PDPSAgent: OneShot track agent submitted to the ANAC 2021 SCM league

David Krongauz and Hadar Ben-Efraim

Department of Computer Science, Bar-Ilan University, Israel

July 2021

Abstract

We present a trading agent dedicated for the OneShot track. Our agent's strategy combines learning, for predicting future market conditions, and dynamical programming for optimal selection of concurrent offers. We show that our agent gains the highest score among the builtin OneShot agents.

1 Introduction

This year, OneShot, a new simulation track in the SCM environment was introduced. In this track an agent's inventory and balance play a much smaller role, as they are reset at every step of the simulation. Such settings gives more emphasis on the negotiation part of the agent since production strategies and other components, present in the 'Standard' and 'Collusion" tracks, are irrelevant. Thus, we chose to develop the negotiation strategy of the agent based on two key components of the OneShot simulated world:

- Exogenous contracts
- Concurrency of negotiations

both aspects are treated in our agent's design.

Utilizing the fact that the trading prices are public information this year, we use learning algorithms to predict future prices. The concurrency aspect is exploited by improving the *GreedySyncAgent* via dynamical programming approach. Combining the two elements together we get a 'Predicting-Dynamical-Programming-Synchronous' agent, or in short: PDPS agent.

2 The Design of PDPSAgent

2.1 Negotiation Choices

Relying on IDEA 11 from the SCML tutorial [2] we decided to try and predict future market conditions, in particular, the future exogenous contract prices based on past information. Since trading prices are public information this year we are be able to extrapolate future market conditions and use this to adapt the agent's behavior. Even though our profits in every step are independent from the future, our concession strategy may depend on our expectation about the future of the market.

We used linear regression algorithm in order to predict future contract prices. If the exogenous price will be greater, and we are on the buying side of the exogenous contract, then we would want to be more flexible in our negotiations. This is implemented by lowering the threshold parameter. On the other hand, if we are a selling to the exogenous buyer, and we predict we will get a higher price tomorrow, we would rather be stiffer in our negotiations today, and vice versa.

The learning was implemented via sklearn.py package [3]. The features that we used are exogenous price of yesterday, exogenous price of today and the day number. the labels were 0 - for a prediction of a greater (or even) exogenous price for tomorrow and 1 for smaller. we reached 93% of accuracy with this algorithm.

2.2 Utility Function

We used the default builtin utility function provided by the SCML environment.

2.3 Concurrent Negotiation

As described in the tutorial [2] the utility function is defined for a complete set of negotiation agreements and not for any single negotiation by itself. Therefore, it makes sense to try to make decisions centrally by collecting offers from partners then responding to all of them at once. This way one can think about all negotiations 'together'. Knowing this advantage we decided to build our agent upon the *GreedySyncAgent* class.

All the concurrent negotiations, each with its quantity and unit price, can be represented an the widely-known *Knapsack* problem [1]. In this case, quantity represents an item's weight, unit price times quantity represents an item's value, and an agent's needs represents knapsack's maximal weight. The greedy agent solves it naively by sorting the items by their values and then accepting all offers until its needs are satisfied or surpassed. In the later case the last offer is rejected, which in turn causes agent's needs to be not fully satisfied.

We propose solving the knapsack problem less naively, by employing DP methods. The original problem is subdivided into sub-problems where in each case the optimal

3 Evaluation

We ran our agent against the following builtin agents: *GreedySyncAgent, RandomOneShotAgent, OneshotDoNothingAgent, SyncRandomOneShotAgent*. Our agent, denoted in Fig. 1 as *DPSyncAgent*, got the highest score among the agents he was simulated against.

```
040 of 040 [1e+02%] completed in  6m:37s [ETA  6m:37s]
Tournament completed successfully
Compiling results from individual world runs
Read: C:\Users\cokron\negmas\tournaments\20210701H193320032077uf0-stage-0001\scores.csv
Calculating Scores
Running statistical tests
Winners: [('scml.oneshot.sysagents.DefaultOneShotAdapter:__mp_main__.DPSyncAgent', 1.1695824195907338)]
Saving results
N. scores = 80  N. Worlds = 40
+----+-----------------------------------------------------------------------------------------------+----------+
|    | agent_type                                                                                    |    score |
|----+-----------------------------------------------------------------------------------------------+----------|
|  0 | scml.oneshot.sysagents.DefaultOneShotAdapter:__mp_main__.DPSyncAgent                          |  1.16958 |
|  1 | scml.oneshot.sysagents.DefaultOneShotAdapter:scml.oneshot.agents.greedy.GreedySyncAgent       |  1.12327 |
|  2 | scml.oneshot.sysagents.DefaultOneShotAdapter:scml.oneshot.agents.random.RandomOneShotAgent    |  0.778433 |
|  3 | scml.oneshot.sysagents.DefaultOneShotAdapter:scml.oneshot.agents.random.SyncRandomOneShotAgent |  0.777628 |
|  4 | scml.oneshot.sysagents.DefaultOneShotAdapter:scml.oneshot.agents.nothing.OneshotDoNothingAgent |  0.729641 |
+----+-----------------------------------------------------------------------------------------------+----------+
```

Figure 1: Scores of tournament result. Our agent is denoted as *DPSyncAgent*

4 Lessons and Suggestions

In order to improve our agent, our suggestion is to improve the linear regression model. we can add more features, search hyper parameters that will maximize the accuracy etc. In addition, we can implement our own ufun.

Conclusions

Looking at the results shown in Section 3, we have managed to create an agent that wins over all of the other builtin OneShot agents. We have used both machine learning technique to maximize our knowledge about the world, and dynamic programming for choosing which offers to accept among the concurrent negotiations. This allowed our agent to gain better results, in particular against the *GreedySyncAgent*.

References

[1] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. USA: John Wiley & Sons, Inc., 1990. ISBN: 0471924202.

[2] Dr. Yasser Mohammad. *Developing an agent for SCML2021 (OneShot)*. 2021. URL: `http://www.yasserm.com/scml/`.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.