Fictitious Play and Heuristics for Multi-Agent Learning

Chris Ge
MIT
cge7@mit.edu

Alan Lee
MIT
alanlee@mit.edu

Sharvaa Selvan
MIT
sharvaa@mit.edu

***Abstract*—Multi-agent team games with imperfect information require teammates to coordinate on correlated strategies while acting under limited, asymmetric observations. The Team Belief DAG (TB-DAG) framework [1] provides a compact, perfect-information representation of such team decision problems (TDPs), but most prior work focuses on realization form vectors and CFR learning algorithms. In this project, we study the Team Leduc Hold'em challenge and develop a pipeline that combines TB-DAGs with fictitious play to compute low-exploitability correlated strategies. Our contributions are threefold. First, we fill in the gaps of the theoretical analysis of TB-DAGs in [1], which primarily works with the realization form vector of strategies. We instead consider strategies to be full descriptions of an action to take from each infoset and extend strategic equivalence for realization form vectors to a similar result for our definition of strategies, culminating in an algorithm to calculate the best response strategy on a TDP using its TB-DAG. Second, we implement fictitious play over this interface, with several practical optimizations (incremental reach-probability updates and caching), making long training runs feasible on limited hardware. Third, we empirically evaluate basic, heuristic, and algorithmic strategies on the course leaderboard, showing that our TB-DAG–based fictitious play reduces exploitability scores from 3.5/3.2 (basic) and 3.5/2.0 (heuristic) to 0.053/0.046 for Teams 13 and 24, respectively. Overall, our work fleshes out the algorithmic and conceptual foundations needed to apply TB-DAGs as a practical tool for multi-agent team learning.**

I. Introduction

Multi-player team games with imperfect information present a fundamental challenge: when teammates share payoffs but cannot communicate during play, standard game theory approaches designed for two-player zero-sum settings fail to take advantage of opportunities for correlation. In this project, we approach the problem of Team Leduc Hold'em, a four-player poker variant where Players 1 and 3 face Players 2 and 4. Importantly, teammates can only coordinate through team-private randomness before play begins.

The Team Belief DAG (TB-DAG) framework [1] generalizes the sequence form to enable efficient computation of correlated team max-min equilibria. This project implements the TB-DAG approach combined with fictitious play to compute approximate correlated equilibria. The implementation aims to accomplish three objectives: (1) Conversion of our scenario into a strategically equivalent perfect-information DAG, (2) Efficient best response computation, and (3) Fictitious play where teams iteratively best respond to opponent correlated mixtures. While the TB-DAG paper focuses almost exclusively on the theoretical results that follow from realization vectors, we instead focus our attention on fleshing out the relationships between strategies on the team decision problem and TB-DAG, in order to improve the foundations of future algorithms involving TB-DAG.

[0]Code at https://github.com/ChrisG777/Leduc-Part-2-Electric-Boogaloo. We will make the code public at 11:59 PM ET December 12, 2025.

II. Project Description

A. Problem Definition

In *Team Leduc Hold'em*, two teams consisting of Players 1 and 3 and Players 2 and 4 play with a reduced deck of 6 cards, consisting of 2 Jacks, 2 Queens, and 2 Kings. Initially, each player must place an ante of $1 into the shared pot, after which they are dealt a private card at random. A community card is also placed at the center, face down.

There are two rounds of betting, with each one commencing with the smallest numbered player still playing, and continuing in increasing player number order. Whenever it is a player's turn, they may:

- Check, which allows them to stay in the hand without any additional monetary commitment. If someone has already raised in the round, however, they must instead Call to stay in the round by matching the bet of the most recent raise.
- Raise, which increases the current bet size by $2 in the first round and $4 in the second round. In this variant of the game, there may be at most 2 raises per betting round, and no player can raise twice.
- Fold, leaving the round and losing any claim to the pot.

Each betting round ends when all remaining players have matched or checked against the highest bet. After the first round of betting, the community card is revealed, and after the second round of betting, if there are multiple players remaining, there is a showdown: all remaining players show their cards, and the player whose private card matches the community card (if any), followed by the highest card (in order of King, Queen, Jack) wins the pot. If two players share a high card, they split the pot.

Within teams, teammates can communicate and produce a correlated strategy prior to being dealt any cards, but may only signal using their actions publicly once the game begins. In this team version, a team's utility is defined as the sum of its players' payoffs.

B. Evaluation Metrics

For this project, we sought to produce two correlated strategies $\sigma = (\sigma_1, \sigma_2)$, one for each team (13 and 24, respectively), to minimize the exploitability of each team. The exploitability for a given correlated strategy is computed as the number of dollars lost in expectation against a team playing the best response to the correlated strategy.

A secondary goal was to win as many head-to-head matchups against other correlated strategies submitted by teams also completing the challenge. The head-to-head matchup result between a project group A submitting strategies $(\sigma_{A1}, \sigma_{A2})$ and B submitting strategies $(\sigma_{B1}, \sigma_{B2})$ was computed as

$$u(A, B) = \frac{u_1(\sigma_{A1}, \sigma_{B2}) + u_2(\sigma_{B1}, \sigma_{A2})}{2},$$

simulating the case where each project group is equally likely to play as either team. Both exploitability and head-to-head matchup results were publicly available to project groups throughout the project period.

C. Author Contribution

Once we decided to participate in the Leduc Poker challenge, we frequently enlisted various friends to simulate the game in real life. These simulations informed our initial heuristic approaches, where we worked closely together to reason through various game outcomes, using a common spreadsheet to define under which circumstances our hard-coded algorithm would check, fold, or raise.

Upon transitioning to a TB-DAG approach, all three team members helped formulate the toy problem and construct the TB-DAG for it. Sharvaa created the necessary infoset and game tree files for the toy game. Alan modified Chris's problem set 2 CFR code to construct team decision problem and TB-DAG classes in Python, and also built the connectivity graphs for both teams. Chris derived the theoretical relationships between strategies on the TDP and the TB-DAG, and designed and implemented the algorithms for best response calculation and fictitious play, resulting in the correlated strategies submitted to the leaderboard.

All authors contributed to writing the paper, and making the presentation.

III. Related Works

The Team Belief DAG (TB-DAG) framework [1] was introduced to address the computational challenges of multi-agent team games with imperfect information. Traditional tools such as the sequence form are efficient for two-player zero-sum games but do not naturally extend to settings where multiple teammates share payoffs but have private information. TB-DAG generalizes the sequence-form representation by constructing a directed acyclic graph over team-belief states, nodes that encode all joint information sets consistent with the team's observations. This representation allows the team to reason about correlated strategies by exploiting shared randomness.

Beyond TB-DAG, alternative approaches are largely driven by column-generation and tree-decomposition algorithms [2]–[6]. However, these methods operate directly on the game tree or require large-scale linear programs. The TB-DAG formulation is exponentially smaller and is conceptually cleaner than these other methods [1]. Therefore, we focus our attention to learning methods which can be applied to the TB-DAG.

Fictitious play (FP) and counterfactual regret minimization (CFR) are two standard approaches for approximating Nash equilbria in exentsive-form games. While CFR and its variants have become the standard in large games, much of the empirical evidence that FP converges more slowly comes from deep RL–based implementations and may not necessarily transfer to our setting [7]. Notably, the original TB-DAG work evaluates performance only using CFR variants defined over realization form vectors [1]. As we discuss in Section IV-E, their output is a mixed plan on the TB-DAG, which does not easily translate to correlated strategies on the original extensive form team game. In contrast, FP naturally produces pure strategies on the original team game. Variations of fictitious play have already been successfully applied to settings where a coordinated team faces an adversary [2], providing further motivation for our choice of using fictitious play.

IV. Methodology

A. Heuristics

We began by playing many rounds of the game, and constructing a strategy purely using heuristics.

Our **Basic Strategy** consisted of the following rules:

- **Round 1:**
  - Raise when holding a King (K).
  - Call otherwise.
- **Round 2:**
  - Raise if holding a pair.
  - Fold otherwise.

In implementation, raising and folding were defined using `hotRaise` (raise if possible, else call) and `checkFold` (check/call if the cost is 0, else fold).

These heuristics were further developed for the team consisting of players 2 and 4 (Team 24), with an intention to extend the resulting strategies to the other team (Team 13) if results were promising. Firstly, we made some key observations based on simple probability calculations, such as it being disadvantageous when teammates have the same card (as this prevents either teammate from matching the community card), and that raising should only be considered as a signaling mechanism or to increase pot odds.

We outlined three strategies based on if on the first round (1) player 1 raised, (2) player 1 did not raise but either player 2 or 3 raised, or (3) players 1-3 all do not raise: these cases come naturally from the actions available to players 2 and 4 on their first turns, as outlined in Table I.

The main difficulties from determining heuristic strategies for each case were instances of imperfect signaling, such as if player 2 had to signal whether they had a K, Q, or J using

TABLE I: Available actions on preflop turn 1

Case	Actions for Player 2	Actions for Player 4
(1)	C, F, R	(variable)
(2)	C, R	(variable)
(3)	C	C, R

just the actions C and R. In most cases, we would choose to assign multiple cards to a single action but leave at least a 1:1 correspondence between one card-action pair, e.g. using a R to signal K and using a C to signal either a Q or J. We decided this would be an easier choice to determine further actions in the game tree, since it minimizes uncertainty for at least one branch. If a turn was not used to signal to the teammate, we would manually calculate the probability of winning the pot (assuming nothing about the opposing team's signaling strategy) and play the action maximizing the expected utility of the team.

Our heuristic approach for the first betting round (preflop) prioritized having at least one player on the team know the identity of their partner's private card as soon as possible. This was usually possible before the end of the first betting round, but when it was not (e.g. all 4 players check the first betting round), we reasoned that the opposing team had little information to work off of as well.

In the second betting round (postflop), our heuristic approach was to generally check or fold when the team did not have a match with the community card, and "slow-play" (check/call for as long as possible until a raise was required to continue the round) when there was a match.

We experimented modestly with both correlation and randomization in our heuristic approaches. Noticing that having a J or Q often differed by little, we tried an equal mix of two correlated strategies, which differed only by actions used to signal a J in one strategy instead signaling a Q in the other, and vice versa. To lower exploitability, we also tried calling with a small probability after the second betting round even if the team did not have a match.

Using heuristic strategies, we produced a strategy for team 24 with minimum exploitability score 2.029; more detailed results are in Section V. Unsatisfied with these results, we decided to pursue a more learning-based approach.

B. Our real approach: Constructing the TB-DAG

For an algorithmic approach to the problem, we implemented a fictitious play on top of the TB-DAG [1]. The TB-DAG converts an imperfect-information team decision problem into a strategically equivalent perfect-information problem, which we can then use standard game theory equilibrium-finding techniques on. We will spend the next three subsections discussing our algorithm and implementation.

Following the construction of the TB-DAG from [1] closely, our implementation first constructs a connectivity graph G over all histories, where two histories h and h' at the same depth are connected if there exists a team information set I such that both $h \preceq I$ and $h' \preceq I$. To efficiently compute this graph, we precompute for each history h the set $L(h)$ of all reachable team information sets in reverse topological order, then connect histories h and h' when $L(h) \cap L(h') \neq \emptyset$.

The TB-DAG consists of active nodes (beliefs) representing sets of indistinguishable histories, and inactive nodes (observations) representing sets of histories after public observations. At active nodes, the team chooses prescriptions, joint actions specifying one action for each intersecting information set, leading to inactive child nodes. At inactive nodes, nature resolves public observations through connected components of the connectivity graph's induced subgraph, with each component becoming an active child node.

C. Best responses on the TB-DAG

In this section, we share our detailed understanding of how to use TB-DAGs as a tool to calculate best responses to opponent strategies on the original TDP (Team Decision Problem, the original extensive form team game). First, we describe **why** we can use the TB-DAG to compute a best response, grounded in theoretical guarantees; a worked example illustrating this is in Appendix B. Then, Algorithm 1 describes **how** we exactly use the TB-DAG to compute a best response.

We believe that this is the bottleneck for theoretical understanding of our approach, as we derived much of this knowledge ourselves from reading in between the lines of the paper. We hope that our description will make implementing TB-DAGs more accessible to those new to the field of multiagent learning and game theory, like us when we started the class.

For the discussion below, we borrow some notation from [1]. Let $z \in \mathcal{Z}$ denote the shared set of terminal nodes in the TB-DAG and its corresponding TDP (team decision problem), and let $x[z]$ denote the **realization form** vector of a (possibly mixed) team strategy: for pure strategies, $x[z] = 1$ if the team takes all actions belonging to that team from the root to z, and 0 otherwise. And for mixed strategies, we take the corresponding mixture of pure realization form vectors. The **reach probability** of a strategy is synonymous with the realization form: both are the probability of selecting a pure strategy from the mixed strategy, such that there exists some sequence of opponent and chance moves that leads to that terminal node under this pure strategy.

Our algorithm relies on the following facts about the relationship between a TB-DAG and its corresponding TDP.

1) **Strategic equivalence** (Theorem 4.2 in [1]): the set of realization form vectors $x'[z]$ attainable by strategies on a TB-DAG (henceforth, we will call strategies on TB-DAGs **plans** to disambiguate) is the same as the set of realization form vectors attainable by strategies on the TDP $x[z]$.
2) **Payoffs in terms of realization form vectors**: In the TDP, let $w[z]$ be the *opponent's and chance's* combined reach probability of terminal node z, given a fixed (possibly mixed) strategy of the opponent and chance (a fixed opponent is indistinguishable from chance). Let $u[z]$ be the payoff to our team of that terminal node.

And as before let $x[z]$ be our strategy's realization form vector. Then

$$\text{payoff} = \sum_{z \in \mathcal{Z}} x[z]w[z]u[z]$$

(The proof of this equation is in Appendix A). In particular, this implies that once the utilities and an opponent strategy are fixed, each terminal node has an **effective payoff** $w[z]u[z]$. Furthermore, finding the optimal best response just means finding the strategy attaining the optimal $x[z]$; call this $x_{opt}[z]$. But by (1), the same $x[z]$'s are attained on the TDP and TB-DAG. So if we **define** the effective utilities $u'[z]$ on the TB-DAG to be $w[z]u[z]$, and **define** the payoff on the TB-DAG to be $\sum_{z \in \mathcal{Z}} x[z]u'[z]$, and then find the optimal strategy (best response) on the TB-DAG, it will have realization form $x'_{opt}[z] = x_{opt}[z]$ since the sets of possible realization form vectors are the same!

To recap, we can ***compute the opponent reach probabilities on the TDP, but calculate the best response plan on the TB-DAG***, since the resulting plan on the TB-DAG is guaranteed by strategic equivalence to attain the optimal $x[z]$ for the TDP as well. But now we need a way to convert from this plan achieving x_{opt} on the TB-DAG to a strategy achieving x_{opt} on the TDP. The next two items will handle that.

3) **Best response on TB-DAG is pure**: We will explicitly give a standard backwards induction algorithm to calculate the best response on the TB-DAG, given the effective payoffs described in (2). The algorithm will clearly always output a pure plan[1].
4) **Pure plan TB-DAG and Pure strategy TDP conversion**: Given either a pure plan on a TB-DAG or a pure strategy on a TDP, there is an algorithm to convert to the other one, such that the two strategies attain the same realization form vectors. This is not a bijection, but it does not need to be. We describe how to do this below.

Combining the observations above leads naturally to Algorithm 1 for calculating the best response to a given opponent strategy. In particular, **Algorithm 1 for computing the best response is the only interface that fictitious play will need to use to interact with the TB-DAG**. Everything else is done purely in the TDP. We assume that the reader knows how to compute reach probabilities on TDPs, through a standard top-down search through the tree and averaging over mixing of strategies.

Converting between pure plans and pure strategies: We back out these algorithms from the proof of Theorem 4.2 in the appendix of [1].

Pure strategy $\implies$ *pure plan*: Converting from a pure strategy to a pure plan requires us to specify an action at each action node, rather than each infoset controlled by the player. However, this is done straightforwardly by defining the prescription "action" taken on the action node based on the pure strategy: for each intersecting infoset, we play according to the action given in the pure strategy.

Pure plan $\implies$ *pure strategy*: Converting a pure plan back into a pure strategy is done by iterating over all active nodes reached under our pure plan in the TB-DAG, and extracting which actions are performed at each infoset by examining the prescription for the node that infoset appears in. No two active nodes reachable under our best response will share a common infoset, so there are no issues of conflicting actions [1]. However, we do have some infosets in the original game tree that are not reached under our best response, so we randomly assign actions to obtain a full pure strategy.

[1]The fact that this best response is pure is corroborated by https://piazza.com/class/mf4r63g8qg94tf/post/65#

Algorithm 1 Best Response via TB-DAG Backward Induction

Input: TB-DAG $\mathcal{D}$ for team i, opponent strategy σ_{-i} on the TDP
Output: Best response pure strategy σ_i^* on the TDP, expected value v^*

function COMPUTEBESTRESPONSE($\mathcal{D}, \sigma_{-i}$)
// Step 1: Compute effective payoffs at terminals
for each terminal node $z \in \mathcal{Z}$ **do**
  $w[z] \leftarrow$ reach probability of z under σ_{-i} and chance (on the TDP)
  $\texttt{eff}[z] \leftarrow w[z] \cdot u_i[z]$ {effective payoff}
end for

// Step 2: Backward induction on the TB-DAG in reverse topological order
for each node v in $\mathcal{D}$ (children before parents) **do**
  if v is a **terminal** node **then**
    $V[v] \leftarrow \texttt{eff}[v]$
  else if v is an **inactive** node **then**
    $V[v] \leftarrow \sum_{c \in \texttt{children}(v)} V[c]$
  else if v is an **active** node **then**
    $V[v] \leftarrow \max_{a \in \mathcal{A}(v)} V[\texttt{child}(v, a)]$
    $\pi^*[v] \leftarrow \arg\max_{a \in \mathcal{A}(v)} V[\texttt{child}(v, a)]$
  end if
end for

// Step 3: Convert pure plan on TB-DAG to pure strategy on the TDP
$\sigma_i^* \leftarrow$ PUREPLANTOPURESTRATEGY($\pi^*, \mathcal{D}$)
$v^* \leftarrow V[r]$ {r is the root of $\mathcal{D}$}
return σ_i^*, v^*

D. Fictitious Play

Now that we have a best response calculator, we can use fictitious play to find a Team Correlated Equilibrium. In each iteration of fictitious play, each team finds a best response to $\sigma_i^{(n)}$, a uniform mixture of the other team's pure strategies so far, and then adds that best response to their own team's list of pure strategies. Instead of recalculating this uniform mixture of

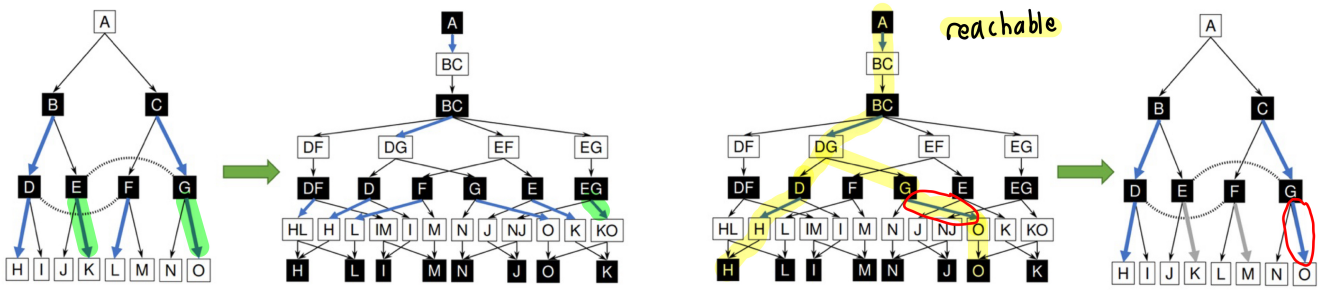


Fig. 1: **Left:** Conversion from a pure strategy on the TDP to a pure plan on the TB-DAG. For each node on the TB-DAG, we prescribe the set of actions taken by the infosets on the TDP: for instance, for active node EG, we know E must go to K, and G must go to O, so EG goes to KO (green highlights). **Right:** Conversion from a pure plan on the TB-DAG to a pure strategy on the TDP. We consider all reachable active nodes on the TB-DAG, and read off the actions prescribed at each infoset. For example, when we reach active node G and the action points to O, that tells us that on the TDP, we should take the action to O (red). We ignore what the pure plan tells us to do on infoset EG, since that is not a reachable active node under the pure plan in the TB-DAG. The gray arrows at E and F on the TDP indicate that the actions at these infosets are not specified by the pure plan, since they are unreachable; we pick an action randomly for the pure strategy. Figure borrowed from [1].

the pure strategies on each iteration, we follow an equivalent form: the Brown/Robinson update equation for fictitious play [8]

$$\sigma_i^{(n+1)} = \left(1 - \frac{1}{n+1}\right)\sigma_i^{(n)} + \frac{1}{n+1}\beta_i\left(\sigma_{-i}^{(n)}\right), \quad (1)$$

where $\sigma_i^{(n)}$ denotes the strategy played by team i at iteration n, $-i$ denotes the opponent of team i, and $\beta_i(\sigma_{-i}^{(n)})$ is the pure best response to the opponent's strategy on the previous iteration:

$$\beta_i(\sigma_{-i}^{(n)}) \in \arg\max_{\sigma_i} u_i(\sigma_i, \sigma_{-i}^{(n)}), \quad (2)$$

It is easy to check that this update keeps the mixture weights over the pure strategies uniform (i.e. from $\frac{1}{n}$ to $\frac{1}{n+1}$). The reason we use this formulation is for computational efficiency, as discussed in Section VI-C.

We give full pseudocode for our algorithm in Algorithm 2.

E. Why Fictitious Play over DCFR

In the initial TB-DAG framework [1], the authors use DCFR rather than fictitious play. However, they only evaluate utilities for their realization form vectors on the TB-DAGs produced, without converting the resulting plans back to strategies on the TDP.

The output of a CFR-based algorithm on the TB-DAG would be a mixed plan with varying prescription probabilities at each node. However, as part of our submission requirements, we must convert any mixed plans on the TB-DAG back into correlated strategies for the TDP. Approximately converting from a mixed plan on the TB-DAG to a mixture of pure strategies on the original TDP is possible. Other teams used strategies such as Monte Carlo sampling, along with other tricks to ensure that each action is sufficiently represented and to fine-tune the mixture weights, but we did not want to proceed with these approximations in the absence of theoretical guarantees on their performance. In contrast, with fictitious play, we can perfectly translate the final learned strategy into the correlated strategy format, because it only requires conversion between pure plans on the TB-DAG and

Algorithm 2 Fictitious Play for Team Correlated Equilibrium

Input: TB-DAGs $\mathcal{D}_1, \mathcal{D}_2$ for teams 1 and 2, number of iterations T

Output: Correlated strategies σ_1, σ_2 (mixtures of pure strategies)

function FictitiousPlay($\mathcal{D}_1, \mathcal{D}_2, T$)

// Step 1: Initialize with arbitrary pure strategies
$\beta_1^{(0)} \leftarrow$ basic pure strategy for team 1
$\beta_2^{(0)} \leftarrow$ basic pure strategy for team 2
$\sigma_1 \leftarrow \beta_1^{(0)}$ with weight 1 {initialize correlated strategy}
$\sigma_2 \leftarrow \beta_2^{(0)}$ with weight 1

// Step 2: Iteratively compute best responses
for $t = 1$ **to** T **do**
  // Team 1 best responds to team 2's correlated strategy
  $\beta_1^{(t)}, v_1^{(t)} \leftarrow$ ComputeBestResponse($\mathcal{D}_1, \sigma_2$)
  $\sigma_1 \leftarrow (1 - \frac{1}{t+1}) \cdot \sigma_1 + \frac{1}{t+1} \cdot \beta_1^{(t)}$ {add to mixture}

  // Team 2 best responds to team 1's (updated) correlated strategy
  $\beta_2^{(t)}, v_2^{(t)} \leftarrow$ ComputeBestResponse($\mathcal{D}_2, \sigma_1$)
  $\sigma_2 \leftarrow (1 - \frac{1}{t+1}) \cdot \sigma_2 + \frac{1}{t+1} \cdot \beta_2^{(t)}$ {add to mixture}
end for

return σ_1, σ_2

pure strategies on the TDP.

We did initially consider taking the pure strategies corresponding to the pure plans with the highest k probabilities (product of probabilities along the branches of the reached TB-DAG action nodes). However, this does not produce a representative set of k pure strategies: consider the example of a mixed plan on a TB-DAG where the first action node allows two prescriptions of probabilities 0.9 and 0.1. The 0.9 branch leads to one more action node with 100 prescriptions of probabilities 0.01, and the 0.1 branch leads to one more action node with 100 prescriptions of probabilities 0.01 as well. Taking the top 100 (even distinct) probabilities in this case would only include pure strategies that choose the actions in the probability 0.9 prescription, although a 90:10 split between each initial prescription would be more favorable.

F. Toy Game

In order to test that our end-to-end fictitious play algorithm was indeed calculating equilibrium correlated strategies, we came up with a small toy game involving two teams, which has a nontrivial team correlated equilibrium. We present the game here, as it may be useful as a preliminary test case for other general team learning methods.

In this game, Players 1 and 3 are on a team together, and Player 2 is on the opposite team. The game proceeds in the following manner:

1) Players 1 and 3 each receive a private card in $\{1, 2\}$, with each outcome having equal probability.
2) Player 1 plays their first move, either 1 or 2.
3) Player 2 sees player 1's action, and then also selects either 1 or 2.
4) Player 3 sees both other players' actions and their own private card, and plays either M (match) or N (no match).

For the **payoffs** of the game, the team with players 1 and 3 earns $+2$ if player 3 correctly guesses if their hidden cards matched, and -2 otherwise. They receive an additional penalty of -10 if player 2 correctly guesses what player 1's hidden card was (their action equals Player 1's card). The reward is split equally among players 1 and 3. Player 2 receives negative of Players 1 and 3's reward to enforce a zero sum game.

Player 1 and player 3 want to pre-determine a signaling strategy for player 1 to express their card to player 3, playing one action for one of the hidden card possibilities, and the other action for the other one. However, if player 1 always plays 1 when they have a 1 and 2 when they have a 2 for example, player 2 can exploit that and predict player 1's card. Thus, in the **equilibrium strategy**, Player 1 and 3 play a correlated strategy, mixing with probability 0.5 over whether player 1 is signaling using the same action as their card, or the opposite action as their card. Player 2, without knowing this signal, gets no information from Player 1's action, so they default to picking 50-50 between 1 and 2.

The toy game tree can be found in Figure 2.

TABLE II: Exploitability Scores of Strategies

Strategy	Team 13	Team 24
Basic	3.527	3.207
Heuristic	3.527	2.029
Algorithmic	0.053	0.046

V. Evaluation/Results

The comparison of exploitability scores for our strategies are depicted in Table II. We found that our heuristic strategy performed very well after implementing just our preflop logic for the case when player 1 raises, with the exploitability score of Team 24 dropping from 3.2 to 2.0. However, after we hardcoded the remaining two preflop scenarios and all the post-flop scenarios, more than doubling the amount of strategy logic, the Team 24 exploitability actually increased, which surprised us. We suspect we had a bug in our implementation, though without a principled way of figuring out what it was, we abandoned the approach.

Our final approach with the TB-DAG fared much better, as we expected based on having theoretical foundations. Exploitability scores and head-to-head scores of our final submission are shown in Figure 3. We observe that despite having far lower explitability than JKN and ChGPT, we were dead even in head-to-head comparison, suggesting that our approach struggles to exploit bad strategies.

For our fictitious play implementation, after going up for the first few iterations, our best response values against the opponent's correlated strategy very reliably and predictably decreased as the iteration count increased, as shown in Figure 4. This is the type of loss curve that a machine learning practitioner dreams of seeing. The reliably decreasing curve was very useful to us, as it confirmed that our algorithm was working properly. By fitting some power laws based on the most recent few hundreds of iterations, we were able to accurately predict where the best response values would end up after adding a given number of iterations to the run. The curves did not seem to have completely converged at 5900 iterations (where we stopped), but they were decreasing slowly enough that it probably would have taken infeasibly long to get to a much lower exploitability.

VI. Discussion

A. Main Takeaways

1) Technical Takeaways: From this project, we have a much better understanding of reach probabilities, and how they are computed and used for utility calculations for mixed strategies that are not behavioral strategies. Previously, we had just thought of reach probabilities as the product of action probabilities on the path from a root to a node, but this does not generalize well to the kind of correlated strategies we were working with in this project, which were not in behavioral form. We learned a lot about team correlated games, or games with imperfect recall, which we knew little about before. We feel confident that if we ever need to work on team correlated games in the future, we will have TB-DAG in our toolbox,

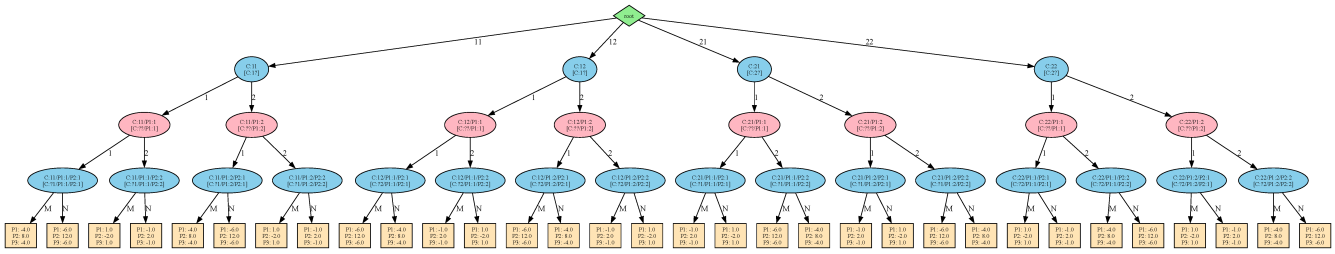
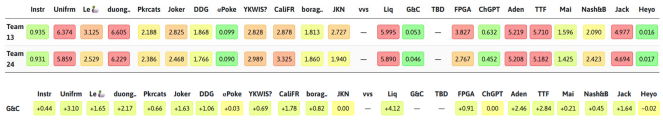


Fig. 2: Full expanded toy game tree. Infosets are shown in brackets beneath each node label.



	Instr	Unifrm	Le🍶	duong_	Pkrcats	Joker	DDG	ePoke	YKWIS?	CaliFR	borag_	JKN	vvs	Liq	GbC	TBD	FPGA	ChGPT	Aden	TTF	Mai	NashbB	Jack	Heyo
Team 13	0.935	6.374	3.125	6.605	2.188	2.825	1.868	0.099	2.828	2.878	1.813	2.727	–	5.995	0.053	–	3.827	0.632	5.219	5.710	1.596	2.090	4.977	0.016
Team 24	0.931	5.859	2.529	6.229	2.386	2.468	1.766	0.090	2.989	3.325	1.860	1.940	–	5.890	0.046	–	2.767	0.452	5.208	5.182	1.425	2.423	4.694	0.017

	Instr	Unifrm	Le🍶	duong_	Pkrcats	Joker	DDG	ePoke	YKWIS?	CaliFR	borag_	JKN	vvs	Liq	GbC	TBD	FPGA	ChGPT	Aden	TTF	Mai	NashbB	Jack	Heyo
GbC	+0.44	+3.10	+1.65	+2.17	+0.66	+1.63	+1.06	+0.03	+0.69	+1.78	+0.82	0.00	–	+4.12	–	–	+0.91	0.00	+2.46	+2.84	+0.21	+0.45	+1.64	−0.02

Fig. 3: Exploitability leaderboard of all teams.

already fully understood and ready to be applied. We also understand fictitious play a little better; at the very beginning, we thought that fictitious play was playing a best response against only the most recent pure strategy, not a mix of all the pure strategies so far, which resulted in very unstable behavior.

2) High Level Takeaways: After unsuccessfully sinking a lot of time into a heuristic strategy, only to find out all at once that it would not work, we realized the importance of having a strategy that we could test the correctness of, and in particular the value of strategies where you can observe incremental improvements or some other measurable progress. For example, the nice training exploitability curves of fictitious play were very refreshing to see. We expect the experience of working around limited compute resources to be a common one in research, so it was also good practice in having good estimates of how long training runs would take, and only starting training runs that could reasonably finish.

In general, during the course of this project, we realized that we could apply the idea of a correlated equilibrium to other team games that we play. For example, Chris and Alan frequently play the 2 vs 2 card game Tractor together, and have thought a little bit about how to correlate strategies for that game. We are also excited to think about correlated strategies for the card game Fish (also known as Literature).

B. Relation to Course Content

The project is similar to the coding exercise from problem set 2 in that we run a learning algorithm for computing low-exploitability strategies. However, in problem set 2, there are just two players and each player has perfect recall. In this project, we have multiple players playing together on teams, and we can view each team as a single player with imperfect recall. By working with a TB-DAG, we get around the problem of imperfect recall. As a result, many of our parsers and helper functions (such as best response and utility calculations) are similar to those from problem set 2, with the key differences being new team decision problem and TB-DAG classes.

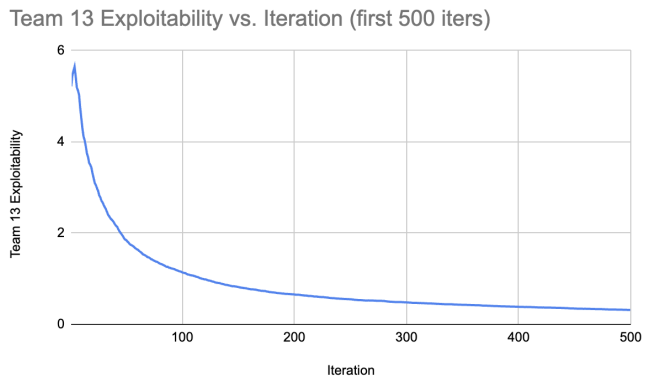
Fictitious play was suggested to us by Professor Farina, in addition to having been introduced in lecture 19.

C. Bottlenecks and Future Work

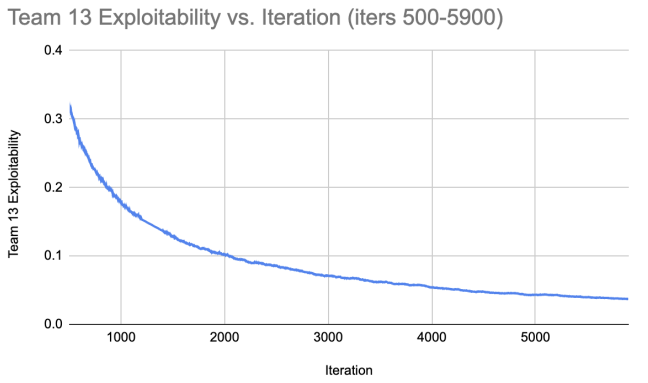
For most of this project after we figured out conceptually how to use TB-DAGs to compute best responses, the main technical challenges were:

1) **Compute constraints**: For compute, we were limited to one Apple M2 MacBook Pro. A single iteration through the game tree took roughly 10 seconds in wall clock time. To train our final strategy, we ended up using $5900 \cdot 2 = 11800$ iterations through the game tree, corresponding to $\frac{11800 \cdot 10}{3600} = 32.8$ real life hours of compute, so we needed to make sure that we had no wasted iterations. In addition, the laptop had to be open while it was computing, and it is prohibitively inconvenient to keep a laptop open for 32.8 consecutive hours.
2) **Testing:** Testing the different modules of our algorithm for correctness was difficult; on the Leduc game itself, the game and the number of infosets are too big to verify the best response strategies by hand, and to analyze and diagnose what types of strategies the algorithm was actually learning. Added on, we faced several nondeterminism issues in our training runs, even with set seeds. We knew that this would be a problem, since lack of testing was one of the main reasons our heuristic strategy failed, so we made sure to address this throughout.

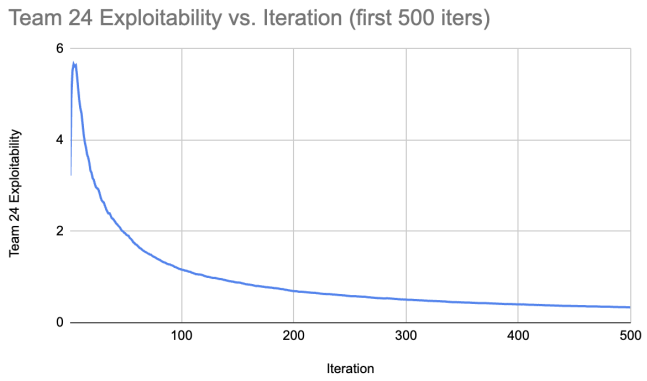
We had two major optimizations for making better use of our limited **compute**. Originally, in each iteration of fictitious play, we were recomputing the reach probabilities of the opponent's mixed strategy by iterating through all of their pure strategies and averaging their reach probabilities. Thus, if we were on iteration t, this took t iterations through a full game tree per team. If we run the algorithm for T iterations, the total compute thus scales as $O(T^2)$. This is prohibitively expensive; on the first fictitious play run where we implemented this, we cut the run off at around $T = 60$ because each successive iteration was just too slow. However, we quickly realized that in order to get the reach probabilities for a uniform mixed strategy of the first $t + 1$ strategies, we can reuse the reach



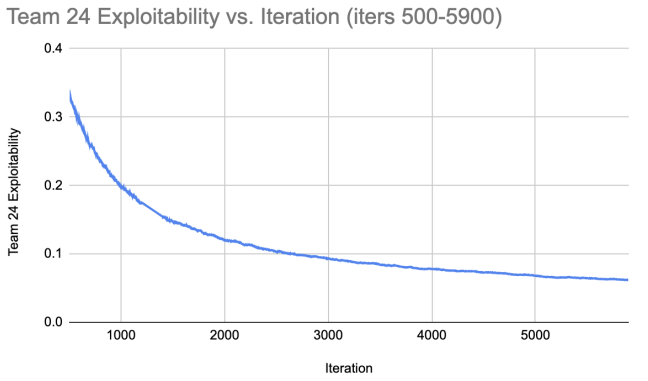
(a) Team 13 — First 500 iterations



(b) Team 13 — Last 5400 iterations



(c) Team 24 — First 500 iterations



(d) Team 24 — Last 5400 iterations

Fig. 4: Plots of the exploitability of the strategies for both teams 13 and 24 vs iteration number of fictitious play. The first 500 iterations and the remaining iterations are shown on separate graphs because they have very different y-axis scales.

probabilities of the first t strategies by using an analogous update to the reach probabilities as the update to the overall strategy in Equation 1. This reduces the time for computing reach probabilities in each iteration to just 1 pass through the tree per team. Our second major optimization had to do with the issue of how long we had to run the code for consecutively. We realized that if we continuously wrote the pure strategies to Numpy files (which we would have to do at the end of the day in order to submit the strategies), we would be able to restart a fictitious play run from where we left off, as that is the only persistent state kept on each iteration of fictitious play. However, this incurs an expensive $O(t)$ computation of the reach probabilities each time we start up the run – unless we **cache the reach probabilities**. After this optimization, all we had to do was run

```
python3 continue_fp.py <run_dir> -i N
```

and the code would start adding N more pure strategies inplace to our existing fictitious play run, unsupervised. In table III, we show how each of these improvements affected the number of passes through the game tree that we needed to take, assuming that we would need to pause the computation every 500 iterations on average.

Algorithm	Passes (general T)	Passes ($T = 5900$)	Hours
Naive recomputation (no optimizations)	$T(T+1)$	3.48×10^7	9.67×10^4
Per-step optimization, no caching	$\frac{T^2}{500} + 3T$	8.73×10^4	243
Per-step optimization, cached reaches	$2T$	1.18×10^4	32.8

TABLE III: Total passes through the game tree (counting both teams) and corresponding wall-clock hours, assuming each pass takes 10 seconds and restarting every 500 iterations.

One last interesting quirk of compute was that inexplicably, sometimes the wall clock time per pass through the tree would go up to roughly 20 seconds. We later discovered that this is because the MacBook Pro was on low power mode, so when it was plugged in it would be at normal speed, and when it was not, the CPU would work at half power.

For testing, we had very few ground truth verifiable answers. The toy problem that we discussed in Section IV-F was

very helpful for writing down verifiable test cases, but even though we made it a nontrivial game, it was still not fully generalizable to the full Leduc poker game. We found that we could use a few other sources of truth to test against. First, the exploitability score leaderboard allowed us to test our best response calculations against correlated strategies. Here, the heuristic correlated strategy that we came up with was actually useful, because it was a very early correlated strategy that we could find the best response value for. We also did some integration tests, such as testing that converting from a pure strategy to a pure plan and back from a pure plan to a pure strategy returns the same strategy. The biggest sign that our algorithm was implemented correctly was that the best response values actually started going down after about 10 iterations, and they very consistently, continuously went down, as discussed in section V.

When we implemented the optimizations involving reusing previous iterations' reach probabilities, we wanted to make sure that the optimized fictitious play runs were still correct, matching the best response values on each iteration of the unoptimized fictitious play runs. But here, we ran into some initially mysterious nondeterminism issues between runs that paused and resumed and runs from scratch. The issue was in floating point error and tiebreaking in best responses: because we multiplied constant factors repeatedly to previous reach probabilities, we had accumulated floating point drift, which caused terminal states that should have had equal effective payoffs to not have exactly equal effective payoffs. To fix this, we considered different actions as tied (allowing us to deterministically pick a seeded random action) as long as their floating point values were not too far apart, instead of being exactly equal.

In terms of performance of our algorithm however, compute was surprisingly not the real bottleneck. Our exploitability values were still decreasing, albeit incredibly slowly, as we approached our final submission. What ended up stopping us from going further was the **60 MB file size submission limit**. Early on, we were misinformed by another team that the strategies submitted could only contain up to 100 pure strategies. However, we discovered on accident that you can get a much smaller zip file by either manually zipping the strategies, or by programmatically using the **maximum compression factor** (which we found out that other teams, including the winning team, never discovered). Our final file ended up being 60.3 MB, which somehow passed through the submission file size limit. This let us submit 5800 more strategy files than every other competing team.

For future work along the path of fictitious play, the first change we want to try is to use different initializations. It is entirely possible that different initializations could have let us converge faster before running into the 60 MB file size limit. Another limitation of our current approach is that all of the strategies we submit are pure strategies, but the files actually allow for each individual strategy in the correlated plan to be a mixed behavioral strategy, which encodes more complex behavior in the same amount of memory. We could combine two pure strategies into one mixed strategy by mixing where they diverge in the TB-DAG, as before that point they encode the same pure strategy, and after that point, their reachable active nodes are distinct from each other. We would also be interested in different ways to use the TB-DAG to generate a team correlated equilibrium, besides fictitious play, which is known to not be a very efficient algorithm.

VII. Conclusion

We summarized our approach to the Team Leduc Hold'em challenge, a two-team four-player variant of poker that allows for correlated strategies between team members. We initially built heuristic strategies through repeated gameplay and probability foundations, then transitioned to constructing a TB-DAG. We also introduced a clear framework for using TB-DAGs to compute best response strategies on TDPs, and using fictitious play, achieved strong results in both exploitability and against opponent groups.

References

[1] B. H. Zhang, G. Farina, and T. Sandholm, "Team belief dag: Generalizing the sequence form to team games for fast computation of correlated team max-min equilibria via regret minimization," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

[2] G. Farina, A. Celli, N. Gatti, and T. Sandholm, "Ex ante coordination and collusion in zero-sum multi-player extensive-form games," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/c17028c9b6e0c5deaad29665d582284a-Paper.pdf

[3] ——, "Connecting optimal ex-ante collusion in teams to extensive-form correlation: Faster algorithms and positive complexity results," in *International Conference on Machine Learning*. PMLR, 2021, pp. 3164–3173.

[4] B. H. Zhang, G. Farina, A. Celli, and T. Sandholm, "Optimal correlated equilibria in general-sum extensive-form games: Fixed-parameter algorithms, hardness, and two-sided column-generation," in *ACM Conference on Economics and Computation (EC)*, 2022.

[5] Y. Zhang, B. An, and J. Černý, "Computing ex ante coordinated team-maxmin equilibria in zero-sum multiplayer extensive-form games," 2021. [Online]. Available: https://arxiv.org/abs/2009.12629

[6] B. H. Zhang and T. Sandholm, "Team correlated equilibria in zero-sum extensive-form games via tree decompositions," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[7] S. Ganzfried, "Empirical analysis of fictitious play for nash equilibrium computation in multiplayer games," 2024. [Online]. Available: https://arxiv.org/abs/2001.11165

[8] T. P. Schulze, "A generalized extensive-form fictitious play algorithm," *arXiv preprint arXiv:2310.09658*, 2023.

Appendix A
Proof of payoff in terms of realization vectors

We prove that if $x[z]$ is the realization form vector of one team's mixed strategy, $y[z]$ is the realization form vector of another team's mixed strategy, $c[z]$ is the reach probabilities of chance on the way to terminal node z, and $u[z]$ is the actual payoff of terminal node z for the team we are interested in, then the expected payoff for that team under these team strategies in the TDP is

$$\sum_{z \in \mathcal{Z}} x[z]y[z]c[z]u[z]$$

In the main text, we combine $y[z]c[z]$ into $w[z]$, the reach probabilities of the opponent's strategy and chance.

Observation: A selection of a pure strategy for our team, a pure strategy for the opponent's team, and a choice of actions for chance at each chance node fixes the terminal node that we end up at.

Why is this important? First, we can write the expected payoff as

$$\sum_{z \in \mathcal{Z}} \Pr \begin{bmatrix} \text{selecting a pure strategy for} \\ \text{my team, opponent, and chance} \\ \text{that deterministically lead to } z \end{bmatrix} u[z]$$

Secondly, it means that if we select a pure strategy for our team that **can** lead to z (for some choice of opponent and chance actions), and a pure strategy for the opponents that can lead to z, and a "strategy" for chance that can lead to z, then the profile of strategies **must** lead deterministically to z.

So actually, we can say that

$$\begin{aligned} &\Pr \begin{bmatrix} \text{selecting a pure strategy for} \\ \text{my team, opponent, and chance} \\ \text{that deterministically lead to } z \end{bmatrix} \\ =&\Pr \begin{bmatrix} \text{selecting a pure strategy for} \\ \text{my team, opponent, and chance} \\ \text{that each make } z \text{ possible} \end{bmatrix} \\ =&x[z]y[z]c[z] \end{aligned}$$

Where the last equality holds by definition of realization form vectors, and the independence of different teams' pure strategy selections. Thus, we can write the total payoff as

$$\sum_{z \in \mathcal{Z}} x[z]y[z]c[z]u[z]$$

Appendix B
Worked Example of Why TB-DAGs compute a TDP Best Response

In Figure 5, we give an example setup of a TDP and opponent/chance strategy. Our goal is to use the TB-DAG to compute a best response strategy on the TDP. We have set the payoffs $u[z]$ for each terminal node. By manual inspection, the optimal strategy always tries to end up at J or O, the only terminal nodes with positive payoff. Since E and G are part of the same infoset, we must take the same action from both: going left is optimal because the probability of being in node E is $\frac{1}{3}$, the probability of being in node G is $\frac{2}{3}$, and thus the expected payoff from going left is $\frac{1}{3} \cdot 3 = 1$ as opposed to $\frac{2}{3} \cdot 1 = \frac{2}{3}$ from going right. Already in this simple example, it is not obvious how to reason about the best response on the TDP, which motivates using a best-response solver on the TB-DAG to convert to a best-response on the TDP.

First, what are the reach probabilities of the opponent and chance, $w[z]$? By definition, it is the probability of the opponent/chance selecting a pure strategy that makes reaching z possible. The opponent's mixed strategy is with probability $\frac{1}{3}$, pick the pure strategy that picks left at the root node, and with probability $\frac{2}{3}$, pick the pure strategy that picks right at the root node. The former results in a realization vector $w[z]$ that is 1 for H, I, J, K and 0 for L, M, N, O, and the latter results in a realization vector that is flipped, so the convex combination is exactly the $w[z]$ in blue.

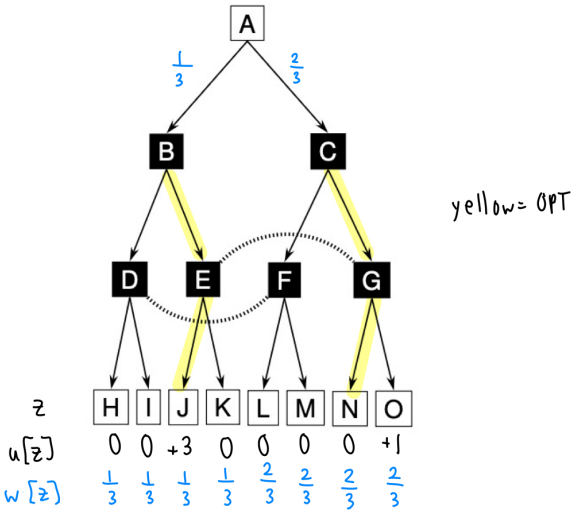


Fig. 5: Example TDP Setup for a best response. Blue indicates the opponent/chance's mixed strategy. z refers to the terminal nodes. $u[z]$ is the payoff from each terminal node. $w[z]$ is the realization form (or reach probabilities) of the opponent/chance's strategy. The yellow highlights are the actions that we can manually determine must be part of the best response. Our goal is to find this best response using a best-response finder on the TB-DAG. Figure borrowed from [1].

Then by Appendix A, the payoff for a strategy with realization form x is

$$\begin{aligned} \sum_{z \in \mathcal{Z}} x[z]w[z]u[z] &= x[J]w[J]u[J] + x[O]w[O]u[O] \\ &= x[J] + \frac{2}{3}x[O] \end{aligned}$$

Thus we are trying to find a strategy with realization form x that optimizes $x[J] + \frac{2}{3}x[O]$; call it x_{OPT}. Note that for pure strategies, this is not the same as just freely choosing $x[J]$ and $x[O]$ to be 0 or 1: because E and G belong to the same infoset and J is a left child and O is a right child, in a pure strategy, we must either have $x[J] = 1$ and $x[O] = 0$ or $x[J] = 0$ and $x[O] = 1$ (but not, for instance, $x[J] = x[O] = 1$).

Now **define** the payoff of a plan on the TB-DAG with realization form x' to be $x'[J] + \frac{2}{3}x'[O]$. Then our best-response algorithm on the TB-DAG will yield a pure plan σ' on the TB-DAG that attains the optimal realization form x'_{OPT}. But by strategic equivalence, $x'_{OPT} = x_{OPT}$! And we know of a way to convert a pure plan σ' on the TB-DAG to a pure strategy σ on the TDP that preserves the same realization form, so we end up with a best-response strategy σ on the TDP that achieves realization form x_{OPT}, as desired.