Technical Report on HardChaosNegotiator Implementation

by: GPT-4, Ayan Sengupta

Introduction

This report details the implementation of the `HardChaosNegotiator` designed for alternating offer negotiation protocol. The negotiator integrates dynamic aspiration functions and opponent modeling to enhance negotiation strategies over time.

Dependencies and Setup

The implementation leverages several Python libraries:

- `scipy` for fitting curves to model opponent behavior dynamically.
- `negmas.sao` and `negmas` for negotiation mechanics and utility functions.
- `numpy` for numerical operations, particularly for the power function used in modeling aspiration and reservation values.

Aspiration Function

The core of the negotiation strategy is based on a customizable aspiration function defined as:

```
def aspiration_function(t, mx, rv, e):
    """A monotonically decreasing curve from mx (maximum) at t=0 to rv (reservation value) at t=1"""
    return (mx - rv) * (1.0 - np.power(t, e)) + rv
```

This function represents the negotiator's utility threshold, which decreases nonlinearly from an initial maximum utility `mx` to a reservation value `rv` as the negotiation progresses (represented by `t` going from 0 to 1). The parameter `e` controls the steepness of the curve, influencing how aggressively the negotiator decreases its expectations. Increasing the exponent makes the negotiator less willing to reduce their aspiration level quickly over time, holding out for better offers for longer.

HardChaosNegotiator Class

Initialization

The `HardChaosNegotiator` class is initialized with an exponent `e`, which is used to steepen the aspiration curve, making the negotiation strategy more aggressive:

```
def __init__(self, *args, e=8.0, **kwargs):
    super().__init__(*args, **kwargs)
    self.e = e
    self._rational = []
    self.last_offer = None
    self.opponent_times = []
    self.opponent_utilities = []
```

Opponent Modeling

The `update_opponent_model` method is used to track the opponent's utility values and corresponding times of offers, essential for dynamically adjusting the negotiator's reservation value:

```
def update_opponent_model(self, offer, relative_time):
    if offer is not None:
        self.opponent_utilities.append(float(self.opponent_ufun(offer)))
        self.opponent_times.append(relative_time)
```

Dynamic Reservation Value Adjustment

Using the `curve_fit` method from `scipy.optimize`, the negotiator fits a curve to the opponent's utilities to estimate their reservation value dynamically, adjusting the negotiator's strategy based on observed opponent behavior:

```
def update_reserved_value(self):
    if len(self.opponent_utilities) >= 3:
        try:
            optimal_vals, _ = curve_fit(
                lambda t, e, rv: (max(self.opponent_utilities) - rv) * (1.0 - np.power(t, e)) + rv*0.9,
                self.opponent_times,
                self.opponent_utilities,
                bounds=((0.5, min(self.opponent_utilities)*0.9), (8.0, max(self.opponent_utilities)))
            )
            self.opponent_ufun.reserved_value = optimal_vals[1]
        except Exception as e:
            pass
```

Offer Generation and Acceptance

The negotiator generates offers based on the current aspiration level plus the reservation value, and accepts offers that meet or exceed a slightly improved aspiration threshold:

```
def generate_offer(self, relative_time) -> tuple:
    aspiration_level = self.aspiration_function(relative_time) + self.ufun.reserved_value
    for outcome, my_util, opp_util in self._rational:
        if my_util >= aspiration_level:
            return outcome
    return self.ufun.best()

def is_acceptable(self, offer, relative_time) -> bool:
    best_outcome_utility = self.ufun(self.ufun.best())
    current_aspiration = self.aspiration_function(relative_time) * (best_outcome_utility - self.ufun.reserved_value) + self.ufun.res
    return self.ufun(offer) >= current_aspiration + 0.05
```

Conclusion

The `HardChaosNegotiator` utilizes dynamic strategies involving opponent modeling and nonlinear aspiration functions to adaptively negotiate in complex environments. This approach allows for a more robust and aggressive negotiation strategy, tailored to the evolving dynamics of opponent behavior and negotiation timelines.