Backward Induction

Game Theory · Module 3 — Extensive Form Games

Node 3.4

Position in Knowledge Graph

System: Game Theory **Structural Domain:** Extensive Form Games **Node:** 3.4

Backward induction is the dynamic-programming solution of finite perfect-information games. Starting from terminal nodes and working toward the root, at each decision node you compute the action that maximizes the acting player's payoff given the optimal continuation strategies already computed for the subtree rooted at that node. The result is a pure strategy profile — the **backward induction equilibrium** — that is guaranteed to be a Nash equilibrium and, further, is subgame perfect (Node 3.5). For games without perfect information, backward induction on node values fails, but a generalization — backward induction on *information sets*, with belief updates and value recursion — underlies every modern poker solver. "River-first solving" is the poker-specific implementation of backward induction.

Formal Definition

Let Γ be a finite perfect-information extensive-form game (all information sets are singletons). Define the **backward induction value** $V_i(t)$ for every node $t \in T$ recursively:

Base case (terminal nodes). If $t \in Z$, $V_i(t) = u_i(t)$ for every player i.

Recursive case (decision nodes). If $P(t) = j$ and t's children are $\{t_a\}_{a \in A(t)}$ (with t_a reached by action a):

$$a^*(t) = \arg\max_{a \in A(t)} V_j(t_a), \qquad V_i(t) = V_i(t_{a^*(t)}) \text{ for every } i.$$

Chance nodes. If $P(t) = 0$ with distribution $p_t \in \Delta(A(t))$:

$$V_i(t) = \sum_{a \in A(t)} p_t(a) V_i(t_a).$$

The **backward induction equilibrium** is the pure strategy profile $s^{BI} = (a^*(t))_t$ restricted to each player's decision nodes, and the **backward induction outcome** is the play of the game under s^{BI} (from root).

Theorem (Zermelo–Kuhn). Every finite perfect-information game has at least one backward induction equilibrium, and every such equilibrium is a Nash equilibrium of the game.

Intuition

Think of backward induction as reasoning "from the end." Start at the leaves where payoffs are known. At the last-to-move player's last decision, they will maximize their own payoff among their available continuations.

Since you know what they'll do, the node *above* that is effectively simpler — its value is the value of the choice you just determined. Work your way up the tree one level at a time.

At the root, you have the value of the game under optimal play. Moreover, the specific actions chosen at each step constitute a pure strategy profile that is the optimal play.

Two key features:

Credibility. At each node, the player whose turn it is chooses the action they would *actually take if the node were reached*. This rules out incredible threats: no player commits to suboptimal off-path behavior, because the backward induction explicitly computes the optimal thing to do if the node were reached. This is exactly the refinement that subgame perfection (3.5) captures in general.

Complete information required. Backward induction on *nodes* works only in perfect-information games. If there are information sets with multiple nodes, you cannot compute "the value of node t" because the acting player doesn't know they're at t rather than at some other node in the same info set. In this case, backward induction must be generalized to work on information sets with belief updates (previewing Node 4.4).

Structural Role

Backward induction is the computational engine of dynamic game theory:

- **3.4 ⇒ 3.5.** Backward induction equilibria are subgame perfect.
- **3.4 ⇒ 3.6.** The centipede game reveals the puzzle of backward induction in finitely repeated interaction.
- **3.4 ⇒ 5.1.** In finitely repeated games, backward induction gives the unique unraveling result (the stage Nash repeated).
- **3.4 ⇒ 4.4.** Perfect Bayesian equilibrium generalizes backward induction to imperfect information.
- **3.4 ⇒ 9.3.** CFR is a learning-theoretic analogue of backward induction: it iteratively updates counterfactual values at each information set.
- **3.4 ⇒ Poker solving.** "River-first solving" is backward induction with beliefs at each information set.

Mathematical Development

Termination and correctness

In a finite tree, backward induction terminates in $O(|T|)$ node evaluations. At each step, the acting player's expected payoff is computed under the assumption that the continuation subtree plays optimally (which was computed in previous steps). Induction on tree depth establishes that the result is a Nash equilibrium.

Zermelo's theorem (1913)

Zermelo proved that in any finite two-player zero-sum game of perfect information without chance, one of the following holds: - Player 1 has a winning strategy. - Player 2 has a winning strategy. - Both have drawing strategies.

The proof is backward induction applied to tic-tac-toe, chess, checkers, etc. The existence of a winning/drawing strategy follows from finiteness alone. Computing the strategy may be infeasible (chess has ~10^{46} positions).

Centipede puzzle (preview)

Consider a finitely repeated Centipede game: two players alternate, at each stage deciding to "Take" (end the game with a split) or "Pass" (double the pot and hand to the opponent). Backward induction predicts that at the last stage, the player who is to move Takes (to get more than passing would yield after the

opponent takes). At the second-to-last stage, knowing the opponent will Take, you Take. Iterating, the unique backward induction equilibrium is "Take immediately" — a Pareto-inefficient outcome.

Empirically, people do not play "Take immediately." This is the puzzle of Node 3.6.

Extensions to imperfect information

For games with non-singleton information sets, backward induction must be replaced by a belief-augmented recursion. The value at an information set depends on:

1. Beliefs about which node within the information set the game is actually at.
2. Optimal play in the subtree beyond the information set.

This is the structure of perfect Bayesian equilibrium (Node 4.4) and sequential equilibrium (Node 4.6). Solver algorithms like CFR compute this recursively.

Worked Example

A three-level perfect-information game

P1 moves at root: A or B. - If A, P2 moves: a or b. - If a, payoff $(3, 1)$. - If b, payoff $(1, 3)$. - If B, P2 moves: c or d. - If c, payoff $(2, 2)$. - If d, payoff $(0, 4)$.

Backward induction.

Level 3 (terminal nodes): values are as given.

Level 2 (P2 nodes): - At the a/b node: P2 compares $b \to 3$ vs $a \to 1$, chooses b. Value $(1, 3)$. - At the c/d node: P2 compares $c \to 2$ vs $d \to 4$, chooses d. Value $(0, 4)$.

Level 1 (P1 root): - A leads to value $(1, 3)$, so P1 gets 1. - B leads to value $(0, 4)$, so P1 gets 0. - P1 chooses A.

Backward induction equilibrium: P1 plays A; P2 plays b (if reached) and d (if reached). Outcome: $(1, 3)$.

Normal form. P1 has 2 pure strategies: $\{A, B\}$. P2 has 4 pure strategies: $\{ab$ conditionals$\}, \{ac\}, \ldots$ — actually $2 \times 2 = 4$ pure plans. Normal form is 2×4 with entries determined by the combination of plans. The backward induction equilibrium is one of its Nash equilibria; there may be others that rely on incredible threats at off-path P2 nodes.

Stackelberg duopoly via backward induction

Two firms, Leader and Follower. Leader picks quantity $q_1 \geq 0$. Follower observes q_1 and picks $q_2 \geq 0$. Demand $P = a - q_1 - q_2$, marginal cost c.

Stage 2 (Follower). $\max_{q_2}(a - q_1 - q_2 - c)q_2 \Rightarrow q_2 = (a - c - q_1)/2$.

Stage 1 (Leader). Knowing q_2 as a function of q_1, maximize $\pi_1(q_1) = (a - q_1 - q_2(q_1) - c)q_1 = (a - c - q_1 - (a - c - q_1)/2)q_1 = \frac{(a-c-q_1)q_1}{2}$. FOC: $q_1 = (a - c)/2$.

Then $q_2 = (a - c - (a - c)/2)/2 = (a - c)/4$. Total output $3(a - c)/4$, price $P = a - 3(a - c)/4 = (a + 3c)/4$, Leader profit $\pi_1 = (a - c)^2/8$, Follower profit $\pi_2 = (a - c)^2/16$.

Comparison to Cournot. Cournot (simultaneous) equilibrium is $q_1 = q_2 = (a - c)/3$, each earning $(a - c)^2/9$. Stackelberg Leader does better $(1/8 > 1/9)$, Stackelberg Follower does worse $(1/16 < 1/9)$. This is the first-mover advantage: by committing credibly to a quantity, Leader changes Follower's best response.

Poker Anchor

Concept mapping

Formal object	Poker instantiation
Backward induction	"River-first solving" — compute optimal river play, then turn given river play, then flop given turn play
Leaf payoffs	Showdown outcomes — chip delta at the end of a hand
Value at node	Expected chip EV of a subtree under optimal continuation play
Credible continuation	A solver's "tree-exploitability" check — would opponent actually play the prescribed line if reached?
Chance nodes	Flop, turn, river deals, averaged with uniform weight

River-first solving = backward induction. Every modern poker solver operates by computing optimal play street-by-street, backward. First, it computes the optimal river play given the range distributions that reach the river. Then, knowing river EVs, it computes optimal turn play. Then flop. Then preflop. Each step replaces a subtree with a value computed from the one below.

This is literally backward induction in the sense of this node. The added wrinkle is the information-set structure: at each decision, you have a *distribution* over hole cards (your range), and the solver must compute value per range element, not per single hand. The recursion is the same; the object at each node is a vector (values for each hand in the range) rather than a scalar.

Concrete situation

Solving a HUNL turn decision. You are on the turn with your entire range vs opponent's range. The board is 9♡8♡3♣$K\diamondsuit$ after BTN c-bet on the flop was called. BTN decides on a turn barrel.

River values (computed first). For each possible river card (44 cards), and for each pair (your hand, villain's hand), the solver computes the EV of the river decision. This gives a lookup table: "if river is Ah and my hand is $A\heartsuit T\heartsuit$ and villain's hand is $K\clubsuit Q\clubsuit$, my river-optimal EV is X chips."

Turn values (computed next). Given the river lookup table, the solver computes the turn decision EV: for each turn action (bet/check/bet size), average over river cards with equity-weighted probabilities, using the river lookup for each. This produces a turn-optimal action distribution for each hand combination at the current turn info set.

Iterate up to flop and preflop. The flop decision is computed knowing the turn-optimal values. Preflop is computed knowing the flop-optimal values. The whole thing is a bottom-up (leaf-to-root) dynamic programming computation on the game tree.

What this understanding buys you

"River-first" is not a heuristic; it's the algorithm. When a solver tells you "bet 62%" on the turn, it computed that frequency by first solving the river under every possible turn outcome and then propagating those values backward. You cannot solve the turn in isolation because turn values *depend on* river values, which depend on the range distributions that reach the river, which depend on your turn action. The circular dependency is resolved by computing from the leaves (river) up.

Subgame re-solving = re-running backward induction from a specific node. If you want to solve just "the turn in this exact spot," you take the turn as your root, fix the input range distributions, and run backward induction on the turn-and-river subtree. This is standard practice in modern poker study.

Centipede-like effects in tournaments. In tournament situations with ICM pressures, iterated backward induction can predict "everyone folds forever" at the bubble, which is manifestly wrong in practice. This is

the same puzzle as the Centipede game (Node 3.6): empirical play deviates from backward induction because players are not common-knowledge rational, and ICM distortions further compound this.

Cross-Links

- **3.1 Game trees** — the object backward induction operates on.
- **3.2 Information sets** — singleton info sets for pure backward induction.
- **3.3 Strategies in extensive form** — backward induction outputs a pure strategy.
- **3.5 Subgame perfect equilibrium** — backward induction equilibria are SPE.
- **3.6 Centipede** — the puzzle of backward induction's behavioral predictions.
- **4.4 Perfect Bayesian equilibrium** — belief-augmented generalization.
- **9.3 CFR** — learning-theoretic analogue.
- **Dynamic programming (CS)** — the algorithmic family.

Structural Compression

Invariant: In a finite perfect-information game, the optimal action at every decision node depends only on the subtree rooted at that node; this locality enables bottom-up computation of optimal continuation values and produces a pure Nash equilibrium that is credible at every node.

Mechanism: Recursive value function $V_i(t)$ defined by maximization over child values at decision nodes and expectation over child values at chance nodes, computed from leaves to root.

Cross-System Echo: Backward induction is the same algorithm as value iteration in dynamic programming, the Bellman equation in Markov decision processes, and the minimax search in game-tree AI. Each computes optimal policy bottom-up using the principle of optimality (Bellman 1957).

Exercises

1. Apply backward induction to a 4-stage centipede game with doubling payoffs. Verify the unique backward induction equilibrium is "Take at stage 1."
2. For the Stackelberg duopoly, verify the profits $(a-c)^2/8$ and $(a-c)^2/16$ by direct substitution.
3. Show that in any finite perfect-information game, every backward induction equilibrium is a Nash equilibrium.
4. Construct a perfect-information game with multiple backward induction equilibria. What causes the non-uniqueness?
5. Apply backward induction to a 3-stage sequential auction (two bidders alternate bids), with valuations drawn at the start.
6. In the Stackelberg setup, what changes if the Leader cannot credibly commit to q_1? Derive the equilibrium of the simultaneous version and compare.
7. **Argue, structurally,** why backward induction fails in games with non-singleton information sets. What replaces the pointwise "optimal action at each node" rule? (Hint: beliefs.)

Game Theory · Module 3 — Extensive Form Games · Node 3.4
Concepts: nash-equilibrium, dynamical-systems
Prerequisites: 3.1, 3.3
Enables: 3.5, 3.6, 5.1