MatchingPennies: An agent submitted to the ANAC 2024 SCM league

Arnie He, Akash Singirikonda, Amy Greenwald

April 15, 2024

Abstract

1 Introduction

2 The Design of MatchingPennies

The design of MatchingPennies is simple, it only has two significant design choices: Concurrent negotiations and calibrated loss function.

2.1 Concurrent Negotiation

MatchingPennies is based on the template of OneShotSyncAgent, for we believe that centrally handling all existing negotiations at the same time will have a huge profit over deciding for each response without a wholistic consideration. The central idea for each *counter_all*() function is to find the best combinations so far, accept them, and then determine responses for the other partners dealing with the quantity we still need. The center of design then lies naturally in the function that searches for the best combination of offers. In short, we define the standard of the value of a bundle to be $quantity\_diff * p - (1 - p) * profit$, where quantity_diff is the gap between the quantity we need and the quantity provided by the bundle. P is a constant that could be toggled, and in the next section we'll talk about the toggling of p.

2.2 Risk Management

The contexts of world are not the same even though they're all one-shot. In some of the worlds the agent should try to maximize its profit in each negotiations, (somewhat equivalent to maximize prices), while in other negotiations the agent might seek to maximize the chance of matching the quantity. (This happens more often because the current awi world has a really small fluctuations in the price range, and despite the times that we want to profit over the dumby bots, we should seek to match the quantity). However, to determine the types of our opponents is both complicated and resources consuming, so we propose another strategy, that we toggle the constant p in our loss function based on the exogenous contracts we need for the day. In short, the more quantity we need, the more oriented towards the 'pennies' we be(maximize for the value); and the fewer quantity we need we maximize for the 'Matching'. That's also why we call the agent the 'MatchingPennies'.

3 Evaluation

4 Lessons and Suggestions

Conclusions

Pseudocode

Algorithm 1 Best Subset Selection

Initialize $best\_total\_los \leftarrow \infty$
Initialize $best\_quantity\_diff \leftarrow \infty$, $best\_indx \leftarrow -1$
for each index i and $partner\_ids$ in $plist$ **do**
    $offered\_quantity \leftarrow \sum_{p \in partner\_ids} offers[p]['\text{QUANTITY}']$
    $diff \leftarrow |offered\_quantity - needs|$
    $total\_contracts\_cost \leftarrow \sum_{p \in partner\_ids} offers[p]['\text{UNIT\_PRICE}'] \times offers[p]['\text{QUANTITY}']$
    $penalty \leftarrow 0$
    if $awi.level = 0$ **then**
        if $offered\_quantity < needs$ **then**
            $penalty \leftarrow penalty + diff \times awi.current\_disposal\_cost$
        else
            $penalty \leftarrow penalty + diff \times awi.current\_shortfall\_penalty$
        end if
    else
        if $offered\_quantity < needs$ **then**
            $penalty \leftarrow penalty + diff \times awi.current\_shortfall\_penalty$
        end if
        if $offered\_quantity > needs$ **then**
            $penalty \leftarrow penalty + diff \times awi.current\_disposal\_cost$
        end if
    end if
    if $offered\_quantity > needs + 1$ **then**
        Continue to next iteration
    end if
    Compute $total\_profit$
    Normalize $total\_profit$ and $diff$
    $loss \leftarrow quantity\_cost\_tradeoff \times diff\_normalized - (1 - quantity\_cost\_tradeoff) \times total\_profit\_normalized$
    if $loss < best\_total\_los$ **then**
        $best\_quantity\_diff \leftarrow diff$, $best\_indx \leftarrow i$, $best\_total\_los \leftarrow loss$
    end if
end for
return $best\_quantity\_diff$, $best\_indx$