NECTAR : Nash Equilibrium CompuTation Algorithms and Resources

Sirshendu Arosh
Arjun Suresh
Aravind S. R.
Deepak Vishwakarma

April 16, 2011

Abstract

Game theoretic modelling has emerged as a popular tool in many areas such as electronic commerce, ad-hoc networks, computational grids, etc. This has led to growing interest in computational problems associated with game theoretic modelling [1]. We have implemented algorithms for some of the game theoretic problems like computation of different mechanism design problems, converting Bayesian games to normal form games and computation of Core, Shapley value and Nucleolus of co-operative games.

1 INTRODUCTION

NECTAR (Nash Equilibrium CompuTation Algorithms and Resources), is a software environment for computing Nash and other equilibrium points [2]. Nectar is designed to serve the computational needs of the game theory researchers and practitioners who apply game theory and mechanism design to solve their design problems. Nectar supported Nash and other equilibrium points for normal and extensive form games. We enhanced the Nectar by adding three more modules - **Mechanism design, Co-operative games, and Bayesian games**.

1.1 NECTAR Architecture

NECTAR has a modular structure as shown in Figure 1. The various modules of it are described below:

1.1.1 Graphical User Interface Module (GUI)

The GUI module provides graphical interface to the user. User can select the input game file, choose the algorithm for computation and see the output using the GUI.

1.1.2 Nectar Module

The Nectar module is central to the whole system. This module interacts with the pre-processing module for generating the Game objects and interacts with the algorithms module, Co-operative game module, Mechanism design module and Bayesian game module.

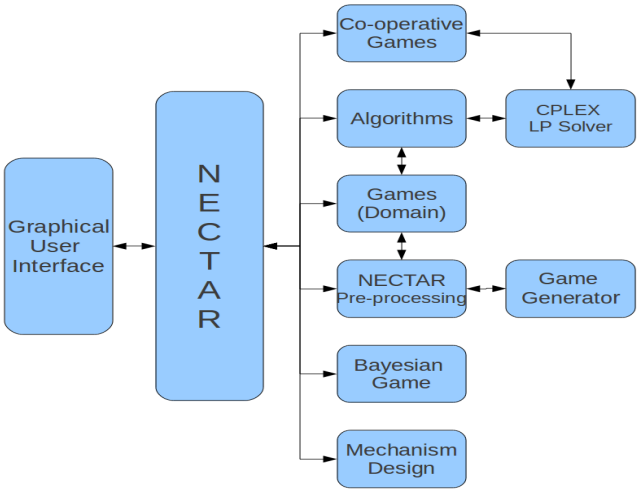

Figure 1: NECTAR: Architecture

1.1.3 Games Module

This module contains the objects which represent various forms of games like normal-form games, extensive-form games, and sequence- form games. This module provides the common interfaces to the concrete objects for modularity.

1.1.4 Algorithms Module

This module can be treated as business module. It contains various algorithms, which are used to compute a sample Nash equilibrium, all Nash equilibrium, and correlated equilibrium points of the given games. This module interacts with games module for getting the utilities of the players and lp-solver interface module for solving a set of linear equations.

1.1.5 LP-Solver Interface Module

The lp-solver interface module provides the interface to any type of linear programming solver with the help of suitable adapters. This interface allows to adapt to any new linear programming solver without modifying the algorithms module.

1.1.6 Mechanism Design Module

We implemented algorithm for computation of Ex-Post efficiency, Dictatorship, Incentive Compatibility, and Individual Rationality.

1.1.7 Co-operative Module

We implemented computation of Core, Shapley value, and Nucleolus of co-operative games.

1.1.8 Bayesian Game Module

We provided the feature to convert a given Bayesian game into a normal form game (Selten game).

1.2 NECTAR User Interface

Nectar user interface is shown in Figure 2.

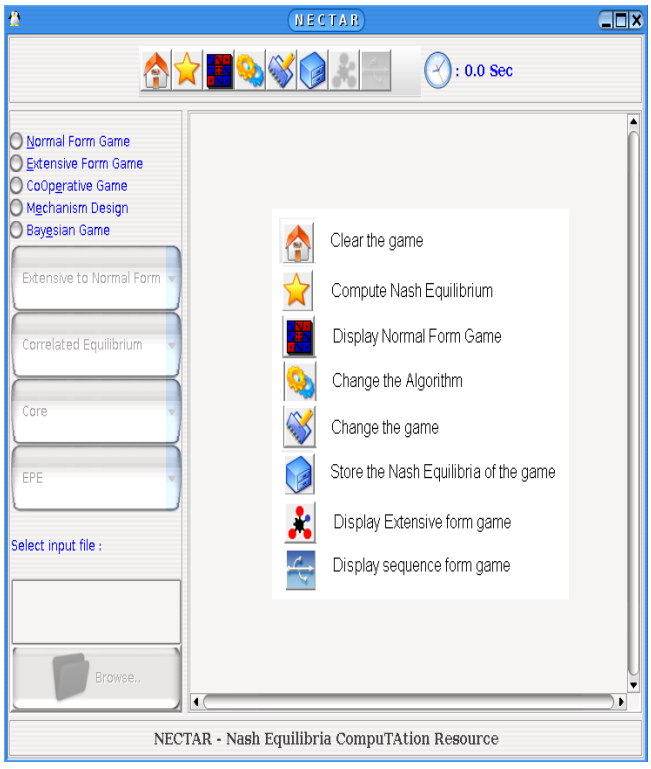

Figure 2: NECTAR: User Interface

2 Implementation

2.1 Co-operative Game

2.1.1 Coalitional Form Game

The coalitional form of an n-person game is given by the pair (N, v),where N is the set of players and v is a real-valued function, called the characteristic function of the game, defined on the power set of N satisfying [4]

- $v(\phi) = 0$ and
- $(S \cap T) = \phi$, then $v(S) + v(T) \leq v(S \cup T)$ where S and T are disjoint coalitions of N

2.1.2 Core

The Core is the set of all pay-off allocations of a Transferable Utility (TU) game that are individually rational, coalitionally rational, and collectively rational [4].
i.e.

$$\text{Core}(N, v) = (x_1, ..., x_n) \in R^n :$$
$$\sum_{i=1}^{n} x_i = v(N);$$
$$\sum_{i \in c} x_i \geq v(C), \forall C \subseteq N$$

2.1.3 Shapley Value : Axioms

- **Efficiency**: $\sum_{i \in N} \phi_i(v) = v(N)$
- **Symmetry**: If i and j are such that $v(S \cup \{i\}) = v(S \cup \{j\})$ for every coalition S not containing i and j, then

$$\phi_i(v) = \phi_j(v)$$

- **Dummy Axiom**: If i is such that $v(S) = v(S \cup \{i\})$ for every coalition S not containing i, then

$$\phi_i(v) = 0$$

- **Additivity**: If u and v are characteristic functions, then

$$\phi(u + v) = \phi(u) + \phi(v)$$

2.1.4 Shapley Value : Definition

Shapley Value is the exactly one mapping $\phi : R^{2^n - 1} \longrightarrow R^n$ that satisfies all the Shapely Value axioms [7]. This function satisfies $\forall i \in N, \forall v \in R^{2^n - 1}$,

$$\phi_i(v) = \sum_{C \subseteq N - i} \frac{|C|!(n - |C| - 1)!}{n!} v(c \cup i) - v(c)$$

2.1.5 Shapley Value : Alternative definition

Alternatively, Shapley value can be expressed in terms of the possible orders of players in N [3].
Let $O : \{1, ...N\} \longrightarrow \{1, ...N\}$ be a permutation that assigns to each position k the player $O(k)$ Given a permutation O, let $Pre^i(O)$ denote the set of predecessors of the player i in the order O
(i.e. $Pre^i(O) = \{O(1), ..., O(K-1)\}$, if $i = O(k)$).
The Shapley value can now be given as

$$sh_i(v) = \sum_{O \in \pi(N)} \frac{1}{n!} v(Pre^i(O) \cup i) - v(Pre^i(O))$$

2.1.6 Estimation of Shapley Value

- The population of sampling process P will be set of all possible orders i,e $P = \pi(N)$
- The vector parameter under study will be $sh = sh_1, ..., sh_n$
- The characteristics observed in each sampling unit $O \in \pi(N)$ are marginal contributions of the players in the order O i.e;

$$x(O) = (x(O)_1, ..., x(O)_n), where$$

$$x(O)_i = v(Pre^i(O) \cup i) - v(Pre^i(O))$$

- The estimate of parameter sh, $\hat{sh}$ will be the mean of the marginal distributions over the sample M, i.e.

$$\hat{Sh} = (\hat{Sh}_1, ..., \hat{Sh}_m), where$$

$$\hat{Sh}_i = (1/m) \sum_{O \in M} x(O)_i$$

- Finally, the selection process used to determine the sample M will take any order with probability $(1/n!)$

$$\Delta = \sum_{i=1}^{N} w_i (x_i - \bar{x})^2$$

Algorithm Auto-Shapley [3]

Begin
Determine m
Cont := 0 and $\hat{Sh}_i$:=0 $i \in N$.
while $Cont < m$
Begin
Take $O \in \pi(N)$ with probability 1/n!
For all $i \in N$
Begin
Calculate $Pre^i(O)$
Calculate $x(O)_i :=$
$v(Pre^i(O) \cup i) - v(Pre^i(O))$
$\hat{Sh}^i := \hat{Sh}^i + x(O)_i$
End
Cont : = Cont+1
End
$\hat{Sh}_i := \hat{Sh}_i / m \ \forall i \in N.$
End

2.1.7 Nucleolus

Excess is a measure of the inequity of an imputation X for a coalition S and is defined as

$$e(x, S) = v(S) - \Sigma_{j \in S} x_j$$

Let $O(x)$ be the vector of excesses arranged in decreasing (non-increasing) order and $X = \{x : \sum_{j=1}^{n} x_j = v(N)\}$ be the set of efficient allocations. We say that a vector $v \in X$ is a **Nucleolus** if for every $x \in X$ we have,

$$O(v) \leq_L O(x),$$

where $\leq_L$ stands for less than in lexicographic ordering [7].

2.1.8 Implementation Details

The classes used in this module are as follows:

- **ReadInput:** This class reads the number of players and utilities of each coalition from the input file.
- **FindShapleyValue:** This class finds the Shapley Value of the input TU game.
- **FindShapleyValueSampling:** This class finds the approximate Shapley Value, using sampling, of the input TU game.
- **FindNucleolus:** This class finds the Nucleolus of the input TU game by solving a series of Linear equations using the CPLEX solver[8].
- **Main:** This class is the main class in this module. It calls the desired class as per the user selection.
- **FindCore:** This class finds the elements of the Core of the input TU game by solving the constraints of the Core using the CPLEX solver[8].

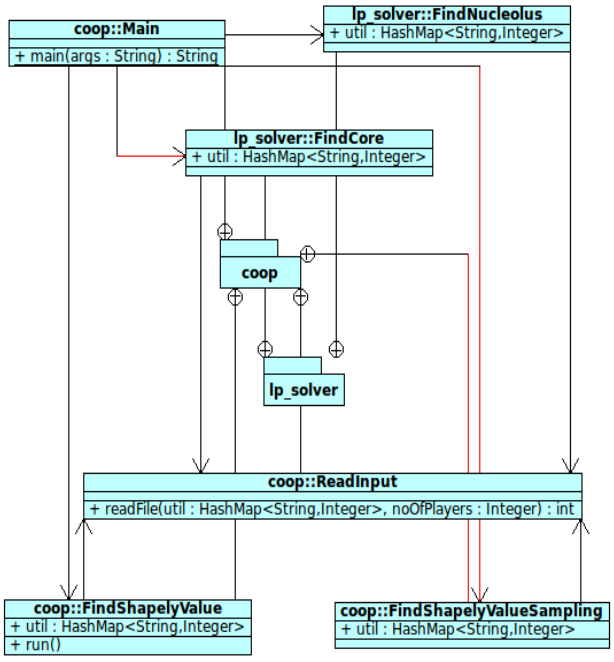

Figure 3: Control Flow of Co-operative game Module

2.1.9 Input format

The first line of the input file specifies the number of players. Each of the following line specifies a coalition and its value. The following is a sample input format for a 3 player game.

3
1 5
2 6
3 4
12 14
23 12
13 15
123 25

2.2 Mechanism Design

2.2.1 Ex-Post Efficiency

The Social Choice Function f: $\Theta \longrightarrow X$ is said to be ex-post efficient if for every profile of agents' types, $\theta \in \Theta$,the outcome, f(θ), is Pareto optimal.The outcome f($\theta_1, ..., \theta_n$) is Pareto optimal if there does not exist any $x \in X$ such that,

$$u_i(x, \theta_i) \geq u_i(f(\theta), \theta_i), \forall i \in N$$

and

$$u_i(x, \theta_i) > u_i(f(\theta), \theta_i), \exists i \in N$$

2.2.2 Dictatorship

The Social Choice Function $f : \Theta \longrightarrow X$ is said to be Dictatorial if there exists an agent d (called Dictator) who satisfies the following property [4]: $\forall \theta \in \Theta, f(\theta)$ is such that,

$$u_d(f(\theta), \theta_d) \geq u_d(x, \theta_d), \forall x \in X$$

2.2.3 Dominant Strategy Incentive compatibility(DSIC)

A social choice function $f : \theta_1 \times ..... \times \theta_n \longrightarrow X$ is said to be Dominant Strategy Incentive Compatible (or truthfully implementable in dominant strategies) if the direct revelation mechanism [4] $D = ((\Theta_i)_{i \in N}, f(.))$ has a weakly dominant strategy equilibrium $s^d(.) = (s_1^d(.), ..., s_n^d(.))$ in which, $s_i^d(\theta_i) = \theta_i, \forall \theta_i \in \Theta, i \in N$. i.e.

$$u_i(f(\theta_i, \theta_{-i}), \theta_i) \geqslant u_i(f(\hat{\theta})_i, \theta_i),$$
$$\forall i \in N, \forall \theta_i \in \Theta_i, \forall \theta_{-i} \in \Theta_{-i}, \forall \hat{\theta}_i \in \Theta_i$$

2.2.4 Bayesian Incentive Compatibility(BIC)

A social choice function $f : \theta_1 \times ..... \times \theta_n \longrightarrow X$ is said to be Bayesian Incentive Compatibility(or truthfully implementable in Bayesian Nash Equilibrium) if the direct revelation mechanism $D = ((\Theta_i)_{i \in N}, f(.))$ has a Bayesian Nash equilibrium [4] $s^*(.) = (s_1^*(.), ..., s_n^*(.))$ in which $s_i^*(\theta_i) = \theta_i \forall \theta_i \in \Theta, i \in N$. i.e.

$$E_{\theta_{-i}}[u_i(f(\theta_i, \theta_{-i}), \theta_i)|\theta_i] \geqslant E_{\theta_{-i}}[u_i(f(\hat{\theta})_i, \theta_i)|\theta_i],$$
$$\forall i \in N, \forall \theta_i \in \Theta_i, \forall \theta_{-i} \in \Theta_{-i}, \forall \hat{\theta}_i \in \Theta_i$$

2.2.5 Ex-Post Individual Rationality Constraints

These constraints become relevant when any agent i is given a choice to withdraw from the mechanism at the ex-post stage that arise after all the agents have announced their types and an outcome in X has been chosen. Let $\bar{u}_i(\theta_i)$ be the utility that agent i receives by withdrawing from the mechanism when his type is θ_i. Then to ensure the agent i's participation, we must satisfy the following ex-post participation(or individual rationality) constraints [4]

$$u_i(f(\theta_i, \theta_{-i})\theta_i) \geq \bar{u}_i(\theta_i)$$

2.2.6 Interim Individual Rationality Constraints

Let the agent i be allowed to withdraw from the mechanism only at an interim stage that arises after the agents have learned their type but before they have chosen their actions in the mechanism. In such a situation, the agent i will participate in the mechanism only if his interim expected utility, $U_i(\theta_i \setminus f) = E_{\theta_{-i}}[u_i(f(\theta_i, \theta_{-i}), \theta_i \setminus \theta_i]$ from the social choice function $f(.)$ when his type is θ_i ,is greater than $\bar{u}_i(\theta_i)$.Thus, interim participation(or individual rationality) constraints for agent i require that [4]

$$U_i(\theta_i \setminus f) = E_{\theta_{-i}}[u_i(f(\theta_i, \theta_{-i}), \theta_i \setminus \theta_i] \geq \bar{u}_i(\theta_i) \ , \theta_i \in \Theta_i$$

2.2.7 Ex-Ante Individual Rationality constraints

Let agent i be allowed to refuse to participate in the mechanism only at ex-ante stage that arises before the agents learn their types. In such a situation, the agent i will participate in the mechanism only if his ex-ante participation utility [4] $U_i(f) = E_\theta[u_i(f(\theta_i, \theta_{-i}), \theta_i)]$, from the social choice function $f(.)$ is greater than $E_{\theta_{-i}}[\bar{u}_i(\theta_i)]$. Thus, ex-ante participation(or,individual rationality) constraints for agent i require that,

$$U_i(f) = E_\theta[u_i(f(\theta_i, \theta_{-i}), \theta_i)] \geq E_{\theta_i}[\bar{u}_i(\theta_i)]$$

2.2.8 Implementation Details

The classes used in this module are as follows:

- **Main:** This is the main module of Mechanism design. This module is integrated to nectar framework. All the algorithms in mechanism design are specified here.
- **mdDSIC:** This is the main DSIC class. This will compute whether a given SCF is DSIC or not. This is done by comparing the utilities when he/she is telling truth with when he/she is not telling the truth
- **mdInitializer:** Given a mdGameObject and a filename, this class will initialize the complete mdGameObject. This is done by calling the corresponding initializers. For more information see documentation of each of the initializers.
- **mdSCFinit:** Given a mdGameObject and a filename, this class will initialize the scf object in mdGameObject. First the scf object is cleared and then assigned with original data values. The clearing is necessary as, if the program is run more than one time, the previous data should not interfere with the current one.
- **mdUtilityInit:** Given a mdGameObject and a filename, this class will initialize the utility object in mdGameObject. First the utility object is cleared and then assigned with original data values. The clearing is necessary as, if the program is run more than one time, the previous data should not interfere with the current one.
- **mdEAIR:** This is the main EAIR class. This will compute whether a given SCF is EAIR or not for each of the players. Given the mdGameObject this fucntion will compute whether the SCF is EPIR for the all the players
- **mdIIR:** This is the main IIR class. This will compute whether a given SCF is IIR or not for each of the players. Given the mdGameObject this fucntion will compute whether the SCF is EPIR for the all the players
- **mdGameObject:** This is the Game Object class. All the input parameters and configurations are stored here. The whole game is represented by this object
- **mdBeliefInit:** Given a mdGameObject and a filename, this class will initialize the belief object in mdGameObject. First the belief object is cleared and then assigned with original data values. The clearing is necessary as, if the program is run more than one time, the previous data should not interfere with the current one.
- **mdEPE:** This is the main EPE class. This will compute whether a given SCF is EPE or not. This is done by comparing the possible epe's
- **mdTrueType:** Given a mdGameObject and a filename, this class will initialize the truetype object in mdGameObject. First the truetype object is cleared and then assigned with original data values. The clearing is necessary as, if the program is run more than one time, the previous data should not interfere with the current one.
- **mdEPIR:** This is the main EPIR class. This will compute whether a given SCF is EPIR or not for each of the players. Given the mdGameObject this fucntion will compute whether the SCF is EPIR for the all the players
- **mdBIC:** This is the main BIC class. This will compute whether a given SCF is BIC or not. This is done by comparing the utilities when he/she is telling truth with when he/she is not telling the truth
- **mdDOM:** This is the main dominator class. This will compute whether a given SCF is dictatorial or not. This is done by comparing the utility of each player against all alternatives

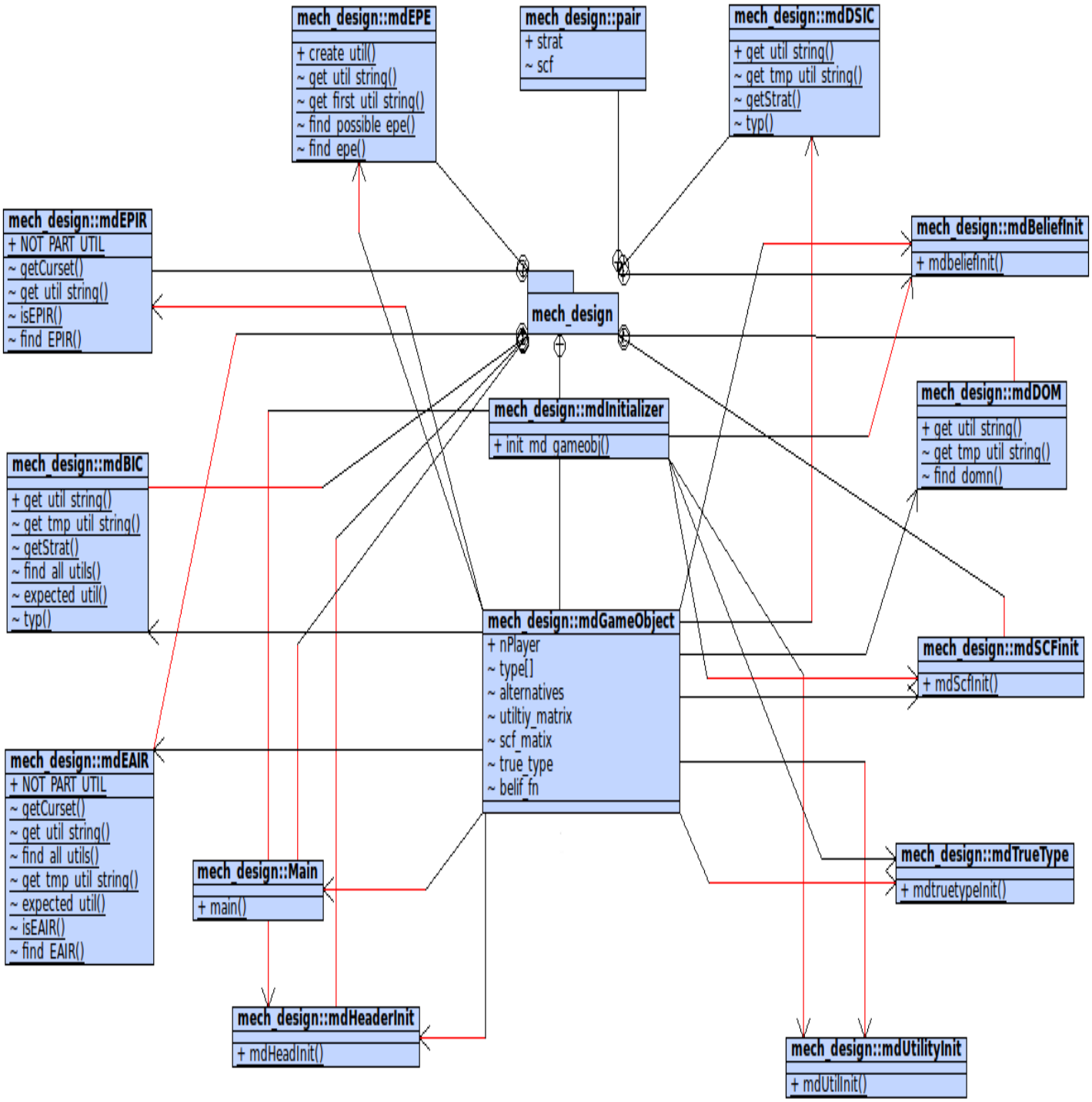

Figure 4: Control Flow of Mechanism Design Module

- **mdHeaderInit:** Given a mdGameObject and a filename, this class will initialize the header object in mdGameObject. First the header object is cleared and then assigned with original data values. The clearing is necessary as, if the program is run more than one time, the previous data should not interfere with the current one.

2.2.9 Snapshot of the Input Format

```
# Generated by NECTAR
# Game Description
# Players: 2
# Types: 2 2
# Alternatives: 3

## Utility Matrix
[1 1 1] : [100]
[1 1 2] : [50]
[1 2 1] : [50]
[1 2 2] : [100]
[1 3 1] : [0]
[1 3 2] : [40]
[2 1 1] : [0]
[2 1 2] : [50]
[2 2 1] : [50]
[2 2 2] : [30]
[2 3 1] : [100]
[2 3 2] : [100]
## SCF
[1 1] = [1]
[1 2] = [1]
[2 1] = [3]
[2 2] = [3]
## True Types
[1] = [1]
[2] = [2]
## Belief Function
[1] = [.2 .8;.6 .4;]
[2] = [.3 .7;.4 .6;]
```

Figure 5: Input Format of Mechanism Design Problem

2.3 Bayesian Games

2.3.1 Introduction

Bayesian games are games with incomplete information [4], i.e., all information is not deterministically and publicly known to the all players. There is at least one player who has some private information about the game that is not known deterministically to the rest of players. Incomplete information games are more realistic and practical.

2.3.2 Definition

A Bayesian game is a tuple $\Gamma = < N, (\Theta_i), (S_i), (p_i), (u_i) >$, where the components are described as below [4]:

- N =1,2,...,n is a set of players
- Θ_i is the set of types of player i, where i = 1, 2,....,n.
- S_i is the set of actions or pure strategies of player i, where i = 1, 2,...,n.
- p_i is a probability (belief) function from Θ_i to the set of probability distribution set of type sets of rest of players Θ_{-i}.
- The payoff function $u_i : \Theta \times S \longrightarrow \Re$ is such that, for any profile of actions and any profile of types $(\theta, s) \in (\Theta \times S), u_i(\theta, s)$ specifies the payoff that player i would get, if the players' actual types were all as in θ and the players all chose their actions according to s.
- Γ is a finite game iff N, $(\Theta_i)_{i \in N}$, and $(S_i)_{i \in N}$ are finite.

2.3.3 Consistency of Beliefs

We say players' belief $p_i,\ i \in N$, in a Bayesian game is consistent iff there is a common prior distribution over the set of type profiles Θ, such that each players' belief, given his type, is the conditional probability distribution that can be computed from the prior distribution by the Bayes formula [4].
In the finite case, beliefs are consistent if there exists a probability distribution $P \in \triangle(\Theta)$ such that

$$p_i(\theta_{-i}|\theta_i) = \frac{p(\theta_i, \theta_{-i})}{\sum_{t_{-i} \in \theta - i} p(\theta_i, t_{-i})},$$

$$\forall \theta_i \in \Theta_i, \forall \theta_{-i} \in \Theta_{-i}, \forall i \in N$$

In a consistent model, the common prior can be used to calculate all the probability functions.

2.3.4 Type Agent Representation

Selten proposed a representation of Bayesian games that enables a Bayesian game to be transformed to a strategic form game (games with complete information)[4]. The idea is to represent every possible type of every player as an agent or player in the new game. Given a Bayesian game $\Gamma = < N, (\Theta_i), (S_i), (p_i), (u_i) >$ the Selten game is an equivalent strategic form game $\Gamma^s = < N^s, {S_j}^s, U_j >$. Hence, the players in the Selten game are $N^s = \bigcup_{i \in N} \Theta_i$
The strategies of the type agents (Selten players) are precisely the same actions as that of corresponding Bayesian player. i.e. $S^s_{\theta_i} = S_i$

2.3.5 Selten Player Utilities

The payoff function U_i for each $i \in N$ is the conditionally expected utility to player i in the Bayesian game given that i

is his actual type. The utilities of Selten players are defined as,

$$u_{\theta_i}(s_{\theta_i}, s_{-\theta_i}) = \sum_{t_{-i} \in \theta - i} p_i(t_{-i}|\theta_i) u_i(\theta_i, t_{-i}, s_{\theta_i}, s_{t_{-i}})$$

2.3.6 Implementation Details

The classes used in this module are as follows:

- **BayesianGame:** This class provides template for creation of Bayesian game player objects.
- **SeltenGame:** This class provides template for players of Selten game, generated for a given Bayesian game.
- **Main:** This class reads the given Bayesian game and builds an array of BayesianGame player objects. Then it builds the SeltenGame players objects from BayesianGame players objects, computing the utilities for every player for each action set of Selten game. The generated Selten game is returned to the Nectar module for computation of Nash and other equilibria points.

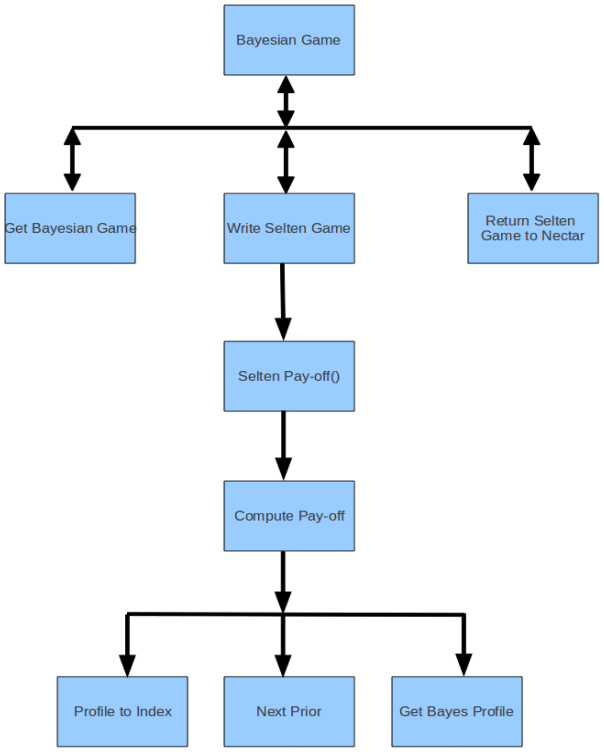

Figure 6: Control Flow of Bayesian Game Module

2.3.7 Input and output format

The Bayesian game input file contains followings :

- Type of game.
- Number of players.
- Action set of players.
- Type set of players.
- The common prior distribution.
- Player utilities for every common prior.

A snapshot of input game file is shown Figure 7.

```
# Bayesian Game file
# Players:      2
# Actions:      2 2
# Type Set:     2 2
# Common Prior:

[ 1 1 ]: 0.200000
[ 2 1 ]: 0.200000
[ 1 2 ]: 0.200000
[ 2 2 ]: 0.400000
# Player Utilities:


 # Utilities for type profile [ 1 1 ]:
 [ 1 1 ]: 1.000000 2.000000
 [ 2 1 ]: 0.000000 4.000000
 [ 1 2 ]: 0.000000 1.000000
 [ 2 2 ]: 1.000000 3.000000

 # Utilities for type profile [ 2 1 ]:
 [ 1 1 ]: 1.000000 2.000000
 [ 2 1 ]: 0.000000 4.000000
 [ 1 2 ]: 0.000000 1.000000
 [ 2 2 ]: 1.000000 3.000000

 # Utilities for type profile [ 1 2 ]:
 [ 1 1 ]: 1.000000 3.000000
 [ 2 1 ]: 0.000000 1.000000
 [ 1 2 ]: 0.000000 4.000000
 [ 2 2 ]: 1.000000 2.000000

 # Utilities for type profile [ 2 2 ]:
 [ 1 1 ]: 1.000000 3.000000
 [ 2 1 ]: 0.000000 1.000000
 [ 1 2 ]: 0.000000 4.000000
 [ 2 2 ]: 1.000000 2.000000
```

Figure 7: Input Format of Bayesian Game

3 Conclusion and Future Works

We are able to enhance Nectar by providing features to support Mechanism Design, Co-operative Games and Bayesian Games. In future, the following additions can also be made to Nectar

- **Additional Solution Concepts for Co-Operative Games:** Currently Core, Shapley Value and Nucleolus are implemented in Nectar. In future additional solution concepts like Kernel, Nash Bargaining solution etc. can be implemented in Nectar.
- **Use of more Approximation Algorithms:** Currently approximate algorithm is used to find Shapley Value only In future, approximate algorithms can be used to find other concepts with faster running time.
- **Improvement of GUI:** The Nectar GUI can be further enhanced to improve user-friendliness.
- **Provision of a web-based Interface:** A web-based interface is highly useful for Nectar so that users can find solutions to their problems online.
- **Replacement of CPLEX with an open-source LP Solver:** Currently, Nectar uses IBM's CPLEX solver for solving Linear programming and it can be replaced with an open-source LP solver in future.

References

[1] Y. Narahari, Dinesh Garg, Ramasuri Narayanam, and Hastagiri Prakash, *Game Theoretic Problems in Network Economics and Mechanism Design Solutions*, Springer Monograph Series in Advanced Information and Knowledge Processing Series, April 2009.

[2] McKelvey, Richard D., & McLennan, Andrew, *Computation of equilibria in finite games*, Handbook of Computational Economics 1996, edition 1, volume 1, chapter 2, pages 87-142 Elsevier

[3] Castro, Javier and Gómez, Daniel and Tejada, Juan, *Polynomial calculation of the Shapley value based on sampling* Elsevier Science Ltd, volume = 36, issue = 5, Year 2009.

[4] Y. Narahari, *"Game Theory Lecture notes"*, Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, India

[5] Sujit Gujar and Rama Suri Narayanam, *Nash Equilibriam CompuTation Algorithms and Resources*, Microsoft Research India, TechVista Delhi 2007

[6] M Kalyan Chakarvarthy, *NECTAR: Nash Equilibrium Computation Algorithms and Resources*, ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, India, 2006.

[7] Thomas S. Ferguson, *"Game Theory Notes"*, Mathematics Department, UCLA

[8] *CPLEX Reference Manual*, `http://www.lix.polytechnique.fr/~liberti/teaching/xct/cplex/usrcplex.pdf`