Machine Learning in Practice

#7: Combinatorial Games: Supplement

Summer 2019

Summary: characterization of combinatorial games

- C : the set of all possible positions (configurations)
- $F_A, F_B \subset C$: the set of ending positions (possibly $F_A \neq F_B$)
- $m_i : C \setminus (F_A \cup F_B) \to 2^C$: possible moves of player i $\quad (i = A, B)$
- $W_i, L_i \subset C$: winning/losing positions of player i
- $s_A : W_A \to L_B$ (with $s_A(c) \in m_A(c) \cap L_B$) : winning strategy of player A
  - $\forall c \in L_A,\ m_A(c) \subset W_B \implies$ winning strategy undefined for $c \in L_A$
- $s_B : W_B \to L_A$ (with $s_B(c) \in m_B(c) \cap L_A$) : winning strategy of player B
  - $\forall c \in L_B,\ m_B(c) \subset W_A \implies$ winning strategy undefined for $c \in L_B$
- For impartial games, $F_A = F_B, m_A = m_B, W_A = W_A, L_B = L_B, s_A = s_B$
  - denote by F, m, W, L, s

A. Informal Characterization of Combinatorial Games

바둑, 장기, 체스, 체커 등의 게임은 다음과 같은 특성을 가진다:

- 두 명의 player A, B가 번갈아 가면서 "move"한다.
  - Move는 게임의 상태를 변화시킨다.
  - 게임을 시작하는 player는 관습적으로 A로 나타낸다.
- Move는 전적으로 player의 의지로 결정되며 "chance device"에 의존되지 않는다.
  - Change device의 예: 주사위, 룰렛, 카드섞기
- 게임의 상태(와 history)는 양 player 모두에게 완벽하게 알려진다.
  - 이를 "perfect information"이라고 한다.
- 게임은 언젠가는 어느 한 player가 이긴 상태로 종료된다.
  - 이 조건을 완화하여 "draw"(비김)인 경우도 허용할 수 있다.
- 게임은 어느 한 player가 더 이상 move할 수 없을 때 종료된다.

이와 같은 게임들을 **combinatorial game**이라 부르는데 포커, 화투, 마이티, 테트리스 등의 게임은 이런 성질을 가지지 않으므로 combinatorial game이 아니다.

두 player는 자기가 이기기 위해 최선을 다 할텐데,

- **winning strategy**(필승전략)이라는 것이 well-defined 되고 존재하는 지 여부
  - 즉, (두 player 모두 super-intellignet하다고 가정하고) 게임의 초기상태가 주어졌을 때, 게임의 **winner**는 이미 정해져 있음
- 두 player 중 누가 winning strategy를 가졌는지를 계산
- winning strategy는 어떻게 정의되고 계산

할 수 있는지를 combinatorial game 분야에서 다룬다. 게임이론이 nontrivial한 이유는 각 player의 최적전략은 다른 player의 최적전략에 상호의존적이기 때문이다.

Winner 및 winning strategy를 계산하는 것은 algorithmic problem으로 이에 대한 계산복잡도, intractability 등을 전산학의 "**algorithmic combinatorial game theory**" 분야에서 다룬다. “Beautiful Mind”의 John Nash와 같이 경제학자들이 다루는 게임은 주로 strategic game으로 combinatorial game과 전혀 다른 특성을 가진다. 물론 실용적 중요성으로 인해 strategic game도 전산학에서 활발히 다루고 있다.

B. Formal Definition of Combinatorial Games

Combinatorial game은 5-tuple (C, F_A, F_B, m_A, m_B)로 정의할 수 있다:

- C : the set of all possible **configurations** ("**positions**")
- $F_A, F_B \subset C$: the set of **ending** positions (possibly $F_A = F_B$)
  - F_i에 먼저 도달하는 player i가 이김 ($i \in \{A, B\}$)
  - 아래에서 정의할 m_i로 move한 직후에 F_i에 도달하는 시점을 기준으로 함 (move 하기 직전에 다른 player에 의해 F_i에 도달하는 시점을 기준으로 하는 것이 아니고)
- $m_i : C \setminus (F_A \cup F_B) \to 2^C$: possible **moves** of player i $(i \in \{A, B\})$
  - 각각의 move는 어떤 configuration $c \in C \setminus (F_A \cup F_B)$에서 다른 configuration $c \in C$로 상태전이
  - m_i의 codomain은 C가 아니고 2^C임에 유의 (즉, 현재 configuration을 입력으로 받아 옮겨갈 수 있는 모든 configuration들의 집합을 출력으로 하는 함수)

아무리 복잡해보이는 게임도 5-tuple (C, F_A, F_B, m_A, m_B) 정형화할 수 있다. $m_A = m_B$, $F_A = F_B$인 게임을 **impartial** game(대칭게임)이라고 부르고 그렇지 않은 게임을 **partizan** game(비대칭게임)이라고 부르는데, impartial game은 다루기 훨씬 간단한 편이다. (“Supplement: Nim & Impartial Games” 참조)

예를 들어, 바둑을 (C, F_A, F_B, m_A, m_B)로 정형화 해보자. 바둑에서 (패로 인한) 동형반복을 금지하지 않는다고 단순화 시키면 바둑게임의 각 configuration은 함수

- $c : [19]^2 \to \{\text{백}, \text{흑}, \text{무}\}$

로 정형화 할 수 있고, configuration의 집합 C는

- $C = \{\text{백}, \text{흑}, \text{무}\}^{[19]^2}$

로 나타낼 수 있다.[1]

Player A가 흑을, player B가 백을 쥔다고 하면,

- $F_A = \{c \in C \mid c\text{에서 더 이상 둘 곳이 없으며 } c\text{에서 흑의 집수가 더 많음}\}$
- $F_B = \{c \in C \mid c\text{에서 더 이상 둘 곳이 없으며 } c\text{에서 백의 집수가 더 많음}\}$

가 된다. 현재 상태 $c \in C$에서 사석들이 발생한다면 이들을 모두 제거해서 얻은 새로운 상태를 $\tau(c)$로 나타내자:

- $\tau : C \to C\ ;\ \big(\tau(c)\big)(x,y) = \begin{cases} \text{무} & \text{if 상태 } c\text{에서 } (x,y) \text{ 위치에 사석이 있으면} \\ c(x,y) & \text{otherwise} \end{cases}$

τ를 이용해 다음과 같이 move를 나타낼 수 있다:[2]

- $m_A(c) = \big\{\tau(c_{ij}) \,\big|\, c(i,j) = \text{무}\big\}$ where $c_{ij}(x,y) = \begin{cases} \text{흑} & \text{if } (x,y) = (i,j) \\ c(x,y) & \text{otherwise} \end{cases}$
  - $m_A(c) = \big\{(i,j) \in [19]^2 \,\big|\, c(i,j) = \text{무}\big\}$로 정의되지 않음에 유의
- $m_B(c) = \big\{\tau(c_{ij}) \,\big|\, c(i,j) = \text{무}\big\}$ where $c_{ij}(x,y) = \begin{cases} \text{백} & \text{if } (x,y) = (i,j) \\ c(x,y) & \text{otherwise} \end{cases}$

게임의 초기상태 $c_0 \in C$는 $c_0(x,y) = \text{무}$이다.

[1] 동형반복을 금지한다면 다시 두면 안되는 자리들의 집합을 $d \subseteq [19]^2$에 함께 기록해나가면 되며 이 경우 configuration의 집합 C는 $C = \{\text{백}, \text{흑}, \text{무}\}^{[19]^2} \times 2^{[19]^2}$ 로 나타낼 수 있다.

[2] 돌이 있었던 자리에 다시 두는 것을 금지하는 식으로 동형반복을 막는 경우에는 move가 $m_A(c,d) = \Big\{\big(\tau(c_{ij}),\, d \cup \{(i,j)\}\big) \,\Big|\, c(i,j) = \text{무 and } (i,j) \notin d\Big\}$와 같이 표현된다.

C. Winning/Losing Positions

어떤 configuration(position) $c \in C$에서 player A가 게임을 시작하거나 이어갈 때, B의 전략과 관계없이 A가 이길 수 있기 위한 필요충분조건은 다음과 같이 정의되는 predicate $\alpha_A(c)$가 성립되는 것이다:

- $\alpha_A(c) \triangleq$ "$\exists c_A^1 \in m_A(c)$, $\forall c_B^1 \in m_B(c_A^1)$,
  $\exists c_A^2 \in m_A(c_B^1)$, $\forall c_B^2 \in m_B(c_A^2)$, $\cdots$, $\exists c_A^k \in m_A(c_B^{k-1})$, $c_A^k \in F_A$".

위의 조건식의 직관적 의미는 다음과 같다:

- c_i^j: player $i \in \{A, B\}$의 j번째 선택 직후 도달되는 configuration
- $\forall c_B^i \in m_B(c_A^i)$: B가 허용되는 범위 $m_B(c_A^i)$내에서 최선의($\forall$) 전략으로 방어하더라도
- $\exists c_A^{i+1} \in m_A(c_B^i)$: A에게 허용되는 범위 $m_A(c_B^i)$내에서 적절한($\exists$) 전략이 존재하여
- $c_A^k \in F_A$: A가 먼저 자신이 이기는 configuration에 도달
  - 위 조건식이 성립되려면, 어떤 $j < k$에 대해 $c_B^j \in F_B$가 되어 B가 이기는 configuration에 먼저 도달하는 경우는 발생할 수 없다. 이는 m_A의 domain이 $C \setminus (F_A \cup F_B)$으로 정의되었기 때문에 $m_A(c_B^j)$가 정의되지 않기 때문이다.

마찬가지로 $c' \in C$에서 player B가 게임을 이어갈 때, A의 전략과 관계없이 B가 이길 수 있기 위한 필요충분조건은 다음과 같이 정의되는 predicate $\alpha_B(c')$가 성립되는 것이다:

- $\alpha_B(c') \triangleq$ "$\exists c_B^1 \in m_B(c')$, $\forall c_A^2 \in m_A(c_B^1)$,
  $\exists c_B^2 \in m_B(c_A^2)$, $\forall c_A^3 \in m_A(c_B^2)$, $\cdots$, $\exists c_B^k \in m_B(c_A^k)$, $c_B^k \in F_B$.

A와 B의 **winning positions**은 α_A, α_B를 만족하는 configuration들의 집합으로 정의한다:

- $W_A \triangleq \{c \in C \mid \alpha_A(c)\}$
- $W_B \triangleq \{c' \in C \mid \alpha_B(c')\}$

위와 비슷한 논리를 사용하여 A와 B의 **losing positions**도 간단히 얻을 수 있다.

- $\beta_A(c) =$ "$\forall c_A^1 \in m_A(c)$, $\exists c_B^1 \in m_B(c_A^1)$,
  "$\forall c_A^2 \in m_A(c_B^1)$, $\exists c_B^2 \in m_B(c_A^2)$, $\cdots$, $\exists c_B^k \in m_B(c_A^k)$, $c_B^k \in F_B$"
  - 즉, A가 어떤($\forall$) 전략을 쓰더라도 B가 적절한($\exists$) 전략으로 막아서 자기가 승리할 수 있는 위치로 돌릴 수 있음
- $\beta_B(c') = \forall c_B^1 \in m_B(c')$, $\exists c_A^2 \in m_A(c_B^1)$,
  $\forall c_B^2 \in m_B(c_A^2)$, $\exists c_A^3 \in m_A(c_B^2)$, $\cdots$, $\exists c_A^k \in m_A(c_B^{k-1})$, $c_A^k \in F_A$"

로 predicate β_A, β_B를 정의하면 A와 B의 losing positions L_A, L_B는

- $L_A \triangleq \{c \in C \mid \beta_A(c)\}$
- $L_B \triangleq \{c' \in C \mid \beta_B(c')\}$

로 나타낼 수 있다.

$c \in F_B$에서 A가 게임을 시작하거나 이어받으면 이미 B가 이긴 채로 게임이 종료되므로

- $F_B \subseteq L_A$

으로 F_B는 A의 losing position에 포함되는 것으로 간주한다. 마찬가지로

- $F_A \subseteq L_B$

로 간주한다.

그렇다면 이렇게 직관적으로 정의한 winning/losing position이 well-defined 된 것일까? 즉, 어떤 configuration $c \in C$가 나의 winning position이기도 하고 losing position이기도 할까? 그리고, winning position과 losing position을 제외한 configuration은 뭘까? 이에 대한 답은 Problem 1-2에서 깨우칠 수 있다.

1. 앞에서 정의한 winning/losing position이 잘 정의되려면

- $W_A \cap L_A = \emptyset$
  - 즉, 각 $c \in C$에 대해, c에서 게임을 시작할 때 A가 필승전략을 가지면서 동시에 필패할 경우는 없음

이 성립되어야 하는데 (같은 이유로, $W_B \cap L_B = \emptyset$도), W_A와 L_A의 정의에 입각하여 $W_A \cap L_A = \emptyset$임을 증명하라. (힌트: $c \in L_A \implies c \notin W_A$를 보이면 됨)

($W_A \cap W_B = \emptyset$ 여부를 따지는 것은 아무 의미가 없음에 유의한다. 이는, W_A는 A가 게임을 이어받는(또는 시작하는) 위치를 기준으로 하고, W_B는 B가 게임을 이어받는 위치를 기준으로 하기 때문이다. 의미가 있는 질문은 $s_A(W_A) \cap W_B = \emptyset$ 정도가 될 것이다.)

2. W_A, L_A가 disjoint함은 앞 문제에서 보였는데, 그럼 이들이 C를 partition할 지 여부도 중요할 것이다.

- $c \in W_A \cup L_A$ for all $c \in C \setminus (F_A \cup F_B)$

이 항상 성립하는지 아니면 그렇지 않을 수도 있는지 여부를 W_A, L_A의 정의에 입각하여 판단하고, 직관적 의미가 무엇인지를 서술하라. (힌트: $\exists \ldots \forall \ldots$ 조건식의 chain의 길이가 유한할 때만 고려함에 주목)

D. Winning Strategies

Player A가 $c \in W_A$에서 게임을 시작하거나 이어갈 때, B가 어떤 전략으로 막아내도 이를 뚫고 이길 수 있는데, 이러한 A의 전략을 **winning strategy**(필승전략)이라고 부르는데,

- $c \in W_A$가 주어졌을 때, A는 $m_A(c)$중에서 L_B에 속하는 configuration을 선택

하면 B가 이후 최선을 다한다고 해도 A는 계속 B의 losing position으로 몰아갈 수 있다.

다시 말하자면, A의 필승전략 $s_A : W_A \to C$는

- $s_A(c) \in m_A(c) \cap L_B$ for all $c \in W_A$

를 만족하는 s_A로 정의한다 (당연히 여러개 존재할 수 있음).

마찬가지로, B의 필승전략 $s_B : W_B \to C$는

- $s_B(c) \in m_B(c) \cap L_A$ for all $c \in W_B$

를 만족하는 s_B로 정의한다.

그렇다면 이렇게 직관적으로 정의한 필승전력 s_A, s_B는 well-defined 된 것일까? 만약 그렇다면 필승전략을 가진 player가 **winner**가 된다. (즉, s_A와 s_B중 하나만 잘 정의된다)

3. Winning strategy $s_A : W_A \to C$가 잘 정의되려면

(a) $\forall c \in W_A,\ \exists c' \in m_A(c),\ c' \in L_B$

가 성립되어야 player A가 $s_A(c) = c'$를 선택하여 configuration을 L_B로 몰아갈 수 있으며,

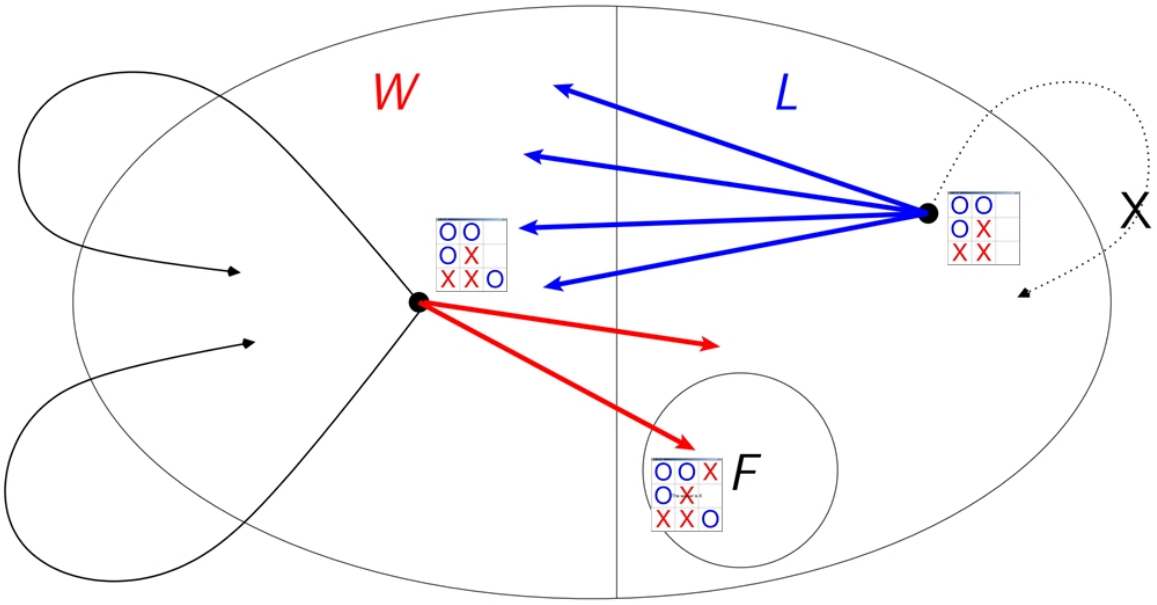
(b) $\forall c' \in L_B,\ \forall c \in m_B(c'),\ c \in W_A$

가 성립되어야 player B가 어떻게 막으려고 하더라도, player A가 그의 필승전략 s_A을 계속하여 사용할 수 있는 configuration인 W_A내부로 헌납할 수 밖에 없게 된다. (이 과정이 반복되다 보면 결국은 A는 $s_A(c) \in F_A \subset L_B$로 몰아가서 이길 수 있을 것이다.)

(a),(b)가 성립함을 W_A, L_B의 정의에 입각하여 증명하라.

$F_A = F_B$, $m_A = m_B$, $W_A = W_B$, $L_A = L_B$인 특수한 경우(즉, impartial game)에 대해, 뒷페이지의 그림은 문제의 의미를 직관적으로 보여준다. 즉,

- winning position에서는 상대를 상대의 어떤 losing position으로 보낼 수 있음
  - W에서는 L로 보내는 길이 항상 존재하며 최종적으로는 F까지 갈 수 있음. (하지만 실수하면 다시 W로 보내서 상대를 이기는 길로 인도할 수도..)
- losing position에서는 어떻게 움직여도 상대를 상대의 winning position으로 상대를 모실 수 밖에 없음
  - L에서는 W로 보내는 길 밖에 없으며 상대를 L로 보낼 방법이 없음



E. Computing Winning/Losing Positions

게임의 winner는 이미 정해져 있는데 player A가 시작하는 configuration c_0가 W_A에 포함되면 A가 winner이고 $c_0 \in L_A$면 B가 winner이다. ($c \in C \setminus (W_A \cup W_B)$면 게임은 비기게 된다.) 그렇다면, winning/losing position은 어떻게 기계적으로 계산할 수 있을까?

4. 다음을 W_A, W_B, L_A, L_B의 정의에 입각하여 증명하라:

- $W_A = \{c \in C \mid \exists c' \in m_A(c),\ c' \in L_B\}$
- $L_A = \{c \in C \mid \forall c' \in m_A(c),\ c' \in W_B\}$

Problem 4의 mutually inductive formula과

- $F_B \subseteq L_A$, $F_A \subseteq L_B$

를 이용하면 L_A, L_B, F_A, F_B를 조금씩 키워나가면서 계산할 수 있다:

$L_A := F_B$
$L_B := F_A$
while (there is a change in one of W_A, W_B, L_A, L_B):
    $W_A := W_A \cup \{c \in C \mid m_A(c) \cap L_B \neq \emptyset\}$
    $W_B := W_B \cup \{c \in C \mid m_B(c) \cap L_A \neq \emptyset\}$
    $L_A := L_A \cup \{c \in C \mid m_A(c) \subseteq W_B\}$
    $L_B := L_B \cup \{c \in C \mid m_B(c) \subseteq W_A\}$

위와 같이 bottom-up 방식을 사용하지 않고 다음과 같이 top-down 방식으로 mutually recursive하게 계산할 수도 있다.

```
def winning_position_for_A(c):
    if c ∈ F_B:
        return False

    for each c' ∈ m_A(c):
        if not winning_position_for_B(c'):
            return True            # c ⇝ c' guarantee A's winning
    return False         # whichever position A selects, A loses

def winning_position_for_B(c):
    if c ∈ F_A:
        return False

    for each c' ∈ m_B(c):
        if not winning_position_for_A(c'):
            return True            # c ⇝ c' guarantee B's winning
    return False         # whichever position B selects, B loses
```

Winning/losing position으로 C를 partition하는 것이 게임분석의 관건이고 전부인데, 위의 알고리즘들로 계산이 가능하기는 하다. 하지만, $|C|$는 일반적으로 exponential-size이므로[3] 위 알고리즘들 모두 exponential-time을 요구하므로 $|C|$가 큰 경우 실제로 사용할 수 없다. 컴퓨터가 인간을 바둑으로 항상 이길 수는 없는 것은 이런 이유 때문이다. (알파고도 가끔씩은 질 수 밖에 없다.)

물론 $|C|$가 exponential 크기라도 알고리즘을 잘 설계하면 polynomial-time에 수행하는 것이 존재할 수도 있을 것이다. 하지만, 계산복잡도 이론의 연구결과 combinatorial-game은 일반적으로 **PSPACE-hard**로 poly-time 알고리즘이 존재할 가능성이 매우 희박하다고 보면 된다.

[3] 바둑의 경우 $|C| = 3^{361}$

그럼에도 불구하고 **Nim** 게임과 같은 일부 게임들은 polynomial-time 알고리즘을 허용한다. 특히 impartial game들은 모두 Nim game으로 reduction 가능하므로 빠른 시간에 계산가능하다 (“Supplement: Nim & Impartial Games” 참조). 수학경시대회나 프로그래밍 경진대회에도 combinatorial game 문제가 자주 등장하는데 winning/losing position 조건이 모두 polynomial-time에 계산가능한 쉬운 경우이다. (Impartial game은 물론이고 일부 partizan game도 poly-time에 계산 가능하다.)

Winning/losing position 조건을 친절히 알려준 문제의 경우

- winning position에서는 상대를 상대의 어떤 losing position으로 보낼 수 있음
- losing position에서는 어떻게 움직여도 상대를 상대의 winning position으로 상대를 모실 수 밖에 없음

만 증명해주면 된다. 하지만 나머지 문제들은 winning/losing position 조건도 찾아서 이를 증명할 수 있어야 한다! (적어도 경시대회에서는 친절하게 winning/losing position을 알려주고 시작하는 문제는 없다.)