MatchingPennies: An agent submitted to the ANAC 2024 SCM league

Arnie He, Akash Singirikonda, Amy Greenwald

April 17, 2024

1 Introduction

MatchingPennies employs concurrent negotiation and a heuristic function to identify the most profitable subsets of current offers. It aims to balance 'matching' quantities with negotiating prices for 'pennies'.

2 Agent Design

MatchingPennies follows a 2-step process:

1. **Concurrent Negotiation:** We select the best subset of offers to accept using a heuristic function and accept them at the offered price. For the remaining partners, with whom we did not reach an agreement in this round, we redistribute our needs as evenly as possible and re-propose offers based on the following rules:

    - For each remaining partner i, propose (p_i, q_i), where:

$$\sum q_i = \text{Needs}$$

$$p_i = \begin{cases} \text{self.BestPrice} & \text{if Needs} \leq 2 \\ \text{Partner proposed price} & \text{if Needs} > 2 \end{cases}$$

2. **Renegotiation:** In certain rare scenarios, we engage in complete renegotiation with our partners during this round, guided by specific conditions and strategies:

    - Let q represent the quantity offered by the current best bundle, Q denote current needs, and $1.5Q$(chosen constant) be the minimum quantity expected in the next concurrent round (a record of which is maintained throughout the simulation).

- If $q > \frac{Q}{2}$ and $q < Q$, and either the expected quantity in the next round is at least $1.2Q$ or the simulation day is less than 3 (to gather information on the record), we redistribute all offers as evenly as possible among current partners and request the best prices.

This strategy is based on empirical findings that the best bundle often nearly fulfills current needs, but a small remaining quantity can be challenging to secure in subsequent rounds. Complete renegotiation at this stage can address this issue, enhancing stability and overall performance.

2.1 Best Offer Subset In Concurrent Negotiation

MatchingPennies utilizes the `counter_all()` function for concurrency, strategically identifying and accepting the best subset of offers from agents still in negotiation. The optimal subset is determined by a heuristic function:

$$\text{normalized\_quantity\_diff} \times p - (1 - p) \times \text{normalized\_profit}$$

where:

- normalized_quantity_diff is the ratio of the quantity difference to the potential maximum difference, calculated as $\frac{\text{quantity\_diff}}{\text{max\_diff}}$. Here, quantity_diff represents the shortfall or excess relative to the contractual quantity requirement.

- normalized_profit scales the total profit relative to potential utility, defined by $\frac{\text{total\_profit}-\text{min\_utility}}{\text{max\_utility}-\text{min\_utility}}$.

- p is a hyperparameter that adjusts the balance between fulfilling quantity requirements and maximizing profit. It is defined as $1 - \frac{exogenous_quantity}{100}$, indicating the agent's risk preference towards contract compliance versus profit maximization.

This function effectively guides the decision-making process, balancing the need to meet contract stipulations with the drive for profitability.

2.2 Risk Management

Our strategy for prioritizing either quantity matching or price negotiation is influenced by the exogenous contracted quantity. As the required quantity increases, our focus shifts towards optimizing for price, referred to as the "pennies" aspect. Conversely, with lower quantity demands, we prioritize the "matching" aspect, aligning our quantities closely with those demanded.

In rare scenarios where the best bundle offered nearly satisfies our current needs but leaves a small yet critical shortfall, we engage in complete renegotiation. As mentioned before, this strategy of complete renegotiation is designed to address difficulties in securing minor quantities in subsequent rounds, thereby enhancing the stability and overall performance of our negotiation process.

3 Evaluation

We tested MatchingPennies by running it against winning agents from previous years under multiple configurations. Specifically, we tested against SyncRandomOneShotAgent, QuantityOrientedAgent, PatientAgent, and GreedySyncAgent.

Here's the mean performance of the agents over 5 trials with 10 steps and 2 configurations.

Agent	Mean Performance (5 games)
MatchingPennies	1.087856
SyncRandomOneShotAgent	1.059264
QuantityOrientedAgent	1.0327694
PatientAgent	1.022377
GreedySyncAgent	0.7784422

4 Lessons and Conclusions

Initially we aimed for an RL agent for the competition, however as we advance in the development we feel it's unnecessary for having an RL strategy in such a simple game setup(We feel that RL approaches could be more impactful in tracks that involve standard or collusion elements). Our strategy rely heavily on empirical data gathered from local competitions and estimates of other participants' strategies. We believe that attention to detail is crucial in the oneshot game format. Among agents that prioritize quantity, our goal is to stand out by maximizing profits through strategic price negotiations.

5 Pseudocode

Algorithm 1 Best Subset Selection

```
1: $best\_total\_loss \leftarrow \infty$
2: $best\_quantity\_diff \leftarrow \infty$
3: $best\_index \leftarrow -1$
4: for each $i$ and $partner\_ids$ in $plist$ do
5:     $offered\_quantity \leftarrow \text{sum}(offers[p][QUANTITY]$ for $p$ in $partner\_ids)$
6:     $quantity\_diff \leftarrow |\text{offered\_quantity} - \text{needs}|$
7:     $penalties \leftarrow \text{CalculatePenalties}(\text{quantity\_diff}, \text{awi.level}, \text{needs})$
8:     if offered_quantity $> \text{needs} + 1$ then
9:         continue
10:    end if
11:    $total\_profit \leftarrow \text{total\_contracts\_cost} - \text{penalties}$
12:    $(\text{normalized\_diff}, \text{normalized\_profit}) \leftarrow \text{Normalize}(\text{quantity\_diff}, \text{total\_profit})$
13:    $t \leftarrow$quantity_cost_tradeoff
14:    $loss \leftarrow \text{t} \times \text{normalized\_diff} - (1 - \text{t}) \times \text{normalized\_profit}$
15:    if loss $<$ best_total_loss then
16:        $best\_quantity\_diff \leftarrow$ quantity_diff
17:        $best\_index \leftarrow i$
18:        $best\_total\_loss \leftarrow$ loss
19:    end if
20: end for
21: return (best_quantity_diff, best_index)
```