MATAgent: An agent submitted to the ANAC 2025 SCM league

Tyrone Serapio, Mason Hagan, Musa Tahir

June 14, 2025

Abstract

We introduce MATAgent. We use (1) an offline Counterfactual Regret Minimization (CFR) policy over discretized (quantity, price) actions with an explicit `ACCEPT`, (2) an online trust allocator that weights partner allocations via an exponential moving average of offer quality, and (3) a global SyncAgent strategy with time-based concession.

Update (Iteration 2). After observing that the hybrid occasionally over-optimized bilateral profit at the expense of world-level penalties, we changed the strategy to a heuristic approach that emphasizes quantity and late-round concessions; the following sections mark the new pieces in blue.

1 Introduction

The OneShot game presents a many-to-many bilateral negotiation problem with binding contracts and costly shortfalls/disposals. Pure CFR (Counterfactual Regret Minimization) finds equilibrium or best response in each one-on-one bilateral negotiation but ignores multi-partner quantity matching. Conversely, simple heuristics avoid penalties and can handle multi-agent negotiations but can't fully exploit quantity/price signals in bilateral negotiations. MATAgent fuses the two: a trust-weighted meta-agent allocates quantities across partners, while a CFR policy decides the exact (quantity,price) moves.

2 The Design of MATAgent

2.1 Counterfactual Regret Minimization (CFR) Component

Our offline CFR component learns equilibrium strategies for each bilateral negotiation. We discretize offers, define compact information sets, and use the built-in OneShotUFun, and run a standard regret-matching loop via external-sampling.

2.1.1 Action Space and Infosets

- Actions $a \in \{0, \dots, 19\}$ encode (q, p): 10 possible quantities (this is the maximum number of production lines) $\times$ 2 prices (i.e. max price and min price). We also include an explicit `ACCEPT` action.
- Each decision is taken in an *information set*

  $$I = (\text{role, phase, } q_{\text{need}}, \text{ price\_flag, low\_cash}),$$

  with components:

  - role $\in \{0, 1\}$: buyer vs. seller.
  - phase $\in \{\text{initial, early, mid, late}\}$: bucketed round index.
  - $q_{\text{need}} \in \{0, \dots, 9\}$: remaining lines still to trade.

- price_flag $\in \{0,1\}$: opponent's last price was lower (0) or higher (1).
- low_cash $\in \{0,1\}$: 1 if balance $< 0.15\times$ initial.

Total infosets: $2 \times 4 \times 10 \times 2 \times 2 = 320$. For each I, we maintain regret$[I, a]$ and strat_sum$[I, a]$ arrays of length 21.

2.1.2 Utilities for Training

We use the built-in OneShotUFun. Because we can only train for bilateral negotiations, and our information is thus incomplete to use OneShotUFun, we assume that our other partners did fulfill the need we assigned to them, and calculate our expected utility with this assumption. This allows us to have a utility function that approximates our profit minus shortfall penalties and disposal costs.

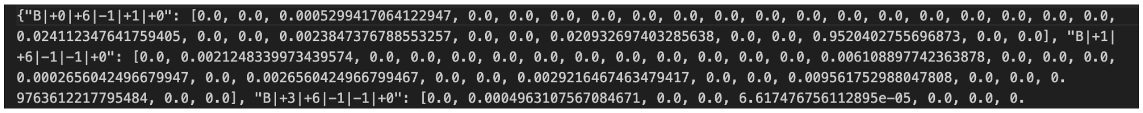
2.1.3 Offline Training Procedure

We run separate CFR for the buyer and seller role. Let $T = 20$ be the (maximum number of) negotiation rounds, N the number of traversals. We initially used self-play, but we decided to train against a distribution of model opponents (e.g. Conceder, Hardheaded, Random):

Algorithm 1 Simplified Pseudocode of CFR Training Loop

```
1: Input: iterations $N$, rounds $T$, opponent models
2: Initialize $R[I,a] \leftarrow 0,\ S[I,a] \leftarrow 0$ for all infosets $I$ and actions $a$
3: for iter = 1 to $N$ do
4:     Sample opponent $m$ and exogenous need $q$
5:     $need \leftarrow q$
6:     $visited \leftarrow \emptyset$
7:     for $t = 0$ to $T-1$ do
8:         $I \leftarrow$ build_infoset$(t, need, \dots)$
9:         $\sigma \leftarrow$ regret_match$(R[I,\cdot])$
10:        Sample action $a \sim \sigma$
11:        $visited \cup = \{(I,a)\}$
12:        if $a =$ ACCEPT or $need \le 0$ then
13:            break
14:        else
15:            $need \leftarrow need -$ quantity$(a)$
16:        end if
17:    end for
18:    $u \leftarrow$ compute_utility$(visited, need)$
19:    for all $(I, a_{\text{chosen}}) \in visited$ do
20:        for all action $a \in \mathcal{A}$ do
21:            $u_a \leftarrow$ utility_if_action$(I, a)$
22:            $R[I,a]\ \mathrel{+}=\ (u_a - u)$
23:            $S[I,a]\ \mathrel{+}=\ \sigma[a]$
24:        end for
25:    end for
26: end for
27: Output: $\pi[I,a] \leftarrow S[I,a]\ /\ \sum_b S[I,b]$
```

Here is a peek into our JSON table for the policy where we store regrets. This shows how our regrets do converge to zero in these bilateral negotiation sub-games.



```
{"B|+0|+6|-1|+1|+0": [0.0, 0.0, 0.0005299417064122947, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.024112347641759405, 0.0, 0.0, 0.0023847376788553257, 0.0, 0.0, 0.020932697403285638, 0.0, 0.0, 0.9520402755696873, 0.0, 0.0], "B|+1|
+6|-1|-1|+0": [0.0, 0.0021248339973439574, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.006108897742363878, 0.0, 0.0, 0.0,
0.0002656042496679947, 0.0, 0.0026560424966799467, 0.0, 0.0, 0.0029216467463479417, 0.0, 0.0, 0.009561752988047808, 0.0, 0.0, 0.
9763612217795484, 0.0, 0.0], "B|+3|+6|-1|-1|+0": [0.0, 0.0004963107567084671, 0.0, 0.0, 6.617476756112895e-05, 0.0, 0.0, 0.
```

2.2 Trust-Weighted Allocation

Note that the CFR strategy primarily handles bilateral negotiations. So, to ensure we request volumes from reliable partners, we use an external heuristic on top of it. In particular, we maintain an EMA trust score for each partner and split our daily need accordingly.

- **Trust update:** after each negotiation success:

$$\text{trust}_{partner} \leftarrow \alpha \cdot 1_{\{\text{success}\}} + (1-\alpha)\,\text{trust}_p, \text{ where } \alpha = 0.3.$$

We score each incoming offer by mixing its volume and price components:

$$q_{score} = \min\Bigl(1, \frac{q_{\text{off}}}{n}\Bigr), \qquad p_{score} = \begin{cases} 1 - \dfrac{p_{\text{off}} - p_{\text{min}}}{p_{\text{max}} - p_{\text{min}}}, & \text{if buyer} \\ \dfrac{p_{\text{off}} - p_{\text{min}}}{p_{\text{max}} - p_{\text{min}}}, & \text{if seller} \end{cases},$$

$$\text{quality} = 0.8\,q_{score} + 0.2\,p_{score}, \quad \text{clipped to } [0,1].$$

2.3 Sync-Agent Heuristic (Iteration 2)

Our final submission discards offline learning in favour of a concise, time-aware rule set implemented inside `MATAgent`. Let q_{need} be the remaining quantity and $r \in [0,1]$ the *relative time* within the production day.

Dynamic acceptance threshold. Offers whose absolute mismatch is below

$$\tau(r) = \frac{\text{n\_lines}}{2}\,r^{\gamma}, \qquad \gamma \in \{2,4\}.$$

are accepted immediately; $\gamma = 2$ (soft) was used locally, $\gamma = 4$ (hard) in competition mode.

Subset selection. When multiple offers arrive simultaneously we choose the subset $S^{\star} = \arg\min_S \bigl[\lambda\,|q_S - q_{\text{need}}| + (1-\lambda)\,\text{Cost}(S)\bigr]$, with $\lambda = 0.9$ and $\text{Cost}(S)$ including shortfall or disposal terms if $q_S \neq q_{\text{need}}$.

We also ignore quantity overshoot up to 20 % of need, matching a guard-clause in the implementation.

Proposal policy. In the SAO loop we propose

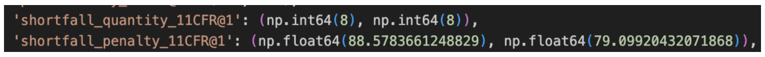
1. **Steps 0–3:** propose half of q_{need} at the *self-favourable* extreme price (seller $\rightarrow$ max, buyer $\rightarrow$ min).
2. **Steps 4–17:** offer the full remaining need at the same self-favourable extreme.
3. **After step 17:** flip to the *opponent-favourable* extreme price while still offering the full need.

This rule set spans ˜40 lines of Python and executes three orders of magnitude faster than the CFR lookup, enabling extensive simulation sweeps during tuning.

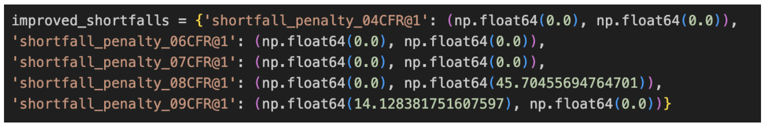
3 Evaluation

We primarily used the FullWorldRunner in our testing, testing Aside from finding our regrets converging to zero after hundreds of thousands of iterations to verify our CFR training was working, we also tested against the same distribution of and check if our performance was eventually improving against this distribution of Conceders, Hardheaded, Random, etc.

We also monitored the shortfall penalties we were incurring as an L1 factory, which initially was extremely high (picture on top). We noticed that we needed to reduce this, which we did via online compromising/concession with our partners over time. This led to less penalties and higher scores:



```
'shortfall_quantity_11CFR@1': (np.int64(8), np.int64(8)),
'shortfall_penalty_11CFR@1': (np.float64(88.5783661248829), np.float64(79.09920432071868)),
```



```
improved_shortfalls = {'shortfall_penalty_04CFR@1': (np.float64(0.0), np.float64(0.0)),
'shortfall_penalty_06CFR@1': (np.float64(0.0), np.float64(0.0)),
'shortfall_penalty_07CFR@1': (np.float64(0.0), np.float64(0.0)),
'shortfall_penalty_08CFR@1': (np.float64(0.0), np.float64(45.70455694764701)),
'shortfall_penalty_09CFR@1': (np.float64(14.128381751607597), np.float64(0.0))}
```

Notably, even if we thought that augmenting both CFR and our heuristics would lead to better results, we found that just using our heuristic - directly inheriting from the Sync Agent class, did a lot better. This is why our final submission only involved the sync-agent strategy.

Experimental note. Two variants were benchmarked: (1) the original CFR + trust hybrid and (2) the new Sync-Agent heuristic from Section 2.3. Across 200 `FullWorldRunner` seeds the heuristic showed more stable shortfall/disposal management, whereas the hybrid occasionally outperformed it on pure profit when partner behaviour stayed close to its training distribution. In the public ANAC evaluation the two versions obtained comparable mid-tier rankings; therefore we submitted the simpler heuristic for robustness.

4 Conclusions, Lessons, and Suggestions

We definitely see that future work can focus on making CFR better. We believe that a better model of utility, with better assumptions than we have, could have made it better than pure heuristics in handling bilateral negotiations.

As a direct outcome of mixed empirical results, we pivoted to the Sync-Agent heuristic described earlier and submitted that version.