Min Nim

Teoría de Juegos

20 de julio de 2023

Integrantes

- Manuel Panichelli, L.U. 72/18
- Ignacio Alonso Rehor, L.U. 195/18

Introducción

El Nim es un juego de dos jugadores en el cual estos se alternan quitando elementos distintas pilas. El jugador que no tenga movimientos disponibles será considerado el perdedor. En la variante analizada en este trabajo, las pilas son tres, y las cantidades que los jugadores pueden quitar de las pilas son tres constantes.

La idea de este trabajo es analizar esta variante del juego, describiendo aquellas configuraciones que resultan "ganadoras" y "perdedoras", y qué significa esto. También, buscamos desarrollar una estrategia de juego para esta variante, y explorar un poco qué garantías tiene.

> En este informe detallamos el desarrollo conceptual del trabajo. Para más detalles sobre cómo correr la implementación e interpretar la salida, consultar el `README.md`

Definiciones

Juego

Definimos un **juego Nim** como una tupla

$$\langle \mathbf{P}, \mathbf{M} \rangle$$

donde

- $\mathbf{P}$ es una terna sin orden que representa la **posición**: las cantidades de cada pila

- $\mathbf{M}$ es un conjunto que representa los **movimientos posibles**: los valores que se pueden sacar de una pila

Algunos ejemplos de juegos son:

- $\langle [0,0,0], \{1,2,3\} \rangle$: Donde $\mathbf{P} = [0,0,0]$, y $\mathbf{M} = \{1,2,3\}$. Esto representa el juego donde las tres pilas tienen 0 elementos, y se podrían intentar sacar 1, 2, o 3 elementos de cada pila.
- $\langle [1,0,2], \{1,3,5\} \rangle$: Donde $\mathbf{P} = [1,0,2]$, y $\mathbf{M} = \{1,3,5\}$. Esto representa el juego donde hay una pila con 1 elemento, otra con 2 elementos y otra sin elementos. También, de cada una de estas pilas se podrían intentar sacar 1, 3, o 5 elementos.

Notar que los juegos $\langle [1,0,0], \{2,5,3\} \rangle$ y $\langle [0,0,1], \{5,3,2\} \rangle$ representan el mismo juego ya que $\mathbf{P}$ y $\mathbf{M}$ no tienen orden.

Movimientos

Definimos un **movimiento** como una tupla $\langle h, m \rangle$. Decimos que es **legal** para un juego $\langle \mathbf{P}, \mathbf{M} \rangle$ si $h \geq m$, $h \in \mathbf{P}$, y $m \in \mathbf{M}$. En otras palabras, un movimiento es legal si no quito más elementos de los que hay en un *heap*, ese *heap* existe en la posición, y la cantidad a quitar está dentro de mis movimientos.

Decimos que realizar un movimiento legal sobre un juego es una operación $(*)$ definida como

$$\langle h_1, m \rangle * \langle [h_1, h_2, h_3], \mathbf{M} \rangle = \langle [h_1 - m, h_2, h_3], \mathbf{M} \rangle.$$

Notar que el resultado es otro juego.

***Outcome Classes* de Posiciones**

Decimos que una posición (o juego) es una $\mathcal{N}$-posición si el jugador al que le toca jugar puede forzar una victoria. Análogamente, decimos que una posición (o juego) es una $\mathcal{P}$-posición si el jugador al que le toca jugar no puede forzar una victoria.

Notar que de esta definición se desprende que para toda posición que sea $\mathcal{N}$-posición, existe al menos un movimiento que resulta en una $\mathcal{P}$-posición para el siguiente jugador. Así mismo, para toda posición que sea $\mathcal{P}$-posición, se tiene que todos los movimientos resultan en una $\mathcal{N}$-posición para el siguiente jugador, o lo que es lo mismo: no existe un movimiento que resulte en una $\mathcal{P}$-posición para el siguiente jugador.

Algunos ejemplos de posiciones:

- La posición $[0, 0, 0]$ es una $\mathcal{P}$-posición pues no hay movimientos legales.

- La posición $[1, 0, 0]$ es una $\mathcal{N}$-posición pues la posición resultante de sacar 1 de la primera pila (si este es un movimiento legal) es una victoria.
- La posición $[1, 0, 1]$ es una $\mathcal{P}$-posición pues a partir de todas las posiciones resultantes (estas son $[0, 0, 1]$ y $[1, 0, 0]$) el siguiente jugador puede forzar una victoria, o lo que es lo mismo, cada una de estas posiciones son una $\mathcal{N}$-posición.

Implementación

En esta sección damos más detalles de la implementación en *Python* del programa.

Representamos a las pilas (`NimHeap`) como enteros y a las posiciones (`Position`) como *listas* de pilas. Modelamos a los juegos como una clase (`Game`) que tiene una posición y una lista de movimientos.

Cálculo de P y N posiciones

Para calcular las P y N posiciones, generamos el *game tree* partiendo de la posición inicial y probando todos los movimientos legales recursivamente. De esta forma generamos solo las posiciones que son alcanzables desde el juego inicial con los movimientos que contamos y no todas las posibles.

Notamos que hay un alto grado de **superposición de subproblemas** (hay muchos caminos de jugadas que llevan a las mismas posiciones) por lo que podemos usar **programación dinámica** para hacer que el cómputo sea más eficiente. Lo implementamos de forma *top-down* usando una estructura de *memoización* auxiliar.

Para ver la *outcome class* de cada posición,

- Si no hay movimientos posibles, es una posición perdedora ($\mathcal{P}$)
- Si existe al menos un movimiento ganador (que lleva a una posición perdedora) es una posición ganadora ($\mathcal{N}$)
- Si no existe ningún movimiento ganador y todos llevan a posiciones ganadoras, entonces es una posición perdedora ($\mathcal{P}$).

En cada nodo primero chequeamos si tenemos memoizado el resultado, en cuyo caso lo retornamos. Sino, lo computamos recursivamente y luego lo memoizamos.

Simetrías

Como implementamos las posiciones como **listas**, tomaríamos como distintas sus diferentes permutaciones, como $[1, 2, 3]$ y $[2, 3, 1]$. Pero como vimos antes,

estos deberían representar al mismo juego por lo que deberían tener la misma *outcome class*.

Por lo tanto, para evitar recomputar las simetrías y aprovechar la memoización tomamos como *clave* de nuestra estructura la *lista ordenada*. De esa forma, todas las permutaciones son tomadas como iguales.

Estrategia óptima

Además de calcular las $\mathcal{P}$ y $\mathcal{N}$ posiciones, implementamos una función que dada una posición para el jugador L, elige una **jugada óptima**:

- Si la posición original es ganadora para L (i.e. es una $\mathcal{N}$-posición) entonces elige alguna jugada ganadora.
- Si no, al estar en una posición perdedora ($\mathcal{P}$-posición) busca "dificultar" la decisión del jugador R en su siguiente turno lo máximo posible.

Nuestra estrategia para dificultar la elección del otro jugador consiste en elegir el juego resultante de forma tal que este tenga menor proporción de jugadas ganadoras para el jugador R.

Valor de una posición

Para poder confeccionar la estrategia, primero definimos la noción de **valor de un juego**, que será la proporción de jugadas ganadoras sobre el total de jugadas. Podemos calcularlo viendo para cada jugada, cual es el *outcome class* de su juego resultante. Luego, podemos calcular la proporción de $\mathcal{P}$-posiciones sobre el total.

Más formalmente, definimos

- La *clase* de un juego como

$$\text{clase}(G) = \text{ la } \textit{outcome class} \text{ de } G$$

- Los *movimientos legales* de un juego como

$$L(G) = \{\langle h, m\rangle \mid \text{ es legal en } G\}$$

- Las $\mathcal{P}$-posiciones resultantes y $\mathcal{N}$-posiciones resultantes como

$$\mathcal{P}(G) = \{G' \mid G' = \langle h, m\rangle * G, \langle h, m\rangle \in L(G),\ \text{clase}(G') = P\}$$
$$\mathcal{N}(G) = \{G' \mid G' = \langle h, m\rangle * G, \langle h, m\rangle \in L(G),\ \text{clase}(G') = N\}$$

Y finalmente el *valor* de un juego como

$$\text{valor}(G) = \frac{\#\mathcal{P}(G)}{\#\mathcal{P}(G) + \#\mathcal{N}(G)}$$

debido a la naturaleza recursiva de la definición y la necesidad de las *outcome classes*, **calculamos el valor de cada posición en conjunto con el cálculo de posiciones P y N**.

Dos observaciones que se desprenden de esta definición son

- Toda $\mathcal{P}$-posición tiene valor 0, pues no tiene jugadas ganadoras
- Como todas las $\mathcal{N}$-posiciones por definición tienen al menos una jugada ganadora (que lleva a $\mathcal{P}$), su valor es estrictamente mayor a 0.

Definición de la estrategia

Finalmente definimos la estrategia de forma conjunta (sin importar en qué *outcome class* estamos parados) como **elegir el movimiento que lleve a la posición de mínimo valor**. Es decir,

$$\arg\min_{\langle h,m\rangle \in L(G)} \text{valor}(\langle h, m\rangle * G)$$

Observamos que esto cumple con la definición que damos al principio. Si se está en una $\mathcal{N}$-posición, la jugada que lleve a un juego de mínimo valor será ganadora, pues este seguro es una $\mathcal{P}$-posición (todas tienen valor 0).

Limitaciones

Si bien esta estrategia es sencilla y óptima, se podrían considerar más cosas a la hora de dificultar una posición. Por ejemplo, puede suceder que haya dos movimientos que resulten en $\mathcal{N}$-posiciones con el mismo valor, pero que la cantidad de jugadas óptimas consecutivas necesarias para ganar en cada una sea diferente.

Estas dos claramente no son equivalentes, pero este método no las diferencia. Al tener que acertar en más jugadas, el jugador humano tiene más chances de equivocarse.

En el siguiente diagrama se puede ver un ejemplo

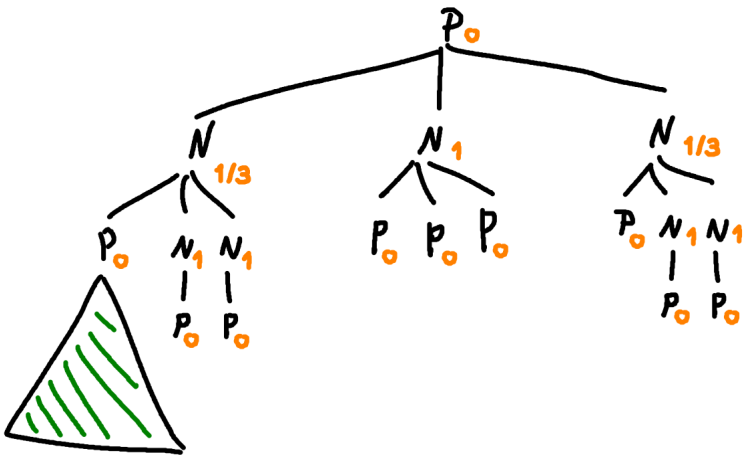

Figura 1: Ejemplo de dos $\mathcal{N}$-posiciones con el mismo valor pero distintos subárboles

Se podría extender el método para que contemple de alguna forma la profundidad del subárbol.

Conclusiones y trabajo futuro

Fue un trabajo práctico muy interesante de implementar. Sobre todo, fue curioso como definiciones tan sencillas llevaron a un juego con un análisis tan complejo.

Como hablamos anteriormente, algo que nos quedó pendiente fue elaborar otro tipo de estrategias para dificultar la elección del otro jugador.

De la mano con esto, nos hubiese gustado elaborar alguna metodología con la cual comparar distintas estrategias. Una idea interesante podría ser armar una especie de torneo entre las distintas estrategias, y ver como se desenvuelven en distintas instancias del juego. Se podrían incluir heurísticas y estrategias de control contra las cuales puedan jugar. Por ejemplo, una que haga movimientos legales aleatorios. Luego, podríamos medir qué tan buenas son comparando el porcentaje de victorias contra cada una (comenzando desde una posición perdedora, para que las óptimas tengan posibilidad de perder).

Por último, queda como trabajo futuro desarrollar un algoritmo que calcule las $\mathcal{N}$ y $\mathcal{P}$ posiciones haciendo uso de los resultados y teoremas del álgebra de juegos definido en clase, en la que se contemplaba la partición de un juego por separado y se analizaba su suma. Aquí probablemente se podría haber utilizado para analizar cada *heap* por separado.