Medidas de dificultad en problemas de ajedrez mediante ordenamiento de jugadas

Mateo Marenco[1], Ariel Arbiser[1]

Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Ciudad Universitaria, Buenos Aires, Argentina
mateomarenco74@gmail.com, arbiser@dc.uba.ar

Resumen Presentamos criterios y métodos computacionales para estimar la dificultad de problemas de ajedrez para jugadores humanos mediante técnicas de aprendizaje automático, extrapolables a otros juegos. En este contexto, el enfoque propuesto busca emular la forma y el orden en que los humanos consideran jugadas candidatas, asignándole puntajes a estas, para luego predecir la dificultad.

Keywords: ajedrez · problema · dificultad · aprendizaje automático · árbol significativo

1. Introducción

El ajedrez es un juego finito [19,18,15] pero computacionalmente complejo para el que no hay perspectivas claras de resolverlo en un futuro cercano, es decir, de dar un algoritmo o método práctico que, dada una posición, permita en un tiempo razonable hallar la mejor jugada disponible, o bien exhibir el valor [15] de la posición[1] [19,15,14,1]. No conocemos el valor de la posición inicial, y aunque lo descubriéramos posiblemente nunca sabremos cuáles son los movimientos iniciales que mantienen ese valor. Sin embargo, sí conocemos los valores de posiciones simples. Por ejemplo, una posición donde las negras tienen una secuencia forzada de jaque mate tiene valor -1, una posición donde sólo quedan los reyes en el tablero tiene valor 0, y una posición donde el blanco ya dio jaque mate o hay mate en 2 jugadas tiene valor 1.

Vinculada con el valor de una posición, pero más orientada a la comprensión de los jugadores humanos, surge la cuestión de la **dificultad** de la misma. El

[1] Estas dos formulaciones se pueden considerar equivalentes. Toda vez que podamos conocer el valor de una posición, el encontrar la mejor jugada es tan sencillo como calcular el valor de las posiciones que resultan de realizar cada una de las (finitas) jugadas legales, para seleccionar cualquiera de las que mantiene el valor (por finitud siempre existe alguna). Por otro lado, si podemos conocer la mejor jugada en una posición dada, entonces podemos conocer el valor de dicha posición efectuando la secuencia de mejores jugadas hasta alcanzar una posición donde la partida está terminada (con jaque mate o tablas); el valor de la posición final es el mismo que el de la inicial, dado que por definición las mejores jugadas preservan el valor.

objetivo de este trabajo es estudiar la noción de dificultad en ajedrez y luego poder calcularla. Más concretamente, abordamos métodos computacionales para estimar numéricamente la dificultad de una posición dada. No usaremos cualquier tipo de posición sino que nuestra fuente de datos constará específicamente de problemas tácticos de ajedrez surgidos en partidas reales. Un problema táctico consiste en una posición que deberá ser analizada por el jugador y en la que deberá encontrar la mejor jugada disponible, para lo cual usualmente hace falta analizar todas y cada una de las respuestas posibles del rival, y tener una respuesta preparada para cada una. En otras palabras, el jugador debe hallar una estrategia ganadora a partir de la posición planteada. Como ejemplo de un problema podemos observar la posición de la Fig. 1, cuya solución comienza con la jugada **f7h6**.

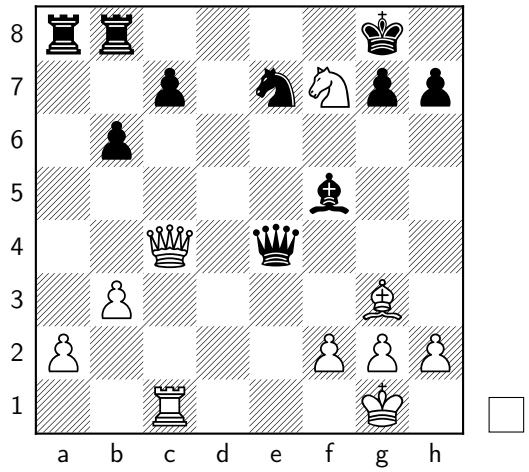


Figura 1: Juegan las blancas y dan jaque mate en 3 jugadas

La tarea de estimar la dificultad de una posición es desafiante incluso para jugadores fuertes: Hristova, Guid y Bratko [6] notaron que los humanos, aun expertos o maestros en este juego, no son especialmente buenos estimando la dificultad de problemas tácticos de ajedrez. También observaron que no existe una fuerte correlación positiva entre el nivel de juego de un ajedrecista y su capacidad para estimar la dificultad de un problema.

Algunas de las medidas de dificultad consideradas en este trabajo se apoyan en un motor de ajedrez para proveer la evaluación de una posición dada. En particular se hace uso del motor **Stockfish 13**, que actualmente figura en los primeros puestos de los rankings de los motores de ajedrez[2].

[2] Estos pueden consultarse en https://computerchess.org.uk/ccrl/4040/.

Si bien nos concentramos en el ajedrez, muchas de las ideas que estudiamos son extensibles a otros dominios. Del mismo modo, ha habido reciente interés en el problema de medir la dificultad de resolver una tarea, y en particular de la búsqueda de jugadas buenas u óptimas, en juegos tan diversos como las Torres de Hanoi [11], Baguenaudier [12], el juego del 15 [17], el problema del viajante de comercio [2], Sokoban [10] y Sudoku [16].

1.1. Aplicaciones de las estimaciones de dificultad

La estimación de la dificultad en ajedrez permite por ejemplo presentar problemas a jugadores humanos apropiados, es decir conociendo las posibilidades esperadas de resolverlos para un tipo y un nivel de entrenamiento requeridos. En este sentido, la plataforma Lichess – https://lichess.org/ – genera problemas automáticamente sobre la base de partidas existentes y, cuando encuentra una posición que cumple con las características de un problema, lo incorpora a su base de datos. Sin una forma de estimar este nivel, se corre el riesgo de plantear el problema a jugadores a quienes resolverlos les resulte demasiado sencillo o demasiado costoso. Dicho esto, Lichess en particular aprovecha su gran número de usuarios para estimar la dificultad de los problemas dinámicamente, según si los jugadores resuelven los problemas correctamente o no, mediante un sistema como Glicko2 [3]. Además, Lichess hace pública su base de datos de problemas, junto con sus soluciones y sus puntajes de dificultad, por lo que constituye una fuente ideal para probar la calidad de nuestras estimaciones.

Para programas que juegan juegos, y específicamente para motores de ajedrez, la posibilidad de incorporar el factor de dificultad da mejores o más amplios criterios de decisión. Así, entre jugadas candidatas que tengan una evaluación similar, podrá ser conveniente elegir la que resulte en la posición más difícil para el rival. Esto puede resultar especialmente útil en situaciones en que la posición es perderdora; en lugar de maximizar la evaluación, podría ser más conveniente maximizar la dificultad de la posición resultante, o bien un punto de compromiso entre ambos objetivos.

Otra aplicación inmediata es la detección automática de problemas: intuitivamente un problema es más interesante si es más difícil. Plataformas como Lichess pueden analizar automáticamente las partidas que se juegan en su sitio y seleccionar aquellas posiciones que tengan las características de un problema – como tener una solución única – y que tengan un puntaje de dificultad suficientemente alto. Además, algunos autores han estudiado nociones de estética y detección de temas en ajedrez [9,8,7,5], las cuales podrían combinarse con medidas de dificultad para generar problemas de calidad.

1.2. Organización de esta presentación

Comenzamos motivando el concepto de dificultad como juego en forma normal para luego introducir el modelo a utilizar en la estimación de la dificultad. Luego exhibimos ejemplos y resultados. Finalmente exponemos conclusiones y

distintas posibilidades futuras. El contenido de este artículo consiste en una adaptación de parte de [13] a lo que se agrega la sección 2.

2. Dificultad como juego en forma normal

Damos en esta primera parte una formulación teórica del problema general de plantear un problema de una dificultad dada, esto es, ofrecer a un jugador la posibilidad de que resuelva un problema con una secuencia de pruebas de jugadas candidatas, sea esta "esperada" o no, para estudiarlo en forma estratégica. Más precisamente, teniendo en cuenta que cada secuencia tiene un costo, proponemos la posibilidad de que un problema tenga distintos niveles de dificultad, y el jugador deberá decidir el plan de acción, esto es, el orden de estas secuencias en la búsqueda de jugadas.

Modelamos entonces la situación como juego en forma normal [15] para dos jugadores, donde el jugador I elige decide la dificultad del problema a proponer entre d posibles, y el jugador II lo deberá resolver. Este precisará considerar distintas posibilidades de secuencias de jugadas, desde la de menor costo hasta la de mayor costo[3], decidiendo el orden en que considerará dichas candidatas. Siendo m las secuencias habrá $m!$ estrategias (en el sentido de decisiones) posibles. Dado que el jugador II toma esta decisión en forma prácticamente simultánea con la del jugador I, puesto que al observar el problema en principio no cuenta con información objetiva de la solución, podemos modelar la situación de este modo. De hecho, como juego de suma cero, dado que consideramos que I busca que la resolución tenga un costo total (eg. tiempo de cálculo) elevado y II desea resolverlo con costo mínimo. Sea w la ganancia del jugador II por resolver el problema, sean $c_1 < c_2 < \ldots < c_m$ los costos de las $m \geq 2$ secuencias posibles de jugadas – para simplificar la formulación y fijar ideas, podemos suponer que sólo hablamos de la clave. Así, la función de pagos se puede presentar en forma de matriz de $d \times m!$ en la que las d filas son las estrategias de I y las columnas – jugadas clave a considerar – las de II. Es natural suponer además que durante el proceso de búsqueda, una vez elegida la columna por II, este interrumpirá las pruebas cuando llegue a la solución. Llamaremos *total* a cada secuencia que considere todas las jugadas candidatas 1 a m.

Lema 1. *En este juego, si $w > \Sigma_{1 \leq i \leq m}\ c_i$, todas las columnas que representan secuencias no totales resultan débilmente dominadas, y no hay otras filas ni columnas dominadas.*

Luego, bajo una hipótesis de racionalidad mutua, estas columnas pueden eliminarse, obteniéndose un juego de dimensión $d \times m!$.

Lema 2. *Bajo las condiciones mencionadas, este juego no tiene equilibrios de Nash puros (ni valor puro).*

[3] Asumimos que el costo de la jugada es fácilmente identificable o, en última instancia, calculable eficientemente, como en el modelo que se propondrá a continuación.

Dado que el juego es de dos jugadores, finito y de suma cero, por el teorema de Von Neumann tiene valor mixto [15].

Un simple ejemplo es tomar $c_1 = c_2 = c_3 = \cdots = c_m$, para el que una estrategia óptima del jugador I será la distribución $(1/m, 1/m, ..., 1/m)$.

Para el caso de 2 órdenes (jugadas) posibles, donde las 2 filas son respectivamente fácil y difícil, y las columnas representan las 4 secuencias (totales y no totales) candidatas: j_1; j_2; j_1j_2; j_2j_1. Podemos simplificar ordinalmente el juego de modo que $w = 0$ y la jugada fácil tenga costo 1, con c el costo de la otra jugada, obteniéndose el juego de matriz de 2×4

w-1	-c	w-1	w-c-1
-1	w-c	w-c-1	w-c

Asumimos que el costo insumido en encontrar la jugada adecuada justifica la ganancia cuando el problema se resuelve, i.e. $1 < c \leq w$.

Lema 3. *Las columnas 1 y 2 están débilmente dominadas por alguna de las otras. Del mismo modo, para todo m, cualquier secuencia no total (en que el jugador no llega a la jugada ganadora) será una columna débilmente dominada.*

Proposición 1. *En las condiciones del juego anterior, existe un equilibrio de Nash mixto dado por las probabilidades* $p = q = 1/(c+1)$.

Demostración. Para calcular el equilibrio mixto, por invariancia bajo transformaciones afines no triviales podemos suponer $w = 0$ y a su vez simplificar considerando la matriz con signo opuesto, que será

1	c+1
c+1	c

Si $(p, 1-p)$ y $(q, 1-q)$ son respectivamente las estrategias mixtas de I y II:

1. Si I elige las filas 1 y 2 con prob. p y $1-p$, la mejor respuesta de II es $argmin\ _{0\leq q\leq 1}\ q(p + (1-p)(c+1)) + (1-q)(p(c+1) + (1-p)c) = argmin\ _{0\leq q\leq 1}\ q(1-p(c+1)) + (p+c)$, mínimo que ocurre en uno u otro extremo según el signo de la pendiente, dando 1 si $p \geq 1/(c+1)$ y 0 en caso contrario.
2. Si II elige las columnas 1 y 2 con prob. q y $1-q$, por simetría la mejor respuesta de I es 1 si $q \leq 1/(c+1)$ y 0 en caso contrario.

El equilibrio se da con ambas desigualdades, i.e. $p = q = 1/(c+1)$.

Nótese que para el caso $c = 1$ se tiene una traslación de *Matching Pennies* (cara y ceca) con pagos no igualitarios.

Encontramos casos de existencia de posiciones y secuencias cruzadas de jugadas con roles de "explosivas" (capturas, jaques, coronaciones) y "no explosivas", lo cual hace relevante el análisis posterior:

Proposición 2. *Existen P_1, P_2, P_3, P_4 problemas de mate en 1 jugada todos con las mismas piezas, sobre posiciones alcanzables en partidas, tales que existen jugadas distintas del blanco j_1, j_2, j_3, j_4 de un mismo tipo de pieza (especificadas como casilla de origen y casilla de destino) donde, para cada $1 \leq i \leq 4$, en P_i las 4 jugadas son legales pero la única solución es j_i.*

Proposición 3. *Existen P_1, P_2, P_3, P_4 problemas de mate en 2 jugadas todos con las mismas piezas, sobre posiciones alcanzables en partidas, tales que existen jugadas distintas del blanco j_1, j_2, j_3, j_4 de la misma pieza (especificadas como en la proposición anterior) donde, para cada $1 \leq i \leq 4$, en P_i las 4 jugadas son legales pero la solución tiene clave única j_i, y tales que entre esas 4 jugadas sólo una es jaque y sólo una (distinta de la anterior) es captura.*

3. Un modelo de aprendizaje con jugadas candidatas

Stoiljkovikj, Bratko y Guid [21,20] propusieron un modelo basado en *árboles significativos* para estimar la dificultad de problemas de ajedrez. En un comienzo hemos replicado parcialmente estos resultados modificando el algoritmo con el objetivo de mejorar la calidad y la eficiencia de las estimaciones [13]. Analizamos los distintos atributos que se extraen de los árboles y su aporte a los modelos de aprendizaje automático para estimar la dificultad. Concluimos que el concepto de árbol significativo posee algunas limitaciones que dificultan obtener mejores resultados, especialmente por el hecho de hay muchos casos de problemas que según este modelo tienen distinta dificultad pero presentan árboles significativos isomorfos [13].

En vista de dichas limitaciones, el modelo de predicción de dificultad que presentamos intenta replicar la forma en que los jugadores humanos consideran las opciones disponibles, basándonos en la siguiente idea: mientras más alternativas incorrectas se consideren antes de llegar a la solución, más difícil será el problema[4]. Imitar el proceso de pensamiento humano en este contexto es una tarea extremadamente compleja, pero ofrecemos un modelo sencillo que, aunque no sea completamente fiel a lo que necesariamente haría un humano, se acerca lo suficiente como para realizar predicciones útiles.

Basándonos especialmente en nuestra experiencia jugando al ajedrez y resolviendo problemas, así como en la implementación de algoritmos de juego y sumando cierto grado de prueba y error, hemos diseñado un esquema de asignación de costos a jugadas. Se basa en la observación de que, durante un proceso de resolución, las jugadas que se suelen considerar antes como posibles soluciones

[4] Uno de los atributos considerados en [21,20] – *SeeminglyGood*, que estima una bondad aparente de la jugada – seguía esta misma idea, pero un inconveniente con su definición es que permite muchos falsos positivos: una jugada que captura un caballo con la dama podría ser ganadora si no fuera porque ese caballo estaba defendido por un peón que puede simplemente recapturar, y así este tipo de jugada era considerada como *aparentemente buena*.

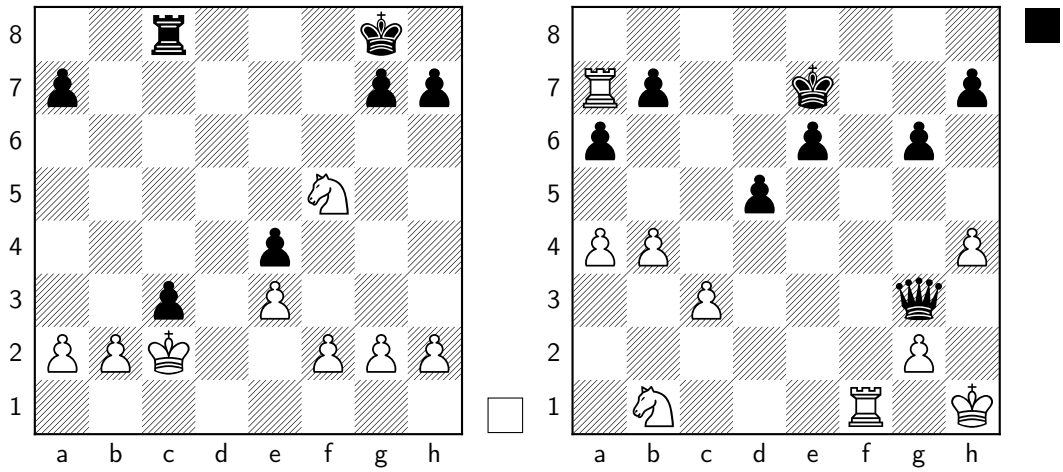
tienden a ser explosivas, y entonces jugadas como jaques y capturas tienen consecuentemente un costo menor (representando el hecho de que son más fáciles de encontrar) que jugadas "tranquilas", mereciendo estas últimas un costo mayor.

El modelo que ilustramos – natural y a priori sencillo pero mejorable y extensible – consiste en una enumeración de las jugadas legales en orden creciente de costo hasta llegar a la (o una) jugada correcta, con el registro de la cantidad de jugadas que se consideran antes de llegar a aquella, esto es cuántas jugadas incorrectas tienen un costo menor que la que consideraremos correcta. A este efecto, también registramos la suma de los costos de las jugadas consideradas. Estos valores se usan como entrada de un modelo de aprendizaje automático. Tras aplicarlo, nos interesará obtener el coeficiente de determinación R^2 como evaluación del proceso propuesto. Podemos entonces pensar en nuestro modelo como un árbol (dirigido, n-ario) junto con pesos en los ejes, a priori con profundidad 1, cuyos nodos representan posiciones y cuyos ejes representan las jugadas por las que nos movemos de una posición a la siguiente. En particular, la raíz del árbol representa la posición inicial del problema.

Las Fig. 3 y 4 muestran los árboles correspondientes a los problemas 2a y 2b respectivamente, los cuales resultan en árboles significativos isomorfos. Esto es, no hay distinción entre uno y otro árbol si se descartan varias jugadas consideradas irrelevantes por el algoritmo discutido en [13,21]. Una vez construido el árbol de jugadas candidatas, al momento de entrenar un modelo de aprendizaje automático, únicamente se le pasarán los valores $(\#ejes, \Sigma ejes)$, que para el caso del árbol 3 es igual a $(1, 185)$, y para el árbol 4 es $(6, 1940)$. En este caso nuestro esquema captura mejor la diferencia de dificultad entre los problemas.

Las reglas para determinar los costos son las siguientes: el costo de una jugada que da jaque mate es 0, y también el de una jugada que es la única jugada legal en la posición. Para cualquier otro caso, el costo se calcula según las reglas del cuadro 1. Notar que las filas de la tabla no generan una partición, sino que una jugada podría entrar en varios casos, y entonces es necesario evaluar si la condición de cada fila se aplica y en tal caso sumar el costo correspondiente. Los valores que se usan en las últimas 3 filas están definidos en la tabla 2, que corresponden a determinaciones estándar en ajedrez, exceptuando al rey, al que normalmente no se le asigna un valor. En este caso decidimos asignarle un valor de 2 pensando en la última regla, la que esencialmente incentiva a mover piezas de mayor valor. Basado en nuestra intuición, consideramos apropiado buscar jugadas que mueven el rey antes que un peón, pero después de cualquier otra pieza.

Como ejemplo, consideremos la Fig. 4, que describe el árbol de jugadas candidatas para el problema de la Fig. 2b. Según nuestro modelo, la primera jugada que consideraría un jugador en la posición es la jugada **g3h4**, mediante la cual la dama captura un peón y da un jaque. El costo de 145 surge de sumar 200 unidades por el hecho de que la jugada es jaque, -10 unidades porque capturamos un peón (que tiene valor 1), y -45 ($-5 \times Valor[Dama] = -5 \times 9$) unidades porque movimos la dama a una casilla donde no está atacada.



(a) Problema de dificultad 845 (b) Problema de dificultad 2208

Figura 2: Dos problemas con árboles significativos isomorfos junto con sus valores de dificultad evaluados en base a dicho modelo.

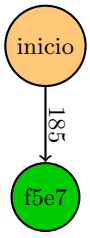


Figura 3: Resolución del problema de la Fig. 2a recorriendo jugadas candidatas.

4. Extensión utilizando las soluciones completas

Dado que los problemas se suelen plantear de modo que sus soluciones no consten tan sólo de una jugada, el mecanismo presentado hasta este punto sólo considera una fracción de la información relevante. Seguimos entonces con una extensión para que se haga uso de un dato que consideramos significativo y hasta ahora dejado de lado: la secuencia del problema completo, incluyendo la primera jugada o *clave* junto con las subsiguientes. Cabe aclarar que hablamos indistintamente del "problema completo" y de la "solución completa", porque en realidad estamos haciendo referencia a la línea de juego en la que se intercalan las jugadas (únicas) que debe encontrar el jugador y las respuestas que va arrojando el programa. Los problemas de la plataforma Lichess, por ejemplo, vienen acompañados no sólo por la clave como parte de la respuesta, sino por

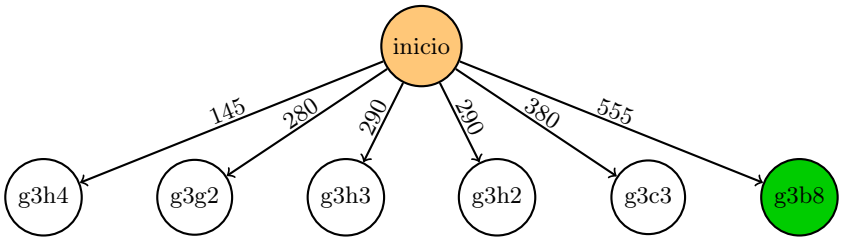


Figura 4: Resolución del problema de la Fig. 2b recorriendo jugadas candidatas.

Si se cumple que	**Sumar al costo**
La jugada es jaque y/o coronación	200
La jugada es una captura que no es jaque ni coronación	300
La jugada no es jaque, captura ni coronación	600
La jugada es una captura	$-10 \times Valor[PiezaCapturada]$
La pieza movida queda atacada	$10 \times Valor[PiezaMovida]$
La pieza movida no queda atacada	$-5 \times Valor[PiezaMovida]$

Cuadro 1: Costo de los distintos tipos de jugada.

la secuencia de jugadas que se presentará al usuario mientras este se mantenga encontrando la mejor jugada en cada turno (y mientras no haya terminado con la secuencia requerida por el problema). El modelo extendido contempla entonces situaciones tales como cuando cuando el jugador encuentra la jugada correcta en el primer nivel, con lo que se avanza en el árbol efectuando esa jugada y la siguiente jugada del rival. En ese punto repetimos el proceso de enumerar jugadas, y seguimos extendiendo el árbol hasta que el problema termine. La Fig. 5 muestra un ejemplo de lo mencionado sobre el problema 1 que termina en jaque mate en 3 jugadas. Notar que las respuestas del rival están fijadas porque hemos utilizado la solución completa del problema. De no haber contado con esta información hubiésemos tenido que considerar otras respuestas posibles.

En este contexto, los datos que incorporamos en el modelo de aprendizaje automático son los mismos que en el caso anterior, pero separándolos por nivel. Así, estamos proveyendo un vector con los ejes y otro con las sumas de sus pesos. Para el ejemplo de la Fig. 5, estos vectores serán $v_{\#ejes} = [6, 4, 1]$ y $v_{\Sigma ejes} = [1335, 995, 0]$.

Un aspecto interesante del modelo extendido es que no resulta computacionalmente demandante. En la versión en la que sólo consideramos jugadas a profundidad 1 y asumimos que no se conoce la solución, es necesario hacer únicamente una invocación al motor de juego para encontrar la mejor jugada. El resto del cómputo consiste en enumerar las jugadas legales de la posición, calcular su costo como ya describimos y ordenar aquellas. Además, cuando se conoce la información de la solución completa ni siquiera es necesario recurrir a un motor de juego (por consiguiente el tiempo de ejecución en la práctica es

Dama	Torre	Alfil	Caballo	Rey	Peón
9	5	3	3	2	1

Cuadro 2: Valor asignado a cada pieza tanto ante un movimiento como ante una captura (de pieza distinta de rey)

mínimo). Nuestra implementación de este modelo extendido puede hallarse en https://pastebin.com/MUzWg5KQ.

Versión	n	R^2	t (s)	Compl(%)
Sólo la primera jugada	16000	0.346	0.13	98
Con la solución completa	16000	0.572	0.02	100

Cuadro 3: R^2 obtenido con *random forest* a partir del modelo basado en jugadas candidatas. El tiempo mostrado es el tiempo promedio que demora procesar un problema. *Compl* es el porcentaje de problemas que se procesaron correctamente, es decir en los que Stockfish encontró una solución ganadora (sólo aplica a la primera fila, porque en la segunda se conoce la solución y no se hace uso de Stockfish).

5. Resultados

La tabla 3 muestra resultados de regresión sobre problemas de Lichess obtenidos por el modelo de jugadas candidatas usando *random forest* como modelo de aprendizaje automático, resultando este superior a los otros considerados. La primera versión sólo busca la mejor jugada con Stockfish, y explora sólo con profundidad 1. La segunda no hace uso del motor al ya conocer la solución, y extiende el árbol hasta el final del problema.

Retomando las ideas de [21,20], podemos aprovechar el hecho de que para cada problema tenemos información producida por dos modelos con enfoques distintos, uno basado en árboles significativos y el otro basado en jugadas candidatas, que podrían aportar información complementaria. Proponemos entonces unir esos datos, esto es entrenar un modelo de aprendizaje automático con los datos generados por uno y otro esquema. A saber, para cada problema hemos generado dos listas de atributos: por un lado, los 10 atributos extraídos de su árbol significativo definidos en [21,20] a los que se agrega otro que mide la altura de este árbol; y por otro los atributos definidos en esta sección, extraídos del árbol de jugadas candidatas. Luego, concatenamos estas dos listas y usamos el resultado como entrada a un modelo de aprendizaje automático. La tabla 4 muestra los resultados del modelo basado en jugadas candidatas combinado con el basado en árboles significativos, específicamente la versión que establece el

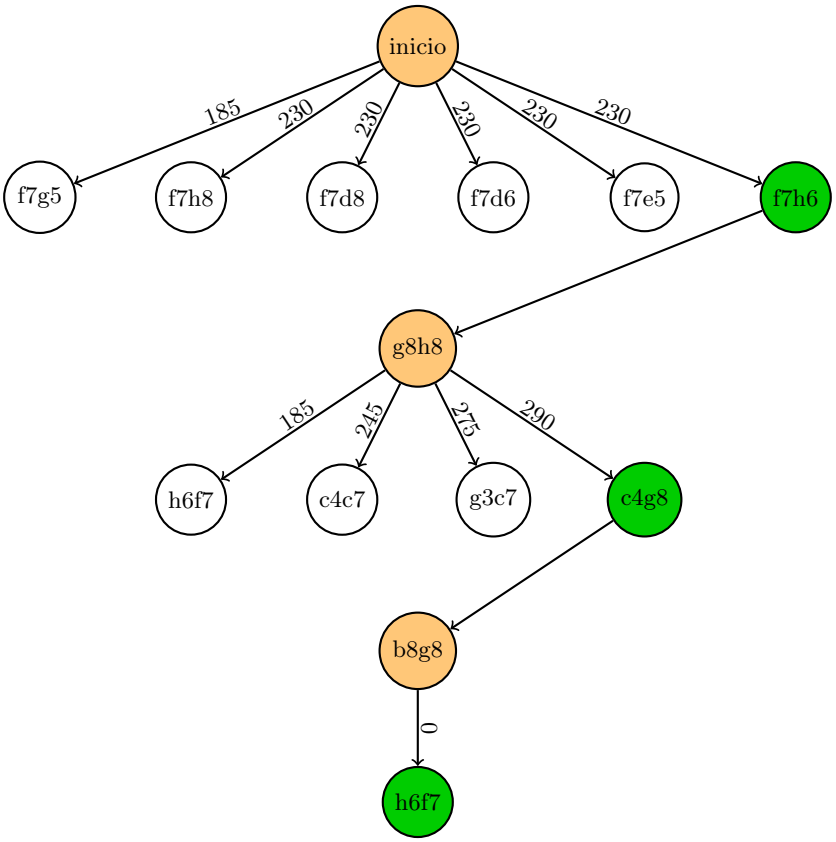


Figura 5: Resolución del problema de la Fig. 1 recorriendo jugadas candidatas y usando la información de la secuencia completa.

atributo que denota máxima profundidad en 7 y permitiendo la posibilidad de *múltiples variantes principales* con valor 3 [13].

Observamos que el modelo basado en árboles significativos hace una contribución importante a la versión que sólo considera la clave, aunque no tanto a la versión que utiliza la solución completa.

5.1. Evaluación de estos resultados

Dado que el valor de dificultad de un problema no es estático sino que se actualiza cada vez que un jugador intenta resolverlo, ahora consideramos para cada problema su valor de dificultad en dos momentos distintos. Queremos determinar con qué precisión podemos predecir la dificultad de un problema en el momento actual si conocemos su dificultad en un tiempo anterior, lo cual en algún sentido nos daría una cota de la calidad de las predicciones que podemos lograr usando información propia del problema. Buscamos entonces entender mejor qué tan

Versión	R^2	Ganancia
Sólo la primera jugada	0.500	+0,154
Con la solución completa	0.592	+0,020

Cuadro 4: R^2 obtenido con *random forest* combinando los atributos obtenidos a partir del modelo de jugadas candidatas con los del basado en árboles significativos, y la ganancia respecto de la tabla 3.

buenas son las predicciones a las que hemos llegado. En particular, el puntaje de $R^2 = 0{,}592$ que mostramos en la tabla 4 es el mejor puntaje obtenido a lo largo de todo este trabajo. Para interpretarlo mejor, en lugar del R^2 también calculamos el *error absoluto medio* (EAM) cometido sobre un conjunto de test. El resultado fue $EAM = 226$. Esto nos permite entender qué tan lejos estamos de una solución óptima usando las mismas unidades de dificultad de los problemas: en promedio nuestras predicciones fallan por 226 unidades. Los problemas usados se encuentran uniformemente distribuidos en el rango $[800, 2400]$. Si bien son deseables predicciones más precisas, creemos que estos resultados son positivos.

Como prueba empírica, consideramos los problemas de Lichess a lo largo del tiempo en dos oportunidades separadas por alrededor de un año y medio[5]. Si bien los datos no traen las fechas en que fueron cargados por última vez, suponemos que son actualizados periódicamente y que por tanto hay alrededor de un año y medio de diferencia entre los primeros y los últimos datos disponibles. Podríamos considerar la diferencia promedio entre los problemas en una y otra fecha, pero esto presentaría sesgos o errores: por ejemplo podría ocurrir que varios de los jugadores se hayan vuelto más fuertes y entonces los problemas habrán sufrido una suerte de disminución en su dificultad relativa. En cambio, preferimos entrenar un modelo de regresión lineal usando como entrada los valores de las dificultades menos recientes para que se predigan las más recientes. Sobre el conjunto de test el resultado fue $EAM = 64$.

El hecho de que conocer la dificultad de un problema en cierto momento no permita acercanos a menos de 64 unidades en promedio a la dificultad del mismo problema un tiempo después, nos da indicios de que el problema encarado es complejo, y que obtener resultados muy superiores a los aquí presentados puede resultar desafiante. De todas formas, creemos que pueden existir mejoras sencillas refinando el modelo de jugadas candidatas.

Otro aspecto que amerita discusión es el siguiente. El hecho de que hayamos obtenido buenos resultados simplemente enumerando jugadas en orden de acuerdo a qué tan costoso es considerarlas podría implicar que para resolver problemas difíciles es conveniente buscar primero jugadas menos "obvias", salteando el análisis de las jugadas más explosivas y por lo tanto menos costosas. Creemos que en cierta medida esto se refleja en la práctica: si estamos resolviendo un problema que sabemos que tiene una dificultad elevada, tendemos a desconfiar de jugadas que parecen demasiado directas, lo cual en la sección 2 nos llevó a ver

[5] Junio de 2022 y noviembre de 2023.

esta confianza como decisión estratégica. De todas maneras, no evaluar las jugadas más explosivas tiene de hecho inconvenientes: primeramente, la cantidad de jugadas explosivas disponibles suele ser significativamente menor a la cantidad de jugadas *tranquilas* – suele haber un número limitado de jaques, capturas y coronaciones –, lo cual se suma a que las primeras suelen llevar a líneas más fáciles de calcular por su naturaleza *forzada*, y entonces las jugadas menos costosas de encontrar también suelen ser menos costosas de analizar. Y en segundo lugar, los problemas suelen requerir encontrar varias jugadas, entonces es posible que un mismo problema difícil requiera encontrar tanto algunas jugadas *tranquilas* como algunas jugadas explosivas, por lo cual descartarlas no sería adecuado.

6. Conclusiones y trabajo futuro

En este trabajo nos dedicamos a explorar la noción de dificultad en problemas de ajedrez con el fin de calcular estimaciones en posiciones arbitrarias. El modelo presentado sugiere un ordenamiento de las jugadas candidatas y basar la evaluación de estas siguiendo dicho orden. A la vez, vinculamos la noción de dificultad con la de decisión estratégica, lo que significó la transformación de secuencias de jugadas candidatas en un juego bipersonal en el que la decisión del jugador que resuelve consiste en un plan de acción para buscar la solución.

Un camino a ser explorado en el futuro es el de usar técnicas de aprendizaje automático para asignar pesos no sólo a los atributos que ya estamos calculando, sino también a cada una de las condiciones del cuadro 1 (y posiblemente a otras condiciones nuevas que describan aspectos de las jugadas). Por ejemplo podríamos tener una única red neuronal que determine los costos de cada jugada y que a la vez use esos costos para estimar la dificultad de la posición. También sería valioso un análisis sobre la importancia de cada atributo usado para entrenar al modelo de aprendizaje automático, similar al hecho en [13] para árboles significativos. Esto proveería información interesante sobre el poder predictivo de cada atributo extraído. Otra extensión natural al modelo presentado implica extender no sólo los nodos que son parte de la solución sino todos o casi todos estos, ya que al resolver el problema el jugador no conoce la jugada correcta y necesita calcular líneas de juego que comienzan con las jugadas candidatas incorrectas antes de descartarlas. Lo ideal es entonces hallar una forma de extender un camino mientras este sea prometedor – es decir, mientras la posición no sea claramente ganadora o claramente no ganadora –. El desafío radica en determinar cuándo un jugador ya no necesita seguir calculando, para lo que tal vez podamos reusar las ideas de [13] sobre la variante con podas del modelo con árboles significativos. Por ejemplo, podríamos decidir que si estamos extendiendo un camino con evaluación no ganadora, dejaremos de extenderlo una vez que el rival tenga cierto número de buenas jugadas disponibles.

Asimismo, esperamos explorar la existencia de problemas con condiciones variadas con secuencias de mate de longitudes mayores, así como aplicar estas ideas a otros juegos de características similares a las del ajedrez.

Referencias

1. Allis, V.: Searching for Solutions in Games and Artificial Intelligence. PhD thesis, University of Limburg, Department of Computer Science (1994)
2. Dry, M., Lee, M., Vickers, D. and Hughesi, P.: Human performance on visually presented traveling salesperson problems with varying numbers of nodes. Journal of Problem Solving, 20–32 (2006)
3. Glickman, M. E.: Example of the Glicko-2 system. Boston University (2011)
4. Guid, M. and Bratko, I.: Search-Based estimation of problem difficulty for humans. Artificial Intelligence in Education (2013)
5. HaCohen-Kerner, Y., Cohen, Y. N. and Shasha, E.: An improver of chess problems. Cybern. Syst., 441–465 (1999)
6. Hristova, D., Guid, M. and Bratko, I.: Assessing the difficulty of chess tactical problems. International Journal on Advances in Intelligent Systems, 7:728–738, (2014)
7. Humble, P. N.: Chess as an art form. Br. J. Aesthetics, 59–66 (1993)
8. Iqbal, A.: Evaluation of Economy in a Zero-sum Perfect Information Games. The Computer Journal (2007)
9. Iqbal, A.: A discrete computational aesthetics model for a zero-sum perfect information game. PhD thesis, Faculty of Computer Science & Information Technology, University of Malaya (2008)
10. Jarusek, P. and Pelánek, R.: Difficulty rating of sokoban puzzle. Proc. of the Fifth Starting AI Researchers' Symposium (STAIRS 2010). IOS Press, 140–150 (2010)
11. Kotovsky, K., Simon H. A. and Hayes, J.: Why are some problems hard? Evidence from tower of Hanoi. Cognitive Psychology, pages 248–294 (1985)
12. Kotovsky, K. and Simon H. A.: What makes some problems really hard: Explorations in the problem space of difficulty. Cognitive Psychology, 143–183 (1990)
13. Marenco, M.: Exploración de métodos para medir dificultad en ajedrez: extensiones, mejoras y limitaciones. Tesis de Licenciatura, Dpto. de Computación, FCEyN, UBA (2023)
14. Marsland, T.: Computer Chess and Search. Encyclopedia of Artificial Intelligence (1992)
15. Osborne, M. and Rubinstein, A.: A course in game theory. The MIT Press, Cambridge, USA (1994)
16. Pelánek, R.: Difficulty rating of sudoku puzzles by a computational model. Proc. of Florida Artificial Intelligence Research Society Conference (FLAIRS 2011). AAAI Press, 434–439 (2011)
17. Pizlo, Z. and Li, Z.: Solving combinatorial problems: The 15-puzzle. Memory and Cognition, 1069–1084 (2005)
18. Richards, D. J. and Hart, T. P.: The Alpha-Beta Heuristic. Massachusetts Institute of Technology (1963)
19. Shannon, C. E.: Programming a Computer for Playing Chess. Philosophical Magazine (1950)
20. Stoiljkovikj, S.: Computer-based estimation of the difficulty of chess tactical problems. PhD thesis, University of Ljubljana, Faculty of Computer and Information Science (2015)
21. Stoiljkovikj, S., Guid, M. and Bratko, I.: A computational model for estimating the difficulty of chess problems. Proceedings of the Third Annual Conference on Advances in Cognitive Systems (2015)