CS 4700: Foundations of Artificial Intelligence

Fall 2017

Instructor: Prof. Haym Hirsh

Lecture 10

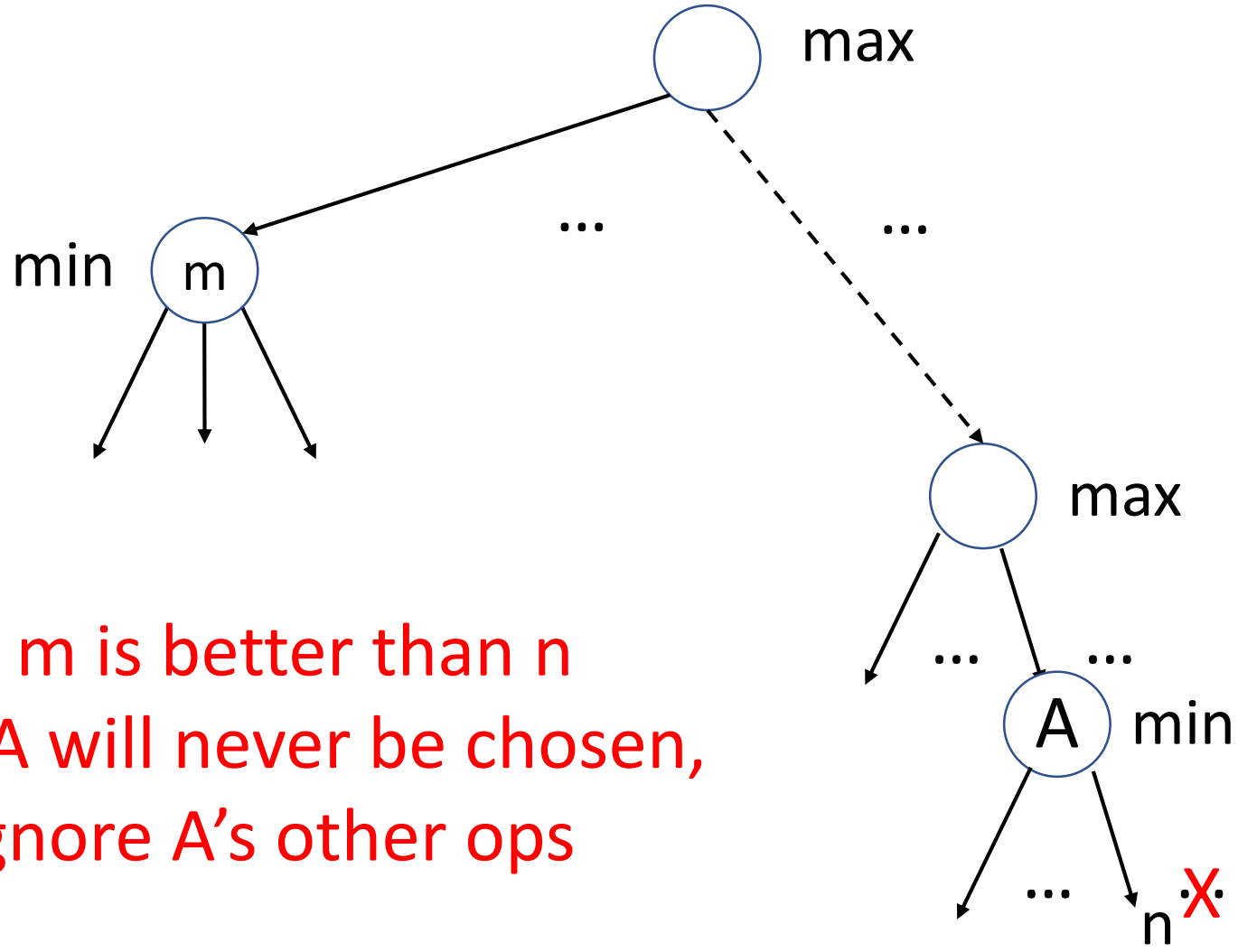
Today

- Adversarial search (R&N Ch 5)

Tuesday, March 7

- Knowledge Representation and Reasoning (R&N Ch 7)

Alpha-Beta Pruning: General Case



If m is better than n
then A will never be chosen,
ignore A's other ops

Minimax Algorithm

```
minimax(s,ops,depth):      {my turn}
    if cutoff(s,depth) then return V(s)
    else
        val ← -∞;
        foreach o ∈ ops
            val' ← maximin(apply(s,o),ops,depth+1);
            if val' > val then
                val ← val';
                bestop ← o;
        return val
```

```
maximin(s,ops,depth):      {opponent's turn}
    if cutoff(s,depth) then return V(s)
    else
        val ← +∞;
        foreach o ∈ ops
            val' ← minimax(apply(s,o),ops,depth+1);
            if val' < val then
                val ← val';
                bestop ← o;
        return val
```

Initial call:

- If I go first: minimax(initial-state,ops,0)
- If opponent goes first: maximin(initial-state,ops,0)

Minimax Algorithm with Alpha-Beta Pruning

```
minimax(s,ops,depth,a,b):  {my turn}
    if cutoff(s,depth) then return V(s)
    else
        val ← -∞;
        foreach o ∈ ops
            val' ← maximin(apply(s,o),ops,depth+1 ,a,b);
            if val' > val then
                val ← val';
                bestop ← o;
            if val ≥ b then return val;
            a ← max(a,val)
        return val
```

```
maximin(s,ops,depth,a,b):  {opponent's turn}
    if cutoff(s,depth) then return V(s)
    else
        val ← +∞;
        foreach o ∈ ops
            val' ← minimax(apply(s,o),ops,depth+1
            if val' < val then                      ,a,b);
                val ← val';
                bestop ← o;
            if val ≤ a then return val;
            b ← min(b,val)
        return val
```

Initial call:

- If I go first: minimax(initial-state,ops,0,-∞,+∞)
- If opponent goes first: maximin(initial-state,ops,0,-∞,+∞)

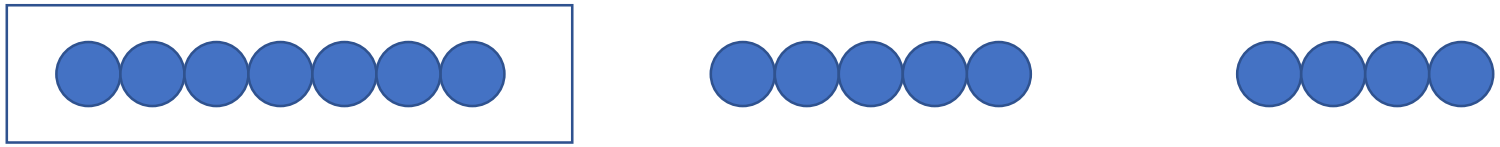
Game of Nim

- Three piles

- Each turn the current player picks one of the piles and removes at least one item from it

Game of Nim

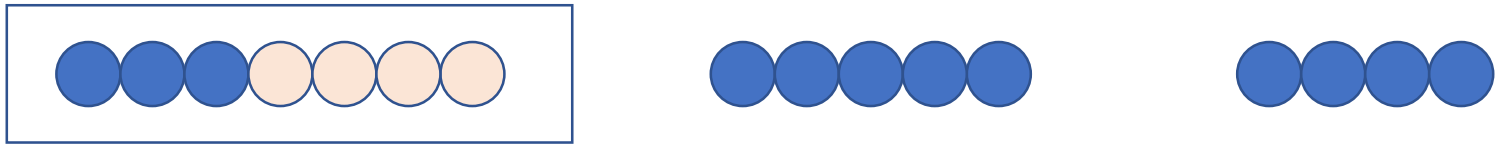
- Three piles



- Each turn the current player picks one of the piles and removes at least one item from it

Game of Nim

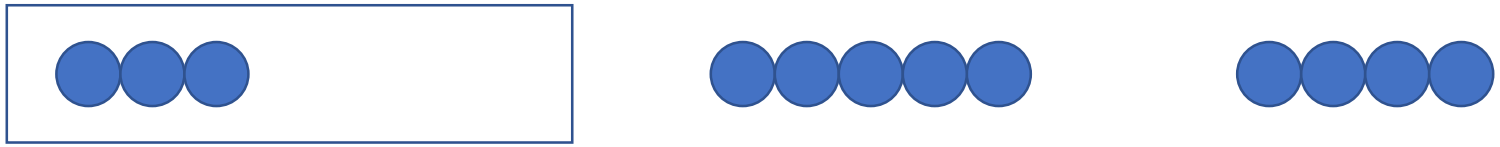
- Three piles



- Each turn the current player picks one of the piles and removes at least one item from it

Game of Nim

- Three piles



- Each turn the current player picks one of the piles and removes at least one item from it

Game of Nim

- Three piles

- Each turn the current player picks one of the piles and removes at least one item from it

Game of Nim

- Three piles

- Each turn the current player picks one of the piles and removes at least one item from it
- Whoever takes the last item wins

Game of Nim

- States: (x_1,x_2,x_3) – amounts in each of the three bins
- Operators:
  - Remove(n,i): remove n items from Pile i ($1 \leq n \leq x_i$)
- Win condition: end in (0,0,0) for player who just moved
  - $V(0,0,0) = +\infty$ if I moved last
  - $V(0,0,0) = -\infty$ if my opponent moved last

- What about V(s) for states that are not terminal nodes?

Game of Nim

- Represent the number of items in each pile in binary

Pile	Size	Binary
1	7	111
2	5	101
3	4	100

Game of Nim

- Represent the number of items in each pile in binary:
- Compute the ones digit of the sum of each columns of digits

Pile	Size	Binary
1	7	111
2	5	101
3	4	100

Game of Nim

- Represent the number of items in each pile in binary
- Compute the ones digit of the sum of each columns of digits

Pile	Size	Binary
1	7	111
2	5	101
3	4	100
		110

Game of Nim

- Represent the number of items in each pile in binary
- Compute the ones digit of the sum of each columns of digits

Pile	Size	Binary
1	7	111
2	5	101
3	4	100
		110

- $V(s) = +\infty$ if the sum is zero on my move, $-\infty$ if opponent's move
- To pick a move apply $V(s)$ to each successor, pick one with $V(s) = +\infty$

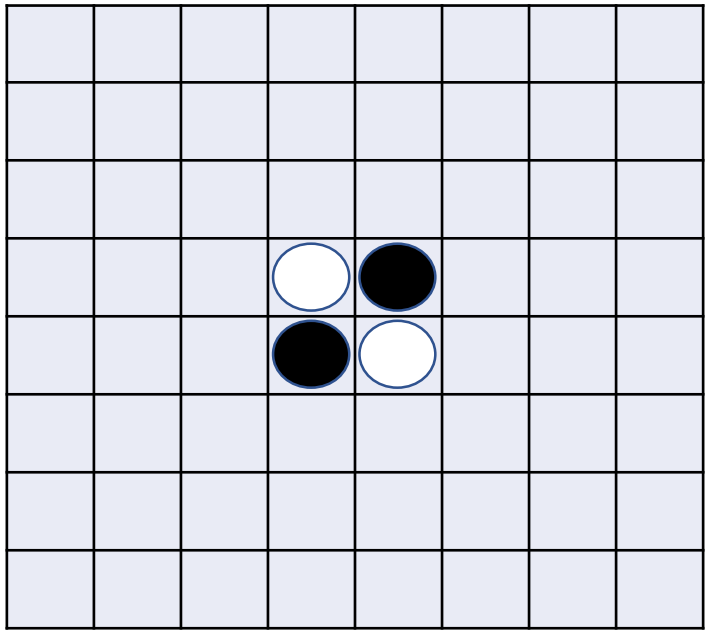
Game of Nim

- Represent the number of items in each pile in binary
- Compute the ones digit of the sum of each columns of digits

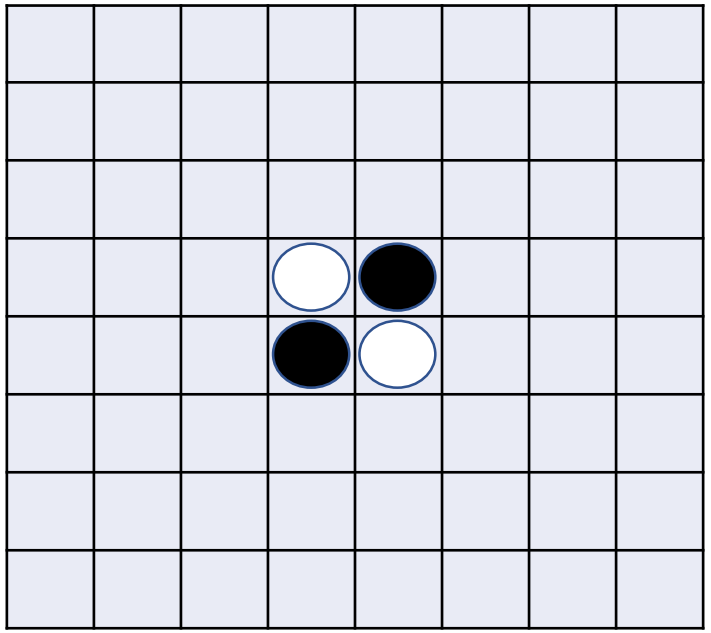
Pile	Size	Binary
1	7	111
2	5	101
3	4	100
		110

- $V(s) = +\infty$ if the sum is zero on my move, $-\infty$ if opponent's move
- To pick a move apply $V(s)$ to each successor, pick one with $V(s) = +\infty$
- Can have $V(s)$ that gives correct value without search

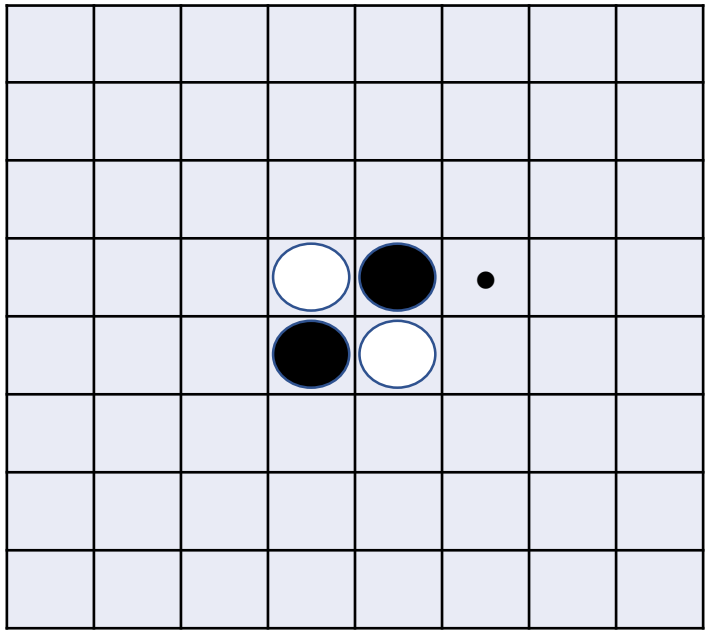
Othello (Reversi)



Othello (Reversi)

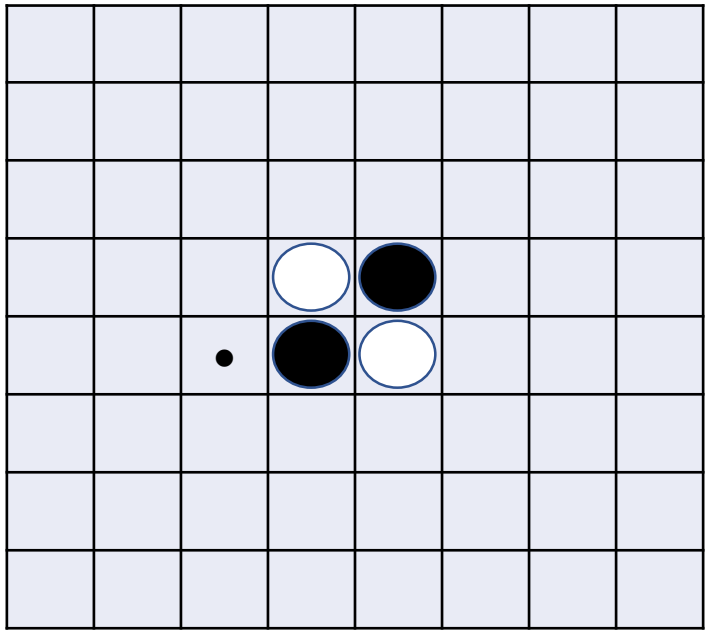


Othello (Reversi)



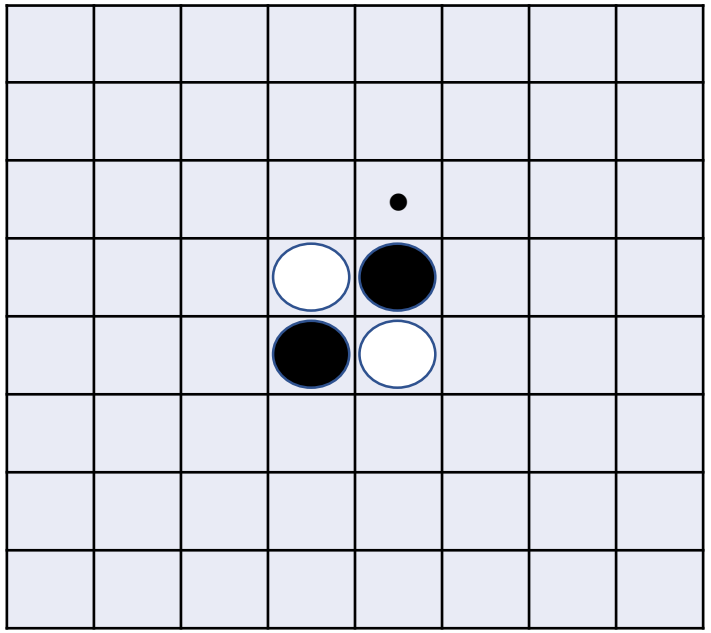
- Place a piece so that on a row, column, or diagonal you surround a contiguous sequence of opponent pieces
- Flip all surrounded pieces

Othello (Reversi)



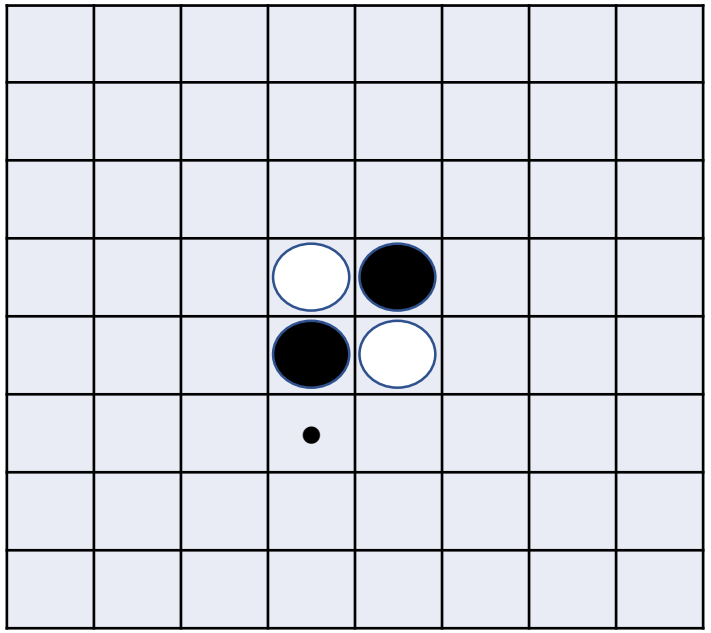
- Place a piece so that on a row, column, or diagonal you surround a contiguous sequence of opponent pieces
- Flip all surrounded pieces

Othello (Reversi)



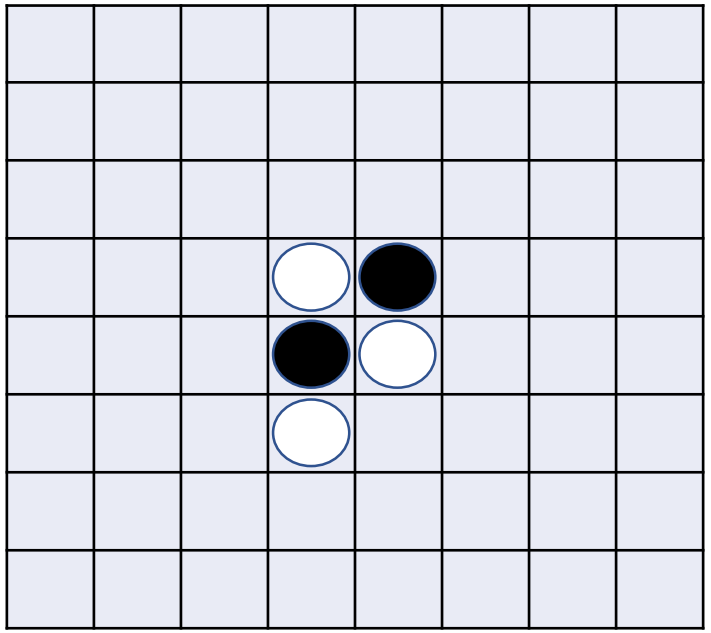
- Place a piece so that on a row, column, or diagonal you surround a contiguous sequence of opponent pieces
- Flip all surrounded pieces

Othello (Reversi)



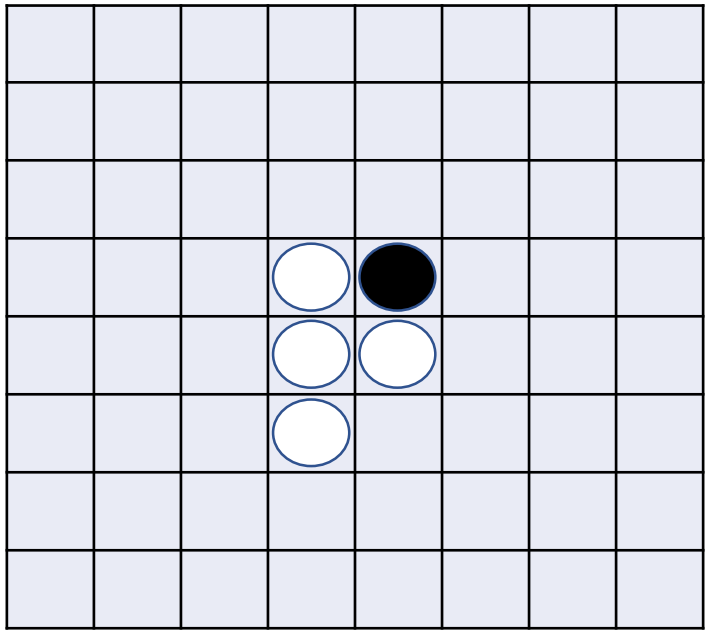
- Place a piece so that on a row, column, or diagonal you surround a contiguous sequence of opponent pieces
- Flip all surrounded pieces

Othello (Reversi)



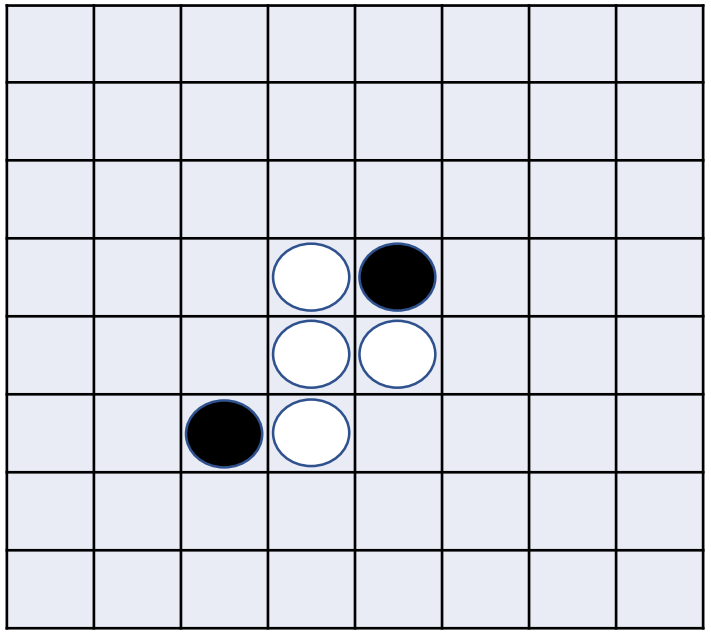
- Place a piece so that on a row, column, or diagonal you surround a contiguous sequence of opponent pieces
- Flip all surrounded pieces

Othello (Reversi)



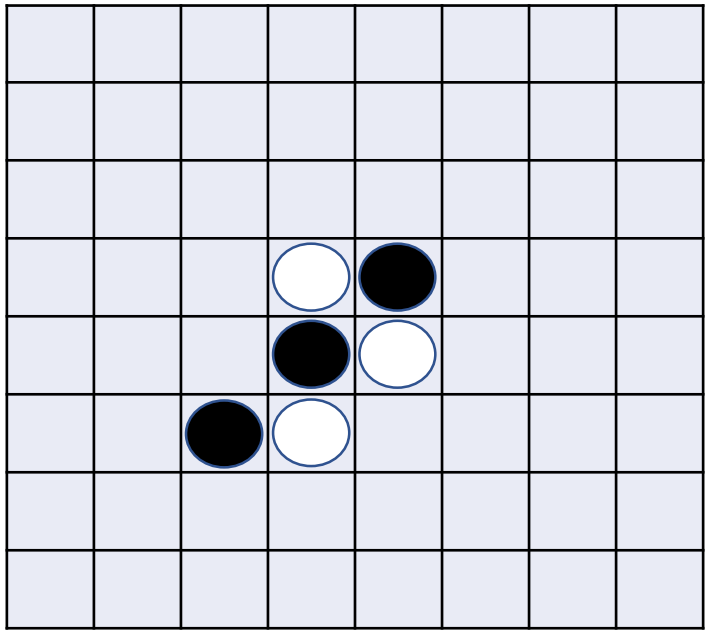
- Place a piece so that on a row, column, or diagonal you surround a contiguous sequence of opponent pieces
- Flip all surrounded pieces

Othello (Reversi)



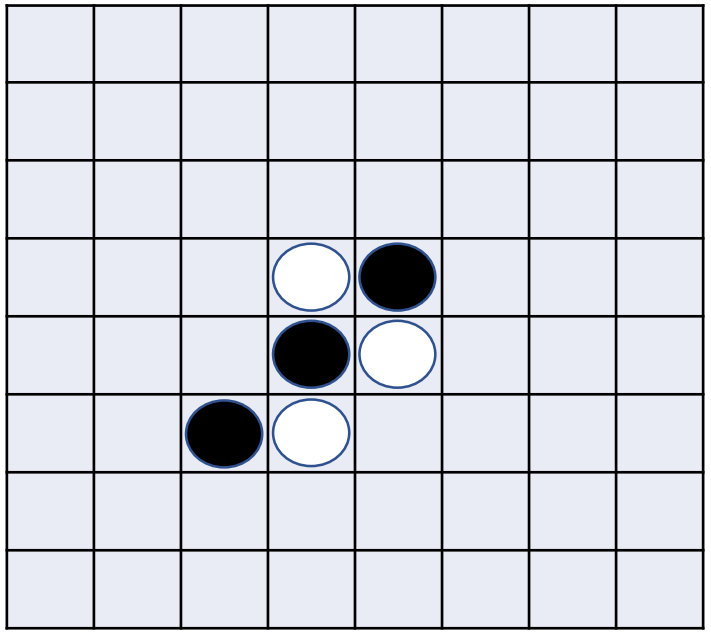
- Place a piece so that on a row, column, or diagonal you surround a contiguous sequence of opponent pieces
- Flip all surrounded pieces

Othello (Reversi)



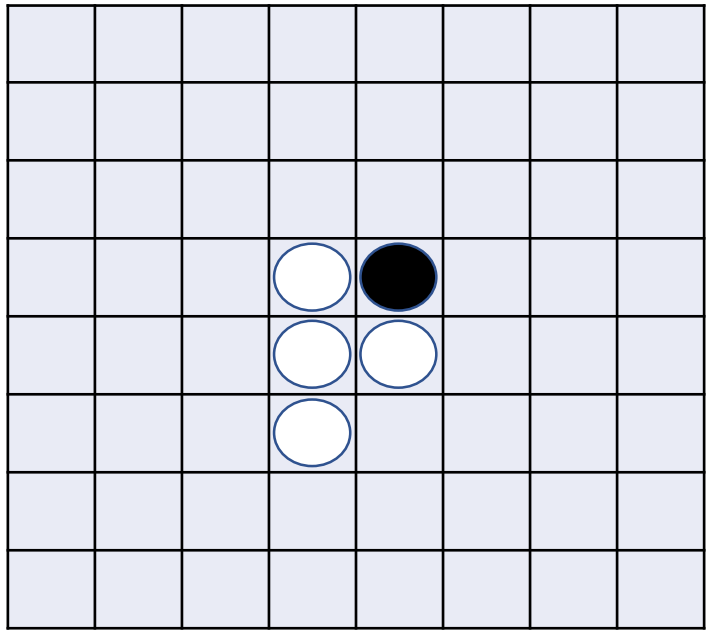
- Place a piece so that on a row, column, or diagonal you surround a contiguous sequence of opponent pieces
- Flip all surrounded pieces

Othello (Reversi)



- V(s) = # of my pieces - # of opponent’s pieces
- V(s) = 0 here

Othello (Reversi)



- V(s) = # of my pieces - # of opponent’s pieces
- V(s) = 3 for white / -3 for black

Claude E. Shannon (1950) XXII. Programming a computer for playing chess, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 41:314, 256-275

XXII. *Programming a Computer for Playing Chess* *.

By Claude E. Shannon,
Bell Telephone Laboratories, Inc., Murray Hill, N.J. †

[Received November 8, 1949.]

1. Introduction.

This paper is concerned with the problem of constructing a computing routine or "program" for a modern general purpose computer which will enable it to play chess. Although perhaps of no practical importance, the question is of theoretical interest, and it is hoped that a satisfactory solution of this problem will act as a wedge in attacking other problems of a similar nature and of greater significance. Some possibilities in this direction are :—

(1) Machines for designing filters, equalizers, etc.
(2) Machines for designing relay and switching circuits.
(3) Machines which will handle routing of telephone calls based on the individual circumstances rather than by fixed patterns.
(4) Machines for performing symbolic (non-numerical) mathematical operations.
(5) Machines capable of translating from one language to another.
(6) Machines for making strategic decisions in simplified military operations.
(7) Machines capable of orchestrating a melody.
(8) Machines capable of logical deduction.

It is believed that all of these and many other devices of a similar nature are possible developments in the immediate future. The techniques developed for modern electronic and relay type computers make them not only theoretical possibilities, but in several cases worthy of serious consideration from the economic point of view.

Machines of this general type are an extension over the ordinary use of numerical computers in several ways. First, the entities dealt with are not primarily numbers, but rather chess positions, circuits, mathematical expressions, words, etc. Second, the proper procedure involves general principles, something of the nature of judgment, and considerable trial and error, rather than a strict, unalterable computing process. Finally, the solutions of these problems are not merely right or wrong but have a continuous range of "quality" from the best down to the worst. We might be satisfied with a machine that designed good filters even though they were not always the best possible.

---

* First presented at the National IRE Convention, March 9, 1949, New York, U.S.A.

† Communicated by the Author.

Claude E. Shannon (1950) XXII. Programming a computer for playing chess, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 41:314, 256-275

$$f(P) = 200(K-K') + 9(Q-Q') + 5(R-R') + 3(B-B'+N-N') + (P-P') - 0.5(D-D'+S-S'+I-I') + 0.1(M-M') + \ldots$$

in which: -

(1)K,Q,R,B,B,P are the number of White kings, queens, rooks, bishops, knights and pawns on the board.

(2)D,S,I are doubled, backward and isolated White pawns.

(3)M= White mobility (measured, say, as the number of legal moves available to White).

Primed letters are the similar quantities for Black.

Claude E. Shannon (1950) XXII. Programming a computer for playing chess, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 41:314, 256-275

APPENDIX. THE EVALUATION FUNCTION FOR CHESS

The evaluation function f(P) should take into account the "long term" advantages and disadvantages of a position, i.e. effects which may be expected to persist over a number of moves longer than individual variations are calculated. Thus the evaluation is mainly concerned with positional or strategic considerations rather than combinatorial or tactical ones. Of course there is no sharp line of division; many features of a position are on the borderline. It appears, however, that the following might properly be included in f(P): -

(1)Material advantage (difference in total material).

(2)Pawn formation:

- (a)Backward, isolated and doubled pawns.
- (b)Relative control of centre (pawns at e4, d4, c4).
- (c)Weakness of pawns near king (e.g. advanced g pawn).
- (d)Pawns on opposite colour squares from bishop.
- (e)Passed pawns.

(3)Positions of pieces:

- (a)Advanced knights (at e5, d5, c5, f5, e6, d6, c6, f6), especially if protected by pawn and free from pawn attack.
- (b)Rook on open file, or semi-open file.
- (c)Rook on seventh rank.
- (d)Doubled rooks.

(4)Commitments, attacks and options:

- (a)Pieces which are required for guarding functions and, therefore, committed and with limited mobility.
- (b)Attacks on pieces which give one player an option of exchanging.
- (c)Attacks on squares adjacent to king.
- (d)Pins. We mean here immobilizing pins where the pinned piece is of value not greater than the pinning piece; for example, a knight pinned by a bishop.

(5)Mobility.

These factors will apply in the middle game: during the opening and end game different principles must be used. The relative values to be given each of the above quantities is open to considerable debate, and should be determined by some experimental procedure. There are also numerous other factors which may well be worth inclusion. The more violent tactical weapons, such as discovered checks, forks and pins by a piece of lower value are omitted since they are best accounted for by the examination of specific variations.

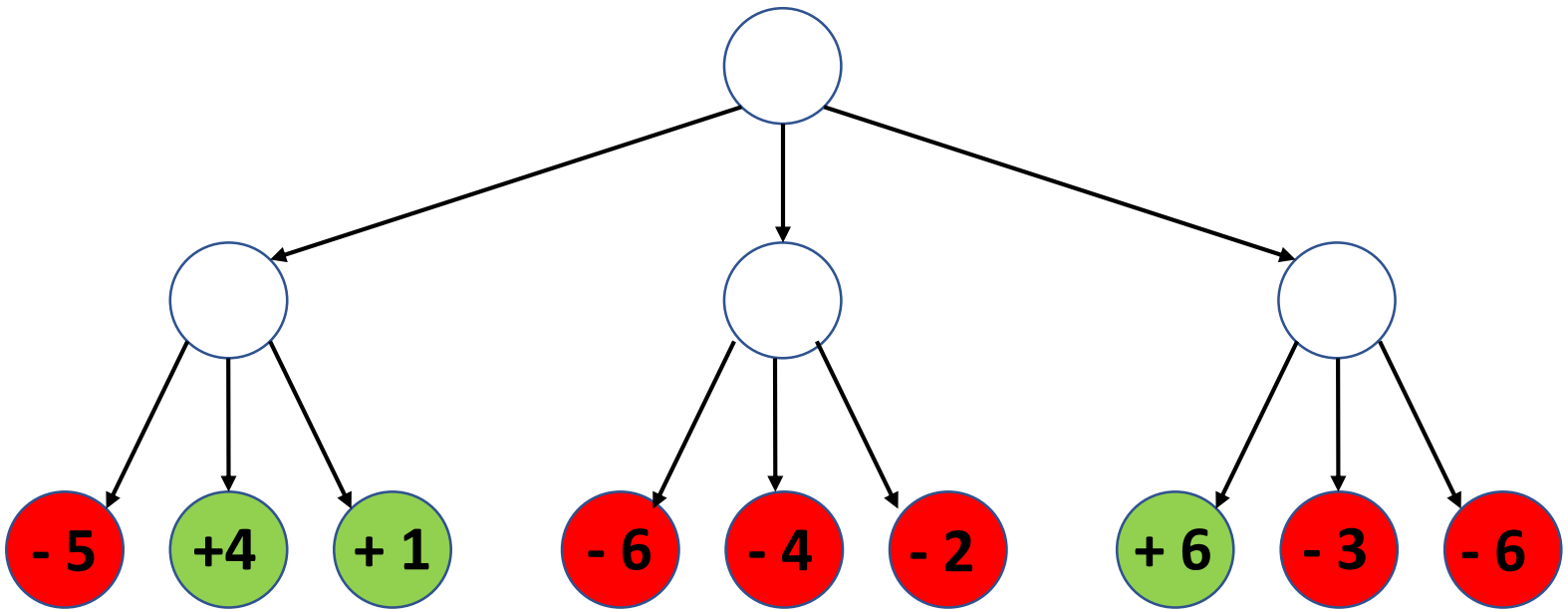
Common Form for Heuristic Evaluation Functions

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

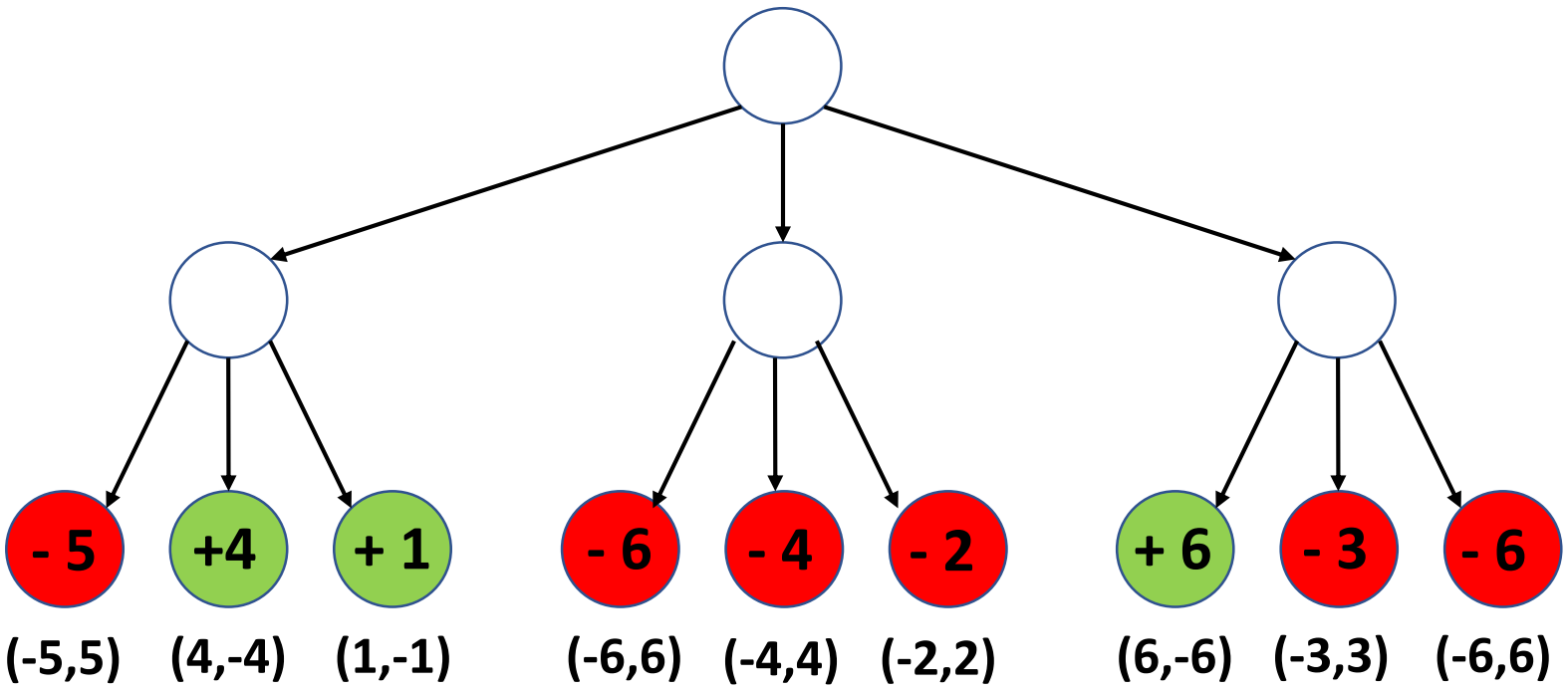
Additional Twists to Adversarial Search

- Non-zero sum
- More than two players
- Ordering/pruning branches
- Horizon effect
- Table lookup
- Stochastic games
- Learning evaluation functions
- Partially observable games

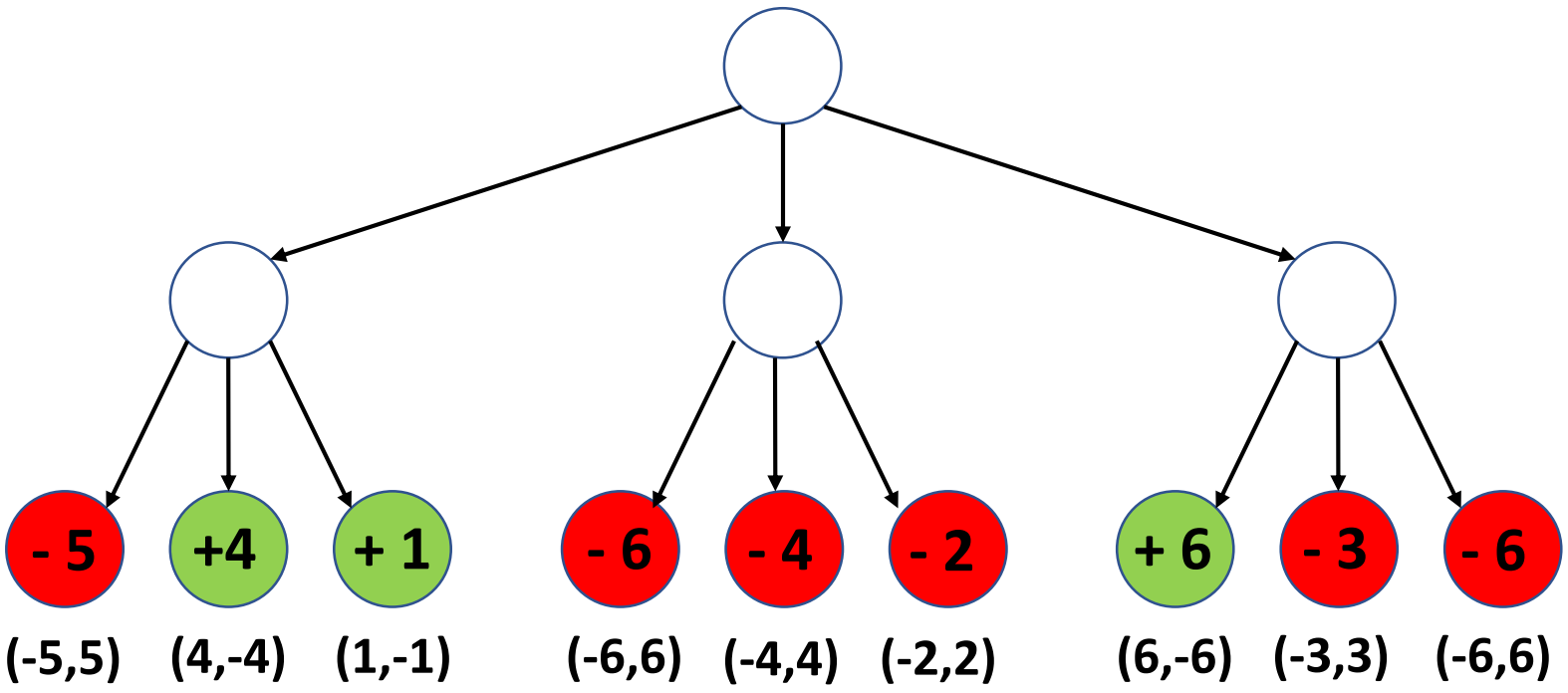
Two Players



Two Players Rewritten

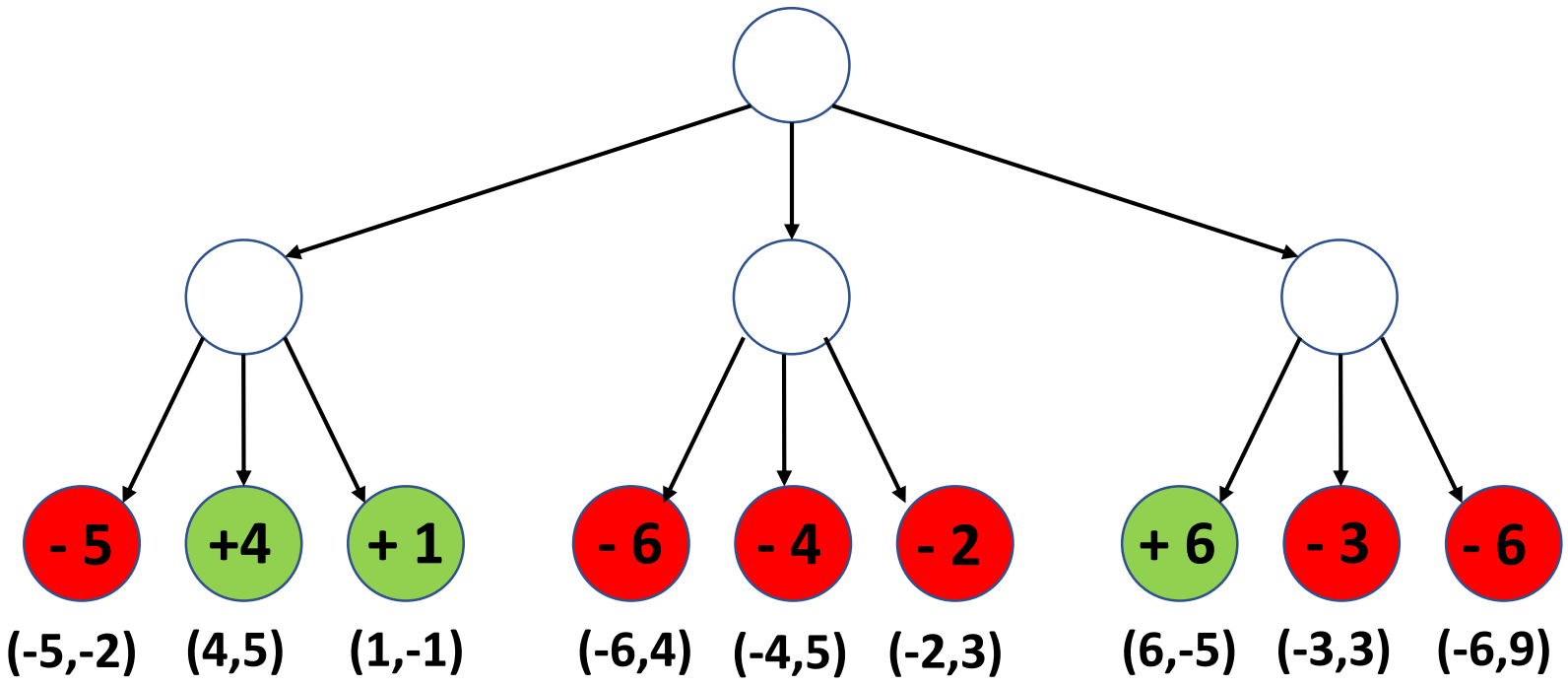


Two Players Rewritten



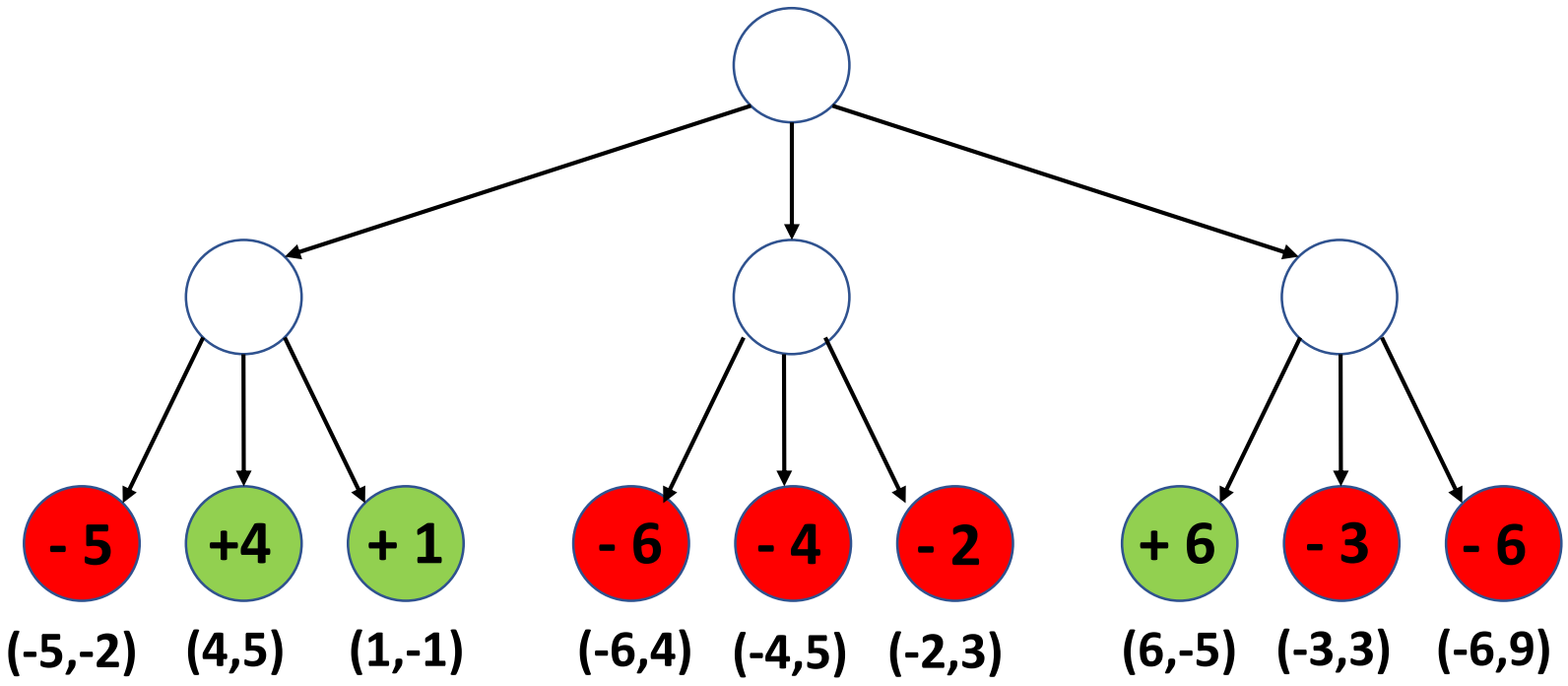
V(s) = (value of s to player 1, value of s to player 2)

Non-Zero Sum



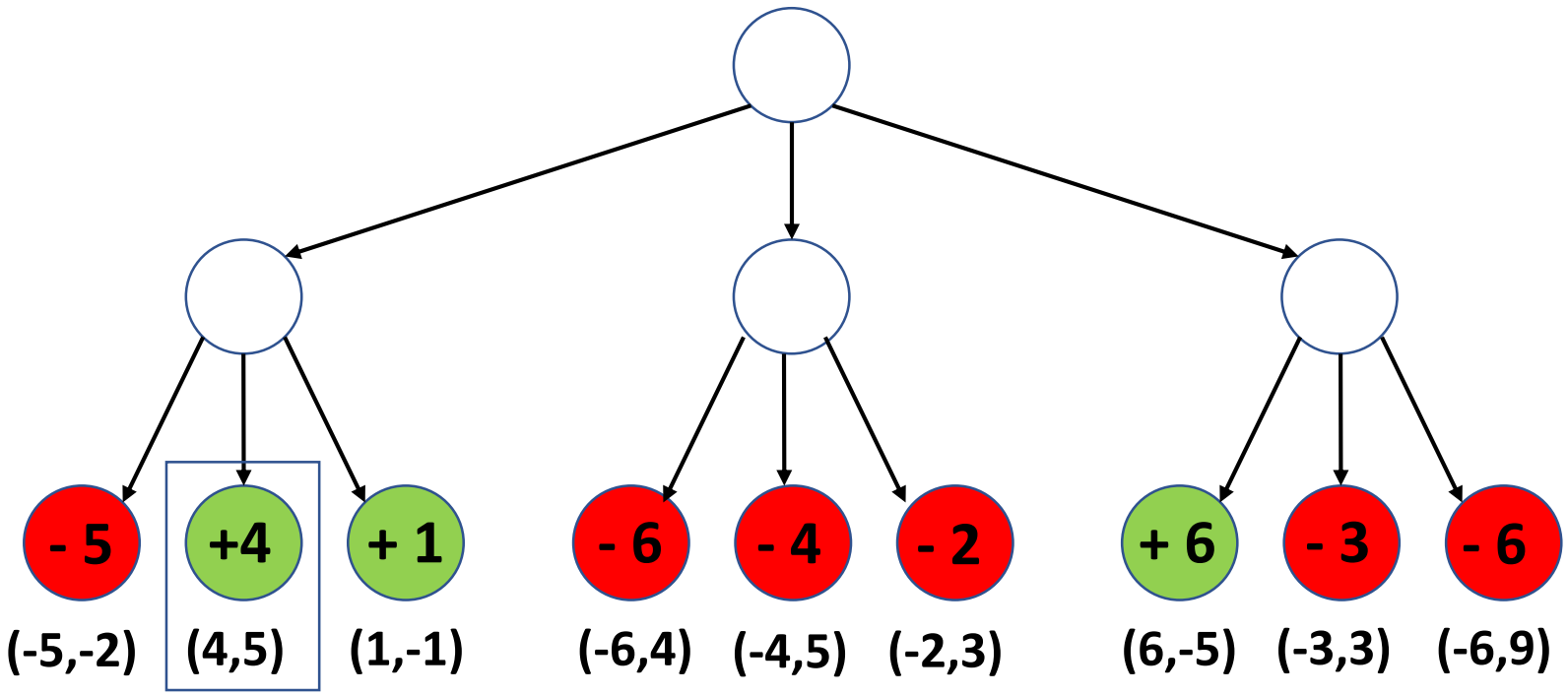
Values no longer add up to 0

Non-Zero Sum



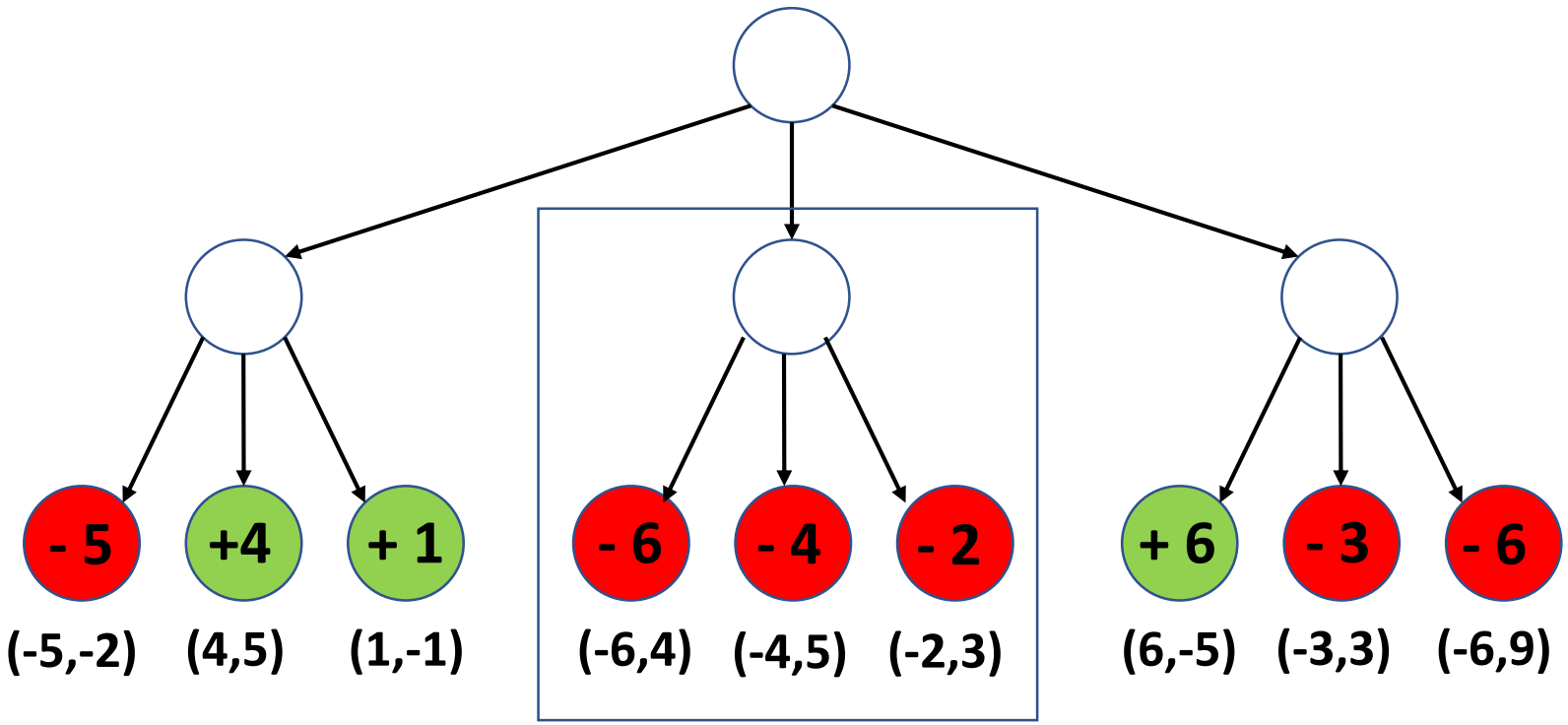
Value of a state for me is my number, opponent picks action based on his number

Non-Zero Sum



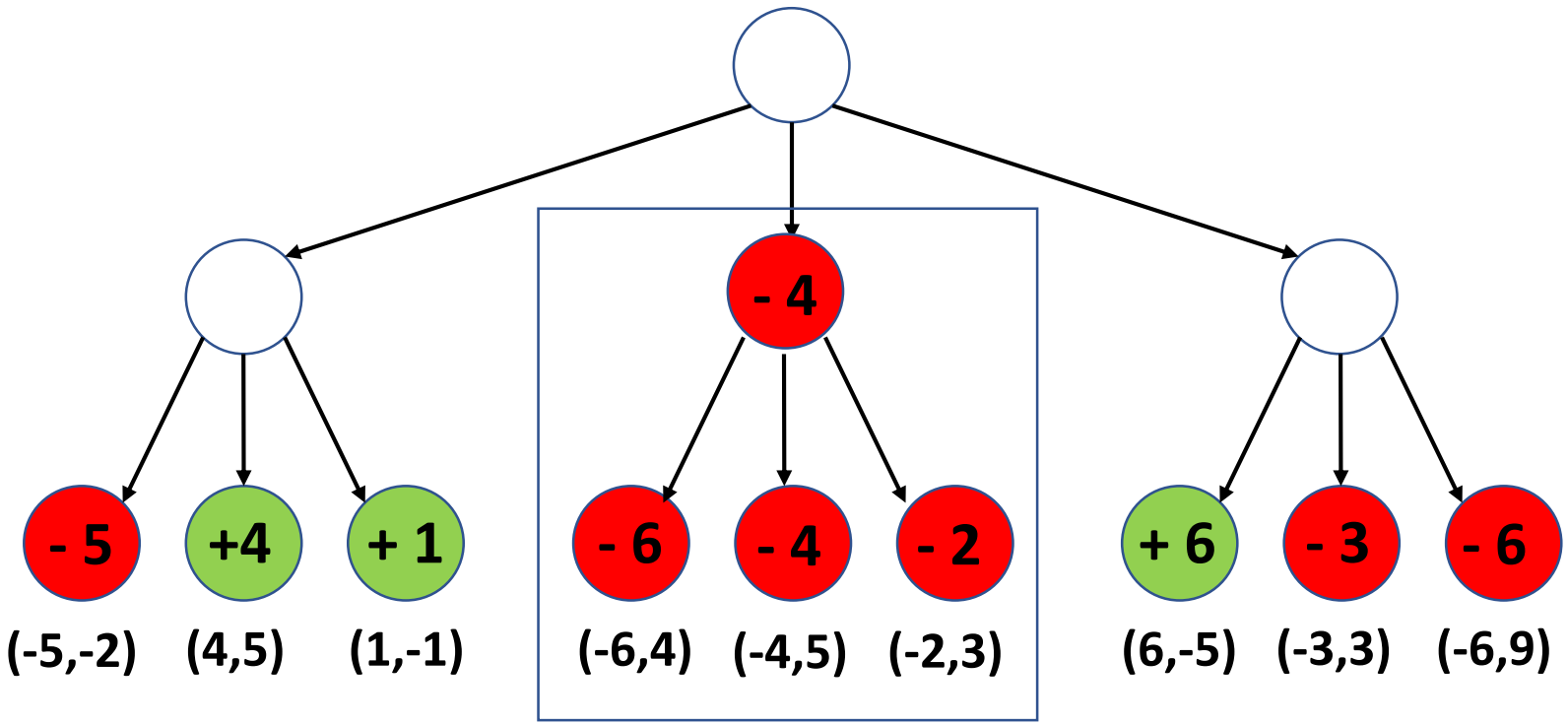
Best move for opponent could be mine, too

Non-Zero Sum



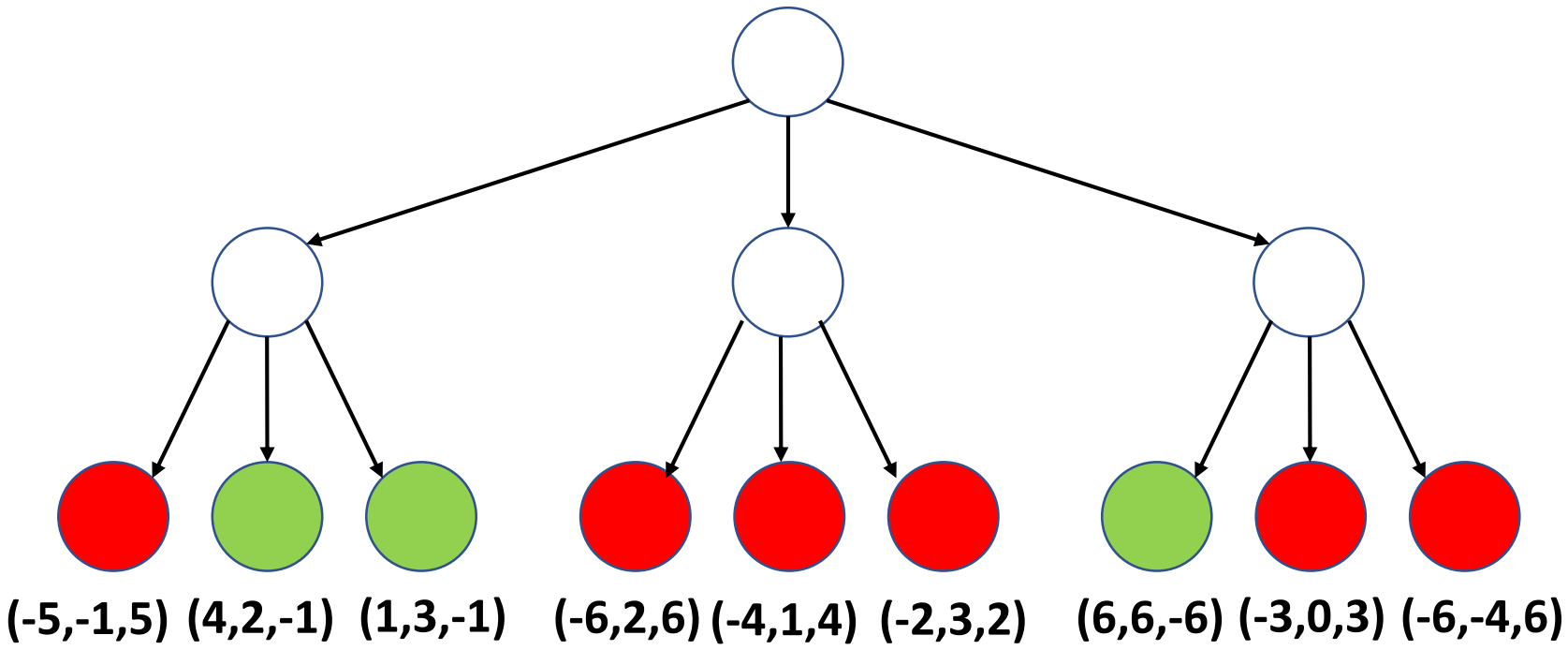
Value of a state for me is my number, opponent picks action based on his number

Non-Zero Sum



Value of a state for me is my number, opponent picks action based on his number

Three Players



V(s) = (value of s to player 1, value of s to player 2, value of s to player 3)

More Generally

- $V(s) = [v_1(s), v_2(s), \ldots, v_n(s)]$
  - n = number of players
  - $v_i(s)$ = value of s to player i
- Value of s for me is $v_1(s)$ [assuming I'm player 1]
- If it is player i's turn, the best move will maximize $v_i(s)$
  - Opponent will do what's best for opponent
  - May not be the worst for me!
- Ignores collaboration other than what arises from the v_i

Ordering / Pruning Branches

- Time complexity for minimax search is $O(b^m)$
  - Branching factor b
  - Tree depth m
- Using alpha-beta pruning
  - Best case: $O\left(b^{\frac{m}{2}}\right)$
    - $b^{\frac{m}{2}}$ is equivalent to $\sqrt{b}^m$
  - Random order: roughly $O\left(b^{\frac{3m}{4}}\right)$

Ordering / Pruning Branches

- Iterative Deepening:
  - Keep track of best move on each iteration
  - Do the best moves first on the next iteration
- Transposition tables:
  - Generalization of checking for revisited states in search
  - Can take into account symmetries

These maintain game tree search outcomes (except for ties) if searching all the way to terminals nodes

Ordering / Pruning Branches

- Forward Pruning - ProbCut:
  - More aggressive than alpha-beta pruning
  - Keep statistics on move variability, do shallow search to compute an estimate of V, then prune it if a state with that value at that depth is highly probable to be outside of the interval (alpha,beta)
- Null move:
  - Let opponent make two moves first to get an initial value for beta

These are *heuristic*, in that they might prune good paths