el método de anagramas es mucho más rápido que el método GADDDAG cuando aparecen comodines.

Un *callejón sin salida* es una trayectoria en un DAWG que no es una palabra válida, pero es el principio de una palabra válida. Por ejemplo, el camino $c \longrightarrow a \longrightarrow r \longrightarrow a \longrightarrow m \longrightarrow b$ denota *caramb* que no es una palabra válida, pero es el principio de una palabra válida.

Los métodos GADDAG y DAWG tienen muchos callejones sin salida, DAWG más que GADDAG. Estos cuestan tiempo en la generación de jugadas. Una ventaja del método de anagramas es que NO hay callejones sin salida, esta cualidad lo ayuda a ser más rápido que los otros dos métodos. Además, es mucho más rápido que el GADDAG cuando aparecen comodines porque, en tal caso, el número de callejones sin salida aumenta considerablemente. Véase la Tabla 3.

Tabla 3. Tiempos para generar todas las jugadas con un tablero no vacío.

Atril	Tiempo de Quackle	Tiempo de Heuri	Razón de Tiempos Q/H
AENRS##	**1431.1 ms.**	**45.7 ms.**	**31.3**
AEINRS#	297.2 ms.	10.9 ms.	27.3
AEINRST	25.3 ms.	2.1 ms.	12

El uso de anagramas, tablas hash, intervalos junto con las restricciones impuestas por las H_σ y V_σ, hacen muy rápido al generador de jugadas de Heuri (el Método de Anagramas). La cantidad de memoria RAM que se necesita es entre 300 MB y 400 MB (dependiendo del lexicón utilizado); es una cantidad razonable para las computadoras actuales.

El método de anagramas se usará mucho más al incorporar la simulación en Heuri, para propósitos de mejorar su defensa. Entonces se volverán a confrontar los motores de Heuri y Quackle (esta vez ambos contarán con simulación).

Esperamos que la velocidad del método de anagramas sea muy útil para ayudar a Heuri a lograr un buen resultado contra Quackle.

Referencias

1. Appel, A.W., Jacobson, G.J.: The World's Fastest Scrabble Program. Communications of the ACM, 31(5), pp. 572-578 (1988)
2. González-Romero, A., Alquézar, R., Ramírez-Flores A., González-Acuña, F.: Heuristics and Fishing in Scrabble. Chairs : Tristan Cazenave, Jean Mehat, Mark Winands (Eds.), In: Proceedings of the European Conference on Artificial Intelligence, (ECAI'12), Computer Games Workshop (CGW12) (2012)
3. Gordon-Steven, A.A.: Faster Scrabble Move Generation Algorithm. SoftwarePractice and Experience, 24(2), pp. 219–232 (1994)
4. Katz-Brown, J., O'Laughlin, J., Fultz, J., Liberty, M.: Quackle is an open source crossword game program (2006)
5. Knuth, D.E.: The Art of Computer Programming. 2, Seminumerical Algorithms (1997)
6. Ramírez, A., González-Acuña, F., González-Romero, A., Alquézar, R., Hernández, E., Roldán-Aguilar, A., García-Olmedo, I.: A Scrabble Heuristic Based on Probability That Performs at Championship Level. In: Proceedings of the 8th Mexican International Conference on Artificial Intelligence MICAI, pp. 112–123 (2009)
7. Scrabble letter distributions: https://en.wikipedia.org/wiki/Scrabble (2012) _letter_distributions
8. Scrabble pages: http://www.scrabblepages.com/scrabble/rules/ (2012)
9. Scrabble wikipedia: https://en.wikipedia.org/wiki/Scrabble (2012)