CMPE 593 - FALL 2017

Report - Project III

MOVIE NEGOTIATOR

Boğaziçi University
2016800003 & 2016800021
Gönül AYCI & Yavuz KÖROĞLU

Instructor: Prof. Dr. Pınar YOLUM

December 7, 2017

1 INTRODUCTION

We propose Pareto Optimal Movie Negotiator (POMN), an agent which performs the Movie Negotiation Game (MNG) for not just two, but also three users. Unless the given MNG is a zero-sum game, POMN always returns a *Pareto efficient* deal. We implement POMN in Perl since visualization of negotiation graphs is convenient in Perl. Our implementation is available at `https://github.com/aycignl/ConferenceHelperBDIAgent/tree/master/movieNegotiator`.

Alternating offers is a known protocol for bilateral negotiations. In this study, we propose two generalizations of *alternating offers* protocol, Gonul Protocol (GP) and Yavuz Protocol (YP) so that these protocols support three negotiators.

First, we discuss how we define an MNG in Section 2. Then, we discuss our proposed protocols in Section 3. We inevitably make use of some simplifying assumptions during the development of POMN and describe these assumptions in Section 4. We show how POMN works by showing nine example cases and make some comments on the results of these examples in Section 5. We compare our protocols by conducting experiments on the FilmTrust dataset in Section 6. Finally, we conclude by giving a summary and future work in Section 7.

2 MOVIE NEGOTIATION GAME (MNG)

```
Gonul,Yavuz,Ali
3,3,4
5,9,3
1,7,2
```

Figure 2.1: Example5.csv: A Preferences Table

In a Movie Negotiation Game (MNG), we have N users and each user has M preferred movies. We get these movies as a table of preference lists from a *.csv* file. Figure 2.1 shows an example *.csv* file. In this file, the number of columns indicates the number of users which is 3 for this example. Each column is a preference list for its corresponding user. A *preference list* of a user is a total ordering of all preferred movies by the user. *Preferred movies* of a user are movies s.t. the utility of all movies in the list is larger than zero for the user. Since a preference list is a total order, each movie in the list must have a larger utility than all subsequent movies in the list. We denote each movie by a unique positive integer, $movieId \in \mathbb{Z}^+$. In Figure 2.1, the first preference of both Gonul and Yavuz is the movie 3, whereas this movie is the second preference of Ali.

First, we convert a preferences table into a circular list of users and preference lists for each user in Algorithm 1. A *circular list* is a list in which the first element comes after the last element. We start with an empty list of users and append all users to this list one by one. For every user, we start with an empty preference list and append all preferred movies to this list, again one by one.

Second, we convert preference lists to utility functions using Algorithm 2. A utility function of a user is a function from a positive integer *movieId* to a natural number *utility*, $U_{\text{user}} : \mathbb{Z}^+ \rightarrow \mathbb{N}$. This function takes a *movieId* and returns an integer denoting the amount of pleasure that the user gets by watching the movie *movieId*. We initially start with a utility function which gives zero utility to all movies. Then we assign unique utilities for each movie in the user's preference list.

Third, we define a user as a four tuple, user = (U, P, S, R), denoting the *utility function*, the *preference list*, the *state* of the user, and the set of *received offers*, respectively. A state of a user is defined as $S_{\text{user}} \in \{ACCEPT, BREAK, CONTINUE\}$. Initial state of every user is the *CONTINUE* state. The set of

Algorithm 1: The First Convertion Algorithm

Inputs

`pTable`: A preferences table

Output

`userList`: A circular list of users

$\mathtt{P}_{\mathtt{user}}$: A mapping of users to their preference lists

```
1: userList ← []
2: for all column ∈ pTable do
3:    user ← pop the first element of column
4:    Append user to userList
5:    P_user ← []
6:    for all movieId ∈ column do
7:       Append movieId to P_user
8:    end for
9: end for
```

Algorithm 2: The Second Convertion Algorithm

Inputs

$\mathtt{P}_{\mathtt{user}}$: A preferences list

`M`: Number of preferred movies

Output

$\mathtt{U}_{\mathtt{user}}$: A utility function, $U_{\mathtt{user}} : \mathbb{Z}^+ \rightarrow \mathbb{N}$

```
1: U_user(movieId) ← 0, ∀movieId
2: i ← M
3: for all movieId ∈ P_user do
4:    U_user(movieId) ← i
5:    i ← i - 1
6: end for
```

received offers of a user denotes the *movieId*s that were offered to the user in the previous turn. Initially, $R_{\mathtt{user}} = \emptyset$ for all users.

Each user has a *conflict deal* utility denoted as U^0, which defines the utility that the user will get if no deals can be made. Every user starts with $U^0 = 0$.

In an MNG, every user offers movies according to their preference lists, P, and their set of received offers, R. At each turn, the user who has the turn offers his/her the most preferred movie and *crosses out* the movie from his/her preference list if the most preferred movie comes from the preference list. We call this the *standard strategy* of playing an MNG.

We define the logical rules of an MNG in Table 2.1. Note that these rules are defined based on the *standard strategy* and won't work if the user never crosses out some movies from his/her preference list. These rules apply for any MNG at all times.

We define an *offer* as a triple (u_1, u_2, m), where the first user u_1 offers the movie m to the user u_2. We keep a history of these offers and count the number of offers to approximate the communication complexity of MNG under different examples and protocols.

We define a *deal* as the final result of the game. A *deal* is a pair (H, m), where H denotes the set of

Table 2.1: Logical Rules of a Movie Negotiation Game

Name		Rule	Modifiers
$S_{\texttt{user}} = BREAK$	$\leftrightarrow$	$U_{\texttt{user}}(\texttt{m}) \leq U^0_{\texttt{user}}$	$\forall \texttt{m} \in R_{\texttt{user}} \cup P_{\texttt{user}}$
$S_{\texttt{user}} = ACCEPT$	$\leftrightarrow$	$U_{\texttt{user}}(\texttt{m1}) \geq U_{\texttt{user}}(\texttt{m2})$	$\exists \texttt{m1} \in R_{\texttt{user}}, \forall \texttt{m2} \in P_{\texttt{user}}$
$S_{\texttt{user}} = CONTINUE$	$\leftrightarrow$	$S_{\texttt{user}} \neq ACCEPT \wedge S_{\texttt{user}} \neq BREAK$	
`Consensus`	$\leftrightarrow$	$\lvert\{\texttt{user} : S_{\texttt{user}} = ACCEPT\}\rvert = N$	
`Majority`	$\leftrightarrow$	$\lvert\{\texttt{user} : S_{\texttt{user}} = ACCEPT\}\rvert > \lfloor N/2 \rfloor$	
`Broken`	$\leftrightarrow$	$\lvert\{\texttt{user} : S_{\texttt{user}} = BREAK\}\rvert > 0$	

Algorithm 3: Main Algorithm

Inputs

`userList`: A circular list of users

`protocol`: A function which defines to which users that a user makes offers.

`Goal`: Either `Consensus` or `Majority`

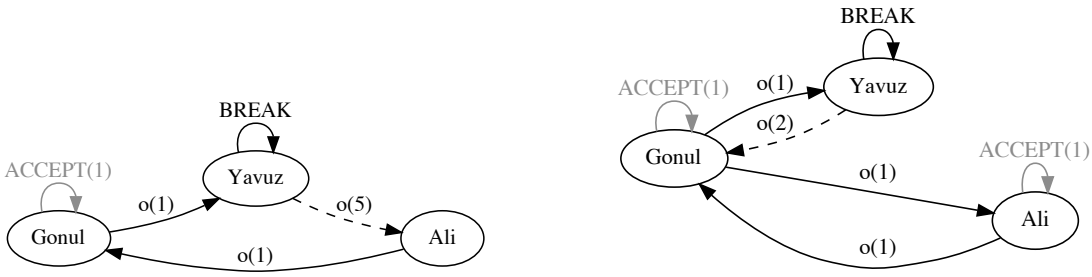
Output

`deal`: A deal

1: `user` $\leftarrow$ `userList[0]`
2: `worstDeal` $\leftarrow \underset{U_{\texttt{user}}(m)}{\arg\max}$ ({`user`}, `m`)
3: **repeat**
4: `selectedMovie` $\leftarrow$ `decide(user)`
5: **Offer** `selectedMovie` **from** `user` **to** `protocol(user, userList)`
6: `acceptingUsers` $\leftarrow \{\texttt{user} : S_{\texttt{user}} = ACCEPT\}$
7: `currentDeal` $\leftarrow$ `(acceptingUsers, selectedMovie)`
8: `user` $\leftarrow$ **Next** `user` **from** `userList`
9: **until** `Goal` **or** `Broken`
10: `deal` $\leftarrow \begin{cases} \texttt{worstDeal} & \texttt{Broken} \\ \texttt{currentDeal} & \texttt{o/w} \end{cases}$

users who mutually accept to go to the movie m.

Finally, we perform the MNG in Algorithm 3. The initial turn always belongs to the first user in the list. Then we initialize the deal as the worst possible deal which is the case that the first user goes to his/her most preferred movie alone. Until the `Goal` is satisfied or someone breaks the negotiations, we continue the negotiations in a turn-based manner. `Goal` can either be `Consensus` or `Majority`. We start with `Goal` = `Consensus` and only if the game is `Broken`, then we change it to `Majority`. At each turn, the user decides on a movie, namely `selectedMovie`, based on the *standard strategy*. We denote this as the `decide` function. Then according to a `protocol`, the `selectedMovie` is offered from `user` to some other users. We describe two such protocols, Gonul Protocol (GP) and Yavuz Protocol (YP) in Section 3. Finally, we construct the current deal by getting the users with state *ACCEPT* and the selected movie and go to the next turn.



(a) with Gonul Protocol (GP)

(b) with Yavuz Protocol (YP)

Figure 3.1: Example Negotiation Graphs

3 PROTOCOLS

We define *a protocol* as a function from a user u and a circular list of users L to a set of users T. T denotes the users in L s.t. u makes offers to whenever u has the turn.

We visualize protocols via negotiation graphs. Overall, a negotiation graph captures the current state of a Movie Negotiation Game. In this graph, nodes denote the users. Solid edges denote the final offer in the current state. Dashed edges denote the last offers of every user which are not the final offer. Each user's state is shown as a self-loop, except we omit drawing the *CONTINUE* state. For *ACCEPT* state, we also give which movie is accepted by the user. We color self-loops as black if the state is changed in the last turn and as gray if the state has been like this for two or more turns. We show examples of negotiation graphs in Figure 3.1.

In general, *fairness* means that at each turn, it is guaranteed that every user is going to have the turn eventually. Our circular list guarantees this condition, hence all our protocols are *fair* in this sense.

We say that a protocol is *strongly fair* if and only if every user makes and receives the same number of offers at every turn. We believe this is an important condition since if some user makes and receives more offers than others, he/she can have more influence on the final deal.

3.1 GONUL PROTOCOL

Gonul Protocol, $T = \text{GP}(u, L)$ is defined simply as $T = \{u'\}$ s.t. u' is the user after u in the circular list L. We give an example negotiation graph that uses GP in Figure 3.1a. Every user makes one offer to the next user at every turn.

It is trivial to see that GP is a generalization of *alternating offers* protocol since they are the same when $N = 2$.

GP is a *strongly fair* protocol since every user gets to make and receive an equal number of offers at each turn.

3.2 YAVUZ PROTOCOL

Yavuz Protocol, $T = \text{YP}(u, L)$ is defined as follows.

$$T = \text{YP}(u, L) = \begin{cases} L - \{L[0]\} & u = L[0] \\ \{L[0]\} & o/w \end{cases}$$

In other words, if the first user has the turn, he/she makes his/her offer to every other user and if any other user has the turn, he/she makes his/her offer to only the first user in YP. We give an example negotiation graph that uses YP in Figure 3.1b.

Surprisingly, YP is also a generalization of *alternating offers* protocol, since when $N = 2$, the first user makes offers to the second user and the second user makes offers to the first.

YP is *NOT a strongly fair* protocol since the first user controls more information in the MNG. We derive this protocol using the assumption that the first user is the one who already scheduled to go to a movie and he/she is trying to convince his/her friends to come along. Therefore, we argue that the first user should have more to say about the final decision. Furthermore, GP assumes that all users are friends with each other whereas YP does not make the assumption that all the first user's friends should also be friends with each other.

3.3 Communication Complexity of Protocols

We define the *communication complexity* of a protocol as the number of offers made in an MNG using that protocol. We do *NOT* use the number of turns since multiple offers can be made in a turn and a protocol can exploit this fact to transfer more information without increasing the turn-based complexity measure.

4 Assumptions

1 We assume there are either two or three users ($N \in \{2,3\}$) in a Movie Negotiation Game (MNG). Although there is no upper bound to this number in real-world conditions ($N \in \mathbb{Z}^+ - \{1\}$), *alternating offers* protocol is restricted to only bilateral (2-user) negotiations ($N = 2$). We generalize the *alternating offers* protocol to support three users as well.

2 We assume that every user has an equal number (M) of preferred movies. Some users may have fewer preferred movies than others in real-world conditions.

3 No user can have the same non-zero utility for two distinct movies, i.e. every preference list implies a total order. Some users may be indifferent between two or more movies in real-world conditions. In short, real-world preference lists are partial orders. We assume that all these users use our Gonul Score based movie selector agent, which return a total order of movies.

4 Every user employs our *standard strategy*. Users may employ different strategies in real-world conditions.

5 We assume that every protocol gives the next turn to the next user in the circular list. We believe circular lists model most of the fair protocols applied in real-world conditions.

6 We assume that the conflict deal utilities of every user to be zero in the beginning. In real-world conditions, some users may have an *incentive* to have a conflict deal. For example, Ali could be a jealous person and may want to go to the movie with only Gonul, eliminating Yavuz in some way. Hence, Ali may get some utility by just breaking the consensus.

7 In the set of received offers, R, we assume that there can be at most one offer from each user. The user could keep a history of all offers made to him/her, though we found out that keeping a history of offers allow *Pareto inefficient* deals in our environment.

8 For any MNG, only the first user in `userList` always has the first turn and not other users. From the description of this project, we gather that the first user is the one who schedules to go to a movie, and therefore he/she must be the starter person of these negotiations. It is trivial to make any user start by just shuffling the `userList`.

5 EXAMPLES

	Case 1			**Case 2**			**Case 3**		
	Gonul	**Ali**	**N/A**	**Gonul**	**Yavuz**	**Ali**	**Gonul**	**Yavuz**	**Ali**
1	2	1	-	2	3	5	8	7	1
2	7	3	-	1	7	3	5	1	2
3	9	8	-	3	2	2	3	8	9
4	15	12	-	-	-	-	-	-	-
5	18	7	-	-	-	-	-	-	-

	Case 4			**Case 5**			**Case 6**		
	Gonul	**Yavuz**	**Ali**	**Gonul**	**Yavuz**	**Ali**	**Gonul**	**Ali**	**N/A**
1	1	4	7	3	3	4	1	7	-
2	2	5	8	5	9	3	4	6	-
3	3	6	9	1	7	2	2	3	-
4	-	-	-	-	-	-	3	4	-
5	-	-	-	-	-	-	5	2	-

	Case 7			**Case 8**			**Case 9**		
	Gonul	**Ali**	**N/A**	**Gonul**	**Yavuz**	**Ali**	**Gonul**	**Yavuz**	**Ali**
1	1	6	-	9	8	2	1	4	6
2	2	7	-	2	1	7	2	5	1
3	3	8	-	3	6	1	3	2	7
4	4	9	-	4	5	5	-	-	-
5	-	-	-	5	9	9	-	-	-

Table 5.1: Preference Tables used in Example Cases 1-9

Table 5.1 shows the example preference tables we tried with POMN. Table 5.2 shows the overall results of each example case.

We use the number of offers made in an MNG (# Offers) as the communication complexity measure of a protocol. Table 5.2 shows that neither GP nor YP dominates the other protocol in terms of communication complexity.

Final utilities of every user are identical in all cases with GP and YP, except Case 8. In Case 8, since GP is *strongly fair*, it does not allow Gonul to get away with a final result which gives more utility to Gonul at the cost of her friends' utilities. Gonul controls the game in YP and therefore convinces both Yavuz and Ali to go to a movie with less utility for them. Although GP is fairer than YP, in this case, YP achieves higher total utility. Hence, we see that *fairness* and *utilitarianism* do NOT go hand in hand

	# Offers		Final Utilities		∑ Utility	
	GP	**YP**	**GP**	**YP**	**GP**	**YP**
Case 1	12	12	G = 4, A = 1	G = 4, A = 1	5	5
Case 2	9	12	G = 1, A = 2, Y = 3	G = 1, A = 2, Y = 3	6	6
Case 3	17	15	G = 3, A = 0, Y = 1	G = 3, A = 0, Y = 1	4	4
Case 4	15	12	*conflict deal*	*conflict deal*	0	0
Case 5	10	8	G = 3, A = 2, Y = 3	G = 3, A = 2, Y = 3	8	8
Case 6	8	8	G = 2, A = 3	G = 2, A = 3	5	5
Case 7	8	8	*conflict deal*	*conflict deal*	0	0
Case 8	14	24	G = 1, A = 2, Y = 2	G = 5, A = 1, Y = 1	5	7
Case 9	9	14	G = 3, A = 2, Y = 0	G = 3, A = 2, Y = 0	5	5

Table 5.2: Communication Complexities and Utilities in Example Cases 1-9

with each other. Note that resulting deals of all cases are *Pareto efficient* unless they end with a *conflict deal*, where Gonul goes to a movie alone.

6 EXPERIMENTS

	# Offers						Utilities			
	GP			YP			GP		YP	
	Min.	**Max.**	**Avg.**	**Min.**	**Max.**	**Avg.**	**# Conflicts**	**Avg. ∑ Utilities**	**# Conflicts**	**Avg. ∑ Utilities**
All Pairs	3	12	9.16	3	12	9.16	20/30	2.26	20/30	2.26
All Triples	6	27	21.14	16	23	21.30	52/120	1.90	52/120	3.98

Table 6.1: Experimental Results on Six Users from the FilmTrust Dataset

We picked six users from the FilmTrust dataset of our previous project. These user ids are 16, 104, 546, 707, 928, and 1508. We took their best five movies according to the Gonul Score. First, we generated preference tables for every pair and every triple, obtaining 30 pairs and 120 triples. Then, we executed both GP and YP on every pair and every triple. We present all results that we obtain from this experimental process in Table 6.1.

Our results show that GP and YP are identical for pairs in terms of all aspects. We expect this since GP and YP are generalizations of the same protocol, namely *alternating offers*. Therefore, GP and YP behave exactly the same for $N = 2$.

For triples, GP and YP have very close average communication complexities (# Offers). However, GP has a smaller minimum and a larger maximum compared to YP. This gives GP a potential to be more efficient than YP, but also we should note that GP's communication complexity is more unpredictable than YP's.

The number of conflicts (# Conflicts) shows the number of cases where only the *conflict deal* is possible, i.e. the first user goes to a movie alone. Both GP and YP giving the same number of conflicts increase our confidence about the soundness of our protocols. Soundness in our case denotes that the ability of a protocol to generate better deals than the conflict deal whenever such a deal exists.

Since YP's concern is utility at the cost of fairness and GP's concern is vice versa, we expect that YP should achieve a higher total utility on average. This is indeed the case as we show in Table 6.1.

7 CONCLUSION

We developed PONM, a Pareto optimal multilateral negotiation agent with customizable protocols. We implement two protocols, Gonul Protocol (GP) and Yavuz Protocol (YP), where GP's main concern is *fairness* and YP's main concern is *utility*. We design case studies and conduct experiments and investigate the properties of both protocols in depth.

In future, we aim to prove that our protocols are complete, i.e. our protocols generate Pareto efficient deals for every possible preference table. We also plan to generalize our protocols to more than three users. We will implement the support for varying number of preferred movies for each user. We will also implement support for the indifference of a user between multiple preferred movies. We will also investigate our protocols under different strategies than the *standard strategy*. We will implement *jealousy* by declaring *varying conflict deal utilities* for every user and investigate its effects on the final deals.