ASMASH

Assaf Marsha, Matanya Freiman, Shmuel Amar
Bar-Ilan University, Israel
assafmarsha@gmail.com, mfreiman97@gmail.com, shmulikamar@gmail.com

July 13, 2020

1 Introduction

For this year's competition our team tried to present an improved version of the *DecentralizingAgent*. The *DecentralizingAgent* composed of the following configuration:

- *PredictionBasedTradingStrategy* - predict inputs/outputs needed assuming a fixed execution rate that does not change for all partners
- *SupplyDrivenProductionStrategy* - A production strategy that converts all inputs to outputs assuming that there was no excessive purchasing of inputs
- *StepNegotiationManager* - Two negotiators for each step, one for buying and another for selling.

To improve the existing agent we analyzed the various agent components. Analyzing the behavior of the Step Negotiation Manager motivated us to look in to several possible improvements. In the start of the competition the Negotiation Manager is generating a requests of negotiations for the next "horizon" steps as we can see in figure 1.

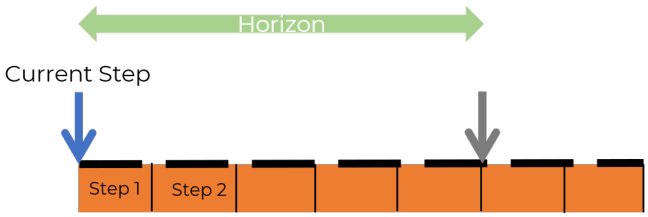


Figure 1: Requests for negotiations in the beginning of the competition

Each another step, the Negotiation manager is generating negotiation to the next $CurrentStep + horizon + 1$ step.
When *StartNegotiations* function From the *StepNegotiationManager* is called a controller is created and get a step as an input. For this given step, the controller is trying to negotiate with the active partners to achieve the given target according to the utility function. That means that the controller that responsible to find the best partner for contract, get as an input the partners that "relevant" for negotiations *horizon* steps before. This way, we actually allow signing a contract with partners that declared bankruptcy few steps before. Furthermore, we are highly wanted to prevent to arrive to situation of *"compensated contracts"*[1].

[1]The contracts of a partners that transferred to the system for compensation of the agents, because of the bankruptcy of the partner. Some of those contracts is partially executed and the rests is nullified.(The contracts honored by their delivery dates)

In this paper, we are proposing our solution Called **ASMASH**(The first chars of our names, **As**saf, **Ma**tanya and **Sh**muel). We used the usual Decentralizing agent and added an improvement trying to minimize the period of time between the agent declared bankruptcy and the last step the agent considered as a legitimate partner for negotiation. In this paper we are described our solution and the algorithm we developed.

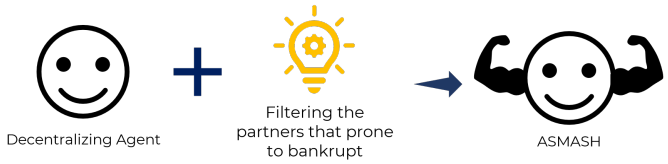
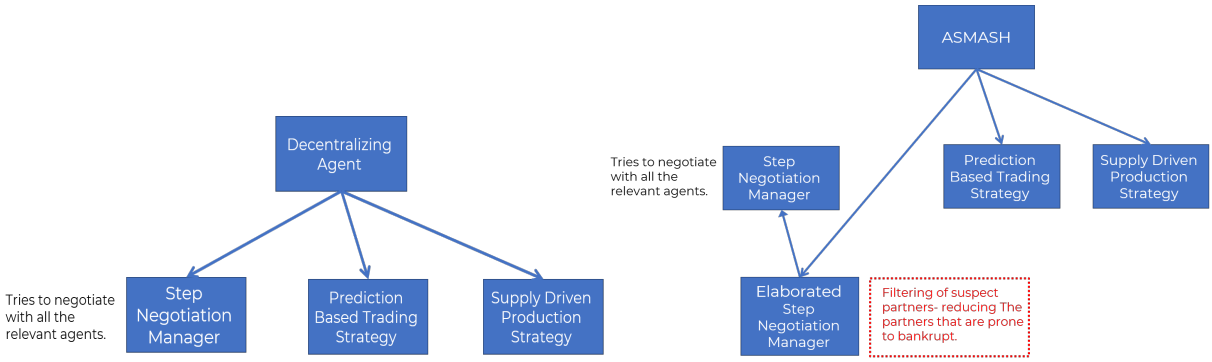


Figure 2: The Idea

2 The Design of ASMASH

As described earlier, ASMASH is actually based on a decentralization agent but the difference is in the motivation to reduce the legitimacy of the partner after declaring bankruptcy. Note that we actually want to get partners to declare bankruptcy by negotiating with them. But, we do not want to negotiate and engage with these partners after the bankruptcy declaration.



(a) Decentralizing Agent Design (b) ASMASH design

Figure 3: The Designs

We strive to minimize the length of time between the partner declaring bankruptcy and the last step that this partner is considered as a legitimate partner in negotiations.

As showed in figure 3b, By creating the class *Elaborated Step Negotiation Manager* we successfully filter out those suspicious partners.

2.1 Negotiation Choices

For choice of what to negotiate about we used Mean Execution Rate Prediction Strategy trying to predict the inputs/outputs needed. As a negation partner we choose to approach agents that obey the following roll: Agent must have less then *thresh* number of breaches in the last N steps.

We developed **F**inding **S**uspect **P**arters (**FSP**) algorithm to find the partners that are likely to declare bankruptcy. We used FSP to filter out those suspect agents which we didn't want to negotiate with in the future. We saw empirically that agents that made few breaches over a small period of steps are expected to bankrupt- We wanted to find those agents and don't start a negotiation with them.

Algorithm 1: FSP - Finding Suspect Partners Algorithm

1: $S \Leftarrow \emptyset$
2: $B \Leftarrow$ N last steps breaches(N)
3: **for** $(perpetrator, step)$ in B **do**
4: **if** Count($perpetrator$) >= $thresh$ **then**
5: $S \Leftarrow S \cup perpetrator$
6: **end if**
7: **end for**
8: RETURN S

FSP algorithm begins by finding the breaches that had been occurred in the last N steps. The partner that made more than *thresh* breaches during the last N steps considered as a suspected partner. We saw empirically that choosing *thresh=3* and *N=5* is effective choices[2]. The algorithm returns all the suspected partners (lines 2-8)

2.2 Utility Function

Similar to Decentralizing agent, ASMASH uses a Linear utility function for multi-issue negotiations.

$$u = \sum_{i=0}^{n_{outcomes}-1} w_i \cdot e_i$$

e_i- The value of the issue i
w_i- The weight of the issue i

In the decentralizing agent documentation the utilities functions were defined for each controller (for each step):

For seller controller:

- **Issues**
  e_1: Quantity- Range of quantity-[1,quantity]
  e_2: Time- Ranges of times - [step of the controller,step of the controller + horizon]
  e_3: Range of prices-[catalog price, (Catalog Price)·2]
- **Weights**
  w_1: 1.
  w_2: 1.
  w_3: 10.

For buyer controller:

- **Issues**
  e_1: Quantity- Range of quantity - [1,quantity]
  e_2: Time- Ranges of times - [Current step +1 , step of the controller -1]
  e_3: Range of prices-[1, Catalog Price]
- **Weights**
  w_1: 1.
  w_2: -1.
  w_3: -10.

[2]The effectiveness of the choice *N=5* is interesting since *horizon* is also equal to 5

3 Evaluation

As described earlier in this paper, in contrast to the Decentralizing agent, ASMASH helps to prevent starting negotiations with partners that are expected to declared bankruptcy. ASMASH is trying to minimize the period between the partner declared bankruptcy and the last time we are seeing the partner as a legitimate to negotiate with.

In figure 4 , we are describing a common scenario of some partner. we assume that the partner made more than 3 breaches between flag 1 and flag 2.

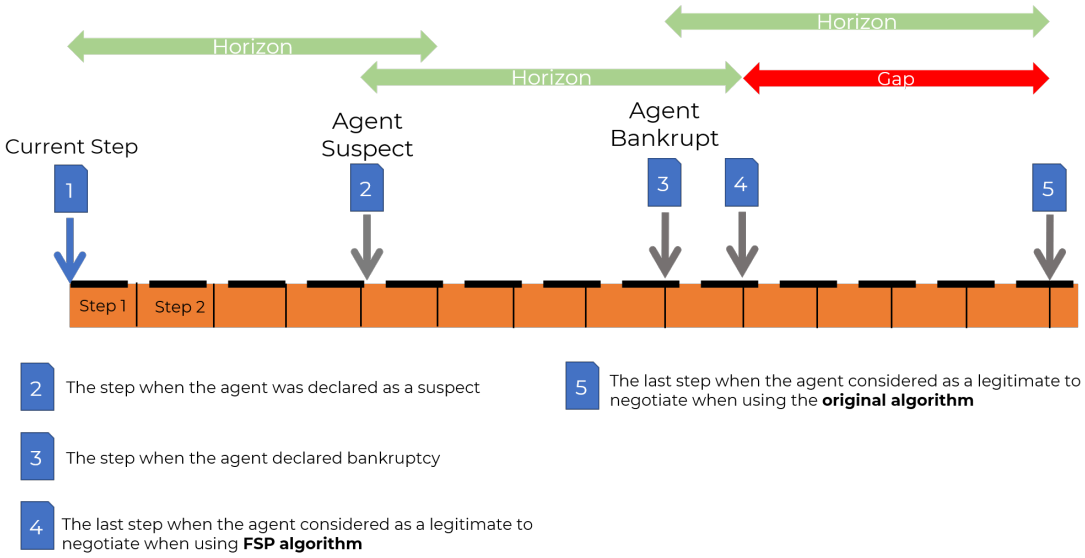


Figure 4: The Scenario

In this scenario, if we used decentralizing agent, our agent will stop to negotiate with this partner only after he will declare bankruptcy + horizon time. This means that if the partner has declared bankruptcy in flag 3, we are still negotiating with him until flag 5 and the contracts already signed in this period changed to be a *compensated contracts*.

However, while using ASMASH, as a result of the sequence of the breaches for which the agent was responsible, he was tagged as a bankruptcy suspect (Flag 2). This means that there has been no negotiation with this partner since Flag 4. Thus, using ASMASH, we made the time between the partner's bankruptcy and the last step the agent was considered legitimate to negotiating, less compared to the situation where the Decentralizing agent was used. The "gap" in figure 4 is reflecting the time between the non legitimacy of the partner by using ASMASH in comparison to the time when using Decentralizing agent used.

Conclusions

In this paper we introduce ASMASH, an agent based primarily on decentralized agent capabilities. We have developed an algorithm that minimize the length of time between a partner's bankruptcy and the final stage in the legitimacy of this negotiating partner. Moreover, the algorithm significantly reduces the number of *compensated contracts*. We have tried to resolve crucial flaws in the agent and by doing so we have created an improved version of the decentralization agent, resolving crucial flaws of the agent.