An Introduction to Counterfactual Regret Minimization

Todd W. Neller
Gettysburg College
Marc Lanctot
Maastricht University

Motivation

- 2000: Hart and Mas-Colell introduced regret matching algorithm
- 2007: Zinkevich et al. introduced counterfactual regret minimization (CFR)
  - dominant in computer poker competitions
- Perceived need:
  - introductory materials for experiential teaching of regret matching, CFR, and more advanced concepts
  - regret-based game-theory teaching that bypasses traditional path (e.g. dominated strategy elimination, simplex method)

Outline

- Regret
- Counterfactual Regret
- Assignment Handout Outline
- Conclusion

Rock-Paper-Scissors (RPS)

- Rock-Paper-Scissors (RPS)
  - 2 players, 3 possible simultaneous actions: rock (R), paper (P), scissors (S)
  - R, P, S beats S, R, P, respectively. Equal actions tie.
  - Win, tie, loss score +1, 0, -1, respectively

Regret

- Suppose you choose rock and your opponent chooses paper. Relative to your choice, how much do you regret not having chosen
  - paper?
  - scissors?
- Regret is the difference in utility between an action and your chosen action.
- Regrets: R→0 P→1 S→2

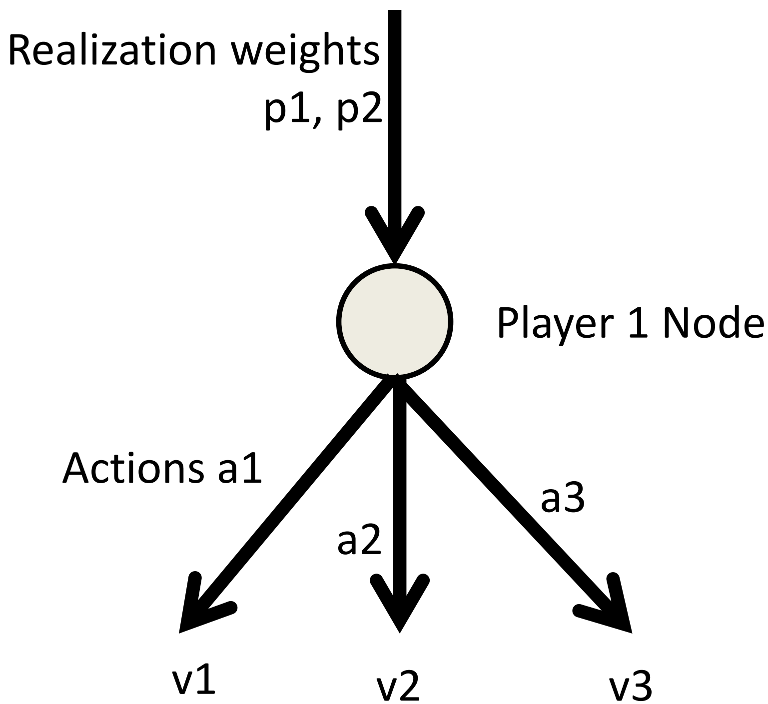
Regret Matching

- Choose an action with probability proportional to positive regrets.
- Regrets (0, 1, 2) normalized to probabilities: (0, 1/3, 2/3)
- Suppose we now choose S while our opponent chooses R.
  - Regrets: (1, 2, 0)
  - Cumulative regrets: (1, 3, 2)
  - Normalized cumulative regrets: (1/6, 3/6, 2/6)

Regret Minimization

- Regret Matching alone will not minimize regrets in the long run.
- However, the average strategy used over all iterations converges to a *correlated equilibrium.*
- In this example, average the strategies (1/3, 1/3, 1/3), (0, 1/3, 2/3), (1/6, 3/6, 2/6), etc.

Counterfactual Regret Example



- Input: realization weights

- Compute node strategy from normalized positive cumulative regret.
- Update avg. output strategy weighted by player realization weight.
- Recursively evaluate strategy to compute action values and node value.

- Compute counterfactual regret.
- Update cumulative regret weighted by opponent realization weight.

Counterfactual Regret Example

	p1	p2	
Realization Weights	0.5	0.25	
Player 1 Node:			
	a1	a2	a3
Cumulative Regret	20	-10	30
Positive Regret	20	0	30
Strategy	0.4	0	0.6
Cumulative Strategy +=	0.2	0	0.3
Player 1 Node Actions:	1	2	3
p1'	0.2	0	0.3
p2'	0.25	0.25	0.25
v1	40	-8	20
Node Value	28		
Action Regrets	12	-36	-8
Counterfactual Regrets	3	-9	-2
Old Cumulative Regret	20	-10	30
New Cumulative Regret	23	-19	28

- Input: realization weights
- Compute node strategy from normalized positive cumulative regret.
- Update avg. output strategy weighted by player realization weight.
- Recursively evaluate strategy to compute action values and node value.
- Compute counterfactual regret.
- Update cumulative regret weighted by opponent realization weight.

Materials Provided

- Starter example Java code explained in a 38 page PDF using Knuth's literate programming style presentation.
- Several tested programming exercises to facilitate experiential learning and deep mastery of material.

To select the actions chosen by the players, we compute the current, regret-matched strategy, and use it to select actions for each player. Because strategies can be mixed, using the same strategy does not imply selecting the same action.

⟨*Get regret-matched mixed-strategy actions*⟩≡

```
double[] strategy = getStrategy();
int myAction = getAction(strategy);
int otherAction = getAction(oppStrategy);
```

Next, we compute the utility of each possible action from the perspective of the player playing `myAction`:

⟨*Compute action utilities*⟩≡

```
actionUtility[otherAction] = 0;
actionUtility[otherAction == NUM_ACTIONS - 1 ? 0 : otherAction + 1] = 1;
actionUtility[otherAction == 0 ? NUM_ACTIONS - 1 : otherAction - 1] = -1;
```

Finally, for each action, we compute the regret, i.e. the difference between the action's expected utility and the utility of the action chosen, and we add it to our cumulative regrets.

⟨*Accumulate action regrets*⟩≡

```
for (int a = 0; a < NUM_ACTIONS; a++)
    regretSum[a] += actionUtility[a] - actionUtility[myAction];
```

For each individual iteration of our training, the regrets may be temporarily skewed in such a way that an important strategy in the mix has a negative regret sum and would never be chosen. Regret sums and thus individual iteration strategies are highly erratic[1]. What converges to a minimal regret strategy is the *average strategy* across all iterations. This is computed in a manner similar to `getStrategy` above, but without the need to be concerned with negative values.

⟨*Get average mixed strategy across all training iterations*⟩≡

```
public double[] getAverageStrategy() {
    double[] avgStrategy = new double[NUM_ACTIONS];
    double normalizingSum = 0;
    for (int a = 0; a < NUM_ACTIONS; a++)
        normalizingSum += strategySum[a];
    for (int a = 0; a < NUM_ACTIONS; a++)
        if (normalizingSum > 0)
            avgStrategy[a] = strategySum[a] / normalizingSum;
        else
            avgStrategy[a] = 1.0 / NUM_ACTIONS;
    return avgStrategy;
}
```

Materials Outline

- Regret Matching and Minimization
  - Worked example: RPS regret minimization versus fixed strategy
  - Exercise: RPS equilibrium, Colonel Blotto
- CFR
  - Worked example: Kuhn Poker equilibrium
  - Exercise: 1-die-versus-1-die Dudo
- “Cleaning” strategy results
- FSICFR
  - Worked example: Liar Die
  - Exercise: 1-die-versus-1-die Dudo with 3 claim memory limit
- Exploiting Opponent Mistakes
  - Exercise: Perturbed Liar Die
- Further Challenges (e.g. Minimum Unique Fingers)

Conclusion

- Regret minimization algorithms are an important part of the modern game theory landscape.
- These literate programming materials provide
  - an expedited, experiential introduction to the main concepts.
  - a starting point for many possible advanced undergraduate / graduate research projects.