Universidad de La Habana
Facultad de Matemática y Computación

El Método de las Palabras Internas, un Nuevo Algoritmo de Generación de Jugadas de Scrabble

Autor:
Davier Sánchez Bello

Tutores:
Fernando Raúl Rodríguez Flores

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

19 de febrero de 2025

Dedicación

Agradecimientos

Agradecimientos

Opinión del tutor

En este trabajo se propone un algoritmo eficiente para la generación de jugadas de Scrabble, y como resultado colateral se propone una estructura de datos para representar multiconjuntos donde las operaciones se pueden realizar en O(1).

Para realizar este trabajo Davier ha usado los conocimientos adquiridos durante su carrera de una manera muy creativa y muy original. Además su comportamiento no ha sido el de un estudiante haciendo su ejercicio de culminación de estudios, sino el de un científico de la computación desempeñándose en su vida profesional.

Estoy muy contento con todos los aspectos del trabajo realizado: la revisión bibliográfica, la propuesta de solución, la presentación de los resultados y la independencia total y absoluta con la que Davier ha trabajado.

Creo que estamos en presencia de un trabajo excelente, realizado por un excelente científico de la computación.

MSc. Fernando Raúl Rodríguez Flores
Facultad de Matemática y Computación
Universidad de la Habana
Febrero, 2025

Resumen

Una IA que juegue al scrabble necesita de un algoritmo lo más eficiente posible que genere todas las jugadas válidas en cualquier situación de juego. En este trabajo se presenta una hipótesis para explicar la caıda en rendimiento hacia los finales de partida del Método de los Anagramas, el más moderno y eficiente de los algoritmos de generación de jugadas de Scrabble. Se presenta un nuevo algoritmo, llamado Método de las Palabras Internas y se presentan resultados experimentales que verifican que es más eficiente que el Método de los Anagramas. Se presenta además la Representación Paridad-Excedencia, una forma compacta de representar multiconjuntos de letras en un solo entero de 64 bits, que soporta el cálculo de multisumas y otras operaciones entre multiconjuntos en tiempo constante respecto al tamaño de los multiconjuntos involucrados.

Abstract

An efficient algorithm that computes all posible plays in any game situation is a central part of any AI that plays Scrabble. This work proposes a hypotesis explaining the performance drop suffered by the Anagrams Method, the latest and more efficient of scrabble move generation algorithms. A new algorithm, called the Intern Words Method, is introduced, along with experimental results demonstrating that it is more efficient than the Anagram Method. Additionally, the Parity-Exceedance Representation is presented—a compact way to represent letter multisets as a single 64-bit integer, enabling the computation of multisums and other multiset operations in constant time with respect to their size.

Índice general

1. Introducción **1**

2. Preliminares **4**
2.1. El Scrabble 4
2.1.1. Elementos del juego del Scrabble 4
2.1.2. Dinámica del Juego 5
2.1.3. Jugadas Válidas 6
2.1.4. Puntuación de una Jugada 9
2.2. Definiciones 9
2.3. Algoritmos de Generación de Jugadas de Scrabble 11
2.3.1. Appel and Jacobson 11
2.3.2. Steve Gordon 14
2.3.3. El Método de los Anagramas 15

3. Representación de Multiconjuntos **19**
3.1. Multiconjuntos 20
3.2. Paridad y Excedencia de un Multiconjunto 21
3.2.1. Paridad de un Multiconjunto 21
3.2.2. Excedencia de un Multiconjunto 22
3.2.3. Correspondencia entre un Multiconjunto y la tupla de su Paridad y su Excedencia 23
3.3. Multiconjuntos y Excedencias Factibles de un Lexicón 24
3.4. Representación Paridad-Excedencia de un multiconjunto de letras . . 25
3.4.1. Representación Compacta de la Paridad y la Excedencia de Multiconjuntos Factibles 26
3.4.2. Representación Paridad-Excedencia de un multiconjunto de letras 27
3.5. Multisumas Factibles 27
3.5.1. Propiedades de la paridad, Excedencia y el Acarreo del resultado de una multisuma 27
3.5.2. Tabla de Tríadas Factibles 33

3.5.3. Tabla de Máscaras de Acarreo 34
3.5.4. Cálculo de Multisumas Factibles 35

4. Los Algoritmos de Generación de Jugadas de Scrabble Basados en Intervalos 37
4.1. Intervalo 37
4.2. Los Algoritmos de Generación de Jugadas de Scrabble Basados en Intervalos 39
4.3. El Método de los Anagramas 40

5. Método de las Palabras Internas 42
5.1. Palabras Internas 42
5.1.1. Palabras Internas 42
5.1.2. Palabras Internas Colocadas 43
5.1.3. Palabra Interna Colocada en un Intervalo 44
5.2. Poda de Instancias del Problema Mano-Intervalo 47
5.3. Selección de Palabras Candidatas 48
5.3.1. Solución de Alto Nivel de un problema Mano-Intervalo 48
5.3.2. Solución de Bajo Nivel de un problema Mano-Intervalo 50
5.4. Umbral de Decisión 52
5.5. Precómputo 52
5.5.1. Tabla de Frecuencias de Palabras Internas Colocadas 53
5.5.2. Tabla de Alto Nivel 53
5.5.3. Tabla de Bajo Nivel 54
5.5.4. Optimización usando el Umbral de Decisión 55
5.5.5. Representaciones Compactas 57
5.6. El Algoritmo 59
5.7. Conclusiones 60

6. Resultados 61

Conclusiones 66

Recomendaciones 67

Bibliografía 68

Capítulo 1

Introducción

El Scrabble es un juego de mesa surgido a mediados del siglo XX, que a pesar de su juventud goza de una enorme popularidad a nivel global [23, 10]. Se caracteriza por una combinación de estrategia, conocimiento lingüístico y competitividad que lo convierten en un desafío intelectual, trascendiendo su concepción inicial como juego familiar hasta convertirse en un deporte de la mente. Ası lo atestiguan varias décadas de tradición de Campeonatos Mundiales y otros eventos de primer nivel en diferentes idiomas [11, 19, 21], donde se codean jugadores que han llegado a la élite tras dedicar décadas de su vida al estudio del vocabulario y de las madejas tácticas del juego [10, 22].

Al igual que sucede con otros deportes de la mente como el Ajedrez o el Go, el entrenamiento de los jugadores de alto rendimiento se puede ver muy beneficiado por el uso de herramientas computarizadas de análisis de partidas. Existen varias generaciones de IA's para el scrabble: Maven [20], Quackle [15], Elise [7], Heuri [12], Macondo [9], sin embargo ninguna de las existentes es capaz de jugar al scrabble de manera sobrehumana [8]. Entiéndase jugar de manera sobrehumana como superar la capacidad humana en cada una de las facetas del juego y demostrar tal dominio venciendo por amplio margen a cualquier jugador humano élite en un match pactado a un número considerable de partidas.

Alcanzar la meta de una IA que juegue de manera sobrehumana es una tarea que se ve complejizada por las siguientes características del Scrabble que hacen imposible una búsqueda exhaustiva:

- El Scrabble es un juego de información incompleta durante la mayor parte de la partida, salvo en los finales, cuando una vez agotada la bolsa es posible inferir las fichas en la mano rival.
- En promedio en un turno, el número de jugadas válidas entre las que el jugador puede escoger, factor de ramificación, es cercano a 1000 en el Scrabble jugado

> en el idioma Inglés [17], siendo considerablemente mayor en idiomas con un diccionario de juego más grande como el Español para el que durante los experimentos realizados para este trabajo, en algunos turnos se encontraron más de 50000 posibles jugadas.

A modo de comparación, el Ajedrez y el Go, además de ser juegos de información completa, tienen factores de ramificación promedio de 35 y 250 jugadas por turno, respectivamente [1]. De hecho, el factor de ramificación en el Scrabble es tan grande que aún no ha sido posible el desarrollo de un motor de juego capaz de resolver con 100 % de precisión los finales de partida a pesar de que en ese momento del juego se dispone de información completa [8].

Generar todas las jugadas válidas para un estado del juego del Scrabble [2]. Los algoritmos diseñados para ello deben generar todas las posibles jugadas combinando las letras que el jugador tiene en su mano y las letras ya colocadas en el tablero para formar palabras válidas, de manera que las palabras laterales formadas por cada jugada sean válidas. Por si fuera poco, deben además calcular la puntuación de cada jugada, cálculo cargado de detalles como casillas de premio, diferentes valores para las letras y presencia de comodines.

En la bibliografía revisada solo aparecen descritos 3 algoritmos para la generación de jugadas de scrabble [2, 13, 17]. Todas las IAs que han gozado de algún éxito frente a jugadores humanos (Maven, Quackle, Heuri, Macondo) implementan alguno de ellos. Los dos primeros algoritmos, descritos por Appel y Jacobson en 1989 y Steve Gordon en 1994, están limitados por las restricciones impuestas a los programadores por la RAM de las computadoras de su época [17].

El Método de los Anagramas, es el tercero de los algoritmos, siendo presentado en 2018 y se basa en un preprocesamiento que no era viable en los años 90. Es el más rápido para las jugadas iniciales y las manos con comodines. Sin embargo su desempeño disminuye a medida que avanza la partida, llegando a ser comparable o ligeramente inferior al de sus predecesores en las etapas finales. Quizás sea esta la razón por la cual no ha sido empleado más allá de Heuri, la IA desarrollada por sus propios autores.

Un nuevo algoritmo de generación de jugadas de scrabble, que aproveche las capacidades de las computadores modernas como el Metodo de los Anagramas pero mantenga un rendimiento estable para todas las etapas del juego, puede contribuir a resolver de manera exhaustiva los finales de partida, así como permitir a las IA's alcanzar mayor profundidad en sus simulaciones [8].

En este trabajo, se presenta un nuevo algoritmo para la generación de jugadas de Scrabble llamado Método de las Palabras Internas. Se presenta la demostración de que su espacio de búsqueda es menor que el del Método de los Anagramas y se realizan varios experimentos numéricos que lo confirman.

También se presenta una forma de codificar multiconjuntos de letras en un solo

entero de 64 bits. Esto permite indexar, adicionar o substraer elementos y realizar uniones o diferencias entre multiconjuntos en tiempo constante, lo cual además del significativo ahorro en memoria, permite un salto de eficiencia.

Capítulo 2

Preliminares

Este capítulo contiene una descripción del juego de Scrabble y una revisión de los algoritmos de generación de jugadas de Scrabble que aparecen en la bibliografía.

2.1. El Scrabble

El Scrabble es un juego de palabras cruzadas donde los jugadores forman palabras en un tablero de 15x15 casillas utilizando fichas con letras, cada una con un valor de puntos. Las palabras deben pertencer al diccionario del juego y se colocan horizontal o verticalmente, conectándose con las ya existentes [18]. El juego comienza desde el centro del tablero, y algunas casillas multiplican el valor de letras o palabras. Cada jugador tiene un atril con 7 fichas y, en su turno, puede formar palabras, cambiar fichas o pasar. Gana quien obtenga la mayor puntuación al final, tras agotar todas las fichas o si ambos jugadores pasan consecutivamente. A continuación en esta sección se describen los elementos que componen un juego de Scrabble, la dinámica de las partidas, el concepto de jugada válida y la puntuación asignada a cada jugada.

2.1.1. Elementos del juego del Scrabble

El juego de Scrabble contiene una serie de elementos que siempre están presentes, ya sea físicamente en el juego presencial, o simulados en el juego en línea. Estos son el lexicón, el tablero, la bolsa y el atril. A continuación se describe cada uno de ellos:

Lexicón Lista de palabras válidas del juego. El lexicón es diferente para cada idioma, siendo más reducido en algunos y más extenso en otros. En la Tabla 2.1 se muestra el tamaño de algunos lexicones de Scrabble.

Tabla 2.1: Cantidad de palabras en los lexicones oficiales de Scrabble por idioma. Un idioma puede tener varios lexicones diferentes, por ello se usa un acrónimo para identificar a los lexicones. En la columna central de esta tabla aparecen los acrónimos que identifican a los diferentes lexicones[6]

Idioma	**Lexicón**	**Cantidad de palabras**
Inglés (Norteamérica)	TWL	192,111
Inglés (Internacional)	SOWPODS	279,496
Francés	ODS	402,324
Rumano	LOC	610,767
Italiano	PARO	674,769
Español	FISE-2	635,270

Tablero El Scrabble se juega sobre un tablero de 15x15 casillas. Algunas casillas son casillas de premio, que multiplican el valor de las letras o palabras que las utilicen. Sobre las casillas de premio se profundizará en la sección de puntuación.

Las palabras se colocan en el tablero de izquierda a derecha a lo largo de una fila, o de arriba hacia abajo a lo largo de una columna, con una letra por casilla. En todo momento del juego se debe cumplir que el tablero luzca como un crucigrama, o sea, que toda sucesión de letras a lo largo de una fila o columna, pueda ser leída de manera natural (hacia la derecha o hacia abajo) como una palabra del lexicón, como se aprecia en la Figura 2.1. A cada fila del tablero corresponde un número y a cada columna una letra, que son usados para identificar a cada casilla como un par de coordenadas.

Bolsa En la bolsa del juego se encuentran todas las fichas al comienzo de la partida. Los jugadores extraen fichas de la bolsa al azar para reponer la cantidad de fichas usadas en cada jugada.

Atril También llamado mano. Ambos términos se usan por igual para referirse a las fichas de cada jugador. Desde el comienzo de la partida cada jugador debe tener siempre 7 fichas en su atril. Por tanto, cada vez que realice una jugada debe reponer las fichas en su mano extrayendo de la bolsa una cantidad igual a la cantidad de fichas que usó. Cuando se acaban las fichas de la bolsa, los jugadores siguen jugando sin reponer sus fichas.

2.1.2. Dinámica del Juego

En una partida de Scrabble pueden participar hasta 4 personas, pero de manera competitiva solo se juega uno contra uno. Los jugadores alternan turnos. En cada turno, el jugador puede:

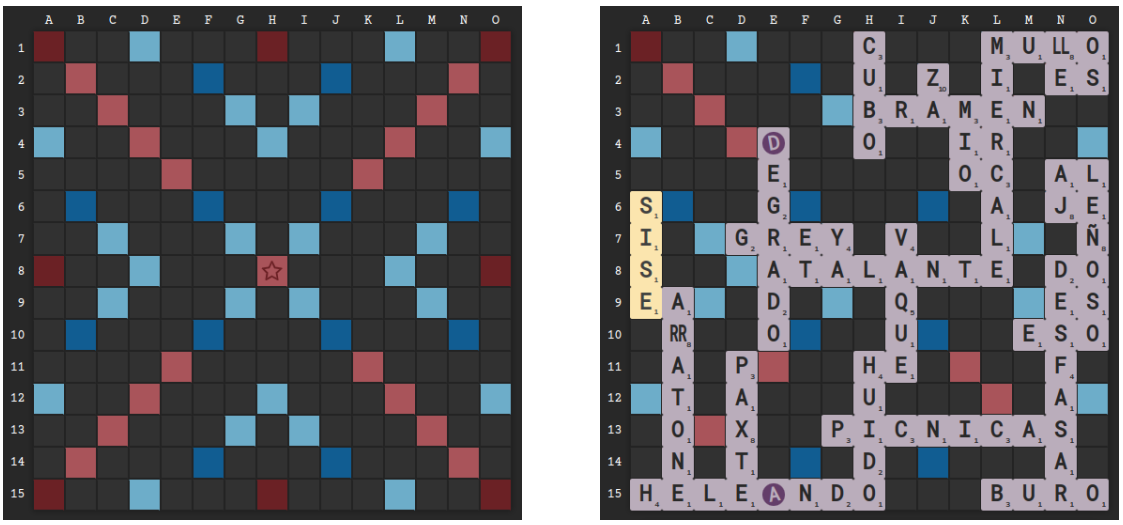

Figura 2.1: Tablero de Scrabble al inicio y al final de una partida

- Jugar: colocar una jugada válida sobre el tablero.
- Pasarse: cediendo automáticamente el turno al rival.
- Cambiar: devolver algunas fichas a la bolsa, sustituyéndolas por igual número de fichas extraídas de la bolsa al azar. En este caso el turno pasa automáticamente al rival.

De estas acciones la única que aporta puntos a un jugador es la primera. El juego termina si dos jugadores se pasan de manera consecutiva. La causa más común de conclusión de la partida es cuando, una vez vacía la bolsa, un jugador agota todas sus fichas. En cualquier caso, al concluir la partida se resta a la puntuación de cada jugador el total del valor de las letras de su mano, y en el caso en que no le hayan quedado fichas, se le suma el total del valor de las letras en la mano del rival. Gana el juego aquel que más puntos acumule.

Un aspecto central de las reglas del juego, es el concepto de jugada válida.

2.1.3. Jugadas Válidas

Una jugada válida es la colocación de una o varias fichas de la mano del jugador sobre el tablero de manera que se cumplan las siguientes condiciones:

- Debe ocuparse al menos una casilla adyacente a una ficha ya colocada. En caso de que aún no se haya colocado ninguna casilla en el tablero, debe ocuparse como parte de la jugada la casilla central del tablero.

- Todas las fichas se colocan en la misma fila o columna, de manera que se forme una palabra nueva (palabra principal) que contenga todas las fichas recién colocadas y quizás algunas otras que estuvieran colocadas antes de la jugada.
- La palabra principal debe ser una palabra válida.
- Todas las palabras que se formen de manera perpendicular a la palabra colocada, como resultado de la concatenación de las fichas recién añadidas con otras fichas ya colocadas, deben ser también palabras válidas.

Las jugadas se denotan por la casilla en que comienza la palabra principal y la palabra principal. Las coordenadas son la concatenación del identificador de la fila y la columna de la casilla y se dan comenzando por el número de la fila en caso de que la jugada sea horizontal o por la letra de la columna en caso de que sea vertical.

A continuación se presentan algunos ejemplos que ilustran el concepto de jugada válida:

Figura 2.2: Tablero de Scrabble antes y después de colocar la jugada *15l BURO*

En la Figura 2.2 se aprecia cómo la jugada *15l BURO* forma dos palabras nuevas en el tablero: BURO y DESFASAR, esta última se obtiene al alargar la palabra DESFASA. Ambas palabras están en el lexicón del juego en Español por lo que la jugada es válida.

En la Figura 2.3 se aprecia cómo la jugada *o5 LEÑOS* forma tres palabras nuevas en el tablero: LEÑOS, DO y ES. DO y ES se forman al combinarse la O y la S de LEÑOS con una D y una E ya puestas en el tablero. Las tres palabras están en el lexicón del juego en Español por lo que la jugada es válida.

Figura 2.3: Tablero de Scrabble antes y después de colocar la jugada *o5 LEÑOS*

Figura 2.4: Tablero de Scrabble antes y después de colocar la jugada *k3 MIO*

En la Figura 2.4 se aprecia como la jugada *k3 MIO* forma tres palabras nuevas en el tablero: MIO, IR y OC. MIO se forma al alargar una *M* ya colocada, mientras que IR y OC se forman al combinarse la I y la O de MIO con una R y una C ya puestas en el tablero. Las tres palabras están en el lexicón del juego en Español por lo que la jugada es válida.

A continuación se describen las reglas para puntuar una jugada

2.1.4. Puntuación de una Jugada

La puntuación de una jugada es la suma de las puntuaciones obtenidas por cada una de las nuevas palabras formadas. Una palabra tiene una puntuación base equivalente a la suma de los valores de las letras que la componen, estuvieran o no ya colocadas sobre el tablero. La puntuación de una palabra es el resultado de incrementar o multiplicar su puntuación base de acuerdo a las casillas de premio usadas en la palabra.

Existen dos tipos de casillas de premio en el tablero:

- Multiplicadores de palabra: al colocar una ficha sobre un multiplicador de palabra sin usar, las palabras que contengan a tal ficha aportarán el doble o triple de su valor.
- Multiplicadores de letra: la letra colocada sobre un multiplicador de letra aportará el doble o triple de su valor.

Los ejemplos usados para ilustrar el concepto de jugada válida también sirven para mostrar el cálculo de la puntuación de las jugadas.

En la jugada *15l BURO* que aparece en la Figura 2.2, la puntuación total es de 39 puntos, que se calculan de la siguiente forma. El valor base de la palabra BURO, 6 puntos, se incrementa a 9, pues la letra *B* se coloca sobre el multiplicador de letra de la casilla *15l*. Luego este valor base debe ser multiplicado por 3, porque la palabra BURO utiliza el multiplicador $\times 3$ de palabra de la casilla *15h*. Siendo así, la palabra BURO por sí sola aporta 27 puntos. Además, en la jugada se formó la palabra DESFASAR cuyo valor base de 12 puntos es todo su valor, ya que no usa ninguna casilla de premio.

En la jugada *o5 LEÑOS* que aparece en la Figura 2.3, la puntuación total es de 47 puntos. El valor base de la palabra LEÑOS es de 12 puntos, pero la palabra aporta 36 pues en su construcción se usa la casilla de premio *8o*, que es un multiplicador de palabra $\times 3$. La palabra DO también usa este multiplicador, por lo que aporta en total 9 puntos. Finalmente la palabra ES aporta solo su valor base, 2 puntos.

Una vez explicadas las reglas básicas del Scrabble, la siguiente sección se presentan algunas estructuras de datos y conceptos que aparecen en el Estado del Arte de los algoritmos de generación de jugadas de Scrabble.

2.2. Definiciones

En esta sección se definen los conceptos de Grafo Acíclico y Dirigido de Palabras (DAWG por sus siglas en Inglés), GADDAG y Multiconjunto.

Grafo Acíclico y Dirigido de Palabras DAWG por sus siglas en inglés (Directed Acyclic Word Graph), es el autómata finito determinista más pequeño que

reconoce el lenguaje de todas las subcadenas de una cadena dada [14, 5]. Se usa para almacenar de manera eficiente en memoria conjuntos de palabras, sin comprometer la eficiencia para responder consultas sobre la presencia o no de una palabra en el conjunto. Sus nodos representan estados intermedios de las palabras y las aristas indican transiciones entre caracteres.

GADDAG Autómata finito determinista que reconoce el lenguaje de todas las palabras de la forma $reverse(x)\#y$, donde # es un delimitador y xy es una palabra válida. Por ejemplo, si la palabra CASA está en en Lexicón, el GADDAG reconocerá #ASAC, A#SAC, SA#AC, ASA#C y CASA#.

El GADDAG recibió su nombre porque es la concatenación del reverso de un DAG y un DAG. El DAG que aparece invertido, es el que reconoce el lenguaje de los prefijos de las palabras válidas. El otro es el DAG que reconoce el lenguaje de los sufijos de las palabras válidas. Entre ambos DAG hay un estado central que corresponde al delimitador. [13].

Una búsqueda en profundidad desde el nodo correspondiente a una subcadena, recorre todas las subcadenas que la contienen, incluyendo a las propias palabras válidas.

Multiconjunto Un multiconjunto es un conjunto que puede contener elementos repetidos. La multiplicidad de un elemento en un multiconjunto es la cantidad de veces que aparece repetido [16].

La multisuma de dos multiconjuntos A y B es el multiconjunto donde aparecen los elementos de A y de B y les corresponde una multiplicidad igual a la suma de sus multiplicidades en A y B [16].

Con la presente definición de multiconjunto y de multisuma basta para la siguiente sección donde se explora el Estado del Arte de los algoritmos de generación de jugadas de Scrabble.

Una definición con mayor nivel de detalle del concepto de multiconjunto y las operaciones definidas sobre los multiconjuntos aparece en el siguiente capítulo que está dedicado por completo a los multiconjuntos.

Una vez expuestas todas las definiciones necesarias, se presentan los algoritmos de generación de jugadas de Scrabble reportados en la literatura.

2.3. Algoritmos de Generación de Jugadas de Scrabble

El problema que abordan los algoritmos de generación de jugadas de Scrabble es el siguiente:

> Dados el tablero de juego y la mano de un jugador, construir la lista con todas las jugadas válidas que el jugador puede realizar en el turno con sus correspondientes puntuaciones.

Los programas más antiguos que jugaban al Scrabble, lo resolvían probando en cada turno todas las palabras del Lexicón, enfoque de fuerza bruta que, exacerbado por las capacidades computacionales de la época, resultaba en una demora de minutos para generar las jugadas válidas de cada turno, de manera que era imposible incorporar ningún tipo de simulación o búsqueda adversarial a su proceso de análisis [2]. Por tal motivo el algoritmo que Appel y Jacobson presentaron en 1989 tiene una importancia seminal, ya que al rebajar en varios órdenes de magnitud el tiempo de cálculo de las jugadas válidas [2] dio paso a la creación de motores de juego de Scrabble mucho más poderosos que combinan el uso de heurísticas con simulaciones.

En 1994, Steve Gordon presentó un algoritmo que, sacrificando uso de memoria por velocidad, alcanza un rendimiento superior al de Appel and Jacobson [13]. El algoritmo de Gordon se ha convertido en el estándar adoptado por la mayoría de las IA's que juegan al Scrabble, incluso la más moderna de ellas, Macondo [8].

Veinticuatro años después de que se publicó el algoritmo de Gordon, en el 2018 se publica el Método de los Anagramas, con un enfoque diferente orientado al preprocesamiento, que aprovecha el salto tecnológico de las décadas posteriores a la aparición del algoritmo de Gordon [17].

A continuación se describen los tres algoritmos, con especial atención al Método de los Anagramas que tiene varios elementos comunes con el Método de las Palabras Internas presentado en este trabajo.

2.3.1. Appel and Jacobson

El primero de los algoritmos que no revisa todo el lexicón en cada turno para generar todas las jugadas válidas fue presentado en el artículo *"The world's fastest scrabble program"* [2] cuyos autores fueron Andrew W. Appel y Guy J. Jacobson. En su trabajo se plantean por primera vez una serie de ideas que serían la base de todos los algoritmos de generación de jugadas de scrabble posteriores y que se enumerarán a continuación:

Unidimensionalidad Puesto que las jugadas verticales son jugadas horizontales en el tablero transpuesto, como se aprecia en la Figura 2.5, puede reducirse

el problema, sin perder generalidad, a generar jugadas horizontales y que por tanto involucran a una sola fila.

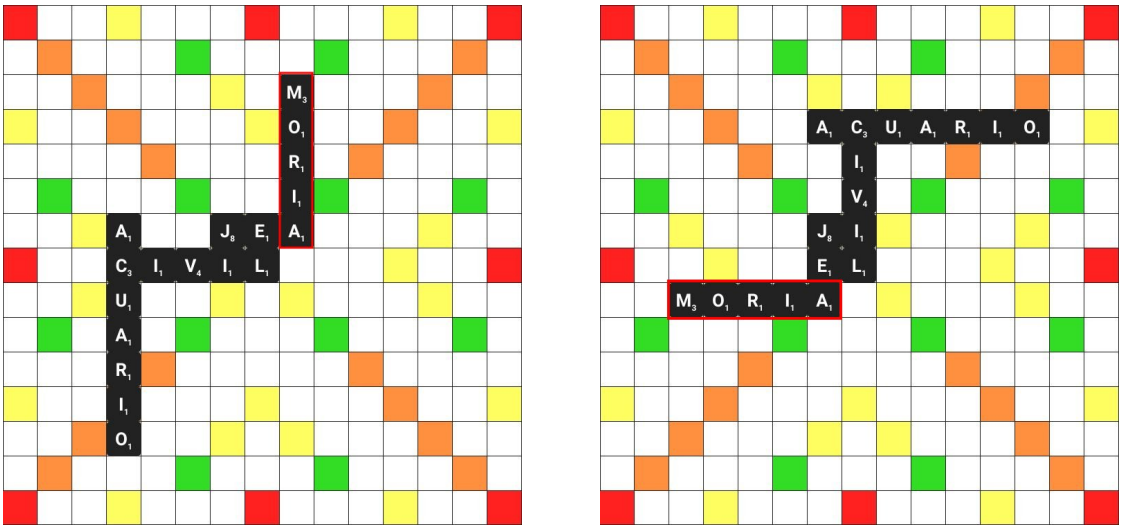

Figura 2.5: Colocar MORIA de manera vertical, es equivalente a colocarla de manera horizontal en el tablero transpuesto

Conjunto de Letras Admisibles Llamado *Cross Checks* en el artículo original, se traduce al español como Conjunto de Letras Admisibles en [17]. El Conjunto de Letras Admisibles de una casilla, es el conjunto de aquellas letras que pueden colocarse en la casilla de tal manera que la columna donde se encuentra la casilla siga conteniendo solo palabras válidas o letras aisladas. Si una casilla no es adyacente a ninguna letra ya colocada, su conjunto de letras admisibles será todo el alfabeto.

Como al jugar de manera horizontal se coloca solo una letra por cada columna, para verificar la validez de una jugada basta comprobar que cada una de las letras jugadas pertenezca al conjunto de letras admisibles de la casilla sobre la cual fue colocada. Por tanto es útil mantener actualizado el conjunto de letras admisibles para cada casilla tanto del tablero original como del tablero transpuesto.

Por ejemplo, veamos los conjuntos de letras admisibles para algunas casillas en el tablero final que aparece a la derecha de la Figura 2.1 de la página 3.

El conjunto de letras admisibles de la casilla *k9* es {A, E, I, O, U}, ya que *k9* está antecedida en la columna k por una T y TA, TE, TI, TO y TU son las únicas palabras válidas de dos letras comenzadas por T.

El conjunto de letras admisibles de la casilla *j1* es {I}, ya que *j1* está sucedida en la columna *j* por la palabra ZA, y la única palabra de 3 letras terminada en ZA es IZA.

El conjunto de letras admisibles de la casilla *n7* es vacío, porque la casilla *n7* está antecedida por la palabra AJ y sucedida por la palabra DESFASAR, y no existe ninguna letra tal que al concatenarle AJ como prefijo y DESFASAR como sufijo se obtenga una palabra del Lexicón.

Casillas Anclas son las casillas adyacentes a fichas ya colocadas en el tablero o la casilla central en el caso del tablero vacío. Cada jugada debe contener al menos una casilla ancla. Las casillas anclas se usan como pivotes para la generación de jugadas en el algoritmo, por lo cual es conveniente mantener calculado el conjunto de casillas ancla.

Appel y Jacobson proponen almacenar el Lexicón en un DAWG. Seleccionaron esta estructura por tener el menor número de nodos posibles entre cualquier autómata capaz de reconocer el lenguaje de las palabras válidas y soportar búsquedas eficientes de todas las palabras comenzadas con un determinado prefijo. [2].

El algoritmo, a grandes rasgos, consiste en iterar por todas las casillas anclas, y para cada una de ellas, usando el DAWG, generar todas las palabras válidas que la contienen.

El algoritmo, para cada casilla ancla C:

1. Genera todos los prefijos terminados en la casilla C, que se puedan construir usando solo letras en la mano del jugador y que en caso de ser colocados no ocupen ninguna casilla ancla a la izquierda de C.

2. Extiende cada prefijo hacia la derecha en el tablero. Para ello realiza un recorrido sobre el DAWG partiendo del nodo que le corresponde al prefijo. Solo se usarán durante el recorrido transiciones denotadas por letras que se encuentren a la vez en la mano del jugador y en el conjunto de letras admisibles de la casilla a ocupar. Cada vez que se usa una transición, se extiende el prefijo con la letra correspondiente a la transición. Cada vez que en el DAWG se alcance un estado de aceptación, se habrán encontrado una jugada válida.

   Cuando el algoritmo llega a un estado donde no existen más transiciones, retrocede al estado anterior y prueba alguna transición que no haya usado aún.

   Resumiendo, se extiende cada prefijo de acuerdo a las restricciones del tablero y de la mano del jugador mediante un *backtrack* sobre el DAWG.

La principal limitación de este algoritmo son los callejones sin salida. Se llaman así los caminos que se recorren en el DAWG sin oportunidad de llegar a una palabra válida, porque las condiciones del tablero lo impiden.

Por ejemplo, si en una fila una casilla ancla está sucedida por casillas cuyo conjunto de letras admisibles es el alfabeto, y luego aparece una casilla que admite solo unas pocas letras, ninguna de las cuales está en la mano del jugador. El algoritmo alargará cada uno de los prefijos de cada posible manera hasta llegar a la casilla restrictiva, sin poder completar la palabra y teniendo que retroceder, una y otra vez.

Un ejemplo se puede apreciar en la Figura 2.6 de la página 14, donde al procesar la casilla ancla *c4*, se extenderá todo prefijo sin limitaciones mientras siga siendo prefijo válido de alguna palabra del Lexicón hasta la casilla *h4*. Pero la única letra admisible en la casilla *h4* es la O, porque la única palabra de dos letras terminada en C en el Lexicón es OC. Luego, si el jugador en turno no tiene una O en la mano, para cada prefijo extendido hasta *h4* se habrán recorrido en vano 4 estados del DAWG (uno por cada letra con que se extendió el prefijo desde *c4* hasta *h4*.

Los autores mencionan en su artículo que quizás con un DAWG en dos sentidos o alguna otra estructura de datos se podía construir las palabras en ambos sentidos partiendo de las casillas anclas y no solo hacia la derecha, lo cual reduciría el número de callejones sin salida. Esta sugerencia fue el punto de partida del algoritmo propuesto por Steve Gordon en 1994.

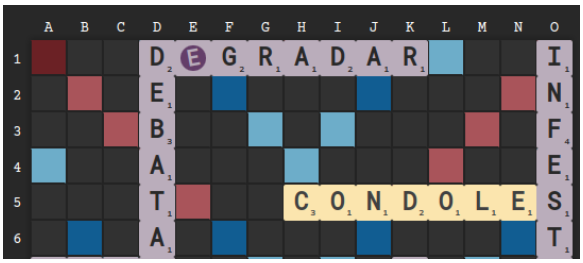

Figura 2.6: Tratar de extender cualquier prefijo desde *c4* hasta *h4* conduce a un callejón sin salida si el jugador en turno no tiene una O en su mano, o esta ya fue empleada en el recorrido por el DAWG.

2.3.2. Steve Gordon

En el año 1994 Steve Gordon propuso una variante del algoritmo anterior, basada en el uso de un GADDAG, en lugar de un DAWG, para almacenar el lexicón.

El GADDAG está diseñado para la búsqueda eficiente de todas las palabras válidas que contienen una determinada subcadena, propiedad que aprovecha el algoritmo de Steve Gordon para generar las jugadas válidas en las dos direcciones partiendo de una casilla ancla. A modo de contraste, el algoritmo de Appel and Jacobson produce un mayor numero de *dead-ends*, porque al solo generar las palabras hacia la derecha, recorre una y otra vez los mismos caminos en el DAWG para cada prefijo.

El algoritmo presentado por Steve Gordon para generar todas las jugadas válidas haciendo uso del GADDAG procede de la siguiente manera:

1. Para cada casilla ancla comienza con la subcadena vacía.

2. Extiende la subcadena hacia la izquierda o derecha, partiendo del estado inicial del GADDAG y usando solo transiciones denotadas por letras que estén tanto en la mano del jugador como en el conjunto de letras admisibles de la siguiente casilla a ocupar.

3. Cada vez que en el GADDAG se alcance un estado de aceptación se habrá encontrado una jugada válida, y cuando llegue a un estado donde no sea posible explorar nuevas transiciones, retrocede al estado anterior y prueba alguna transición que no haya usado aún.

Los autores plantean que esta variante usa 5 veces más memoria que la del DAWG, pero es el doble de rápida porque necesita de menos casillas anclas y conduce a menos callejones sin salida, lo que demuestran con resultados experimentales [13].

En la práctica, el algoritmo de Steve Gordon es el algoritmo para la generación de jugadas de Scrabble usado en Quackle [15], Elise [7], y Macondo [9], tres IA's diferentes presentadas en las tres diferentes décadas del siglo XXI, lo cual es una prueba de que ha superado la prueba del tiempo. No obstante, fue diseñado en base a restricciones (en cuanto a uso en memoria) impuestas por las computadoras de su época. El siguiente algoritmo, publicado casi 2 décadas más tarde, aprovecha el incremento en la capacidad RAM de las computadoras en ese período.

2.3.3. El Método de los Anagramas

En 2009 un colectivo de autores hispanoamericanos presentó a Heuri en la 8va Conferencia Mexicana de Inteligencia Artificial. Heuri es una IA que juega al Scrabble basada en heurísticas probabilísticas. Dentro del artículo sobre Heuri, los autores mencionaban el uso de un nuevo método de generación de jugadas que describían brevemente. En el 2018 este nuevo método de generación de jugadas se publicó en la revista *Research in Computing Science*, bajo el nombre de Método de los Anagramas, en el artículo *"The Anagram Method: A Fast and Novel Scrabble Move Generator Algorithm"* [17] .

Para la descripción del algoritmo se necesitan las siguientes definiciones:

Halo conjunto de las casillas adyacentes a una ficha o el conjunto que solo contiene a la casilla central para el caso del tablero vacío . O sea, el conjunto de las casillas anclas.

Intervalo sección de casillas contiguas que cumple los siguientes requisitos:

- Todas las casillas pertenecen a la misma fila o a la misma columna.
- Está antecedida y sucedida en la fila o columna común por casillas vacías o por el borde del tablero.
- Contiene al menos una casilla del halo.
- Puede contener casillas usadas.

O sea, un intervalo es toda sección de casillas sobre la que se puede colocar una jugada. En la Figura 2.7 aparecen encuadrados algunos ejemplos de intervalos.

Figura 2.7: Algunos intervalos aparecen resaltados usando recuadros verdes

Anagramas Un anagrama de una cadena es una permutación de sus letras. Por ejemplo COSTERA y RETOCAS son anagramas de la cadena ACEORST.

El Método de los Anagramas debe su nombre a que precomputa la lista de anagramas que constituyen palabras válidas para cada multiconjunto de fichas de Scrabble de tamaño entre 2 y 15. Según los autores, los datos precomputados ocupan entre 300 y 400 MB [17] dependiendo del lexicón, cantidad enorme comparada a la memoria consumida por los programas que implementan los algoritmos precedentes que no llega al MB [2, 13], pero cómoda para las computadoras modernas.

El Método de los Anagramas, a diferencia del algoritmo del DAWG y el del GADDAG, no genera las jugadas válidas de manera creciente a partir de casillas anclas, sino que las genera de manera independiente para cada intervalo del tablero.

Es posible calcular las jugadas válidas para un intervalo con v casillas vacías, calculando para cada subconjunto de tamaño v de las letras en la mano del jugador, las jugadas válidas del intervalo que emplean todas y solo las letras de ese subconjunto.

Dado un intervalo y un multiconjunto de letras a emplear, sea M la multisuma de las letras ya colocadas en el multiconjunto y las letras a emplear. Cualquier jugada válida es un anagrama de M. El Método de los Anagramas considera como jugadas candidatas a todos los anagramas de M. Las jugadas válidas serán aquellas jugadas candidatas que sea posible validar, es decir, que cada letra pertenezca al conjunto de letras admisibles de la casilla donde fue colocada.

O sea, el Método de los Anagramas, para cada intervalo con v casillas vacías y cada subconjunto de las letras en la mano del jugador de tamaño v:

1. Calcula la multisuma M de las letras del subconjunto y las letras ya colocadas en el intervalo.

2. Considera jugadas candidatas todas las resultantes de colocar un anagrama de M en el intervalo. Los anagramas de M estarán precomputados.

3. Descarta las jugadas candidatas que implicarían colocar una letra en una casilla donde no es admisible y por tanto no son válidas.

Para generar jugadas válidas, los algoritmos de Appel y Jacobson y Steve Gordon, siguen caminos en un grafo. El Método de los Anagramas no, y por tanto no llega a callejones sin salida. [17].

Como los anagramas fueron calculados para cualquier conjunto de fichas del Scrabble, incluyendo a los que contienen comodines, no es necesario hacer ninguna distinción en el algoritmo a la hora de trabajar con los comodines. No obstante, precisamente la mayor parte del espacio ocupado por el precómputo corresponde a multiconjuntos que contienen comodines, dado que para un multiconjunto de 7 letras diferentes por ejemplo hay 28 multiconjuntos resultantes de sustituir una o dos de sus letras por comodines.

En este trabajo se propone, junto al Método de las Palabras Internas, una forma de trabajar con los comodines que, de ser aplicada en el Método de los Anagramas, reduciría el uso de memoria.

Los autores del Método de los Anagramas, midieron su velocidad contra el generador de jugadas de *Quackle*, que cuenta con una implementación optimizada del método del GADDAG de Steve Gordon. Para ello simularon 1000 partidas sobre la edición del 2006 del Lexicón del idioma inglés. Presentaron los resultados en la tabla mostrada en la Figura 2.8.

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

Figura 2.8: Comparativa de los tiempos de la implementación del Método de los Anagramas presente en Heuri, con la implementación del GADDAG presente en Quackle

Es posible notar en la tabla de la Figura 2.8 como la ventaja que el Método de los Anagramas toma sobre el Método del GADDAG se desvanece a medida que el tablero se va llenando. En este trabajo se presenta una hipótesis que explica esta caída en rendimiento.

Capítulo 3

Representación de Multiconjuntos

Si bien el Método de los Anagramas y el Método de las Palabras Internas funcionan independientemente de la representación que usen para los multiconjuntos, la representación influye en su rendimiento y uso de memoria.

En [17] los multiconjuntos de letras se representan como cadenas donde las letras están ordenadas lexicográficamente y cada letra se repite tantas veces como su multiplicidad en el multiconjunto, como se puede apreciar en los ejemplos de la Tabla. 3.1.

Tabla 3.1: Multiconjuntos de letras en algunas palabras

palabra	**multiconjunto**
CASA	AACS
SCRABBLE	ABBCELRS
PALANGANA	AAAALNNGP
MULTICONJUNTO	CIJLMNNOOTTUU

Esta representación de multiconjuntos se usará como notación para mencionar ejemplos de multiconjuntos de letras a lo largo del resto de este texto, ya que es más cómoda de usar que la notación extensiva tradicional de los multiconjuntos, que enumera los elementos entre llaves y separados por comas.

La representación de multiconjuntos como cadenas presenta una serie de desventajas computacionales:

- A la hora de indexar usando un multiconjunto como llave, la representación de los multiconjuntos como cadenas exige calcular el hash de la cadena que representa al multiconjunto, lo cual constituye una sobrecarga.
- Las operaciones de multisumas entre multiconjuntos representados como cadenas toman tiempo lineal con respecto al tamaño de los multiconjuntos involu-

crados.

- El uso de memoria para almacenar multiconjuntos representados como cadenas es lineal con respecto al tamaño de los multiconjuntos.

La representación de multiconjuntos como cadenas permite representar de manera biunívoca cualquier multiconjunto de letras. Pero el Método de los Anagramas y el Método de las Palabras Internas no necesitan poder representar a cualquier multiconjunto de letras, sino solo a aquellos que pueden ser usados para formar alguna palabra válida.

En este capítulo se presenta la Representación Paridad-Excedencia, una codificación biunívoca en 64 bits de los multiconjuntos de letras que pueden formar alguna palabra válida y además permite realizar operaciones de multisuma e indexación con multiconjuntos en tiempo constante respecto al tamaño de los multiconjuntos involucrados.

El capítulo comienza con un recuento del concepto de multiconjunto y las operaciones definidas sobre multiconjuntos. Las secciones subsecuentes están dedicadas a los conceptos de paridad y excedencia de un multiconjunto, los conceptos de multiconjunto y excedencia factibles, la Representación Paridad-Excedencia y finalmente la multisuma de multiconjuntos representados usando la Representación Paridad-Excedencia.

3.1. Multiconjuntos

En lo adelante, U se usará para representar el Universo de los potenciales elementos de un multiconjunto.

Un multiconjunto es un conjunto que puede contener elementos repetidos.

La multiplicidad de un elemento en un multiconjunto es la cantidad de veces que aparece repetido en el multiconjunto [16].

Se puede tratar a la multiplicidad como una función que a cada elemento presente en el multiconjunto le hace corresponder la cantidad de veces que aparece en el multiconjunto, mientras que a los elementos no presentes le hace corresponder 0.

Cada multiconjunto tiene una y solo una función de multiplicidad, que lo describe completamente. Así mismo cualquier función cuyo codominio sean los enteros no negativos describe un único multiconjunto [4, 3].

La función de multiplicidad de un multiconjunto es una función de U en $\mathbb{N}_{\geq 0}$

A continuación se definen las operaciones de multiconjuntos que serán utilizadas en este trabajo:

Multisuma : La multisuma de dos multiconjuntos A y B es el multiconjunto donde a los elementos de A y de B les corresponde una multiplicidad igual a la suma

de sus multiplicidades en A y B [16].

O sea, dados dos multiconjuntos A y B con funciones de multiplicidad f_a y f_b respectivamente, la multisuma de A y B, denotada $A \uplus B$ es un multiconjunto cuya función de multiplicidad f_m cumple que para toda $x \in U$:

$$f_m(x) = f_a(x) + f_b(x)$$

La multisuma es conmutativa y asociativa por lo que es fácil extender su definición de una operación binaria a una operación n-aria [16].

La multisuma de n multiconjuntos A_1, A_2, ..., A_n cuyas funciones de multiplicidad son f_1, f_2, ..., f_n, es el multiconjunto cuya función de multiplicidad f_m cumple que para toda $x \in U$:

$$f_m(x) = \sum_{i=1}^{n} f_i(x)$$

Subconjunto : Un multiconjunto B será subconjunto de otro multiconjunto A sí y solo sí la multiplicidad de cada elemento de B es menor o igual que su multiplicidad en A.

O sea, dados dos multiconjuntos A y B cuyas funciones de multiplicidad son f_a y f_b respectivamente. B será subconjunto de A, denotado como $(B \subseteq A)$ sí y solo sí para toda $x \in U$ se cumple que:

$$f_b(x) \leq f_a(x)$$

En la siguiente sección se definen los conceptos de paridad y excedencia de un multiconjunto.

3.2. Paridad y Excedencia de un Multiconjunto

En esta sección se definen la paridad y excedencia de un multiconjunto así como algunas de sus propiedades, y luego se demuestra que puede representarse cada multiconjunto de manera biunívoca por la tupla de su paridad y su excedencia.

3.2.1. Paridad de un Multiconjunto

La **paridad** de un multiconjunto M, es un multiconjunto donde la multiplicidad de cada elemento es el resto módulo 2 de su multiplicidad en M.

En la Tabla 3.2 aparecen ejemplos de multiconjuntos de letras y sus paridades:

Tabla 3.2: Algunos multiconjuntos de letras y sus paridades.

Multiconjunto	**Paridad**
AACS	CS
ABBCELRS	ACELRS
AAAALNNPG	LPG
ACEGILMORU	ACEGILMORU
CIJLMNNOOTTUU	CIJLM
CDEEENNR	CDER
EJNOOOPSSS	EJNOPS

La paridad de un multiconjunto M es otro multiconjunto P que contiene una vez a cada letra que tiene multiplicidad impar en el multiconjunto original.

O sea, dado un multiconjunto con función de multiplicidad f_m, su paridad es el multiconjunto cuya función de multiplicidad f_p cumple que para todo $x \in U$:

$$f_p(x) = \begin{cases} 1 & \text{si } f_m(x) \text{ es impar} \\ 0 & \text{de otra forma} \end{cases}$$

o, escrito de otra forma:

$$f_p(x) = f_m(x) - 2\left\lfloor \frac{f_m(x)}{2} \right\rfloor$$

La paridad de un multiconjunto es única, porque se tiene su función de multiplicidad, y una función de multiplicidad define solo un multiconjunto.

3.2.2. Excedencia de un Multiconjunto

La **excedencia** de un multiconjunto M es un multiconjunto donde la multiplicidad de cada elemento es la parte entera de la mitad de su multiplicidad en M.

En la Tabla 3.3 aparecen ejemplos de multiconjuntos de letras y sus excedencias:

Tabla 3.3: Algunos multiconjuntos de letras y sus excedencias.

Multiconjunto	**Excedencia**
AACS	A
ABBCELRS	B
AAAALNNPG	AAN
ACEGILMORU	−
CIJLMNNOOTTUU	NOTU
CDEEENNR	EN
EJNOOOPSSS	OS

En la Tabla 3.3 puede notarse que por cada dos veces que una letra aparece en un multiconjunto aparece una vez en su excedencia. El símbolo − denota al multiconjunto vacío, que puede también ser una excedencia.

O sea, dado un multiconjunto con función de multiplicidad f_m, su excedencia es el multiconjunto cuya función de multiplicidad f_p cumple que para todo $x \in U$:

$$f_p(x) = \left\lfloor \frac{f_m(x)}{2} \right\rfloor$$

Luego, por cada dos veces que un elemento aparece en un multiconjunto M, aparecerá una vez en la excedencia de M.

La excedencia de un multiconjunto es única porque se tiene su función de multiplicidad, y una función de multiplicidad define solo un multiconjunto.

3.2.3. Correspondencia entre un Multiconjunto y la tupla de su Paridad y su Excedencia

Teorema 1 *Todo multiconjunto M se puede respresentar de manera biunívoca mediante una tupla formada por su paridad y su excedencia.*

Demostración: De las propias definiciones de paridad y excedencia se tiene que a cada multiconjunto corresponde una única paridad y una única excedencia. Solo queda entonces probar que la tupla paridad, excedencia de dos multiconjuntos diferentes no puede ser la misma.

Sean M_1 y M_2 dos multiconjuntos con funciones de multiplicidad f_1 y f_2, y que tienen la misma paridad y la misma excedencia. Sea P con función de multiplicidad f_p la paridad de ambos y sea E con función de multiplicidad f_e la excedencia de ambos.

Por las definiciones de paridad y excedencia se tiene que para toda $x \in U$ se cumplen las siguientes ecuaciones:

$$f_e(x) = \left\lfloor \frac{f_1(x)}{2} \right\rfloor \tag{3.1}$$

$$f_p(x) = f_1(x) - 2\left\lfloor \frac{f_1(x)}{2} \right\rfloor \tag{3.2}$$

Al duplicar 3.1 y sumarla a 3.2 tendremos:

$$f_1(x) = 2f_e(x) + f_p(x)$$

También por las definiciones de paridad y excedencia se tiene que para toda $x \in U$ se cumplen las siguientes ecuaciones:

$$f_e(x) = \left\lfloor \frac{f_2(x)}{2} \right\rfloor \tag{3.3}$$

$$f_p(x) = f_2(x) - 2\left\lfloor \frac{f_2(x)}{2} \right\rfloor \tag{3.4}$$

Al duplicar 3.3 y sumarla a 3.4 tendremos:

$$f_2(x) = 2f_e(x) + f_p(x)$$

Luego $f_1 = f_2$, de donde $M_1 = M_2$, porque una función de multiplicidad describe un único multiconjunto. □

Corolario 2 *Sea M un multiconjunto con función de multiplicidad f_m. Sean E y P, con funciones de multiplicidad f_e y f_p respectivamente la excedencia y la paridad de M. Entonces para todo $x \in U$ se cumple que:*

$$f_m(x) = 2f_e(x) + f_p(x)$$

3.3. Multiconjuntos y Excedencias Factibles de un Lexicón

Existen infinitos multiconjuntos de letras, pero para los algoritmos de generación de jugadas de Scrabble solo son relevantes un grupo finito de multiconjuntos: aquellos que al serle añadidas una o más letras pueden usarse en su totalidad para formar palabras válidas. A continuación se definen tales multiconjuntos y sus excedencias.

Un **multiconjunto factible** de un lexicón, es un multiconjunto M de letras, para el cual existe al menos una palabra P en el lexicón, tal que M es subconjunto del multiconjunto de las letras en P.

Por ejemplo, AABCD es un multiconjunto factible del lexicón del Español, ya que es subconjunto de las letras que forman la palabra ACABADO. VYZZZ por su parte no es un multiconjunto factible del lexicón Español, sin embargo si lo es del lexicón del Inglés, ya que esta contenido por la palabra ZYZZYVA[1].

Una **excedencia factible** de un lexicón, es la excedencia de algún multiconjunto factible.

Los multiconjuntos factibles de un lexicón de Scrabble se cuentan por millones. Por ejemplo, para el lexicón NWL23 del idioma Inglés existen más de 9 millones de multiconjuntos factibles. Sin embargo, el número de excedencias factibles en los lexicones de Scrabble es mucho menor, como puede apreciarse en la Tabla 3.4. Esto se debe a las características de los idiomas, que hacen infrecuentes las palabras con muchas repeticiones de una misma letra, o muchas letras repetidas más de una vez.

Tabla 3.4: Cantidad de excedencias factibles diferentes en el lexicón oficial para Scrabble del idioma Español, y en el lexicón nwl23 del idioma Inglés.

Idioma	**excedencias factibles**
Español	7106
Inglés	6433

3.4. Representación Paridad-Excedencia de un multiconjunto de letras

En esta sección se presenta la Representación Paridad-Excedencia (RPE), una técnica para representar biunívocamente multiconjuntos factibles como enteros de 64 bits, que permite además, realizar operaciones de multisuma de multiconjuntos en tiempo constante respecto al tamaño de los multiconjuntos involucrados. En la siguiente sección se describe el cálculo de la multisuma sobre la RPE de un multiconjunto.

Del Teorema 1 se tiene que cualquier multiconjunto se puede representar de manera biunívoca como la tupla formada por su paridad y su excedencia. Para que tal tupla quepa en 64 bits deben aprovecharse algunas propiedades de la paridad y la excedencia de multiconjuntos factibles.

[1] zyzzyva: género de escarabajos sudamericanos encontrados generalmente cerca de palmas

3.4.1. Representación Compacta de la Paridad y la Excedencia de Multiconjuntos Factibles

La paridad de un multiconjunto de letras puede ser representada como una máscara de bits del tamaño del alfabeto, porque su función de multiplicidad es binaria. Se define entonces la máscara de paridad de un multiconjunto de letras como sigue:

Máscara de Paridad : La máscara de paridad de un multiconjunto de letras M es la representación de su paridad como una máscara de bits. En la máscara de paridad a cada letra del alfabeto le corresponde un bit, donde se almacena su paridad en M. En la Tabla 3.5 aparecen algunos ejemplos de multiconjuntos y sus máscaras de paridad.

Tabla 3.5: Máscaras de paridad de algunos multiconjuntos en Español. Puede apreciarse que solo están encendidos los bits correspondientes a letras que tienen multiplicidad impar en el multiconjunto y por tanto están presentes en la paridad del multiconjunto

Multiconjunto	**Máscara de paridad**																												
	A	**B**	**C**	**CH**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**L**	**LL**	**M**	**N**	**Ñ**	**O**	**P**	**Q**	**R**	**RR**	**S**	**T**	**U**	**V**	**X**	**Y**	**Z**
AACS	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
ACEGILMORU	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0
CDEEENNR	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
AAAAGLNNP	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
AAAAEERSTTTT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Por otra parte, a cada excedencia factible de un lexicón se le puede asignar un identificador, porque la cantidad de excedencias factibles de un lexicón es finita y constante. Se define entonces el identificador de una excedencia como sigue

Identificador de una Excedencia : El identificador de cada excedencia factible es un número único y distinto para cada excedencia factible, entre 1 y la cantidad de excedencias factibles en el lexicón. El identificador de las excedencias que no sean excedencias factibles es un código de error.

En los lexicones del Español y del Inglés por ejemplo, la cantidad de excedencias factibles es menor que 2^{13}, por lo cual el identificador de cualquier excedencia factible cabe en menos de 13 bits. El alfabeto que usa el lexicón del Scrabble en Español tiene 27 letras, mientras que el usado por el lexicón del Scrabble en Inglés tiene 26 letras, por lo que la máscara de paridad y el identificador de la excedencia de cualquier multiconjunto en estos idiomas toma menos de 64 bits.

3.4.2. Representación Paridad-Excedencia de un multiconjunto de letras

La Representación Paridad-Excedencia (RPE) de un multiconjunto de letras, es un entro de 64 bits que contiene la máscara de paridad y el identificador de la excedencia de un multiconjunto.

Por convención pudieran dedicarse los 32 bits menos significativos de la RPE a la máscara de paridad y los 32 bits más significativos al identificador de la excedencia.

3.5. Multisumas Factibles

En esta sección se presenta un algoritmo, basado en la RPE para calcular multisumas factibles en tiempo constante con respecto al tamaño de los sumandos. Para ello, primero se enuncian y demuestran las propiedades de las multisumas que aprovechará el algoritmo. Luego, se describe el precómputo que usa el algoritmo y que está basado en el reducido número de excedencias factibles en los lexicones de Scrabble. Finalmente se presenta el algoritmo.

Multisuma factible : multisuma de dos multiconjuntos factibles cuyo resultado es también un multiconjunto factible.

O sea, se define como multisuma factible a las multisumas de multiconjuntos de letras que producen resultados relevantes para los algoritmos de generación de jugadas de Scrabble.

3.5.1. Propiedades de la paridad, Excedencia y el Acarreo del resultado de una multisuma

En la Tabla 3.6 aparecen algunos ejemplos de multisumas.

Tabla 3.6: Ejemplos de Multisumas. Cada ejemplo está formado por tres columnas, en la izquierda están los multiconjuntos, en el centro sus excedencias y en la derecha su paridad.

Multisuma 1

	AC	$-$	AC
$\uplus$	AS	$-$	AS
	$AACS$	A	CS

Multisuma 2

	$AACS$	A	CS
$\uplus$	$ABBCLERS$	B	$ACELRS$
	$AAABBCCELRSS$	$ABCS$	$AELR$

Multisuma 3

	$MOOS$	O	MS
$\uplus$	$AOOORS$	O	$AORS$
	$AMOOOOORSS$	OOS	$AMOR$

Multisuma 4

	$CDEEEENNR$	EN	$CDER$
$\uplus$	ENN	N	E
	$CDEEEENNNNR$	$EENN$	CDR

Una primera observación que se puede realizar de las multisumas en la Tabla 3.6 es que en el multiconjunto resultante de una multisuma, cada elemento aparece dos veces por cada vez que aparece en la excedencia de alguno de los sumandos, y una vez por cada vez que aparece en la paridad de alguno de los sumandos.

Así por ejemplo, la letra E aparece 4 veces en el resultado de la cuarta multisuma, 2 correspondientes a su aparición en la excedencia del primer sumando, y una por cada aparición en las paridades de los sumandos.

Lema 1 *La función de multiplicidad f del resultado de una multisuma cumple que para toda $x \in U$:*

$$f(x) = 2e_a(x) + 2e_b(x) + p_a(x) + p_b(x)$$

donde:

- *e_a y e_b son las funciones de multiplicidad de las excedencias de los sumandos, y*
- *p_a y p_b son las funciones de multiplicidad de las paridades de los sumandos.*

Demostración: Sean A y B dos multiconjuntos con funciones de multiplicidad f_a y f_b. Por definición de multisuma, la función de multiplicidad f_m de la multisuma de A y B cumple que para toda $x \in U$:

$$f_m(x) = f_a(x) + f_b(x) \tag{3.5}$$

Sean e_a y p_a las funciones de multiplicidad de la excedencia y la paridad de A respectivamente. Por el Corolario 2 se cumple que para toda $x \in U$:

$$f_a(x) = 2e_a(x) + p_a(x) \tag{3.6}$$

Sean e_b y p_b las funciones de multiplicidad de la excedencia y la paridad de B respectivamente. Por el Corolario 2 se cumple que para toda $x \in U$:

$$f_b(x) = 2e_b(x) + p_b(x) \tag{3.7}$$

Sustituyendo 3.6 y 3.7 en 3.5 se obtiene:

$$f_m(x) = 2e_a(x) + 2e_b(x) + p_a(x) + p_b(x) \tag{3.8}$$

□

Otra observación que se puede realizar de las multisumas en la Tabla 3.6 es que la paridad del multiconjunto resultante de una multisuma, contiene a los elementos que están presentes en la paridad de alguno de los sumandos pero no en la de ambos.

Así por ejemplo, la paridad del resultado de la tercera multisuma es AMOR, porque la letra M aparece en la paridad del primer sumando pero no en la del segundo, mientras las letras A, O y R aparecen en la paridad del segundo sumando y no en la del primero. La Letra S no aparece en la paridad del resultado de la multisuma, porque aparece en la paridad de ambos sumandos.

En este trabajo se usan los operadores binarios XOR y AND, representados con los símbolos $\oplus$ y & respectivamente.

Teorema 3 *La paridad del resultado de una multisuma, es un multiconjunto cuya función de multiplicidad p cumple que para toda $x \in U$:*

$$p(x) = p_a(x) \oplus p_b(x)$$

donde p_a y p_b son las funciones de multiplicidad de las paridades de los sumandos.

Demostración: Sean A, B y M multiconjuntos tales que $M = A \uplus B$.

Sea f la función de multiplicidad de M. Entonces por el Lema 1 se tiene que:

$$f(x) = 2e_a(x) + 2e_b(x) + p_a(x) + p_b(x) \tag{3.9}$$

donde:

- e_a y e_b son las funciones de multiplicidad de las excedencias de A y B respectivamente
- p_a y p_b son las funciones de multiplicidad de las paridades de A y B respectivamente.

Por definición de paridad de un multiconjunto, la paridad de M es un multiconjunto cuya función de multiplicidad p tiene la forma:

$$p(x) = f(x) - 2\left\lfloor \frac{f(x)}{2} \right\rfloor \tag{3.10}$$

Sustituyendo 3.9 en 3.10, y realizando trabajo algebraico se llega a:

$$p(x) = p_a(x) + p_b(x) - 2\left\lfloor \frac{p_a(x) + p_b(x)}{2} \right\rfloor$$

Como p_a y p_b solo toman valores binarios, el valor de $p(x)$ para cualquiera sea la combinación de valores de $p_a(x)$ y $p_b(x)$ es equivalente a $p_a(x) \oplus p_b(x)$, como se aprecia en la Tabla 3.7

Tabla 3.7: Valores de p para las diferentes combinaciones de valores de p_a y p_b

$p_a(x)$	$p_b(x)$	$p(x)$
0	0	0
0	1	1
1	0	1
1	1	0

Luego, para toda $x \in U$ se cumple que:

$$p(x) = p_a(x) \oplus p_b(x)$$

□

Corolario 4 *La máscara de paridad de una multisuma se puede calcular como el XOR de las máscaras de paridad de los sumandos.*

Puede observarse en las multisumas en la Tabla 3.6, que generalmente un elemento aparece tantas veces en la excedencia del resultado de una multisuma, como la suma de las veces que aparece en la excedencia de sus sumandos. Así sucede por ejemplo con las letras A, O y N en la segunda, tercera y cuarta multisumas de la Tabla 3.6.

Sin embargo en la propia Tabla 3.6 pueden encontrarse excepciones como las siguientes:

- En la primera multisuma la letra A no aparece en la excedencia de ninguno de los sumandos, pero aparece una vez en la excedencia del resultado.

- En la segunda y la tercera multisuma, la letra S no aparece en la excedencia de ninguno de los sumandos, pero aparece una vez en la excedencia del resultado.
- En la cuarta multisuma la letra E aparece una vez en la excedencia del primer sumando, y ninguna en la excedencia del segundo, pero aparece dos veces en la excedencia del resultado.

Las letras mencionadas en estas excepciones tienen en común que aparecen en la paridad de ambos sumandos de su multisuma, o sea, tienen multiplicidad impar en ambos sumandos. Tienen en común también que su multiplicidad en la excedencia del resultado de la multisuma es igual a la suma de sus multiplicidades en las excedencias de los sumandos, incrementada en 1.

Las excepciones ocurren, porque al realizar la multisuma, se acumulan los restos módulo dos de cada letra, de manera que si una letra tiene multiplicidad impar en cada uno de los sumandos, sus restos, que fueron ignorados en el cálculo de las excedencias de los sumandos, deben ser tomados en cuenta para el cálculo de la excedencia del resultado.

Se define entonces como acarreo de una multisuma al multiconjunto que contiene una vez a todas las letras que están presentes en la paridad de ambos sumandos. O sea:

Acarreo: El acarreo de la multisuma de dos multiconjuntos A y B es un multiconjunto cuya función de multiplicidad f cumple que para toda $x \in U$:

$$f(x) = p_a(x) \& p_b(x)$$

Los acarreos de las multisumas 1, 2, 3 y 4 de la Tabla 3.6 son respectivamente los multiconjuntos A, CS, S y E.

Puede apreciarse que para todas las multisumas en la Tabla 3.6, se cumple que la multiplicidad de un elemento en la excedencia de la multisuma es igual a la suma de sus multiplicidades en la excedencia de cada uno de los sumandos y en el acarreo. O sea, la excedencia de una multisuma es la multisuma del acarreo y las excedencias de los sumandos.

Teorema 5 *La excedencia del resultado de una multisuma, es un multiconjunto cuya función de multiplicidad e cumple que para toda $x \in U$:*

$$e(x) = e_a(x) + e_b(x) + r(x)$$

donde e_a y e_b son las funciones de multiplicidad de las excedencias de los sumandos y r es la función de multiplicidad del acarreo de la multisuma.

Demostración: Sean A, B y M multiconjuntos tales que $M = A \uplus B$.

Sea f la función de multiplicidad de M. Entonces por el Lema 1 se tiene que:

$$f(x) = 2e_a(x) + 2e_b(x) + p_a(x) + p_b(x), \tag{3.11}$$

donde:

- e_a y e_b son las funciones de multiplicidad de las excedencias de A y B respectivamente.
- p_a y p_b son las funciones de multiplicidad de las paridades de A y B respectivamente.

Por definición de excedencia de un multiconjunto, la excedencia de M es un multiconjunto cuya función de multiplicidad e cumple que para toda $x \in U$:

$$e(x) = \left\lfloor \frac{f(x)}{2} \right\rfloor. \tag{3.12}$$

Sustituyendo 3.11 en 3.12 y realizando trabajo algebraico se llega a:

$$e(x) = e_a(x) + e_b(x) + \left\lfloor \frac{p_a(x) + p_b(x)}{2} \right\rfloor. \tag{3.13}$$

Sea r la función de acarreo de la multisuma de A y B. Por definición de acarreo se cumple que para toda $x \in U$

$$r(x) = p_a(x) \& p_b(x).$$

Como p_a y p_b son funciones binarias, entonces se cumple para todo $x \in U$ que:

$$r(x) = \left\lfloor \frac{p_a(x) + p_b(x)}{2} \right\rfloor. \tag{3.14}$$

Luego, sustituyendo 3.14 en 3.13 se obtiene que pata toda $x \in U$:

$$e(x) = e_a(x) + e_b(x) + r(x)$$

□

Corolario 6 *La excedencia de una multisuma se puede calcular como la multisuma de las excedencias de sus sumandos y el acarreo de la multisuma.*

3.5.2. Tabla de Tríadas Factibles

Lema 2 *Tanto el acarreo de una multisuma factible, como la excedencia de su resultado son excedencias factibles.*

Demostración: Por definición de multisuma factible, el resultado de una multisuma factible es un multiconjunto factible. Por tanto la excedencia del resultado de una multisuma factible debe ser una excedencia factible.

Por el Corolario 6 se tiene que la excedencia del resultado de una multisuma se puede obtener como la multisuma del acarreo y las excedencias de los sumandos de la multisuma. Luego, en cualquier multisuma el acarreo es un subconjunto de la excedencia del resultado. Como la excedencia del resultado de una multisuma factible es una excedencia factible y el acarreo es su subconjunto, entonces el acarreo de una multisuma factible es una excedencia factible. □

Luego, para toda multisuma factible, la tríada formada por su acarreo y las excedencias de sus sumandos, es una tríada de excedencias factibles cuya multisuma es también una excedencia factible. Se puede definir estas tríadas como sigue:

Tríada Factible : Una tríada factible es una tupla de 3 excedencias factibles tal que su multisuma es también una excedencia factible.

En la Tabla 3.8 aparecen algunos ejemplos de tríadas factibles:

Tabla 3.8: Algunos ejemplos de tríadas factibles y sus multisumas. El símbolo − denota al multiconjunto vacío, que es también una excedencia factible y por tanto puede formar parte de tríadas factibles

Tríada Factible	**Multisuma**
(−, −, −)	−
(OS, O, −)	OOS
(A, C, BS)	ABCS

Como el número de excedencias factibles de cualquier lexicón es limitado y las excedencias factibles se caracterizan por ser multiconjuntos con baja cardinalidad, entonces el número de tríadas factibles de cualquier lexicón es manejable. En la Tabla 3.9 aparecen el número de tríadas factibles en los lexicones de los idiomas Español e Inglés que se usaron en este trabajo.

Tabla 3.9: Número de excedencias y tríadas factibles en Inglés y Español.

Idioma	**Excedencias Factibles**	**Tríadas Factibles**
Inglés	6433	85,141
Español	7106	112,803

Para cada lexicón, se puede precomputar una **tabla de tríadas factibles** donde a cada tríada factible se le haga corresponder su multisuma. Esta tabla puede usarse para obtener en tiempo constante el identificador de excedencia de cualquier multisuma factible, disponiendo solo de los identificadores de excedencia de los sumandos y el identificador de excedencia del acarreo.

3.5.3. Tabla de Máscaras de Acarreo

Cuando se calcula la multisuma factible de dos multiconjuntos codificados en su RPE, se dispone del identificador de excedencia de los sumandos, pero el identificador de excedencia del acarreo de la multisuma debe computarse.

De la definición de acarreo, se tiene que la función de multiplicidad del acarreo de una multisuma cumple que para toda $x \in U$:

$$f(x) = p_a(x) \& p_b(x)$$

Luego, al calcular el AND de las máscaras de paridad de los sumandos de una multisuma, se obtiene una máscara de bits, que solo tiene encendidos los bits correspondientes a las letras presentes en el acarreo de la multisuma. Se define entonces como:

Máscara de Acarreo : La máscara de acarreo de una multisuma es una máscara de bits donde están encendidos los bits correspondientes a las letras presentes en el acarreo de la multisuma. Se puede obtener como el AND de las máscaras de paridad de los sumandos de la multisuma.

Del lema 2 se tiene que en las multisumas factibles los acarreos son excedencias factibles. En particular, los acarreos son excedencias factibles que no repiten letras, pues su función de multiplicidad toma solo valores binarios.

Cada excedencia factible sin letras repetidas puede ser representada como una máscara de bits que solo tiene encendidos los bits correspondientes a las letras que contiene, o sea, como una máscara de acarreo.

Para cualquier lexicón se puede precomputar una **tabla de máscaras de acarreos**, donde a cada máscara de acarreo que represente a una excedencia factible, se le haga corresponder el identificador de la excedencia factible que representa. Esta

tabla puede usarse para obtener el identificador de excedencia del acarreo de una multisuma en tiempo constante a partir de la máscara de acarreo de la multisuma, que se obtuvo como el AND de las máscaras de paridad de sus sumandos.

3.5.4. Cálculo de Multisumas Factibles

En esta subsección se describe el algoritmo para calcular multisumas factibles en tiempo constante con respecto al tamaño de los sumandos.

El algoritmo toma la RPE de dos multiconjuntos factibles y devuelve la RPE del resultado de su multisuma si es una multisuma factible, y un código de error en otro caso.

El algoritmo calcula componente a componente la RPE de la multisuma. Primero calcula la máscara de paridad del multiconjunto resultante, luego calcula el identificador de excedencia del acarreo de la multisuma y finalmente calcula el identificador de excedencia del multiconjunto resultante.

En la Tabla 3.10 aparece un ejemplo del flujo del algoritmo.

Tabla 3.10: Ejemplos de la multisuma de los multiconjuntos AACS y ABBCELRS calculada paso a paso.

Inicio		**Paso 1**		**Paso 2**		**Paso 3**	
A	CS	A	CS	A	CS	A	CS
$\uplus$ B	$ACELRS$	$\uplus$ B	$ACELRS$	$\uplus$ B	$ACELRS$	$\uplus$ B	$ACELRS$
$-$	$-$	$-$	$AELR$	$-$	$AELR$	$ABCS$	$AELR$
				acarreo = CS			

El algoritmo usa dos tablas precomputadas: la tabla de máscaras de acarreo y la tabla de tríadas factibles.

En el algoritmo, la máscara de paridad del multiconjunto resultante se calcula como el XOR de las máscaras de paridad de los sumandos, aprovechando el Corolario 4.

Para calcular el acarreo de la multisuma se calcula la máscara del acarreo como el AND de las máscaras de paridad de los sumandos, y luego se busca en la tabla de máscaras de acarreo cual es el identificador de la excedencia factible representada por la máscara de acarreo. Si la máscara de acarreo no tuviera entrada en la tabla, entonces el acarreo de la multisuma no es un multiconjunto factible, y por tanto la multisuma no es una multisuma factible, por lo que se para la ejecución del algoritmo y se devuelve un error.

Una vez que se dispone del identificador de excedencia del acarreo, se busca en la tabla de tríadas factibles si la tríada formada por los identificadores de excedencia de los sumandos y el identificador de excedencia del acarreo es una tríada factible. De ser así, de la propia tabla se obtiene el identificador de excedencia del resultado de la

multisuma. De otra manera se devuelve un error porque el resultado de la multisuma no es un multiconjunto factible.

Cualquiera sean los multiconjuntos a sumar, el algoritmo realiza un XOR, un AND, una consulta a la tabla de máscaras de acarreo y una consulta a la tabla de tríadas factibles. Por tanto, el algoritmo calcula multisumas factibles en tiempo constante con respecto al tamaño de los sumandos.

Luego, el algoritmo realiza el mismo número de operaciones independientemente del tamaño de los multiconjuntos factibles a suma.

De un modo muy semejante a la multisuma, apenas redefiniendo el acarreo y consultando la tabla de triadas factibles de una manera diferente, se puede obtener verificar en tiempo constante si un multiconjunto es subconjunto de otro, lo cual es de especial utilidad para el Método de las Palabras Internas.

En este capítulo se presentó una forma de representación en solo 64 bits de los multiconjuntos de letras, con la cual se pueden realizar operaciones como la multisuma o la verificación de relación de que un multiconjunto es subconjunto de otro en tiempo constante con respecto al tamaño de los multiconjuntos involucrados.

Ambas operaciones se repiten una y otra vez durante el proceso de generación de jugadas válidas de cualquier algoritmo de generación de jugadas de Scrabble basados en intervalos.

Capítulo 4

Los Algoritmos de Generación de Jugadas de Scrabble Basados en Intervalos

Los algoritmos de generación de jugadas de Scrabble resuelven el problema de:

> Dados el tablero de juego y la mano de un jugador, construir la lista con todas las jugadas válidas que el jugador puede realizar en el turno, con sus correspondientes puntuaciones.

Los primeros algoritmos de generación de jugadas de Scrabble, generan las jugadas válidas en tiempo de ejecución, basándose en un autómata finito determinista que comprime el lexicón. El Método de los Anagramas supuso un cambio de paradigma: es el primer y hasta ahora único algoritmo de generación de jugadas de Scrabble por intervalos.

En este capítulo se presenta el concepto de Algoritmo de Generación de Jugadas de Scrabble basado en Intervalos y las definiciones que se usan en este trabajo para tratar con los algoritmos de generación de jugadas de Scrabble basados en intervalos.

4.1. Intervalo

Un **intervalo** es una sección de casillas del tablero sobre la que se puede colocar una palabra durante una jugada válida. En la Figura 4.1 aparecen algunos ejemplos de intervalos.

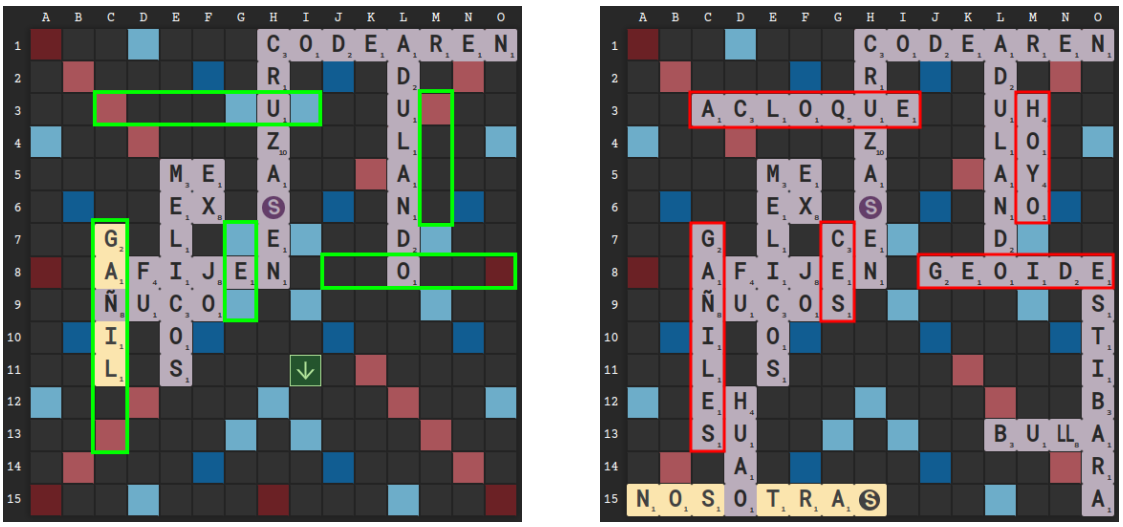

Figura 4.1: Algunos intervalos, antes y después de ser ocupados. Después de ser ocupados ya no se consideran intervalos

En la parte izquierda de la Figura 4.1 se pueden constatar las siguientes propiedades de los intervalos:

- Todas las casillas de un intervalo son consecutivas y pertenecen a la misma fila o columna.
- Un intervalo contiene al menos una casilla adyacente a una casilla ya usada del tablero.
- Un intervalo está antecedido y sucedido en la fila o columna por la que discurre por casillas vacías o el borde del tablero.

En [17], para calcular todos los intervalos de un tablero, se recorre el tablero, verificando qué secciones de casillas cumplen con las condiciones mencionadas y por tanto constituyen intervalos.

En una partida de Scrabble inicialmente hay 54 intervalos y todos contienen a la casilla central del tablero. La cantidad de intervalos cambia con cada jugada, en dependencia de las palabras jugadas.

Un intervalo puede contener casillas ocupadas por letras. Dentro del intervalo estas letras pueden estar aisladas o conectadas con otras letras formando palabras válidas que quedan completamente contenidas por el intervalo. Un intervalo puede tener varias palabras o letras aisladas en su interior, pero siempre tiene al menos una casilla vacía.

La Figura 4.2 muestra como los intervalos pueden llegar a ser realmente largos y contener combinaciones complejas de palabras y letras aisladas en su interior. Los

intervalos de esta clase suelen surgir hacia los finales de partida, y generalmente no existe ninguna palabra válida que pueda ser colocada sobre ellos.

Figura 4.2: Algunos intervalos realmente largos, sobre el tablero de los compases finales de una partida

La siguiente subsección describe de manera general a los algoritmos de generación de jugadas de Scrabble basados en intervalos.

4.2. Los Algoritmos de Generación de Jugadas de Scrabble Basados en Intervalos

Cualquier jugada válida de un juego de Scrabble consiste en la colocación de una palabra sobre un intervalo, por ello un enfoque para computar todas las jugadas válidas para un tablero de Scrabble es computar por separado todas las jugadas válidas para cada uno de sus intervalos.

La generación de todas las jugadas válidas que pueden ser colocadas en un intervalo se realiza en dos etapas:

- **Generación de Palabras Candidatas**: consiste en generar una lista de palabras cuya colocación en el intervalo potencialmente constituye una jugada válida.
- **Validación**: para cada una de las palabras candidatas, verificar la validez de la jugada resultante de su colocación en el intervalo.

Se define como **palabra candidata** de un intervalo, a una palabra que cumple una serie de restricciones que sugieren que su colocación en el intervalo es una jugada válida.

El **criterio de selección de jugadas candidatas** de un algoritmo de generación de jugadas de Scrabble basado en intervalos, es el conjunto de restricciones que una palabra debe cumplir para considerarse jugada candidata de un intervalo según el algoritmo.

Se define como **problema Mano-Intervalo**, al problema de dado un intervalo y la mano de un jugador, obtener la lista con todas las palabras candidatas del intervalo.

Para cada algoritmo de generación de jugadas de Scrabble basado en intervalos, la complejidad de resolver un problema Mano-Intervalo depende de su criterio de selección de jugadas candidatas.

Para determinar si una jugada es válida, es necesario comprobar que cada letra colocada pertenece al Conjunto de Letras Admisibles de la casilla donde fue colocada. Como consecuencia, la verificación de la validez de la colocación de una palabra candidata tiene, en el peor caso, una complejidad lineal con respecto a su longitud.

Por ello es importante para la eficiencia de los algoritmos de generación de jugadas de Scrabble basados en intervalos, que su critrerio de selección de jugadas candidatas escoja pocos candidatos por cada instancia del problema Mano-Intervalo, sin dejar de considerar por supuesto como palabra candidata a ninguna jugada válida.

Un algoritmo de generación de jugadas de Scrabble basado en intervalos será más eficiente en la medida en que su criterio de selección de jugadas candidatas le permita considerar menos palabras candidatas y resolver más eficientemente las instancias del problema Mano-Intervalo.

4.3. El Método de los Anagramas

El Método de los Anagramas es un algoritmo de generación de jugadas de Scrabble. Como tal, genera todas las jugadas válidas que se pueden colocar en un tablero intervalo a intervalo. Considera una instancia del problema Mano-Intervalo por cada intervalo del tablero.

Se define como **multiconjunto combinado** de un problema Mano-Intervalo a la multisuma de las letras de la mano y las letras ya colocadas en el intervalo.

Para cualquier instancia del problema Mano-Intervalo, el Método de los Anagramas considera como palabras candidatas a aquellas palabras tales que el multiconjunto de sus letras es subconjunto del multiconjunto combinado.

Para resolver un problema Mano-Intervalo donde el intervalo tiene v casillas vacías, el Método de los Anagramas realiza $\binom{7}{v}$ consultas a una tabla de anagramas, tal y como se describió en el capítulo de Preliminares.

El número de intervalos sobre un tablero de Scrabble hacia los finales de partida es mucho mayor que en los comienzos. En la Figura 4.3 se puede apreciar el número promedio de intervalos antes de cada jugada de un partido de Scrabble.

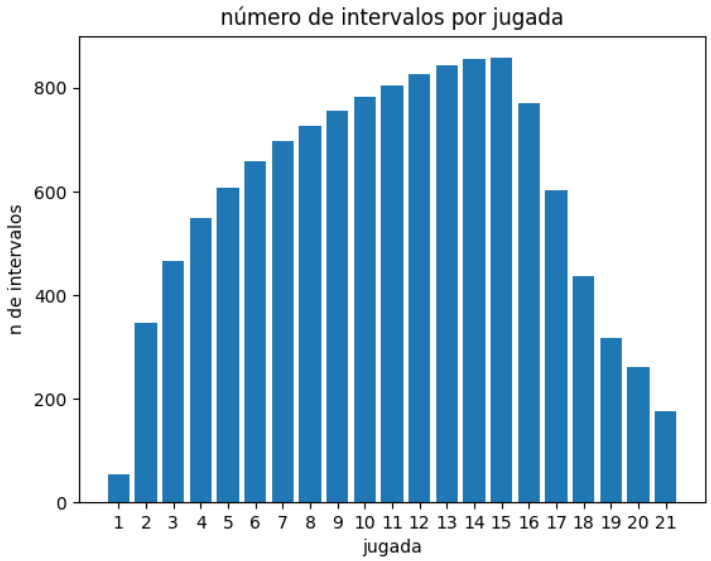

Figura 4.3: Cantidad promedio de Intervalos en el Tablero de Scrabble antes de cada jugada.

Como el Método de los Anagramas considera una instancia del problema Mano-Intervalo por cada intervalo del tablero, y resuelve a todas las instancias del problema Mano-Intervalo de la misma manera, entonces su rendimiento disminuye a medida que el número de intervalos en el tablero aumenta.

En el siguiente capítulo se presenta un algoritmo de generación de jugadas de Scrabble basado en intervalos, cuyo criterio de selección de jugadas candidatas le permite podar el número de intervalos para los que considera instancias del problema Mano-Intervalo y además considerar menos palabras candidatas por instancia.

Capítulo 5

Método de las Palabras Internas

En este capítulo se propone el Método de las Palabras Internas, un algoritmo de generación de jugadas de Scrabble basado en intervalos.

Para su selección de jugadas candidatas, el algoritmo considera las restricciones impuestas por las letras y palabras ya colocadas dentro de los intervalos, lo cual le permite considerar menos jugadas candidatas que el Método de los Anagramas y resolver de manera más eficiente todas las instancias de Problema Mano-Intervalo.

5.1. Palabras Internas

En esta sección se presenta el concepto de palabra interna, y otros conceptos relacionados, que constituyen el fundamento del Método de las Palabras Internas presentado en este capítulo.

5.1.1. Palabras Internas

Se define como **palabra interna** de una palabra, a cada una de sus subcadenas que constituye una palabra válida o una letra.

Por ejemplo, la palabra CORAZON tiene 12 palabras internas:

- C, COR[1] y CORA[2], comenzadas en la posición 0.
- O y ORA comenzadas en al posición 1.
- R, RA[3] y RAZON comenzadas en la posición 2.

[1] cor: forma en desuso de referirse al corazón

[2] cora: En la España musulmana, división territorial poco extensa.

[3] ra: interjección, normalmente repetida tres veces, para animar a un equipo deportivo o a una persona.

- A, Z, O y N comenzadas en las posiciones 3, 4, 5 y 6.

De las 12 palabras internas de CORAZON, solo 2 tienen longitud mayor que 3. En general, al recorrer cualquier lexicón contando las ocurrencias de palabras internas, se puede verificar que la mayor parte de las ocurrencias de palabras internas corresponden a palabras muy pequeñas, como se aprecia en la Tabla 5.1.

Tabla 5.1: Porcentaje del total de las ocurrencias de palabras internas atribuidos a palabras internas de tamaños 1, 2, 3 y 4 en adelante en los lexicones de los idiomas Inglés y Español

Longitud	Inglés	Español
1	49,8%	44,0%
2	21,6%	18,7%
3	12,8%	11,1%
4+	15,8%	26,2%

Pero en cualquier lexicón el número de palabras pequeñas es minúsculo comparado al tamaño del lexicón, como se aprecia en la Tabla 5.2. Ello provoca que unas pocas palabras internas, tengan una frecuencia de ocurrencia enorme, mientras la mayor parte de las palabras internas sean subcadenas de solo unas pocas palabras.

Tabla 5.2: Número de palabras de longitud 2, 3 y 4 en adelante en los lexicones del Español y del Inglés.

Longitud	Inglés	Español
2	107	91
3	1086	548
4+	195 447	638 654

Se define como **frecuencia** de una palabra interna en el lexicón, a la cantidad de palabras del lexicón que la contienen como palabra interna.

En la Tabla 5.3 aparecen algunos datos sobre las palabras internas de los lexicones del Español y el Inglés:

5.1.2. Palabras Internas Colocadas

Se define como **palabra interna colocada** de una palabra, a la tupla formada por una de sus palabras internas, la posición donde comienza y la longitud de la palabra.

En la Tabla 5.6 aparecen las palabras internas colocadas de CORAZON:

Tabla 5.3: Algunas estadísticas sobre las palabras internas y sus frecuencias en los lexicones del Inglés y el Español. Las siglas P. I. se refieren a las Palabras Internas

	Inglés	Español
Palabras del Lexicón	196 640	639 293
Ocurrencias de Palabras Internas	3 567 631	14 441 912
N. de Palabras Internas Diferentes	87 835	303 073
% de P. I. con frecuencia 1	54.2%	42.7%
% de P. I. con frecuencia 2 o menos	65.9%	49.2%
% de P. I. con frecuencia 5 o menos	82.8%	76.6%
% de P. I. con frecuencia 10 o menos	89.5%	85.2%
% de P. I. con frecuencia 30 o menos	95.4%	92.8%
Máxima frecuencia de una P. I.	202 247	1 057 232
P. I. con mayor frecuencia	E	A

Tabla 5.4: Palabras Internas Colocadas de CORAZON. El 7 al final de cada una se refiere a que la longitud de CORAZON es 7

$(C,0,7)$	$(A,3,7)$	$(N,6,7)$	$(CORA,0,7)$
$(O,1,7)$	$(Z,4,7)$	$(RA,2,7)$	$(ORA,1,7)$
$(R,2,7)$	$(O,5,7)$	$(COR,0,7)$	$(RAZON,2,7)$

Para cualquier lexicón, el número de palabras internas colocadas diferentes es mucho menor que la cantidad de ocurrencias de palabras internas colocadas.

Esto se debe a que en cualquier lexicón, las mismas palabras internas se repiten una y otra vez a lo largo de todo el lexicón, ocupando las mismas posiciones en palabras de la misma longitud, o sea, las mismas palabras internas colocadas se repiten una y otra vez.

Se define como **frecuencia** de una palabra interna colocada en el lexicón, a la cantidad de palabras del lexicón que la contienen como palabra interna colocada.

En la Tabla 5.5 aparecen algunos datos sobre las palabras internas de los lexicones del Español y el Inglés:

A modo de curiosidad puede notarse que la palabra interna colocada de mayor frecuencia es la letra S colocada en la última posición de palabras de longitud 9 y 11.

5.1.3. Palabra Interna Colocada en un Intervalo

Los intervalos pueden tener letras o palabras en su interior.

Se define como **palabra interna** de un intervalo a cualquier letra o palabra contenida en su interior.

Tabla 5.5: Algunas estadísticas sobre las palabras internas y sus ocurrencias en los lexicones del Inglés y el Español. Las siglas P. I. se refieren a las Palabras Internas

	Inglés	Español
Palabras del Lexicón	196 640	639 293
Ocurrencias de Palabras Internas Colocadas	3 567 631	14 441 912
N. de Palabras Internas Colocadas Diferentes	315 743	1 059 816
% de P. I. C. con frecuencia 1	57.1%	51.6%
% de P. I. C. con frecuencia 2 o menos	73.0%	72.0%
% de P. I. C. con frecuencia 5 o menos	86.4%	83.4%
% de P. I. C. con frecuencia 10 o menos	89.5%	92.0%
% de P. I. C. con frecuencia 30 o menos	96.5%	97.6%
Máxima frecuencia de una P. I. C.	12 131	54 049
P. I. C con la mayor frecuencia	$(S, 8, 9)$	$(S, 10, 11)$

Las palabras internas de un intervalo imponen restricciones sobre las palabras que pueden construirse en el intervalo. Por ejemplo, en el intervalo *a4-h4* de la Figura 5.1 solo pueden construirse palabras de longitud 8 comenzadas por PARPAD[4], como es el caso de las palabras PARPADOS y PARPADEO.

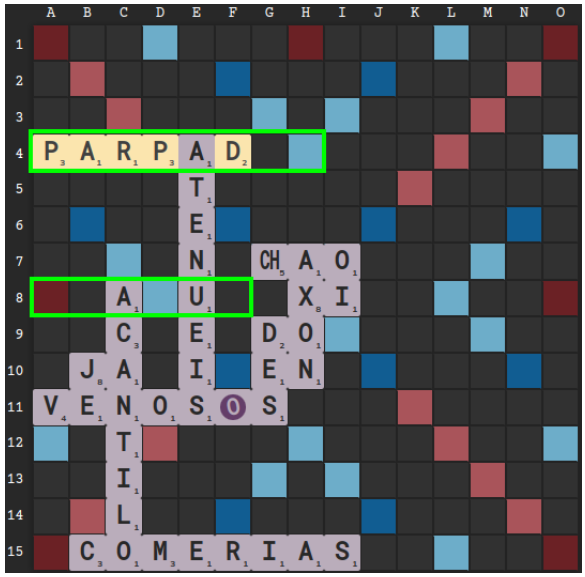

Figura 5.1: Dos intervalos que contienen palabras internas

Se define entonces como **palabra interna colocada** de un intervalo, a las tuplas

[4]parpad: imperativo del verbo parpar. Parpar es dar graznidos un pato

formadas por una de sus palabras internas, la posición del intervalo donde comienza y la longitud del intervalo.

Cada palabra interna colocada en un intervalo, restringe las palabras que pueden ser construidas sobre el intervalo a aquellas palabras del lexicón que la contienen como palabra interna colocada.

Como se vio en la subsección anterior, la frecuencia de una palabra interna colocada está relacionada inversamente con su longitud. Luego, generalmente la palabra interna colocada de mayor longitud de un intervalo es la que más restringe la lista de palabras que pueden ser construidas sobre el intervalo.

Se define entonces como **palabra interna principal** de un intervalo, a su palabra interna colocada de mayor longitud.

La mayor parte de los intervalos que aparecen durante un juego de Scrabble contienen exactamente una palabra interna colocada. En la Figura 5.2 se muestra la frecuencia promedio de aparición de intervalos por su cantidad de palabras internas a lo largo de las jugadas de una partida de Scrabble. Estos datos fueron calculados simulando 1000 partidas.

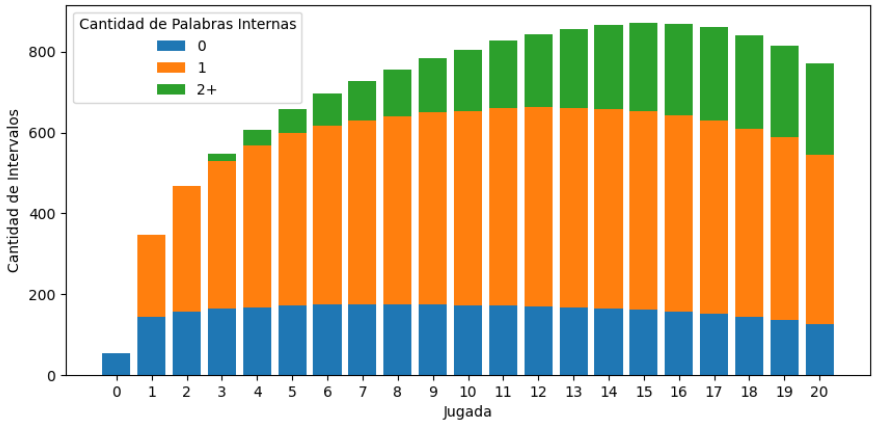

Figura 5.2: Se muestra, jugada a jugada de un partido de Scrabble, el promedio de intervalos con 0, 1 y 2 palabras internas sobre el tablero

Entonces, para la mayor parte de los intervalos que aparecen durante una partida de Scrabble, la restricción impuesta por la palabra interna principal del intervalo no solo es la más restrictiva de las impuestas por sus palabras internas, sino que es la única.

El Método de las Palabras Internas, toma en cuenta la restricción impuesta por la palabra interna principal de un intervalo, tanto para determinar que problemas Mano-Intervalo deben ser considerados, como para la selección de las jugadas candidatas de cada uno de ellos.

5.2. Poda de Instancias del Problema Mano-Intervalo

El Método de las Palabras Internas es un algoritmo de generación de jugadas de Scrabble basado en intervalos, o sea, divide el problema general de los algoritmos de generación de jugadas de Scrabble por intervalos.

Existen intervalos en los que, no importa cual sea la mano del jugador en turno, no es posible realizar ninguna jugada.

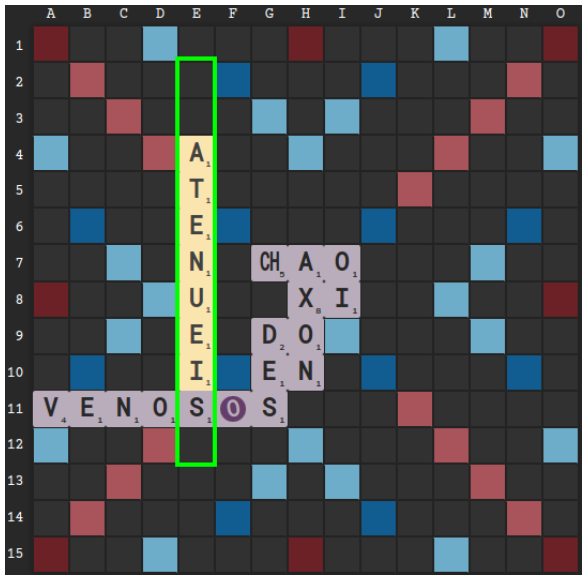

Figura 5.3: Ninguna palabra válida puede ser construida en el intervalo enmarcado

Por ejemplo, en el intervalo enmarcado en la Figura 5.3, la palabra interna principal es $(ATENUEIS, 2, 11)$ y no existe ninguna palabra del lexicón del idioma Español que contenga tal palabra interna colocada. Por lo tanto cualquier problema Mano-Intervalo que tenga como palabra interna principal a $(ATENUEIS, 2, 11)$, puede ser descartado de inmediato del proceso de generación de jugadas.

Tampoco es posible realizar una jugada válida en un intervalo donde alguna de las casillas tenga un conjunto de letras admisibles vacío.

El Método de las Palabras Internas en su proceso de generación de jugadas válidas solo considera los problemas Mano-Intervalo correspondientes a intervalos que cumplen que:

- Su palabra interna principal es una palabra interna colocada del lexicón.
- Todas sus casillas tienen un conjunto de letras admisibles no vacío.

Para verificar el cumplimiento de ambas condiciones basta tener precomputadas la lista de palabras internas colocadas que aparecen en el lexicón y mantener correctamente calculados a lo largo de la partida los conjuntos de letras admisibles de cada casilla.

En la siguiente sección se describe como el Método de las Palabras Internas selecciona las jugadas candidatas para los problemas Mano-Intervalo que considera.

5.3. Selección de Palabras Candidatas

Para cualquier instancia del problema Mano-Intervalo, el Método de las Palabras Internas selecciona como palabras candidatas a aquellas que:

- Contienen a la palabra interna principal del intervalo.
- El multiconjunto de sus letras es subconjunto del multiconjunto combinado del problema.

Para calcular la lista de palabras que cumplen ambas restricciones, el Método de las Palabras Internas elige entre dos métodos diferentes de solución del problema Mano-Intervalo, el de bajo nivel, que debe su nombre a que divide el problema en instancias más pequeñas, y el de alto nivel que recibe su nombre por contraposición.

En esta sección se presentan las soluciones de alto y bajo nivel de un problema Mano-Intervalo.

Para elegir entre ambos métodos, el Método de las Palabras Internas usa un hiperparámetro, llamado umbral de decisión, sobre el que se discute en la siguiente sección.

5.3.1. Solución de Alto Nivel de un problema Mano-Intervalo

La **solución de alto nivel** de un problema Mano-Intervalo, consiste en:

1. Obtener la lista de palabras que contienen a la palabra interna principal del intervalo.
2. Filtrar la lista obtenida, conservando aquellas palabras cuyas letras formen un subconjunto del multiconjunto combinado del problema.

Por ejemplo, en el problema Mano-Intervalo representado en la Figura 5.4, la palabra interna principal del intervalo es $(PARPAD, 0, 8)$ y la mano del jugador está formada por las letras del multiconjunto AEHLNOU.

Figura 5.4: Un problema Mano-Intervalo, donde la mano del jugador aparece al pie de la foto y el intervalo es el enmarcado

Existen solo 4 palabras en todo el lexicón del Español que contienen a la palabra interna colocada $(PARPAD, 0, 8)$, y son:

Tabla 5.6: Palabras del lexicón del Español que contienen a la palabra interna colocada $(PARPAD, 0, 8)$

PARPADEA	PARPADEE
PARPADEO	PARPADOS

El multiconjunto combinado del problema es AAADEHLNOPPRU, o sea la multisuma del multiconjunto de las letras en la mano del jugador y el multiconjunto de las letras ya colocadas en el intervalo

PARPADEA y PARPADEO son las únicas palabras de esa lista que cumplen que el multiconjunto de letras que las forma es subconjunto del multiconjunto combinado del problema, por tanto la lista de palabras candidatas contiene solo a estas dos palabras.

En la sección de Precómputo se describe como calcular de antemano para cualquier palabra interna colocada del lexicón, la lista de palabras que la contiene. Por ello, se puede obtener en tiempo constante la lista de palabras que contiene a la palabra interna principal de cualquier Problema Mano-Intervalo.

Del Capítulo de multiconjuntos se tiene que se puede verificar si un multiconjunto es subconjunto de otro en tiempo constante respecto a sus tamaños haciendo uso de sus representaciones paridad-excedencia.

Luego, la cantidad de operaciones que realiza la solución de alto nivel del problema Mano-Intervalo depende únicamente de la cantidad de palabras que contienen a la palabra interna principal del multiconjunto. Esto la hace ideal para aquellos problemas Mano-Intervalo cuya palabra interna principal tenga una frecuencia baja en el lexicón

5.3.2. Solución de Bajo Nivel de un problema Mano-Intervalo

Para calcular todas las palabras candidatas de un problema Mano-Intervalo, se puede calcular de manera independiente las palabras candidatas que aportan cada uno de los subconjuntos de la mano. Se define entonces:

Problema Multiconjunto-Intervalo : Dado un multiconjunto y un intervalo, calcular todas las palabras que se pueden construir combinando las letras del multiconjunto y del intervalo, y que contienen a la palabra interna principal del intervalo.

La **solución de bajo nivel** de un problema Mano-Intervalo donde el intervalo tiene v casillas vacías, consiste en descomponer el cálculo de jugadas candidatas en instancias del problema Multiconjunto-Intervalo, una por cada subconjunto de tamaño v de la mano.

Por ejemplo, en el problema Mano-Intervalo representado en la Figura 5.5, la palabra interna principal del intervalo es $(I, 3, 7)$ y la mano del jugador está formada por las letras del multiconjunto CELMORS.

Como el intervalo tiene 6 casillas vacías, la solución de bajo nivel del problema Mano-Intervalo, descompone el problema en una instancia del problema Multiconjunto-Intervalo por cada multiconjunto de tamaño 6 de la mano del jugador.

De estas instancias algunas aportan palabras candidatas y otras no. Por ejemplo, con el subconjunto de las letras de la mano CELMOS, y la I del intervalo se pueden formar dos palabras que contienen la palabra interna $(I, 3, 7)$ y por tanto son palabras candidatas: COLIMES[1] y MELICOS[2].

En cambio, con el subconjunto de las letras de la mano CELORS, y la letra I del intervalo se puede construir la palabra LICORES, pero esta no es candidata porque no contiene a la palabra interna principal del intervalo.

En la Tabla 5.7 aparecen los 6 multiconjuntos de las letras de la mano CELMORS, y junto a ellos las palabras candidatas que aporta la solución de los problemas Multiconjunto-Intervalo que le corresponden, de haber alguna.

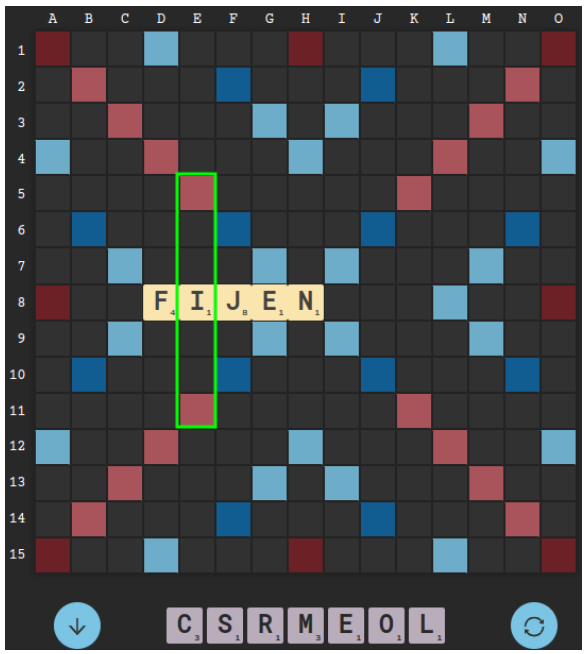

Figura 5.5: Un problema Mano-Intervalo. La mano del jugador aparece al pie de la imagen y el intervalo es el enmarcado

Tabla 5.7: Multiconjuntos de tamaño 6 de la mano del Problema 1, y las soluciones que aportan al Problema 1, de haber alguna.

Multiconjunto	**Soluciones**
CELMOR	-
CELMOS	COLIMES, MELICOS
CELMRS	-
CELORS	-
CEMORS	CREIMOS
ELMORS	MORILES[3]

En la sección de Precómputo se describe como calcular de antemano las palabras candidatas para cada instancia del problema Multiconjunto-Intervalo, de manera que puede obtenerse su solución en tiempo constante.

La solución de bajo nivel de un problema Mano-Intervalo considera $\binom{7}{v}$ instancias

[1]colimes: segunda persona del subjuntivo del verbo colimar. Colimar es obtener o concentrar un haz de rayos paralelos a partir de un foco luminoso.

[2]mélicos: pertenecientes o relativos al canto.

[3]moriles: vino fino que se cría y elabora en el término municipal de Moriles, en la provincia de Córdoba, España.

del problema Multiconjunto-Intervalo, donde v es la cantidad de casillas vacías del intervalo. Como la solución de cada instancia del problema Multiconjunto-Intervalo se puede obtener en tiempo constante, entonces el número de operaciones que realiza la solución de bajo nivel del Problema Mano-Intervalo depende solo de la cantidad de casillas vacías del intervalo.

5.4. Umbral de Decisión

Cuál de los dos métodos de solución de un problema Mano-Intervalo es más eficiente para una instancia en específico depende de las características del intervalo de la instancia, en particular, de la frecuencia en el lexicón de su palabra interna principal y de su cantidad de casillas vacías.

En este trabajo se propone un enfoque sencillo para seleccionar el método de solución de cada instancia del problema Mano-Intervalo. Consiste en fijar un umbral de decisión. Todas las instancias cuya palabra interna principal tenga una frecuencia en el lexicón por debajo del umbral de decisión, se resuelven usando la solución de alto nivel, mientras que el resto se resuelve usando la solución de bajo nivel.

Como el umbral de decisión es fijo, se puede conocer de antemano para cada palabra interna colocada del lexicón, el método por el que serán resueltas las instancias de la cual sea palabra interna principal. Ello permite optimizar el uso de memoria del precómputo tal y como se detallará en la sección del Precómputo.

El umbral de decisión para el cual se alcanza el mejor rendimiento en cuanto a velocidad varía de un lexicón a otro y de una etapa del juego a otra, y debe calcularse experimentalmente.

Al usar el umbral de decisión para seleccionar el método por el cual se resuelve una instancia del problema Mano-Intervalo, se acota el número de operaciones que se realiza al máximo entre:

- La cantidad de operaciones que realiza la solución de alto nivel de un problema Mano-Intervalo donde la palabra interna principal tiene una frecuencia en el lexicón igual al umbral de decisión.
- La cantidad de operaciones que realiza la solución de bajo nivel.

5.5. Precómputo

De las sección de Selección de Palabras Candidatas, se tiene que para el cálculo de las palabras candidatas, el Método de las Palabras Internas necesita precomputar:

- Todas las palabras internas colocadas del lexicón.

- Para cada palabra interna colocada del lexicón, la lista de las palabras del lexicón que la contienen.
- Las palabras candidatas para cada instancia del problema Multiconjunto-Intervalo.

En esta sección se describe el cálculo y almacenamiento de estas tres clases de datos, en un diccionario de frecuencias, una tabla de alto nivel y una tabla de bajo nivel respectivamente. Se analiza el tamaño de estas estructuras y como puede ser optimizado aprovechando el umbral de decisión fijo. Finalmente se describe una forma compacta de codificar palabras, palabras internas colocadas e instancias del problema multiconjunto-intervalo.

5.5.1. Tabla de Frecuencias de Palabras Internas Colocadas

Para calcular todas las palabras internas colocadas de una palabra basta iterar por todas sus subcadenas, verificado cuales son palabras del lexicón.

Para computar una tabla que contenga la frecuencia en el lexicón de cada palabra interna colocada, basta recorrer el lexicón, contando para cada una de sus palabras, las palabras internas colocadas que contiene.

5.5.2. Tabla de Alto Nivel

Se llama **tabla de alto nivel** a una estructura de datos que se utiliza para almacenar para cada palabra interna colocada del lexicón todas las palabras que la contienen.

Para evitar manejar muchas listas por separado, la tabla de alto nivel consiste en una **lista de candidatos**, y una **tabla de índices** que a cada palabra interna colocada le hace corresponder los índices de la lista de candidatos entre los que se encuentran ubicadas las palabras que la contienen.

Para computar la tabla de alto nivel se procede como sigue:

1. Se computa la lista de ocurrencias de palabras internas colocadas del lexicón.

   Una lista de ocurrencias de palabras internas colocadas es una lista de tuplas, donde cada tupla está compuesta por una palabra interna colocada y una palabra del lexicón que la contiene. Para computar la lista de ocurrencias basta recorrer todo el lexicón y para cada una de sus palabras, calcular todas sus palabras internas colocadas.

2. Se ordena la lista de ocurrencias, de manera que todas las tuplas correspondientes a las mismas palabras internas colocadas queden agrupadas juntas.

3. Se crea la tabla de índices, asignando a cada palabra interna colocada, los índices de la lista de ocurrencias ordenada entre los que aparecen las tuplas que le corresponden.

4. Se crea la lista de candidatos, compuesta por los segundos componentes de cada tupla de la tabla de ocurrencias, en el mismo orden en que aparecen en la tabla de ocurrencias.

La tabla de alto nivel debe su nombre a que es empleada por la solución de alto nivel del problema Mano-Intervalo para encontrar todas las palabras que contienen a determinada palabra interna colocada en tiempo constante.

5.5.3. Tabla de Bajo Nivel

Las soluciones de una instancia del problema Multiconjunto-Intervalo son las palabras que son anagramas del multiconjunto y contienen a la palabra interna principal del intervalo como palabra interna colocada. Por tanto una palabra es solución de una instancia del problema Multiconjunto-Intervalo por cada una de sus palabras internas colocadas. Una palabra también es solución de la instancia del problema Multiconjunto-Intervalo donde el intervalo está vacío y el multiconjunto es el multiconjunto de sus letras.

Por ejemplo, en la Tabla 5.8 aparecen algunas instancias del problema Multiconjunto-Intervalo de las que la palabra CORAZON es solución. Todas están compuestos por una palabra interna colocada de CORAZON y el multiconjunto de las letras de CORAZON. No es necesario representar la longitud del intervalo, ya que en cualquier instancia del problema Multiconjunto-Intervalo, la longitud del intervalo es igual al tamaño del multiconjunto de letras a emplear:

Tabla 5.8: Algunas instancias del problema Multiconjunto-Intervalo de las que CORAZON es solución. La cuarta se refiere a la instancia que surge en los intervalos vacíos, cuando el multiconjunto de letras de la mano del jugador a utilizar es el multiconjunto de las letras que forman la palabra CORAZON.

$$\begin{array}{cc} (C, 0, ACNOORZ) & (A, 3, ACNOORZ) \\ (RA, 2, ACNOORZ) & (-, -, ACNOORZ) \end{array}$$

Se llama **tabla de bajo nivel** a una estructura de datos que se utiliza para almacenar para cada instancia del problema Multiconjunto-Intervalo que tiene alguna solución, todas sus soluciones.

Para evitar manejar muchas listas por separado, la tabla de bajo nivel está compuesta por:

lista de candidatos : una lista de palabras que es la concatenación de las soluciones de todas las instancias del problema Multiconjunto-Intervalo que tienen alguna solución.

tabla de índices : una tabla, donde a cada instancia del problema Multiconjunto-Intervalo, se le hace corresponder los índices de la lista de candidatos entre los que se encuentran ubicadas sus soluciones.

Para calcular ambos componentes se siguen los siguientes pasos:

1. Se computa una lista de ocurrencias de problemas Multiconjunto-Intervalo.

   Una lista de ocurrencias de problemas Multiconjunto-Intervalo, es una lista de tuplas, donde cada tupla está compuesta por una instancia del problema Multiconjunto-Intervalo y una de sus soluciones. Para computar la lista de ocurrencias basta iterar por todo el lexicón y para cada una de sus palabras, generar todas las instancias del problema Multiconjunto-Intervalo de las que es solución, como se describió anteriormente.

2. Se ordena la lista de ocurrencias, de manera que todas las tuplas correspondientes a las mismos instancias del problema Multiconjunto-Intervalo queden agrupadas.

3. Se crea la tabla de índices, iterando sobre la tabla de ocurrencias y asignando a cada instancia de problema Multiconjunto-Intervalo los índices entre los que se encuentran todas sus soluciones.

4. Se crea la lista de candidatos, sustituyendo cada tupla de la tabla de ocurrencias por solo su segundo elemento, o sea, la palabra que es solución de la instancia del problema Multiconjunto-Intervalo contenida en la tupla.

La tabla de bajo nivel debe su nombre a que es empleada por la solución de bajo nivel de los problemas Mano-Intervalo, para obtener en tiempo constante las soluciones de cualquier instancia del problema Multiconjunto-Intervalo.

Sobre la representación compacta de las instancias de los problemas Multiconjunto-Intervalo y de sus soluciones se discute en la subsección dedicada a las representaciones compactas.

5.5.4. Optimización usando el Umbral de Decisión

Como el umbral de decisión es fijo, las palabras internas colocadas del lexicón se pueden separar en dos clases, las comunes, y las poco comunes, en dependencia de si su frecuencia en el lexicón es mayor o no que el umbral de decisión.

Todas las instancias del problema Mano-Intervalo cuya palabra interna principal sea una palabra interna colocada poco común, son resueltas por el Método de las Palabras Internas usando la solución de bajo nivel del problema Mano-Intervalo.

Todas las instancias del problema Mano-Intervalo cuya palabra interna principal sea una palabra interna colocada común, son resueltas por el Método de las Palabras Internas usando la solución de bajo nivel del problema Mano-Intervalo.

Luego, las tablas de bajo y alto nivel, pueden ser computadas teniendo en cuenta solo a las palabras internas colocadas comunes y poco comunes respectivamente; en lugar de todas las palabras internas colocadas para cada una.

De la forma en que se realiza el precómputo, se obtienen las siguientes dos propiedades relativas a las dimensiones del precómputo:

- La suma de las longitudes de las listas de candidatos del precómputo es igual a la cantidad de ocurrencias de palabras internas colocadas en el lexicón, independientemente de cuál sea el umbral de decisión.
- La cantidad total de entradas en las tablas de índices del precómputo es siempre menor que la cantidad de ocurrencias de palabras internas colocadas en el lexicón.

O sea, el tamaño del precómputo del Método de las Palabras Internas es del orden de la cantidad de ocurrencias de palabras internas en el lexicón.

De hecho, la cantidad total de entradas en tablas de índices del precómputo disminuye ligeramente al aumentar el umbral de decisión, como se puede apreciar en la Figura 5.6

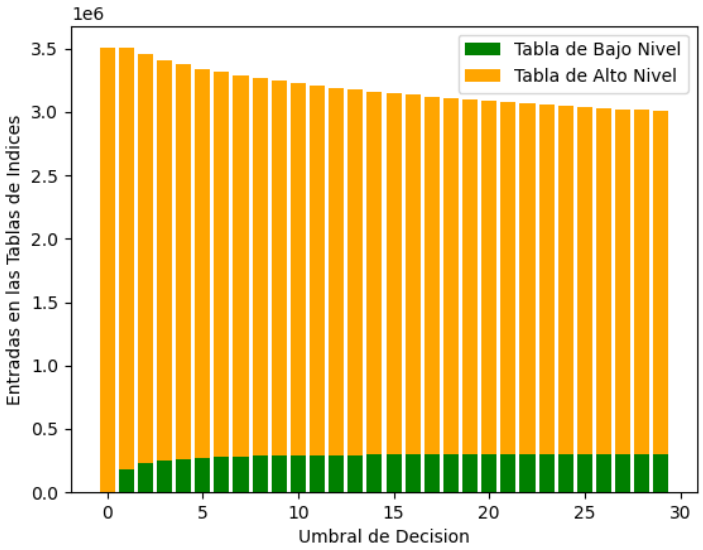

Figura 5.6: Distribución de entradas entre las tablas de índices de bajo y alto nivel para los valores de umbral de decisión entre 1 y 30

5.5.5. Representaciones Compactas

En el Método de las Palabras Internas se le asigna un identificador a cada palabra del lexicón, a cada letra del alfabeto y además a la cadena vacía de la siguiente manera:

- A la cadena vacía se asigna el identificador 0.
- A las letras del alfabeto se les asigna los identificadores entre 1 y la longitud del alfabeto.
- A las palabras que aparecen alguna vez como palabras internas del lexicón, se les asignan identificadores consecutivos a partir de la longitud del alfabeto en adelante.
- A las palabras del lexicón que nunca aparecen como palabra interna, se le asignan los identificadores más bajos aún sin ocupar.

Por ejemplo, en la Tabla 5.9 aparecen los identificadores asignados para las palabras del Español y del Inglés.

De esta manera, cada palabra del lexicón puede ser representada con apenas 20 bits, al menos en los casos de los lexicones de Español e Inglés.

Tabla 5.9: Identificadores asignados por el Método de las Palabras Internas a la cadena vacía, las letras, las palabras que aparecen alguna vez como palabra interna y las que no.

	Inglés	Español
cadena vacía	0	0
letras	1 - 26	1 - 28
palabras internas	27 - 87 862	29 - 303 102
otras palabras	87 863 - 196 667	303 103 - 639 322

Como en la generalidad de los lexicones menos de la mitad de las palabras ocurren como palabra interna, al haberles asignado los identificadores más bajos, se puede representar a las palabras internas con un bit menos que el necesario para representar cualquier palabra. Por tanto, cualquier palabra interna puede ser representada con apenas 19 bits, al menos en los casos de los lexicones de Español e Inglés.

Las palabras internas colocadas son tuplas formadas por una palabra interna, la posición donde aparece en la palabra que la contiene y la longitud de la palabra que la contiene. Como las palabras usadas en el Scrabble tienen una longitud hasta 15, sus longitudes y los índices correspondientes a sus posiciones interiores pueden ser representados usando apenas 4 bits. Entonces, una palabra interna colocada se puede representar ubicando como sea conveniente para la implementación que se realice los 19 bits de la palabra interna, los 4 bits de la posición que ocupa y los 4 bits de la longitud de la palabra que la contiene.

Cada instancia del problema Multiconjunto-Intervalo puede ser descrita por el multiconjunto disponible y la palabra interna principal del intervalo.

Del capítulo de los multiconjuntos se tiene que un multiconjunto puede ser representado en 64 bits usando su representación paridad-excedencia. Tal representación divide los 64 bits en dos mitades, dedicando la mitad de los bits menos significativos a la máscara de paridad del multiconjunto y la otra mitad al identificador de excedencia del multiconjunto. En ambas mitades quedan bits sin usar:

- En la mitad correspondiente a los bits menos significativos, quedan $32 - l$ bits sin usarse donde l es la longitud del alfabeto y por tanto la longitud de la máscara de bits de un multiconjunto.
- En la otra mitad quedan $32 - e$ bits sin usarse, donde e es la cantidad de bits necesarios para representar el identificador de excedencia de un multiconjunto.

En cualquier idioma, donde hayan 28 letras o menos, habrá 4 bits sin ocupar en la mitad de los bits menos significativos, lo cual es espacio suficiente para representar la posición donde comienza la palabra interna principal de un intervalo.

Para los lexicones del Español y del Inglés, el identificador de excedencia de cualquier multiconjunto cabe en 13 bits, por lo que en los 19 restantes se puede colocar el identificador de una palabra interna, que usa precisamente 19 bits.

Luego, tanto para el Español como para el Inglés, se puede representar ambos componentes de un problema Multiconjunto-Intervalo dentro de un solo entero de 64 bits. En la Tabla 5.10 se describe como quedaría la distribución de los bits.

Tabla 5.10: Distribución de los 64 bits de un entero para representar un problema Multiconjunto-Intervalo

Bits	Contenido
0 - 27	Máscara de Paridad del Multiconjunto
28 - 31	Posición de la Palabra Interna
32 - 44	Identificador de Excedencia del Multiconjunto
45 - 63	Identificador de la Palabra Interna

5.6. El Algoritmo

El Método de las Palabras Internas es un algoritmo de generación de jugadas de Scrabble. Como tal, genera todas las jugadas válidas que se pueden colocar en un tablero intervalo a intervalo. Considera instancias del problema Mano-Intervalo solo en aquellos intervalos cuya palabra interna principal sea una palabra interna del lexicón, y que tengan alguna casilla con conjunto de letras admisibles no vacío.

Para resolver una instancia del problema Mano-Intervalo, el Método de las Palabras Internas selecciona entre el método de solución de alto nivel y el método de solución de bajo nivel, en dependencia de si la palabra interna principal tiene una frecuencia de ocurrencia en el lexicón por debajo o por encima del umbral de decisión.

El método de solución de alto nivel de una instancia del problema Mano-Intervalo, lo descompone en instancias del problema Multiconjunto-Intervalo, una por cada multiconjunto de la mano con tantas letras como casillas vacías tenga el intervalo. Para cada instancia del problema Multiconjunto-Intervalo resultante, obtiene directamente de la tabla de alto nivel todas sus palabras candidatas.

El método de solución de bajo nivel de una instancia del problema Mano-Intervalo, obtiene de la tabla de bajo nivel la lista de todas las palabras que contienen a la palabra interna principal del intervalo. Luego filtra la lista, tomando solo como palabras candidatas aquellas cuyo multiconjunto de letras sea subconjunto del multiconjunto combinado del problema Mano-Intervalo.

Tras obtener la lista de palabras candidatas de un intervalo, el Método de las Palabras Internas selecciona como jugadas válidas para el intervalo solo a aquellas

que de ser colocadas en el intervalo, todas sus letras quedarían ubicadas en casillas a cuyo conjunto de letras admisibles pertenecen.

5.7. Conclusiones

En este capítulo se presentó el Método de las Palabras Internas, un algoritmo de generación de jugadas de Scrabble basado en intervalos, cuyo criterio de selección de palabras candidatas se centra en la palabra interna principal de cada intervalo.

Se presentaron los conceptos de palabra interna, palabra interna colocada del lexicón y palabra interna principal de un intervalo y se describió como estos conceptos pueden ser explotados para generar eficientemente una lista de palabras candidatas reducida para cualquier problema Mano-Intervalo.

Capítulo 6

Resultados

El Método de los Anagramas y el Método de las Palabras Internas son ambos algoritmos de generación de jugadas basados en intervalos, o sea, solo difieren en su forma de solucionar los problemas Mano-Intervalo. Ello permitió implementarlos a ambos, sobre su base común, y realizar pruebas que comparan no solo el tiempo que toman para la generación de jugadas, sino también la cantidad de instancias del problema Mano-Intervalo, o la cantidad de candidatos que consideran en el proceso.

Para las pruebas se simularon las mismas 1000 partidas para ambos algoritmos. Se usó el lexicón del Inglés. Como los algoritmos se limitan a listar todas las jugadas posibles, entonces las pruebas consistieron en partidas donde el jugador en turno siempre juega de manera *greedy*, o sea, la jugada de mayor puntuación. El umbral de decisión del Método de las Palabras Internas usado en las pruebas fue 10.

En la Figura 6.1 se aprecia la evolución del promedio de instancias del problema Mano-Intervalo que ambos algoritmos consideran en cada turno.

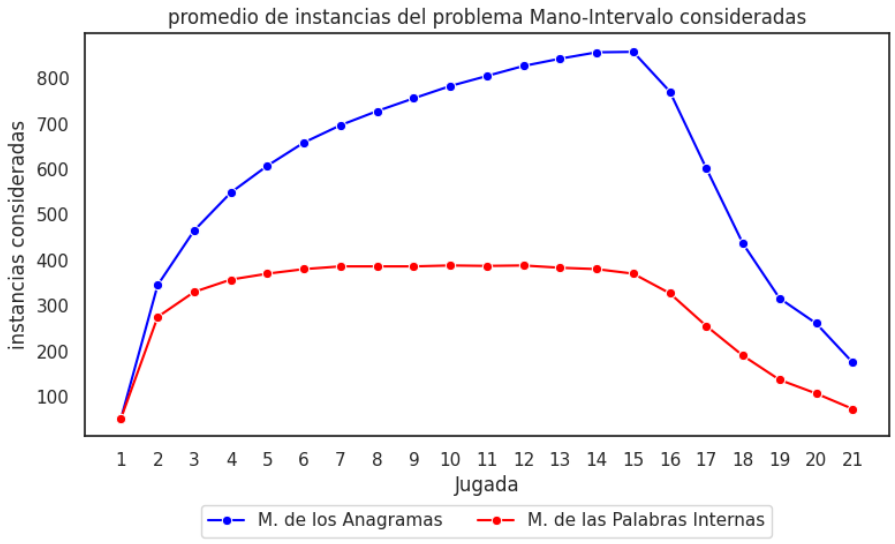

Figura 6.1: Comparación de los algoritmos en cuanto a la cantidad promedio de instancias del problema Mano-Intervalo consideradas en cada turno

Para el Método de los Anagramas, como se había expuesto anteriormente, la cantidad promedio de instancias del problema Mano-Intervalo consideradas en cada turno, crece a lo largo de toda la partida, decreciendo solo hacia el final.

El Método de las Palabras Internas, en cambio, considera una cantidad más o menos constante de instancias del problema Mano-Intervalo a lo largo de toda la partida. Sucede así porque solo considera aquellos problemas Mano-Intervalo donde ninguna de las casillas tiene un conjunto de letras admisibles vacío y la palabra interna principal es una palabra interna colocada del lexicón.

En la Figura 6.2 se aprecia que el Método de las Palabras Internas considera una cantidad menor de palabras candidatas por cada problema Mano-Intervalo considerado que el Método de los Anagramas. Esto se debe a que el criterio de selección de jugadas candidatas del Método de las Palabras Internas es más restrictivo que el del Método de los Anagramas.

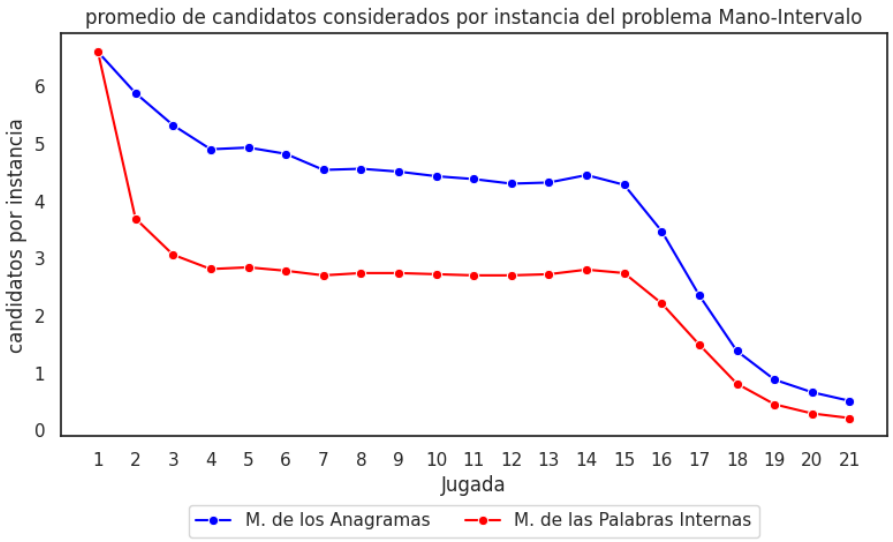

Figura 6.2: Comparación de los algoritmos en cuanto a la cantidad promedio en cada turno, de palabras candidatas consideradas por problema Mano-Intervalo.

En la Figura 6.3 se puede apreciar que la cantidad de palabras candidatas consideradas por el Método de las Palabras Internas es aproximadamente constante a lo largo de la partida y marcadamente menor que la cantidad de palabras candidatas consideradas por el Método de los Anagramas.

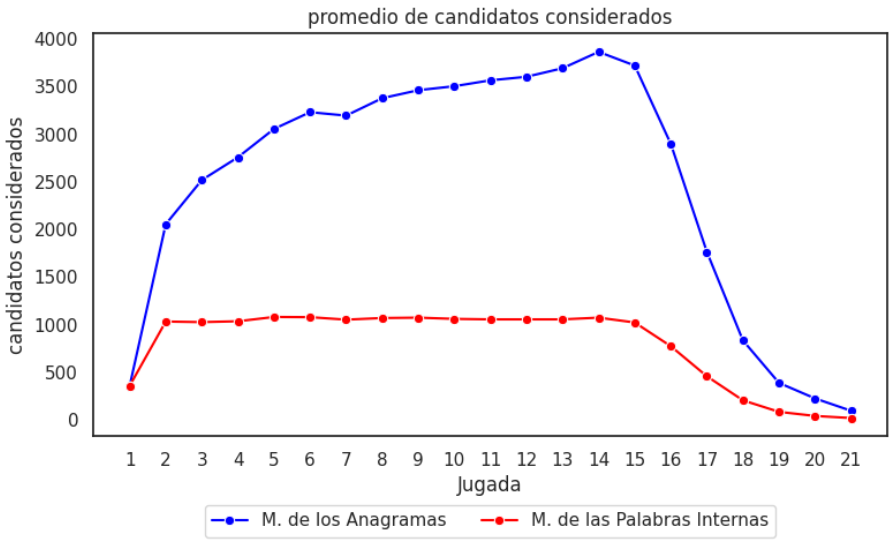

Figura 6.3: Comparación de los algoritmos en cuanto a la cantidad promedio de palabras candidatas consideradas en cada turno

Este resultado es una combinación de los resultados anteriores, o sea, de que el Método de las Palabras Internas considera una cantidad menor de instancias del problema Mano-Intervalo, y además para cada instancia considera menos palabras candidatas.

Todos estos resultados redundan en que el Método de las Palabras Internas obtenga mejores tiempos que el Método de los Anagramas, como se puede apreciar en la Figura 6.4

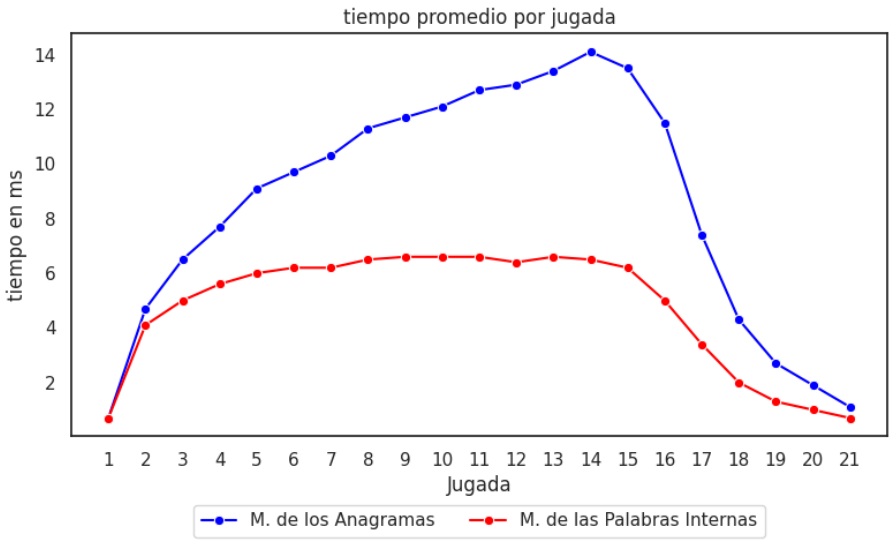

Figura 6.4: Comparación de los algoritmos en cuanto al tiempo promedio que emplean en cada turno para generar todas las jugadas válidas

Los resultados de la experimentación permiten constatar que a debido a su criterio de selección de palabras candidatas más restrictivo, el Método de las Palabras Internas, considera menos instancias del problema Mano-Intervalo y menos palabras candidatas que el Método de los Anagramas, lo que se traduce en un mejor en rendimiento.

Conclusiones

En este trabajo se presentó un nuevo algoritmo para la generación de jugadas en Scrabble: el Método de las Palabras Internas.

Se realizó una revisión bibliográfica exhaustiva sobre los algoritmos existentes en la literatura, identificando al Método de los Anagramas como el más eficiente reportado hasta la fecha, aunque su rendimiento decae en las últimas jugadas. El Método de los Anagramas es el primer y único algoritmo en la literatura que divide el cálculo de jugadas válidas por intervalos.

En este trabajo, se propuso una definición general para los algoritmos de generación de jugadas de Scrabble basados en intervalos, que incluye al Método de los Anagramas.

Se diseñó y presentó el Método de las Palabras Internas, un algoritmo basado en intervalos cuyo criterio de selección de jugadas candidatas se centra en la restricción impuestas por la palabra más larga situada dentro de cada intervalo.

Se realizaron experimentos para comparar el rendimiento del Método de las Palabras Internas con el del Método de los Anagramas en distintos aspectos, como la cantidad de jugadas candidatas consideradas, la cantidad de instancias del Problema Mano-Intervalo consideradas y el tiempo requerido para la generación de jugadas válidas. Los resultados mostraron que Método de las Palabras Internas mantiene un rendimiento uniforme a lo largo de toda la partida, y superior al del Método de los Anagramas.

La necesidad de una representación eficiente para los multiconjuntos en el Método de las Palabras Internas, dio lugar a la concepción de la Representación Paridad-Excedencia, que también se discutió en este trabajo. La Representación Paridad-Excedencia es una manera de representar multiconjuntos de letras en solo 64 bits, que permite realizar multisumas y otras operaciones entre multiconjuntos de letras en tiempo constante con respecto a su tamaño.

Recomendaciones

La investigación realizada en este trabajo puede ser extendida en varias direcciones.

Sería conveniente investigar, en distintos lexicones, los valores del umbral de decisión que optimizan el rendimiento del Método de las Palabras Internas. Asímismo se puede investigar una variante del Método de las Palabras Internas, donde el umbral de decisión no sea fijo.

La Representación Paridad-Excedencia propuesta en este trabajo, puede emplearse para una representación eficiente de grafos dirigidos donde los nodos son multiconjuntos y las aristas son letras. Tales grafos pueden usarse para facilitar el trabajo con comodines en los algoritmos de generación de jugadas de Scrabble.

Los algoritmos de Generación de Jugadas de Scrabble basados en Intervalos, incluido el Método de las Palabras Internas pueden ser implementados en paralelo, porque resuelven de manera independiente cada instancia del problema Mano-Intervalo. Una línea de investigación por explorar es el estudio de las posibilidades que una implementación en paralelo brindaría al Método de las Palabras Internas.

Finalmente, se sugiere el empleo del Método de las Palabras Internas en la implementación de algún motor de juego de Scrabble, que podrá beneficiarse de la mejora en rendimiento aportada por el Método de las Palabras Internas para realizar simulaciones más exhaustivas.

Bibliografía

[1] L. V. Allis. «Chess, Shogi, Go, natural developments in game research». En: *ICGA Journal* 19.2 (1996), págs. 103-112 (vid. pág. 2).

[2] A. Appel y S. Jacobson. «The World Fastest Scrabble Program». En: *ACM SIGACT News* 20.1 (1989), págs. 50-52 (vid. págs. 2, 11, 13, 16).

[3] W. D. Blizard. «The Development of Multiset Theory». En: *Modern Logic* 1 (1991), págs. 319-352 (vid. pág. 20).

[4] W. D. Blizzard. «Multiset Theory». En: *Notre Dame Journal of Formal Logic* 30.1 (1989), págs. XX-XX (vid. pág. 20).

[5] A. Blumer, J. Blumer y D. Haussler. «The smallest automaton recognizing the subwords of a text». En: *Theoretical Computer Science* 40 (1985), págs. 31-55 (vid. pág. 10).

[6] International Scrabble Club. *Dictionaries*. `https://www.isc.ro/en/commands/dictionaries.html` (vid. pág. 5).

[7] CodeHappy. *ELISE: Análisis y simulación de Scrabble*. `https://www.codehappy.net/elise` (vid. págs. 1, 15).

[8] 14 Domino. *Scrabble is Nowhere Close to a Solved Game*. `https://medium.com/@14domino/scrabble-is-nowhere-close-to-a-solved-game-6628ec9f5ab0` (vid. págs. 1, 2, 11).

[9] Domino14. *Macondo: A Scrabble Solver*. `https://domino14.github.io/macondo/` (vid. págs. 1, 15).

[10] Stefan Fatsis. «Winning Scrabble and the Nature of Expertise». En: *Scientific American* () (vid. pág. 1).

[11] Filexico. *Torneos de Scrabble*. `https://www.filexico.com/torneos` (vid. pág. 1).

[12] A. González et al. «Heuri: A Scrabble Playing Engine Using a Probability-Based Heuristic». En: *ICGA Journal* 44.4 (2022), págs. 190-202 (vid. pág. 1).

[13] S. Gordon. «A Faster Scrabble Move Generation Algorithm». En: *Software: Practice and Experience* 24.2 (1994), págs. 219-232 (vid. págs. 2, 10, 11, 15, 16).

[14] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997, p. 132 (vid. pág. 10).

[15] J. Katz-Brown. *How Quackle Plays Scrabble*. `https://people.csail.mit.edu/jasonkb/quackle/doc/how_quackle_plays_scrabble.html` (vid. págs. 1, 15).

[16] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3rd. Addison-Wesley, 1998, p. 488 (vid. págs. 10, 20, 21).

[17] A. Romero et al. «El método de anagramas: un rápido y novedoso algoritmo para generar jugadas de Scrabble». En: *Research in Computing Science* 147 (2018), págs. 41-55 (vid. págs. 2, 11, 12, 15-17, 19, 38).

[18] Federación Internacional de Scrabble en Español (FISE). *Modalidad Clásica - Reglamentos*. `https://fisescrabble.org/reglamentos/modalidad-clasica/` (vid. pág. 4).

[19] Fédération Internationale de Scrabble Francophone (FISF). *Présentation des Championnats du Monde Francophones de Scrabble*. `https://www.fisf.net/competitions/championnats-du-monde-francophones/presentation.html` (vid. pág. 1).

[20] B. Sheppard. «World Championship Caliber Scrabble». En: *Artificial Intelligence* 134.1–2 (2002), págs. 241-275 (vid. pág. 1).

[21] Crossword Solver. *Scrabble Championship: WESPA Championship List*. `https://www.crosswordsolver.com/scrabble-word-finder/scrabble-championship/world#wespac` (vid. pág. 1).

[22] Jerome Wolfe. «The World Competitive Scrabble». En: *Psychology Today* () (vid. pág. 1).

[23] WordsRated. *Scrabble Statistics: Facts, Data, and Trends for 2024*. `https://wordsrated.com/scrabble-statistics/` (vid. pág. 1).