N E C T A R : Nash Equilibria CompuTAtion Resource

A Thesis

Submitted in partial fulfilment of the

requirements for the Degree of

Master of Engineering

in the Faculty of Engineering

by

M. Kalyana Chakravarthi

Under the supervision of

Prof. Y. Narahari

June, 2006

Computer Science and Automation
Indian Institute of Science
Bangalore, India

Dedicated

To

My Parents

Acknowledgments

I am deeply indebted to Prof. Y. Narahari for his meticulous guidance and constant encouragement throughout the course of this project. His extraordinary patience, and his approach to research, writing, teaching, and mentoring have been a great source of inspiration. A lot of his effort and time went into the making of this project. If the following pages make a better reading, it has a lot to do with his clear thinking and his ability to express clearly. My association with him for the past one year has been one great learning experience.

I thank the Department of Computer Science and Automation (CSA) for providing excellent ambiance, faculty, and facilities. I am happy to be a member of the Electronic Enterprices Laboratory (EEL), CSA which provided me a pleasant atmosphere for doing my work.

I thank Keyur who helped for the implementation of vertex enumeration for Mangasarian's algorithm. I thank Suri, Radhani, Sowjanya, Sujit and Siva. Because of their valuable suggestions, critical feedback, and the moral support, the NECTAR has its own shape with in a short period of time. A very special thanks for all of them.

I owe a lot to my parents, and my brothers for their unconditional support and encouragement to carry out my work.

Abstract

Game theoritic modeling 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. Computing Nash equilibria is a fundamental computational problem at the interface of computer science and game theory. The computational complexity of many algorithms for computing Nash equilibria remains an interesting open problem.

NECTAR is an open source, platform-independent software tool for computing Nash equilibria of strategic form games and extensive form games. It also provides an environment where researchers can carry out experiments using different algorithms. The design and implementation of NECTAR has been carried out following best practices in object oriented software engineering.

Contents

List of Figures

Chapter 1

Introduction

The central game-theoretic solution concept is the Nash equilibrium, a fixed-point in mixed strategy space existing for every finite game. This concept was first introduced by Nash [24] in 1951. It remains unknown whether a Nash equilibrium can be found in polynomial time, even in two-person games. Recently, many researchers have started looking at computational problems in game theory and are developing many algorithms for computing Nash equilibria. The next section answers the some of the questions that could be frequently about NECTAR.

1.1 An FAQ on NECTAR

- ***What is NECTAR?***
  NECTAR, Nash Equilibriuam CompuTAtion Resource, is an open source software tool to compute the Nash equilibrium points of non-cooperative games.

- ***Why NECTAR?***
  Game theory is applicable in many areas of computer science and economics. In all these problems, the central challenge is to find the equilibria. NECTAR is designed for computing the equilibrium points efficiently and effectively.

- ***What are the previous efforts?***

  – **Gambit** is a library of game theory software and tools for the construction and analysis of finite extensive and strategic games available at *http://econweb.tamu.edu/gambit* [20].

  – **GameTracer** is an application for calculating Nash equilibria in strategic form games. It implements the global Newton method of Govindan and Wilson (2001), as well as their iterative polymatrix approximation algorithm available at *http://dags.stanford.edu/G*

- ***How do you compare NECTAR with GAMBIT?***
  GAMBIT has not implemeted the newly published algorithms, but NECTAR has algorithms which are recently published.

- ***How do you compare NECTAR with GAMETRACER?***
  GAMETRACER is for only strategic form games and implemented global Newton method and iteratively polynomical algorithms. Whereas, NECTAR is the complete suite of algorithms for both strategic form and extensive form games.

- ***Which algorithms have been implemented?***
  Pure Nash equilibria, two-person zero-sum algorithm, Lemke-Howson algorithm, Mangasarian algorithm, simple search method, mixed integer programming, sequence form algorithm, and correlated equilibrium etc..

- ***What are the platform details of NECTAR?***
  NECTAR is implemented in Java, which is object oriented, platform independent, and architectural-neutral language.

- ***What are the design patterns used in NECTAR?***
  Factory method, Singleton, Command, Facade, Mediator, and Adapter etc., are used for NECTAR.

- ***What type of data structrues used for NECTAR?***
  Multidimensional array for strategic form games, and general tree structure for extensive form games.

- ***What are the other tools used for NECTAR?***
  For some algorithms, currently we used ILOG's CPLEX software tool for solving the set of linear equations. We also provided an interface in such way that can be easily adapted to any other linear programming solver with a suitable adapter class.

- ***What is the input to the NECTAR?***
  Currently NECTAR takes all inputs in the form of files. NECTAR uses GAMBIT input format for both strategic form games and extensive form games.

1.2 Contributions

Recently, Jain [13] worked on computation of equilibrium points of non-cooperative games and Stackelberg games. She implemented the branch and bound algorithm for bilevel programming problem and the classical Lemke-Howson algorithm. Our project is the next generation version

of the above project. Our objective is to come up with complete suite of programs for computing Nash equilibria.

Our objective is to build a comprehensive suite of implementations of classical and emerging algorithms for Nash equilibrium computation. The algorithms that we have identified for implementation on include

- Lemke-Howson algorithm developed by Lemke and Howson [17]
- Enumeration of all equilibria developed by Mangasarian [18]
- Simple search methods developed by Porter, Nudelman, and Shoham [27]
- Mixed integer programming method by Sandholm, Gilpin, and Conitzer [30]
- Simplicial subdivision method for n-person games by Talman [33]
- Global Newton method for n-person by Govindan and Wilson [11]
- Iterated polymatrix approximation method for n-person by Govindan and Wilson [12]
- Correlated equilibrium for n-person game [23]

The design of NECTAR is such that new algorithms and procedures can be easily added to the existing suite as and when they become available.

1.3 Outline

The outline of this report is as follows. In Section 2, we present non-cooperative game theoretic concepts and equilibrium concepts. In Section 3, computational complexity issues. In Section 4, a survey of existing algorithms for Nash equilibrium computation is provided. In Section 5, we discuss the architecture of NECTAR and algorithms implemented so far. Section 6 concludes with directions for future development of NECTAR.

Chapter 2

Preliminaries

This chapter presents a brief introduction of game theory, explains various forms of game representation, and the concept of Nash equilibrium and gives the formulation of a naive method for computing Nash equilibrium.

2.1 Game Theory

Game Theory is the study of problems of conflict and cooperation among independent decision-makers. The ingredients of a game theory problem include players (decision-makers), choices (feasible actions), payoffs(benefits, prizes or awards) and preferences to payoffs (objectives). Problems in game theory can be classified in a number of ways like static Vs dynamic games, cooperative Vs non-cooperative games, complete Vs incomplete information *etc.*. A *player* is a decision-maker who is participant in the game and whose goal is to choose the action that produce his most preferred outcomes or lotteries over outcomes. We assume that *players are rational*. The rational players choose actions that maximize their expected utilities.

Games can be represented by extensive form (game tree) or normal form (strategic). Here we are studying only non-cooperative games. For a complete introduction to this class of games, the reader is referred to [23].

2.2 Extensive Form Game (Game Tree)

The extensive form (also called a game tree) is a graphical representation of a sequential game. It provides information about the *players*, *payoffs*, *strategies*, and the order of *moves*. The game tree consists of nodes (or vertices), which are points at which players can take actions, connected by edges, which represent the actions that may be taken at that node. An initial (or root) node represents the first decision to be made. Every set of edges from the first node through the tree eventually arrives at a terminal node, representing an end to the game. Each terminal node is

labeled with the payoffs earned by each player if the game ends at that node.

Formally, the extensive form of a game can be represented as

$$\Gamma = \{I, (X, \succ), \iota(), C(), H, u\}$$

which contains the following elements:

- A set of players denoted by $i \in I$, with $I = \{N, 1, 2, ...\}$, with N representing the pseudo-player Nature;
- A tree, $(X, \succ)$, which is a finite collection of nodes $x \in X$ endowed with the precedence relation $\succ$, where $x \succ x'$ means x *is before* x'. This relation is transitive and asymmetric, and thus constitutes a *partial order*. This rules out cycles where the game may go from node x to a node x', from x' to node x'', and from x'' back to x. In addition, each node x has exactly one immediate predecessor, that is, one node $x' \succ x$ such that $x'' \succ x$ and $x'' \neq x'$ implies $x' \succ x''$ or $x'' \succ x'$. Thus, if x' and x'' are both predecessors of x, then either x' is before x'' or x'' is before x'.
- A set of terminal nodes, denoted by $z \in Z$ consisting of all nodes that are not predecessors of any other node. Because each z determines the path through the tree, it represents an outcome of the game. The payoffs for outcomes are assigned by the Bernoulli payoff functions $u_i : Z \to \Re$, and $u = (u_1(), \ldots, u_I())$ is the collection of these functions, one for each player.
- A map $\iota : X \to I$, with the interpretation that player $\iota(x)$ moves at node x. A function $C(x)$ that denotes the set of feasible actions at x.
- Information sets $h \in H$ that partition the decision nodes of the tree such that every node is exactly in one set. The interpretation of $h(x)$ is that information set h contains the node x. Suppose if the information set h contains the another node x', then $C(x') = C(x)$. In general, C_h denote the set of feasible actions at the information set h.
- A probability distribution over the set of alternatives for all chance nodes.

2.2.1 Perfect Recall

Given a game is said to be *perfect recall*, that each player never forgets the information he once knew, and each player knows the actions he has chosen previously. This is accomplished by requiring that

- If two decision nodes are in the same information set, then neither is a predecessor of the other.
- If two nodes x' and x'' are in the same information set and one of them has a predecessor x, then the other one has a predecessor $\hat{x}$ in the same information set as x and the action taken at x that leads to x' is the same as the action taken from $\hat{x}$ that leads to the x''.

2.2.2 Strategies

A *strategy* is a complete contingent plan which specifies what a player will do in each of his information sets, where he is called upon to play. The set of information sets of a player represents the set of all possible distinguishable circumstances in which the player may be required to play.

In the given game Γ, a *pure strategy* for player $i \in I$ is a function $s_i : H \to C$ such that $s_i(h) \in C_i(h)$ for all $h \in H$. Letting H_i denote the collection of information sets for player i, the number of pure strategies he has is

$$|S_i| = \prod_{h \in H_i} |(C_i(h))|$$

2.2.3 Behaviour Strategies

Unlike strategic form games, extensive form games allow two distinct types of randomization: a player can either randomize over his pure strategies (mixed strategy) or he can randomize over the actions at each of his information sets.

A *behavior strategy* β_i for player i, which specifies a probability distribution over actions $C(h_i)$ at each information set h_i. These distributions are independent. Thus, a pure strategy s_i is a special kind of behavior strategy β_i where the distribution at each information set is degenerate.

A *sequence* of player i is the sequence of his choices, on the unique path from the root to some node t of the tree, and is denoted by $\sigma_i(t)$. The empty sequence is denoted by $\emptyset$. Player i has *perfect recall* iff $\sigma_i(t) = \sigma_i(s)$ for any nodes $s, t \in h$ and $h \in H$. Then the unique sequence $\sigma_i(t)$ leading to any node t in h will be denoted σ_h.

Let β_i be a behaviour strategy of player i. The move probabilities $\beta_i(c)$ fulfill

$$\sum_{c \in C_h} \beta_i(c) = 1, \quad \beta_i(c) \geq 0 \text{ for all } h \in H, c \in C_h$$

The *realization probability* of a sequence σ of player i under β_i is

$$\beta_i[\sigma] = \prod_{c \in \sigma} \beta_i(c)$$

An information set h in H_i is called *relevant* under β_i iff $\beta_i[\sigma_h] \geq 0$, otherwise *irrelevant*.

2.3 Strategic (Normal) Form game

A strategic form game G is a tuple

$$G = (N, (S_i)_{i \in N}, (u_i)_{i \in N})$$

where N = {1,2,. . . ,n} is a finite set of players. $S_1, S_2, \ldots, S_n$ are the strategy sets of the players. $u_i : S_1 \times S_2 \times \cdots \times S_n \to \Re$ for $i = 1, 2, \ldots, n$ are utility functions $u_i : X_{i \in N} S_i \to \Re$. We often denote $S = S_1 \times S_2 \times \cdots \times S_n$, called the set of strategy profiles of the players. The strategies here are also called "*actions*" or "*pure strategies*" and the utility of a player depends not only on his own strategy but on an entire strategy profile. Utility functions are also called *payoff functions*. Every profile of strategies induces an *outcome* in the game.

A mixed strategy for player i is a probability distribution over S_i. That is, $\sigma_i : S_i \to [0, 1]$, assigns to each pure strategy $s_i \in S_i$, a probability $\sigma_i(s_i)$ such that $\sum_{s_i \in S_i} \sigma_i(s_i) = 1$.

If $S_i = \{s_{i1}, s_{i2}, \ldots, s_{im}\}$, then the set of all mixed strategies of player i is the simplex:

$$\Delta(S_i) = \{(\sigma_{i1}, \ldots, \sigma_{im}) \in \Re^m : \sigma_{ij} \geq 0 \text{ for } j = 1, \ldots, m \text{ and } \sum_{j=1}^{m} \sigma_{ij} = 1\}$$

Here, $\Delta(S_i)$ is the set of all probability distributions over S_i. This simplex is called the mixed extension of S_i. Using mixed extension of strategy sets we would like to define a mixed extension of the pure strategy game $G = (N, (S_i), (u_i))$. The mixed extension of G is denoted by

$$G_{ME} = (N, (\Delta(S_i)), (U_i))$$

There for $i = 1, 2, \ldots, n$,

$$U_i : X_{i \in N} \Delta(S_i) \to \Re$$

Given $\sigma_i \in \Delta(S_i)$ for $i = 1, \ldots, n$, be compute $U_i(\sigma_1, \ldots, , \sigma_n)$ as follows. We implicitly assume that the randomizations of individual players are mutually independent. This implies that given a profile $(\sigma_1, \ldots, \sigma_n)$, the random variables $\sigma_1, \ldots, \sigma_n$ are mutually independent.

Therefore the probability of a pure strategy profile $(s_i, \ldots, s_n)$ is given by

$$\prod_{i \in N} S_i \to [0, 1]$$

$$\sigma(s_1, \ldots, s_n) = \prod_{i \in N} \sigma_i(s_i)$$

The payoff functions U_i are defined in a natural way as

$$U_i(\sigma_1, \ldots, \sigma_n) = E_\sigma[u_i(s)] = \textstyle\sum_{(s_1, \ldots, s_n) \in S} \sigma(s_1, \ldots, s_n) u_i(s_1, \ldots, s_n)$$

The tuple $(N, (\Delta(S_i)), (U_i))$ is called the mixed extension of $(N, (S_i), (u_i))$.

2.4 Pure Strategy Nash Equilibrium

Given a game $G = (N, (S_i), (u_i))$ with pure strategies, the strategy profile

$$s^* = (s_1^*, s_2^*, \ldots, s_n^*)$$

is said to be a pure strategy Nash equilibrium of G if,

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*) \quad \forall s_i \in S_i, \ \forall i = 1, 2, \ldots, n$$

where $s_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$. That is, each player's Nash equilibrium strategy is a best response to the Nash equilibrium strategies of the other players. Given a game $G = (N, (S_i), (u_i))$, the best response correspondence for player i is the mapping $b_i : S_{-i} \to 2^{S_i}$ defined by

$$b_i(s_{-i}) = \{s_i \in S_i : u_i(s_i, s_{-i}) \geq u_i(s_i', s_{-i}), \ \ \forall s_i' \in S_i\}$$

That is, given a profile s_{-i} of strategies of the other players, $b_i(s_{-i})$ gives the set of all best response strategies of player i. Given a pure strategy game $G = (N, (S_i), (u_i))$, the strategy profile $(s_1^*, \ldots, s_n^*)$ is a Nash equilibrium iff,

$$s_i^* \in b_i(s_{-i}^*), \quad \forall i = 1, \ldots, n$$

For $(s_1^*, \ldots, s_n^*)$ to be a Nash equilibrium, it must be that no player can profitably deviate from his Nash equilibrium strategy given that the other players are playing their Nash equilibrium strategies.

2.5 Mixed Strategy Nash Equilibrium

A mixed strategy profile $(\sigma_1^*, \ldots, \sigma_n^*)$ is called a Nash equilibrium if $\forall i \in N$,

$$u_i(\sigma_i^*, \sigma_{-i}^*) \geq u_i(\sigma_i, \sigma_{-i}^*) \quad \forall \sigma_i \in \Delta(S_i)$$

Define the best response functions as follows.

$$b_i : X_{j \neq i} \Delta(S_j) \to 2^{\Delta(S_i)}$$

$$b_i(\sigma_{-i}) = \{\sigma_i \in \Delta(S_i) : u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma_i', \sigma_{-i}) \; \forall \sigma_i' \in \Delta(S_i)\}$$

Then, a mixed strategy profile $(\sigma_1^*, \ldots, \sigma_n^*)$ is a Nash equilibrium iff

$$\sigma_i^* \in b_i(\sigma_{-i}^*) \quad \forall i = 1, 2, \ldots, n.$$

2.6 Computation of Nash Equilibrium

Given a mixed strategy σ_i of player i, the *support* of σ_i is the set of strategies of i which have positive probabilities under σ_i.

$$\delta(\sigma_i) = \{s_i \in S_i : \sigma_i(s_i) > 0\}$$

The number of supports of a mixed strategy of a player i is $2^{|S_i|} - 1$ (the number of non-empty subsets of S_i). Therefore total number of supports of mixed strategy profiles are $\Pi_{i \in N}(2^{|S_i|} - 1)$.

A naive method of finding a Nash equilibrium is to iterate through all the supports available. Let $\delta_i \subseteq S_i$ be a non-empty subset of S_i. If there exists a Nash equilibrium σ^* with this support, then there must exist numbers $v_1, \ldots, v_n$ and mixed strategies $\sigma_1, \ldots, \sigma_n$ such that

$$(\forall i)(\forall s_i \in \delta_i) \quad \textstyle\sum_{s_{-i} \in \delta_{-i}} \sigma(s_{-i}) u_i(s_i, s_{-i}) = v_i \tag{2.1}$$

$$(\forall i)(\forall s_i \notin \delta_i) \quad \textstyle\sum_{s_{-i} \in \delta_{-i}} \sigma(s_{-i}) u_i(s_i, s_{-i}) \leq v_i \tag{2.2}$$

$$(\forall i \in N) \quad \textstyle\sum_{s_i \in \delta_i} \sigma(s_i) = 1 \tag{2.3}$$

$$(\forall i)(\forall s_i \in \delta_i) \quad \sigma(s_i) \geq 0 \tag{2.4}$$

$$(\forall i)(\forall s_i \notin \delta_i) \quad \sigma(s_i) = 0 \tag{2.5}$$

Condition (1) asserts that each player must get the same payoff, denoted by v_i, by choosing any of his pure strategies having positive probability in his mixed strategy σ_i. Condition (2) ensures that the pure strategies $s_i \in \delta_i$ are better than pure strategies $s_i \notin \delta_i$. (3) and (4) are

the standard requirements of probability distributions. (5) asserts that the probability of all pure strategies of a player not in the support of the mixed strategy must be zero.

If we find a solution $(\sigma_1, \ \ldots \ , \sigma_n, v_1, \ \ldots \ , v_n)$ to the equations (1)–(5), then $(\sigma_1, \ \ldots \ , \sigma_n)$ is a Nash equilibrium and v_i is the expected payoff to player i in the Nash equilibrium. On the other hand, if there is no solution that satisfies (1)–(5), then there is no equilibrium with support $\times_{i \in N} \delta_i$.

Here the number of unknowns are $n + 2\sum_{i \in N} |S_i|$, where n corresponds to the variables $v_1, v_2, \ \ldots \ , v_n$. While $|S_i|$ corresponds to the variables $\sigma_i(s_i)$, $s_i \in S_i$.

We make the following observations.

- From (1) and (5), it is seen that the equations are in general non-linear form because of the term
$$\sigma(s_i) = \Pi_{j \neq i} \sigma_j(s_j)$$
- If there are only two players, then we will have only linear equations. The number of these equations will be $2 + 2|S_1| + 2|S_2|$ for each support combination.
- So even for 2 player games, the number of equations to be solved is quite large.
- If the number of players > 2, not only do we have a large number of equations, we also have to deal with non-linearity.
- For two player games, the resulting equations are said to constitute a *linear complementarity problem* (LCP).

2.7 Linear Complementarity Problem (LCP)

It is a general problem that unifies linear programming, quadratic programming, and bimatrix games. Complementary pivot algorithm is a popular algorithm for solving LCPs. Let M be a given square matrix of order n and q is a column vector in $\Re$.

In an LCP, there is no objective function to be optimized. The problem is to find

$$\begin{aligned} w &= (w_1, \cdots, w_n)^T \quad \text{and} \\ z &= (z_1, \cdots, z_n)^T \quad \text{satisfying} \\ w - Mz &= q \\ w &\geq 0 \\ z &\geq 0 \\ w_i z_i &= 0 \quad \forall i \end{aligned}$$

Computing Nash equilibria in two-person games leads to LCP.

2.8 Nonlinear Complementarity Problem (NLCP)

For $j = 1, \ \ldots \ , n$, let $f_j(z)$ be a real valued function on $\Re^n$. Let $f(z) = (f_1(z), \ \ldots \ , f_n(z))$. The problem of finding $z \in \Re$ satisfying

$$\begin{aligned} & z \geq 0 \\ & f(z) \geq 0 \\ & z_j f_j(z) = 0 \qquad f_n j = 1, \ \ldots \ , n \end{aligned}$$

is known as NLCP.

Computing Nash equilibrium in n-person games turns out to be an NLCP.

Chapter 3

Review of Relavent Work

This chapter provides the review of literature survey and points the computational complexity issues for computing Nash equilibrium points.

In this chapter we present the state-of-the-art survey of algorithms for computing Nash equilibria. The methodology followed according to one of the factors like the number of equilibria points to compute, number of players involved in the game, the type of Nash equilibria (equilibria refinement). The algorithms for solving a game involve linear and nonlinear approaches according to number of players in game. N-player games involve nonlinear methods like simplicial subdivision, for approximating fixed points of systems of inequalities of $(N-1)$-degree polynomials. McKelvey and Mclennan [19] and Stengel [35] provide a comprehensive survey of n-person and two-person games respectively

For the two-person game linear methods are applicable due to the linear nature of constraints in the problem. The ideas developed to solve game are also useful for other problems in mathematical optimization, like linear complementarity problems(LCP). To solve a LCP classical algorithm was given by Lemke and Howson (1964) [17]. The algorithm gives a constructive proof for the existence of equilibrium points in a bimatrix game. For non-degenerate games equilibrium points are found in finitely many steps. Rosenmuller [29] extended this algorithm to strategic form n-person finite games and Aggarwal [3] generated all equilibrium points using this algorithm. Shapley [32] gave a nice geometric interpretation.

For N-person games one state of the art general-purpose algorithm is the continuation method of Govindan and Wilson [11], a gradient following algorithm which is based on topological insight into the graph of the Nash equilibrium correspondence by Kohlberg and Mertens [14]. They have applied Smale's (1976) global Newton method to computing equilibria. They have even improved the performance of the method by providing a "fast star" that jumps quickly to the vicinity of the target game by iterated polynomial approximation technique [12]. They shows that all the standard algorithms for calculating Nash equilibria of N-player games actually use homotopy methods to trace equilibria in the graph of the Nash correpondence. Eaves, Eaves and

Schmedders trace the equilibria over a path of games from a starting point that is a game whose unique equilibrium is known, to the terminal point that is the target game whose equilibria are wanted. Eaves, Eaves and Scarf, Scarf and Hansen trace homotopy explicity on the vertices of the sub-simplices in a pseudo-manifold of the games space. The method of approximation via quantalresponse equilibria is developed by McKelvey and Palfrey (1995) and the interior point method developed by van den Elzen and Talman (1991).

Empirically, many games have Nash equilibria with small support (strategies with positive support). Based on this observation, Porter, Nudelman, and Shoham [27] developed a couple of simple search algorithms to find a Nash equilibrium in a normal form game. Their algorithms are based on the fact that given a support, it is fairly easy to find a Nash equilibrium consistent with the support, if one exists. The algorithms basically search over all possible supports, starting from small ones. The search terminates when a Nash equilibrium is found. There are exponential number of supports, so the algorithms could take exponential time in the number of pure strategies to terminate. But if the game has a Nash equilibrium with small support, the algorithms would find it quickly. The authors tested the algorithms on a set of games, and they performed surprisingly well, outperforming Lemke-Howson for two-player games. Sandholm, Gilpin, and Conitzer [30] developed a new approach to find the Nash equilibrium using *mixed integer programming* method. Mangasarian [18] developed an algorithm for computing all Nash equilibria of two-person games.

The main software tools for computing equilibria today is the free software package GAMBIT of McKelvey, McLennan, and Turocy [20], GAMETRACER of Koller's group, and GALA system by Koller and Pfeffer. Where Gambit uses the Normal form games, GAMETRACER for n-person normal form games and Gala provides representation and analysis tool for extensive form games. The techniques for solving systems of polynomial equations (Nash equilibria of finite games characterized as solutions to systems of multilinear equalities and inequalities) used in GAMBIT package, have been investigated independently in past.

3.1 Computational Complexity

There has been growing interest in the computational complexity of natural quetions in game theory from 1970 onwards. Rosenbaum [28] provide a detailed survey on the computational complexity of Nash equilibria. Finding a Nash equilibrium with specific characteristics seems hard, but the complexity of finding one Nash equlibrium is unknown. For two person non-zero sum games the complexity is unknown. Megiddo and Papadimitriou [21] originated a new complexity class, TFNP (Total Function for NP), to encompass such problems. There are no complexity results even for symmetric two player games [7]. Some of the complexity results are

listed below.

- Finding the best response strategy, given a game and the opponents' strategies, is in P as long as the number of players is fixed.
- For a finite game, it is possible to verify in polynomial time whether a given strategy is a best response to the opponent's strategies.
- For an arbitrary number of players, there is no polynomial time algorithm for finding a best response strategy.
- Finding Nash equilibria of n-player games with particular characteristics are NP-hard, such as finding the Nash equlibria with the maximal payoff, maximal or minimal support, containing a particular subset of strategies, and proving uniqueness of Nash equilibrium.
- For two person games, these problems are all in NP, except for uniquesness, which is in $Co - NP$.
- Counting the Nash equlibria is $\#P$ and showing the existence of Nash equilibria subject to certain constraints is NP-hard.
- Determining whether pure strategy Bayes-Nash equilibria exist is NP-hard, and showing the existence of a pure strategy Nash equlibria for a stochastic game is $PSPACE$-hard.
- A game may be transformed from normal to extensive form in linear time, transforming from extensive to normal form is an exponential function of the size of the game tree. From extensive form to sequence form is in linear to the size of game tree.
- The computational complexity of the Lemke's algorithm is unknown for the two person case, but in general case it is NP-complete [22].
- If the game is zero-sum, it can be expressed as a linear program which can be solved in polynomial time.
- Finding a sample equlibrium for Nash equilibria refinements (i.e., satisfy particular criteria) is difficult; generally, Nash equilibria refinements are found by first finding all Nash equlibria and only then imposing refinements on the set of all Nash equilibria.
- Saddle points of two person zero-sum extensive form game with perfect recall can be computed in polynomial time, but with imperfect recall computing any prudent strategy is NP-hard.

- What is the game's most desirable equilibrium? Is there a unique equilibrium? If not, how many equilibria are there? Is there exists an algorithm where a certain player plays a certain strategy? etc., all these questions are NP-hard.

Chapter 4

Algorithms Implemented for NECTAR

There are many methods for finding Nash equilibria, which depend on how many equilibria are desired (one or all), the number of players in the game (two or more), and whether the game is in strategic or extensive form. Here we present some of the algorithms which were implemented as part of the NECTAR project.

4.1 Von Neumann-Morgenstern Algorithm

Let (A, B) be a two-person game in normal form where A is an $m \times n$ matrix of payoffs for player 1 and B is an $m \times n$ matrix of payoffs for player 2. The m rows are the pure strategies of player 1 and the n columns are the pure strategies of player 2. A mixed strategy x for player 1 can be represented by the vector x of probabilities for playing pure strategies. Thus x is an m-vector fulfilling $x \geq 0$ and $l_m^T x = 1$ where l_m is an m-vector with all components equal to one. Similarly, a mixed strategy y for player 2 can be represented as an n-vector y fulfilling $y \geq 0$ and $l_n^T y = 1$.

When player 1 and player 2 use the mixed srategies x and y, their expected payoffs are $x^T Ay$ and $x^T By$, respectively. Strategy x is a *best response* to y if it maximizes the expression $x^T(Ay)$, for fixed y. Similarly a best response y of player 2 to x maximizes $(x^T B)y$. An *equilibrium* is a pair (x, y) of mutual best responses.

If the strategy y of player 2 is known, then a best response x of player 1 of y is a solution to the following LP:

$$\max \quad x^T(Ay)$$

$$\text{subject to} \quad x^T l = 1 \tag{4.1}$$

$$x \geq 0$$

The dual of this LP, which has only a single dual variable u:

$$\min \quad u$$

$$\text{subject to} \quad lu \geq Ay \tag{4.2}$$

Both LPs are feasible. By strong duality, they have the same optimal value. That is, maximal expected payoff $x^T Ay$ to player 1 is the same as the smallest u fulfilling $lu \geq Ay$. The latter is obviously the maximum of the entries of the vector Ay, which are the expected payoffs to player 1 for his strategies.

Consider now a *zero-sum game*, where $B = -A$. Player 2, when choosing y, has to assume that his opponent plays rationally and maximizes $x^T Ay$. This maximum payoff to player 1 is the optimal value of the LP (4.1), which is equal to the optimal value u of the dual LP (4.2). Player 2 is interested in minimizing u by his choice of y. The constraints are linear in u and y even if y is treated as a variable, which has to represent a mixed strategy. So a minmax strategy y of player 2 (minimizing the maximum amount he has to pay) is a solution to the mixed LP with variables u, y

$$\begin{aligned} \min \quad & u \\ \text{subject to} \quad & l_n^T y = 1 \\ & l_m u - Ay \geq 0 \\ & y \geq 0 \end{aligned} \tag{4.3}$$

The dual of the LP with variables v and x, which has the form

$$\begin{aligned} \max \quad & v \\ & x^T l_m = 1 \\ & v l_n^t - x^T A \leq 0 \\ & x \geq 0 \end{aligned} \tag{4.4}$$

Thus, a zero-sum game with payoff matrix A for player 1 has the equlibrium (x, y) iff u, v is an optimal solution of the LP and v, x is an optimal solution of its dual LP. Thereby, u is the maxmin payoff to player 1, v is the minmax payoff to player 2, and $u = v$, denoting the value of the game.

4.2 Lemke and Howson Algorithm

The most commonly-used algorithm for finding a Nash equilibrium in a two-player game is the Lemke and Howson's [17] algorithm, which is a special case of Lemke's method for solving linear complementarity problems. The Lemke-Howson algorithm is a complementary pivoting algorithm, where an arbitrary selection of an action for the first player determines the first pivot, after which every successive pivot is determined uniquely by the current state of the algorithm, until an equilibrium is found. Thus, each action for the first player can be thought of as defining a path from the starting point (the extraneous solutions of all players assigning probability zero to all strategies) to a Nash equilibrium. Savani and von Stengel [31] showed that Lemke-Howson algorithm takes exponential time even in the best case.

4.2.1 Shapley's Interpretation

Shapley [32] gave an intuitive explanation of this algorithm which is easily visualized for games of small dimension. Stengel [35] has given nice explanation with example about Shapley's [32] interpreation. Let, the given bimatrix game is (A, B) and has size $m \times n$. The sets of pure strategies are $I = \{1, \ldots, m\}$ for player 1 and $J = \{m+1, \ldots, m+n\}$ (easier distinction) for player 2. The sets of mixed strategies for player 1 and 2 are denoted by

$$X = \{x \in \Re^m \mid l_m^T x = 1,\ x \geq 0\}$$

$$Y = \{y \in \Re^n \mid l_n^T y = 1,\ y \geq 0\}$$

Let, a *label* is any element of $I \cup J$. The labels are used to mark the points in X and Y, as follows. The $Y(i)$ for $i \in I$ where the pure strategy i of player 1 is a best response and $X(i)$ is the set of strategies that are unplayed or have probability zero. Similarly, for player 2, the sets $X(j)$ and $Y(j)$ are the sets of pure strategies which are best reponses or unplayed strategies. These can be denoted by

$$X(i) = \{x \in X \mid x_i = 0\}$$

$$X(j) = \{x \in X \mid b_j x \geq b_k x \ \ \forall k \in J\}$$

$$Y(i) = \{y \in Y \mid a_i y \geq a_k y \ \ \forall k \in I\}$$

$$Y(j) = \{y \in y \mid y_j = 0\}$$

Let y has label k if $y \in Y(k)$, for $k \in I \cup J$, and define the set of labels of y as

$$L(y) = \{k \in I \cup J \mid y \in Y(k)\}$$

similary for x

$$L(x) = \{k \in I \cup J \mid x \in X(k)\}$$

With the help of these labels, it is easy to identify the equilibria of the game. These are exactly the pairs (x, y) in $X \times Y$ that are *completely labeled*, that is, $L(x) \cup L(y) = I \cup J$. The reason is that in equilibrium, a pure strategy is either a best response or has probability zero (or both, which can only happen if the game is degenerate). A *missing* label k represents a pure strategy (of either player) that has positive probability but is not a best response, which is forbidden in equilibrium.

This algorithm is combinatorial and can it its general form be described in graph-theoretic terms. Let G_1 be the graph whose vertices are those points x with $|L(x)| = m$. Except for $\mathbf{0}$ in $X(I)$, these points belong to X. Any two such vertices x, x' are joined by an edge if they differ in exactly one label. The set $K = L(x) \cap L(x')$ of the $m-1$ labels that x and x' have in common define the line segment $x(K)$ which represents the edge in $\Re^m$. The vertices x and x' are the endpoints of this edge since it is not possible to get additional labels by taking convex combinations. Similarly, let G_2 be the graph with vertices y where $|L(y)| = n$, and edges joining those vertices that have $n-1$ common labels. Finally, consider the *product graph* G of G_1 and G_2. Its vertices are (x, y) where x is a vertex of G_1, and y is a vertex of G_2. The edges of G are given by $\{x\} \times e_2$ for vertices x of G_1 and edges e_2 of G_2, or $e_1 \times \{y\}$ for edges e_1 of G_1 and vertices y of G_2.

The algorithm goes along; Starting from a completely labeled vertex $(0, 0)$ (artificial equilibrium) of G, the edges in G of the path that is followed are alternatingly edges of G_1 and G_2, leaving the vertex in the other graph fixed. The first edge is defined by the label k that is dropped initially, the subsequent edges are defined by the duplicate labels encountered along the way. It stops at a vertex of G which is completely labeled. The (x, y) is the equilibrium point where x and y are the mixed strategies of player 1 and player 2. The expected utilities of player1 and player 2 at this equilibrium are $x^T Ay$ and $x^T By$ respectively.

4.2.2 Linear Complementarity

A best response x of player 1 against the mixed strategy y is a solution to the LP (4.1). By strong duality, a feasible solution x is optimal iff there is a dual solution u fulfilling $l_m u \geq Ay$

and $x^T(Ay) = u$, that is, $x^T(Ay) = (x^T l_m)$ or equivalently

$$x^T(l_m u - Ay) = 0 \tag{4.5}$$

This condition states that x and $l_m u - Ay$ are orthogonal. Because these vectors are nonnegative, they have to be *complementary* in the sense that they cannot both have positive components in the same position. This characterization of an optimal primal-dual pair of feasible solutions is known as "complementary slackness" in linear programming [22]. Here, x being a mixed strategy, has at least one positive component, so the respective component of $l_m u - Ay$ is zero and u is the maximum of the entries of Ay. Any pure strategy i, $1 \leq i \leq m$, of player 1 is a best response to y iff the ith component of the slack vector $l_m u - Ay$ is zero. So condition (4.5) amounts to the following well-known Nash's property [24]: A strategy x is a best response to y iff it only plays pure strategies that are best responses with positive probability.

For player 2, strategy y is a best response to x iff it maximizes $(x^T B)y$ subject to $l_n^T y = 1$, $y \geq 0$. This gives an LP analogous to (4.1). Its dual LP has a single scaler v as variable and says: minimize v subject to $l_n v \geq B^T x$. A primal-dual pair y, v of feasible solutions is optimal iff, analogous to (4.5),

$$y^T(l_n v - B^T x) = 0 \tag{4.6}$$

The vector pair (x, y) is an equilibrium of the bimatrix game (A, B) iff there are reals u, v such that

$$\begin{gathered} l_m^T x = 1 \\ l_n^T y = 1 \\ l_m u - Ay \geq 0 \\ l_n v - B^T x \geq 0 \\ x, y \geq 0 \end{gathered} \tag{4.7}$$

and (4.5), (4.6) hold.

The constraints (4.7) are linear in the variables u, v, x, y. The orthogonility conditions (4.5) and (4.6) state that the nonnegative vector of slacks for these constraints, $(l_m^T x - 1, l_n^T y - 1, l_m u - Ay, l_n v - B^T x)$, is complementary to the vector (u, v, x, y) of variables. Such a problem is called a *linear complementarity problem*(LCP).

4.2.3 Implementation

This algorithm is implemented by iteratively computing *basic solutions* to a system of linear equations equivalent to (4.7). These basic solutions represent the graph vertices used above, and moving along a graph edge is a basis change known as *pivoting*.

Pivoting is well known as the main step of the Simplex algorithm for solving linear programs given in equality form. The linear constraints (4.7) are also converted to equality form by introducting vectors of slack variables: If there is z in $\Re^n$ with

$$l_m^T x = 1$$

$$-l_n v + B^T x + I_n z = 0$$

$$x, \quad z \geq 0$$

(where I_n is the $n \times n$ identity matrix), and w in $\Re^m$ with

$$l_n^T y = 1$$

$$-l_m u + Ay + I_m w = 0$$

$$y, \quad w \geq 0$$

then u, v, x, y is a solution. Where x and y are the mixed strategies of player 1 and player 2 at the equilibrium. The expected payoffs to player 1 and player 2 are $x^T Ay$ and $x^T By$ respectively.

4.3 A Simple Search Method

Porter, Nudelman and Shoham [27] presented two simple search methods for computing a sample Nash equilibrium in a normal-form game; one for 2-player games and one for n-player games. These algorithms bias the search towards supports that are small and balanced, and employ a backtracking procedure to efficiently explore these supports. They applied simple heuristic methods which are quite effective, and significantly outperformed the relatively sophisticated algorithms developed in the past. They tested these algorithms, along with existing ones, extensively through the use of a new computational testbed GAMUT developed by Nudelman, *et. al*, [2], [26].

The basic idea behind these two search algorithms is simple. Computing a Nash equilibrium is a complementarity problem, computing whether there exists a Nash equilibrium with a particular support for each player is a relatively easy feasibility program. These algorithms explore the space of support profiles using a backtracking procedure to instantiate the support for each

player separately. After each instantiation, they prune the search space by checking for actions in a support that are strictly dominated, given that the other agents will only play actions in their own supports.

Both of the algorithms are biased towards simple solutions through their preference for small supports. These algorithms are inspired by the procedure described by Dickhaut and Kaplan [9] for finding all Nash equilibria which enumerates all possible pairs of supports for a two-player game. For each pair of supports, it solves a feasibility program to check whether there exists a Nash equilibrium consistent with this pair. A similar enumeration method was suggested earlier by Mangasarian [18], based on enumerating vertices of a polytope.

Equations (1)–(5) of chapter 2 are used to test the feasibility of a given solution being a Nash equilibrium. For the two-person case, there are three keys to efficiency. The first two are the factors used to order the search space, favoring support size that are balanced and small. The third key to this algorithm is that it separately instantiates each players' supports, making use of what they called conditional (strict) dominance to prune the search space.

An action $s_i \in S_i$ is conditionally dominated, given a profile of sets of available actions $\delta_i \subseteq S_{-i}$ for the remaining agents, if the following condition holds: $\exists s'_i \in S_i \quad \forall s_{-i} \in \delta_{-i} : u_i(s_i, s_{-i}) < u_i(s'_i, s_{-i})$.

The preference on small support sizes amplifies the advantages of checking for conditional dominance. This algorithm is complete, because it considers all support size profiles and only prunes actions that are strictly dominated.

Both algorithms for 2-player and n-player check whether the input game has a pure strategy equilibrium. This step can be performed very fast even on large games. Both algorithms employs backtracking approaches (augmented with pruning) to search the space of support profiles, favoring supports that are small and balanced.

They tested these algorithms over various game distribution data and showed that 2-player algorithm outperformed Lemke-Howson on all distributions and n-player algorithm outperformed both Simplicial Subdivision [33] and Govindan-Wilson [11] algorithms.

4.4 A Mixed Integer Programming Based Method

Sandholm, Gilpin, and Conitzer [30] have presented a new approach to find the Nash equilibrium using *mixed integer programming* method. This is the first mixed integer program formulation for finding Nash Equilibria in normal form games. This paper has four types of formulations for computing Nash equilibrium points for which the first formulation gives exactly the equilibria of the game, whereas the other three formulations have feasible solutions other than the equilibria as well. On various kinds of data, first formulation showed better performance than the rest.

The first formulation is represented in this report.

Let, the *regret* of pure strategy s_i is the difference in utility for player i between playing an optimal strategy and playing s_i. In any equilibrium, every pure strategy is either played with probability 0 or has 0 regret. Also, any vector of mixed strategies for the players where every pure strategy is either played with probability 0, or has 0 regret, is an equilibrium.

Formulation 1: Only equilibria are feasible

Find $p_{s_i}, v_i, v_{s_i}, r_{s_i}, b_{s_i}$ such that

$$(\forall i) \sum_{s_i \in S_i} \quad p_{s_i} \quad = 1 \tag{4.8}$$

$$(\forall i)(\forall s_i \in S_i) \quad v_{s_i} \quad = \sum_{s_{-i} \in S_{-i}} p_{s_{-i}} u_i(s_i, s_{-i}) \tag{4.9}$$

$$(\forall i)(\forall s_i \in S_i) \quad v_i \quad \geq v_{s_i} \tag{4.10}$$

$$(\forall i)(\forall s_i \in S_i) \quad r_{s_i} \quad = v_i - v_{s_i} \tag{4.11}$$

$$(\forall i)(\forall s_i \in S_i) \quad p_{s_i} \quad \leq 1 - b_{s_i} \tag{4.12}$$

$$(\forall i)(\forall s_i \in S_i) \quad r_{s_i} \quad \leq U_i b_{s_i} \tag{4.13}$$

where $p_{s_i} \geq 0, v_i \geq 0, v_{s_i} \geq 0, r_{s_i} \geq 0, b_{s_i} \in \{0, 1\}$

Here, b_{s_i} is a binary variable which is set to 1, if probability placed on the strategy must be 0, otherwise it is set to 0. v_i indicates the highest possible expected utility that player i can obtain given the other player's mixed strategy and the variable v_{s_i} is the expected utility of playing that strategy. The variable r_{s_i} is the regret of playing s_i. The constant U_i is the maximum difference between two utilities in the game for player i.

Constraints (4.8)–(4.11) ensure that the p_{s_i} values constitute a valid probability distribution and define the regret of a strategy. (4.12) ensures that b_{s_i} can be set to 1 only when no probabilities placed on s_i. (4.13) ensures that the regret of a strategy equals 0, unless $b_{s_i} = 1$, in which the regret can never exceed U_i.

In their results, formulation 1 performed significantly better than the other three formulations. They tested these formulations on CPLEX 9.0 [1] (a commercial MIP software package). The solving method in CPLEX is a branch-and-bound algorithm with several sophisticated techniques incorporated. The authors used different techniques to improve the performance of

their algorithm by defining objective function to the first formulation which helps for bias on the search and they also incorporated some strategies over node selection in branch-and-bound algorithm, disabling heuristics at nodes and applying cutting plane techniques.

This algorithm outperforms Lemke and Howson [17] but not Porter, Nudelman and Shoham [27] on GAMUT [2] data. Most of GAMUT generated games have equilibria with small size supports. Whereas in some games which have equilibrium points at medium size supports, this algorithm drastically outperforms Porter, Nudelman, Shoham [27] but not Lemke and Howson [17]. In cases, where we have to find the optimal Nash equilibrium (for example social welfare), this algorithm outperforms both Mangasarian [18] and Porter, Nudelman and Shoham [27] by 2–3 orders of magnitude.

4.5 Mangasarian Algorithm

The problem of finding all equilibria can be phrased as a vertex enumeration problem for polytopes.

4.5.1 Introduction

A *polyhedron* H is a subset of $\Re^d$ defined by a finite number of linear inequalities. If the dimension of H is d, then H is called *full-demensional* A *polytope* is a bounded polyhedron. A *face* of a polyhedron H is a subset of H of the form $\{x \in H \mid c^T x = c_0\}$ where $c^T x \leq c_0$ is a valid inequality for H (holds for all x in H), for some $c \in \Re^d, c_0 \in \Re$. A face of dimension zero (or its unique element) is called *vertex* of H. This is the same as an *extreme point* of H, that is, a point in H not representable as a convex combination of other points in H. A face of dimension one is called an *edge* of H. A face of dimension $d-1$ is called a *facet* of H if H has dimension d. If $H = \{x \in \Re^d \mid Ax \leq b\}$ for some matrix A and vector b, and a_i is a row of A and b_i the corresponding component of b, and $x \in H$, then the inequality $a_i x \leq b_i$ is called binding for x if $a_i x = b_i$.

4.5.2 The Algorithm

Mangasarian [18] proposed an algorithm to compute all extreme equilibrium points of a bimatrix game. Consider a bimatrix game with the $m \times n$ payoff matrices A and B to player 1 and player 2 respectively. let the $m \times 1$ probability vector x be a mixed strategy of player 1, the $n \times 1$ probability vector y a mixed strategy of player 2. α be the expected payoff to player 1 and β be the expected payoff to player 2. A necessary and sufficient condition that $(x^0, y^0, \alpha^0, \beta^0)$ be an equilibrium point is that it is a solution of the programming problem

$$\max_{x,y,\alpha,\beta}\{x^T(A+B)y-\alpha-\beta \mid (x,\beta)\in S,(y,\alpha)\in T\},$$

where S and T are the convex polyhedral sets

$$S=\{(x,\beta) \mid B^Tx-\beta l\leq 0,\ e^Tx=1,\ x\geq 0\}$$

$$T=\{(y,\alpha) \mid Ay-\alpha e\leq 0,\ l^Ty=1,\ y\geq 0\}$$

Where e and l are $n\times 1$ and $m\times 1$ vectors of ones respectively. It is shown that the extreme equilibrium points of a bimatrix game which are finite in number and all other equilibrium points can be determined from these extreme equilibrium points.

The algorithm for determining all extreme equilibrium points is as follows,

- Find all the vertices of convex polyhedral sets of S and T.
- The extreme equilibrium points are those vertices of S and T that satisy $x^T(A+B)y-\alpha-\beta=0$.

4.5.3 Vertex Enumeration

D. Avis and K. Fukuda have [6] described an reverse search algorithm for computing all the vertices of a polyhedron described by a system of linear inequalities. The reverse search algorithm works as follows. Let, for a system of m linear inequalities defining a d-dimentional polyhedron in $\Re^d$ and a vertex of that polyhedron given by the indices of d inequalities whose bounding hyperplanes intersect at the vertex. These indices define a *cobasis* for the vertex. The complementary set of $m-d$ indices are called a *basis*. For any given linear objective function, the simplex method generates a path between adjacent bases (or equivalently cobases) which are those differing in one index. The path is terminated when a basis of a vertex maximizing this objective function is found. The path is found by pivoting, which involves interchanging one of the hyperplanes defining the current cobasis with one in the basis. The path chosen from the initial given basis depends on the pivot rule used, which must be finite to avoid cycling. This implementation used Bland's least subscript rule. The set of all such paths from all bases of the polyhedron, can be represented by a spanning forest of the graph of adjacent bases of the polyhedron. The root of each subtree of the forest is a basis of an optimum vertex. The reverse search algorithm start at each root and traces out its subtree in **depth first order** by *reversing* the pivot rule.

A revised version [5] resolves degeneracy by use of the well-known lexicographic pivot selection rule for the simplex method. This rule is defined for a subset of the bases, known as lex-positive. The subgraph of lex-positive bases forms a connected subgraph of the basis graph

which covers all vertices of the polyhedron. Furthermore an objective function can be chosen so that the simplex method initiated at any lex-positive basis terminates at a unique lex-positive optimum basis. The reverse search method initiates at this basis and reverse the lexicographic pivot rule and generates a spanning tree of the graph of all lex-positive bases.

A remarkable feature of the algorithm is that no additional storage is needed at intermediate nodes in the tree. Going deeper in the tree, all valid "reverse" pivots in order by basic index from any given intermediate node are explored. For backtracking, it uses the pivot rule to return it to the parent node along with the current pivot indices. From there it is simple to continue by considering the next basic index as a "reverse" pivot, etc. The algorithm is therefore non-recursive and requires no stack or other data structure.

4.6 Sequence Form

4.6.1 Introduction

The sequence form is a matrix scheme similar to the normal form but where pure strategies are replaced by sequences of consecutive moves. Rather than planning a move for every information set h_i of the player i, looks at each leaf of the game tree and considers the choices he needs to make so that the leaf can be reached in the game. These choices are prescribed by the respective *play*, i.e., path from the root to the leaf, whenever that path goes through an information set h_i of the player i. They represent a "sequence" that will be considered instead of a pure strategy.

Let S_i be the set of sequences (of moves) of player i. Since player i has perfect recall, any sequence σ in S_i is either the empty sequence $\emptyset$ or uniquely given by its last move c at the information set h of player i, that is, $\sigma = \sigma_h c$, so

$$S_i = \{\emptyset\} \cup \{\sigma_h c \mid h \in H,\ c \in C_h\}$$

This implies that the number of sequences of player i, apart from the empty sequence, is equal to his total number of moves, that is, $|S_i| = 1 + \sum_{h \in H_i} |C_h|$. This number is linear in the size of the game tree.

The large number of mixed strategy probabilities does not arise when using behavior strategy probabilities (2.2.3). Let β_1 and β_2 denote behaviour strategies of the two players, and let β_0 be the known behaviour of the chance player. Let $a(t)$ and $b(t)$ denote the payoffs to player 1 and player 2, respectively, at a leaf t of the tree. The probability of reaching t is the product of move probabilities on the path to t. The expected payoff to player 1 is therefore

$$\sum_{leaves\ t} a(t)\beta_0[\sigma_0(t)]\beta_1[\sigma_1(t)]\beta_2[\sigma_2(t)]$$

and the expected payoff to player 2 is the same expression with $b(t)$ instead of $a(t)$. However, the expected payoff is nonlinear in terms of behavior strategy probabilities $\beta_i(c)$ since the terms $\beta_i[\sigma_i(t)]$ are products of $\beta_i(c)$ for all $c \in \sigma_i$.

The realization probabilities $\beta_i[\sigma]$ can be represented by the functions of *sequences* σ in S_i. They can also be defined for mixed strategies μ_i of player i, which choose each pure strategy π_i of player i with probability $\mu_i(\pi_i)$. Under π_i, the realization probability of σ in S_i is $\pi_i(\sigma)$, which is equal to one if π_i prescribes all the moves in σ and zero otherwise. Under μ_i, the realization probability of σ is

$$\mu_i[\sigma] = \sum_{\pi_i} \mu_i(\pi_i)\pi_i[\sigma] \tag{4.14}$$

For player 1, this defines a map x from S_1 to $\Re$ by $x(\sigma) = \mu_1[\sigma]$ for $\sigma \in S_1$. We call x the *realization plan* of μ_1 or a realization plan for player 1. A realization plan for payer 2, similarly defined on S_2 by a mixed strategy μ_2, is denoted y. Realization plans have two important properties

- A realization plan x of a mixed strategy of player 1 fulfills $x(\sigma) \geq 0$ for all $\sigma \in S_1$ and

$$x(\emptyset) = 1 \text{ and } \sum_{x}(\sigma_h c) = x(\sigma_h) \text{ for all } h \in H_1 \tag{4.15}$$

  Conversely, and $x : S_1 - > \Re$ with these properties is the realization plan of a behavior strategy of player 1, which is unique except at irrelevant information sets. A realization plan y of player 2 is charcterized analogously.

- Two mixed strategies μ_i and μ_i' of player i are realization equivalent iff they have the same realization plan, that is, iff $\mu_i[\sigma] = \mu_i'[\sigma]$ for all $\sigma \in S_i$.

- For a player with perfect recall, any mixed strategy is realization equivalent to a behaviour strategy.

A realization plan represents all the relevant strategic information of a mixed strategy. This compact information is obtained with the linear map in (4.14). This map assigns to any mixed strategy μ_i, regarded as a tuple of mixed strategy probabilities $\mu_i(\pi_i)$, its realization plan, regarded as a tuple of realization probabilities $\mu_i[\sigma]$ for σ in S_i. The simplex of mixed strategies is thereby mapped to the polytope of realization plans defined by the linear constraints (4.15). The vertices of this polytope are the realization plans of pure strategies. Here, the number of these vertices may be exponential. The number of defining inequalities and the dimension of the polytope, however, is linear in the tree size. For player i this dimension is the number $|S_i|$

of variables minus the number $1 + |H_i|$ of equations (4.15) (which are linearly independent), so it is $\sum_{h \in H_i}(|C_h| - 1)$.

Consider realization plans as vectors in $x \in \Re^{|S_1|}$ and $y \in \Re^{|S_2|}$, that is, $x = (x_\sigma)_{\sigma \in S_1}$ where $x_\sigma = x(\sigma)$, and similarly $y = (y_\tau)_{\tau \in S_2}$. The linear constraints (4.15) are denoted by

$$Ex = e, \;\; x \geq 0 \;\text{ and }\; Fy = f, \;\; y \geq 0 \tag{4.16}$$

where E and F are called *constraint matrices*.

Define the *sequence form payoff* matrices A and B, each of dimension $|S_1| \times |S_2|$, as follows. For $\sigma \in S_!$ and $\tau \in S_2$, let the matrix entry $a_{\sigma\tau}$ of A be defined by

$$a_{\sigma\tau} = \sum_{\text{leaves } t:\ \sigma_1(t)=\sigma,\ \sigma_2(t)=\tau} a(t)\beta_0[\sigma_0(t)]$$

The matrix entry of B is the term with b instead of a.

4.6.2 Computing Equilibria

The computation of equilibria with the sequence form uses the same approach as presented for the normal form. Consider a fixed realization plan y of player 2. An optimal realization plan x of player 1 maximizes his expected payoff $x^T(Ay)$, subject to $Ex = e,\ x \geq 0$.

If the game is *zero-sum*, then player 2 is interested in minimizing the optimal payoff to player 1, which is the value of this dual LP. So a minmax realization plan y of player 2 solves the LP with variables $u,\ y$ which is analogous to (4.3),

$$\text{minimize} \quad e^T u$$

$$\text{subject to} \quad Fy = f'$$

$$E^T u - Ay \geq 0,$$

$$y \geq 0$$

It is easy to see that the dual of this LP, analogous to (4.4), determines a maxmin realization plan of player 1.

For non-zero-sum games, it is analogous to Section (4.2.2). For player 2, the realization plan y is a best response to x iff it maximizes $(x^T B)y$ subject to $Fy = f,\ y \geq 0$. Its dual LP has the vector v of variables and says: minimize $f^T v$ subject to $F^T v \geq B^T x$. The optimality of this primal-dual pair of LPs, as well as that for player 1, is characterized by complementarity conditions analogous to (4.5), (4.6) and (4.7). This is as follows

Consider the two-person extensive form game with sequence form payoff matrices $A,\ B$ and constraint matrices $E,\ F$ in (4.16). Then the pair (x, y) of realization plans defines an equilibrium iff there are vectors $u,\ v$ such that

$$\begin{aligned} Ex &= e \\ Fy &= f \\ E^T u - Ay &\geq 0 \\ F^T v - B^T x &\geq 0 \\ x,\quad y &\geq 0 \end{aligned}$$

and

$$x^T(E^T u - Ay) = 0 \qquad y^T(F^T v - B^T x) = 0 \tag{4.17}$$

The size of this LCP with variables u, v, x, y is linear in the size of the game tree.

4.7 Global Newton Method

Govindan and Wilson [11], presented a new algorithm for computing Nash equilibria of finite games using Kehlberg and Mertens [14] srtructure theorem. They have shown that a homotopy method can be represented as a dynamical system and implemented by Smale's global Newton method. The methods that facilitates the construction is a fundamental topological property of the graph of the Nash equilibrium correspondence.

The structure theorem of Kohlberg and Mertens(1986, Theorem 1), [14] shows that this graph is homeomorphic to the space of games. The homeomorphism enables a homotopy method to be applied in a subspace of games whose dimension is the same as the number of pure strategies. The path of the homotopy can traced by a dynamical system using the global Newton method, then after wards one applies the inverse of the homeomorphism to obtain the players' mixed strategies at the equilibrium. One of implications of the theorem is that, above each generic ray emanating from the true game (represented as a point in a Euclidean space), the graph of the equilibrium correspondence is a 1-dimensional manifold; further, at a sufficient distance from the true game there is a unique equilibrium. Therefore, starting from a sufficiently distant game along any generic ray, one can traverse the line segment to the true game, tracing the 1-dimensional manifold of equilibria along the way, to find an equilibrium of the true game at the terminus. For practical computation, however, one must eliminate the huge discrepancy between the dimensions of the spaces of games and mixed strategies. The structure theorem

again provides an answer, the homeomorphism shows that it is sufficient to consider a subspace of deformed games of the same dimension as the space of mixed strategies, and it provides a simple parameterization, Indeed, it shows that one can represent both deformed games and their equilibria within this subspace, deferring until last the calculation of the true game's equilibrium strategies by applying the homeomorphism. But now, since the two dimensions are the same, the global Newton method can be used as the engine for calculations.

All this leads to a formulation in which one traces the 1-dimensional manifold above a generic ray (emanating from the true game positioned at the origin) via the dynamical system of the global Newton method. The result is an explicit alogirithm that accomplishes the reverse deformation suggested by the homotopy prinicple. The analysis establishes the conditions for application of the homotopy method and then shows how it can be implemented by the global Newton method[GNM]. The authors followed usual convention that GNM is formulated as a dynamical system with a continuous time parameter. The theoretical simplicity comes at some cost because the technical issue of differentiability must be addressed. Formally, a dynamical system is constructed that involves restarts. A byproduct of this approach is a generalization to many-player games of Shapley's [32] demonstration that the index is always +1 for an equilibrium at a terminal point of the Lemke-Howson algorithm for generic n-player games. In this case a terminal point of GNM has index +1, while equilibria with index -1 are locally unstable and obtained by GNM with the time direction reversed.

4.8 Correlated Equlibrium

There are many games, like the Prisoners' Dilemma, in which the Nash equilibria yield very low payoffs for the players, relative to other nonequilibrium outcomes. In such situations, the players would want to transform the game such that to extend the set of equilibria to include better outcomes. Players might seek to transform a game by trying to communicate with each other and coordinate their moves, perhaps even by formulating contractual agreements.

But, there are many situations in which players cannot commit themselves to binding contracts. Players' strategies might be unobservable to the legal enforcers of contracts, etc., In such situations, it might be possible for the players to communicate and coordinate with each other. A game is said to be *with communication* if, in addition to the strategy options explicitly specified in the structure of the game, the players have a very wide range of implicit options to communicate with each other. Aumann[4] showed that the solutions for game with communication may have remarkable properties, even without contracts.

Example : Consider the following familiar two-person non-zero sum game.

2, 1	0, 0
0, 0	1, 2

There are exactly 3 Nash equilibrium points: 2 in pure strategies, yielding (2, 1) and (1, 2) respectively, and one in mixed strategies, yielding (2/3, 2/3). The payoff vector (3/2, 3/2) is not achievable at all in mixed strategies. It is, however, achievable in "correlated" strategies, as follows; One fair coin is tossed; if it falls heads, player 1 and 2 play top and left respectively; otherwise, they play bottom and right.

The interesting aspect of this procedure is that it not only achieves (3/2, 3/2), it is also in equilibrium; neither player can gain by a unilateral change. From the above example by the use of correlated strategies one can achieve a payoff vector that is in equilibrium, which is outside the convex hull of the Nash equilibrium payoffs. In fact, it is better for all players than any Nash equilibrium payoff.

In general, for any finite strategic-form game $G = (N, (S_i)_{i\in N}, (u_i)_{i\in N})$, a mediator who was trying to help coordinate the players' actions would need to let each player i know which strategy in S_i was recommended for him. When the mediator can communicate separately and confidentially with each player, however, no player would have to be told the recommendations for any other players. Without contracts, player i would then be free to choose any strategy in S_i, after hearing the mediator's recommendation. So in the game with mediated communication, each player i would actually have an enalrged set of communication strategies that would include all mappings from S_i into S_i, each of which represents a possible rule for choosing an element of S_i as a function of the mediator's recommendation in S_i.

Now, suppose that is common knowledge that the mediator will determine his recommendations according to the probability distribution μ in $\Delta(S)$; so $\mu(s)$ denotes the probability that any given pure strategy profile $s = (s_i)_{i\in N}$ would be recommended by the mediator. Then it would be an equilibrium for all players to obey the mediator's recommendations iff

$$U_i(\mu) \geq \sum_{s\in S} \mu(s) u_i(s_{-i}, \delta_i(s_i)), \quad \forall i \in N, \ \forall \delta_i : S_i \to S_i$$

The above condition is equivalent to the following system of inequalities:

$$\sum_{s_{-i}\in S_{-i}} \mu(s)(u_i(s) - u_i(s_{-i}, s_i')) \geq 0$$

$$\forall i \in N, \ \forall s_i \in S_i, \ \forall s_i' \in S_i \tag{4.18}$$

To interpret this inequality, given any s_i dividing the left-hand side by $\sum_{s_{-i}\in S_{-i}} \mu(s)$, would make it equal to the difference between player i's conditionally expected payoff from obeying the mediator's recommendation and his conditionally expected payoff from using the strategy

s_i', given that the mediator has recommended s_i, when all other players are expected to obey the mediators's recommendations. Thus, above condition asserts that no player i could expect to increase his expected payoff by using some disobedient strategy s_i' after getting any recommendation s_i from the mediator. These inequalities can be called *strategic incentive constraints*, because they represent the mathematical inequalities that a mediator's correlated strategy must satisfy to guarantee that all players could rationally obey his recommendations.

The set of correlated equilibria is a compact and convex set, for any finite game in strategic from. Furthermore, it can be characterized by a finite collection of linear inequalities, because a vector μ in $\Re^N$ is a correlated equilibrium iff it satisfied the strategic incentive constraints and the following *probability constraints*.

$$\sum_{s \in S} \mu(s) = 1 \;\; and \;\; \mu(s) \geq 0 \;\; \forall s \in S \tag{4.19}$$

By maximizing a linear objective ($\sum_{i \in N} U_i(\mu)$) subject to linear constraints (4.18) and (4.19), finds the correlated equilibrium that maxmizes the sum of the players' expected payoffs in G and this is linear programming problem even for n player game.

Chapter 5

NECTAR: Architecture and Design

When building interactive applications as with other programs, modularity of components gives enormous benefits. Isolating functional units from each other makes it easier for the application designer to understand and modify each unit, without having to know much about the other units. This chapter describes about the architecture of NECTAR and various design patterns used for NECTAR.

5.1 MVC Architectural Pattern

Many of the GUI based software projects use Model-View-Controller (MVC) architecture. MVC programming is the application of three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (the controller). Smalltalk-80 is the most successful application in which the user interface environment has been developed using MVC architecture [16].

Figure 5.1 shows the block diagram of MVC architecture. A typical interaction cycle is as follows. The user takes some input action and the active controller notifies the model to change itself accordingly. The model carries out the prescribed operations, possibly changing its state, and broadcasts to its dependents (views and controllers) that it has changed, possibly telling them the nature of the change. Views can then inquire of the model about its new state, and update their display if necessary. Controllers may change their method of interaction depending on the new state of the model.

The Swing has slightly different architecture called a *separable model architecture* which is similar to MVC architecture. In this architecture the model part of a component is treated as a separate element, just as the MVC design does. But Swing collapses the view and controller parts of each component into a single UI (user-interface) object.

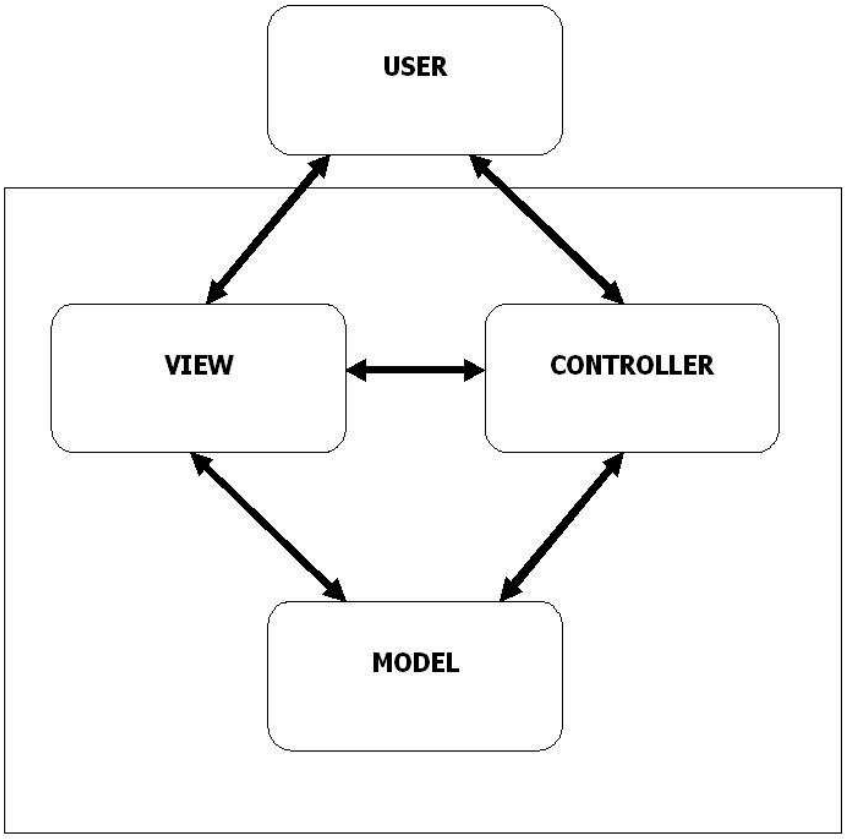

Figure 5.1: Model-View-Controller Architecture Pattern

5.2 NECTAR: Software Architecture

The NECTAR project is divided into modules logically and physically. The architecture diagram of NECTAR is shown in Figure 5.2. The following subsections describes about each module.

5.2.1 Graphical User Interface Module

The *Graphical User Interface* (GUI) module is shown in Figure 5.3. The GUI module provides graphical interface to the end user. The objects in this module are organised effectively with the help of singleton, mediator and command design patterns which are explained in the following section. Here, the mediator object controls the whole state of the GUI. The GUI module is aware of only the *Nectar* interface, it is not aware of any other modules as shown in the architecture diagram 5.2. Because of this architecture, the other modules can be changed or modified without affecting the GUI Module.

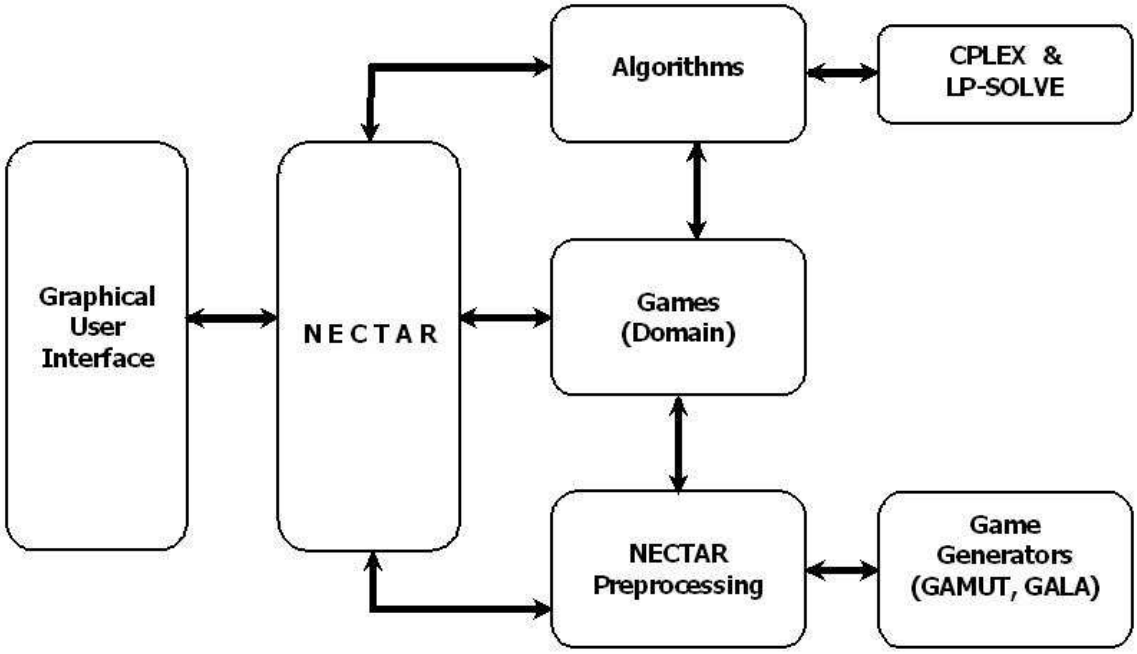

Figure 5.2: Architecture diagram of NECTAR

5.2.2 Nectar Module

The Nectar module is shown in Figure 5.4. The Nectar module is central to the whole system. It isolates the GUI module from other modules in order to get the flexibility and modularity. The GUI module interacts with this module only. This module interacts with the *preprocessing* module for generating the *Game*(domain) objects *i.e.* normal-form, extensive-form, sequence-form game representations and interacts with the *algorithms* module for computing the Nash equilibrium points for the given games. This is implemented with *facade* design pattern explained in the next section.

5.2.3 Games Module

The *Games* module is shown in Figure 5.5. This module can be treated as *domain*, because all the operations are performed on these objects. This module contains the objects which represents 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. This module is implemented with *factory method* design pattern. With the help of this design pattern, we can modify existing concrete objects or develop the new concrete objects without affecting of the other modules. The *factory method* design pattern will be explained in the

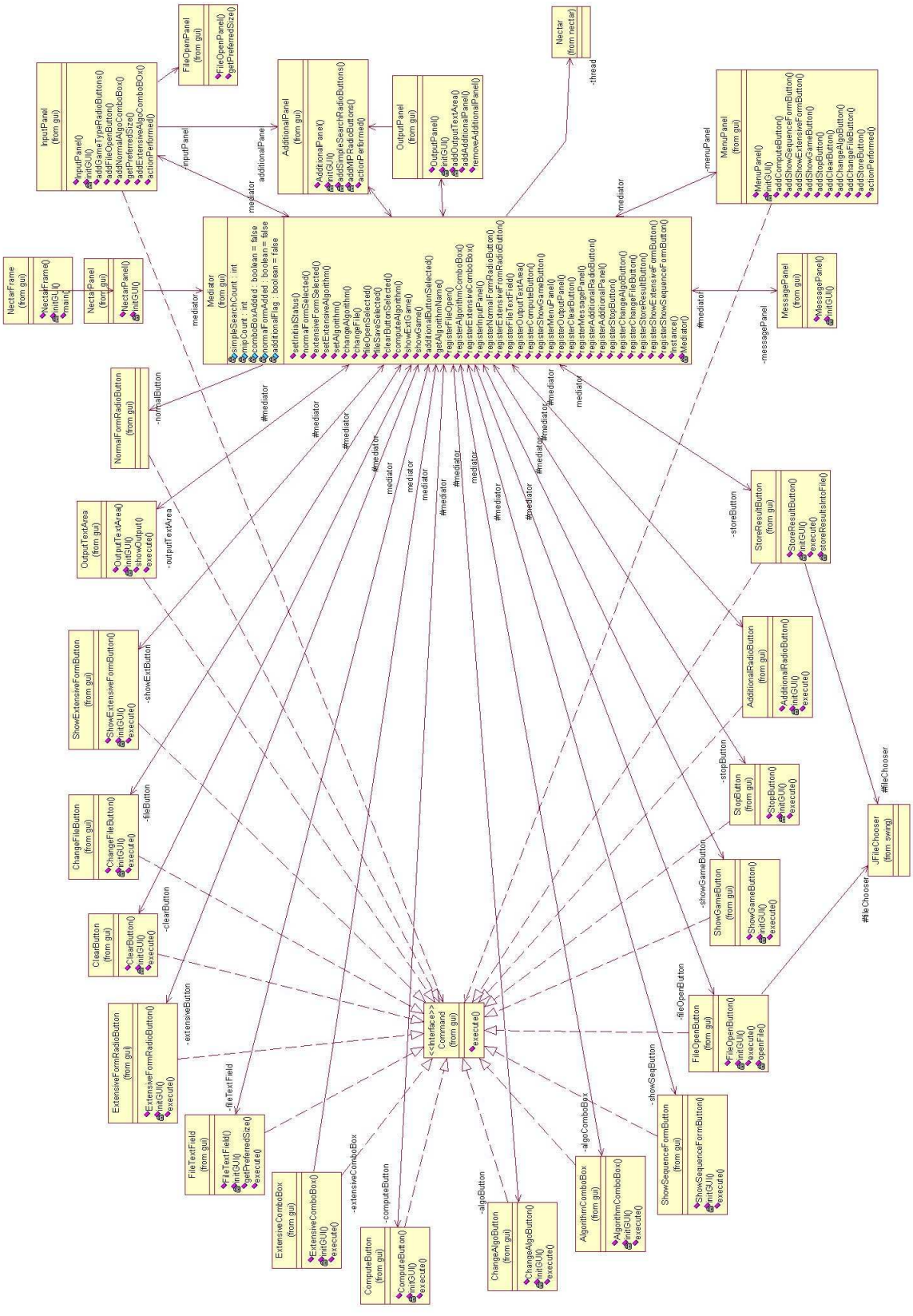

Figure 5.3: Class diagram of Graphical User Interface module of NECTAR

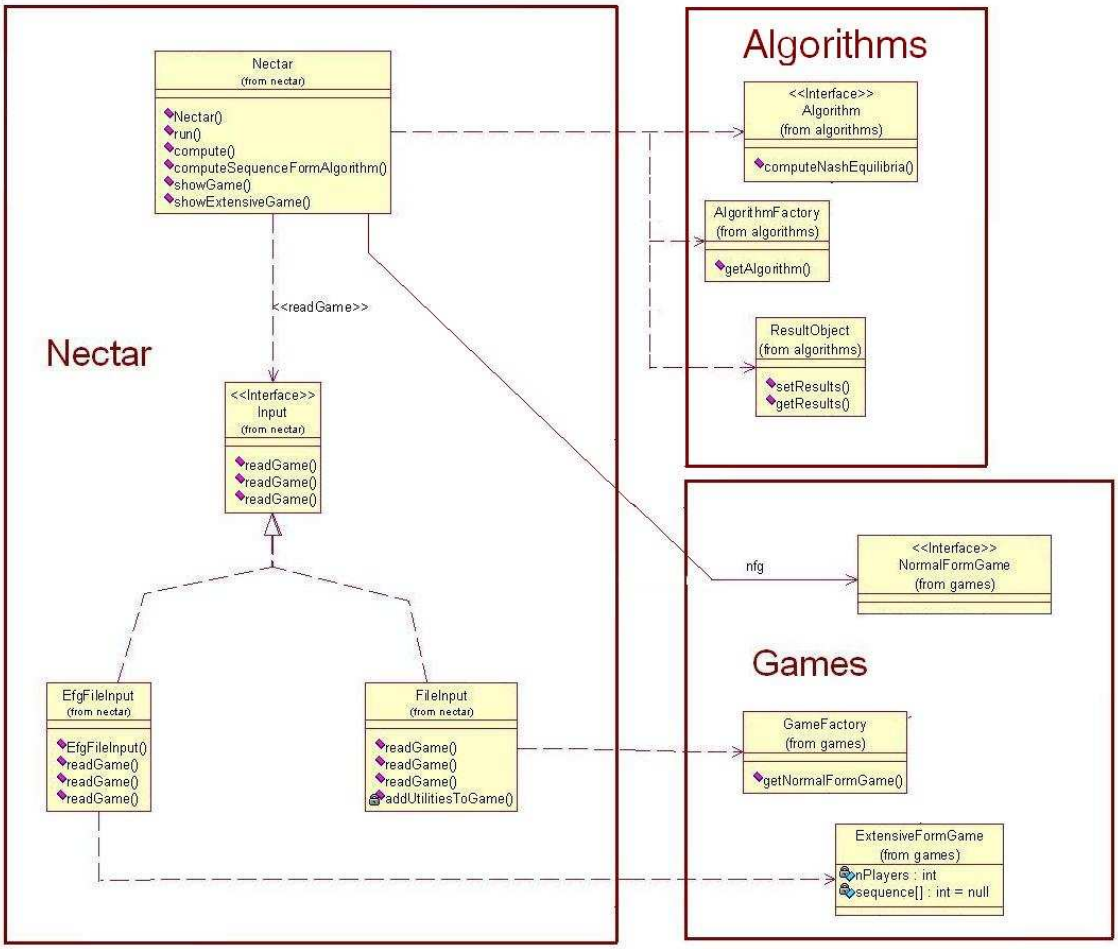

Figure 5.4: Class diagram of Nectar module of NECTAR

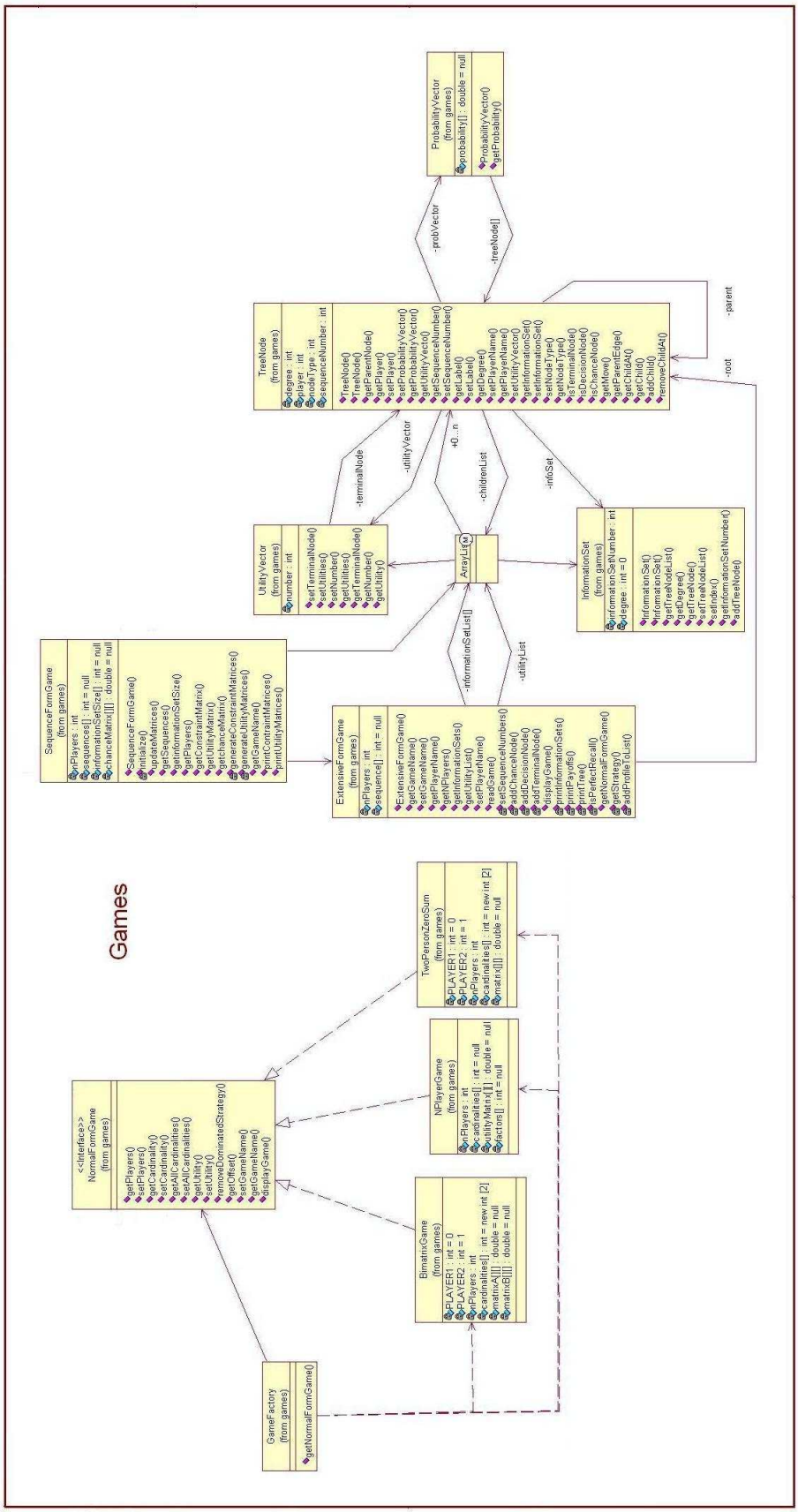

Figure 5.5: Class diagram of Games module of NECTAR

section 5.4.1.

5.2.4 Algorithms Module

The *algorithms* module is shown in Figure 5.6. This module can be treated as *business* module. It contains various algorithms, 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. This module is also implemented with *factory method* design pattern. This module provides a common interface to all the existing algorithms, due to this nature, new algorithms can be ported easily by implementing this common interface without affecting the other modules.

5.2.5 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. Due to this nature, we can adapt any new linear programming solver without modifying the *algorithms* module. This module is implemented with *abstract factory* and *adapter* design patterns which are explained in the section 5.4.4.

5.3 Control Flow

A typical sequence diagram for computing a Nash equilibrium is shown in Figure 5.7. The User selects the type of algorithm for computing Nash equilibrium and selects the file which contains the game data. This information is forwarded to *Nectar*. *Nectar* translates corresponding game data into standard *game* object and returns this *game* object to *Nectar*. Then *Nectar* instantiates required *algorithm* object and passes the *game* object to it. The *algorithm* object processes the *game* object, computes Nash equilibrium and returns the results to the *Nectar*. The *Nectar* module gives these results back to the *GUI* module. Then the *GUI* module displays these results to the end user.

5.4 NECTAR: Design Patterns

This section explains some of the key design patterns (developed by Gamma, Helm, Johnson, and Vlissides [10]) implemented for NECTAR software using object-oriented analysis and techniques for easy pluggability and modifiability. For more information about the design patterns, the reader is referred to [10] and [8].

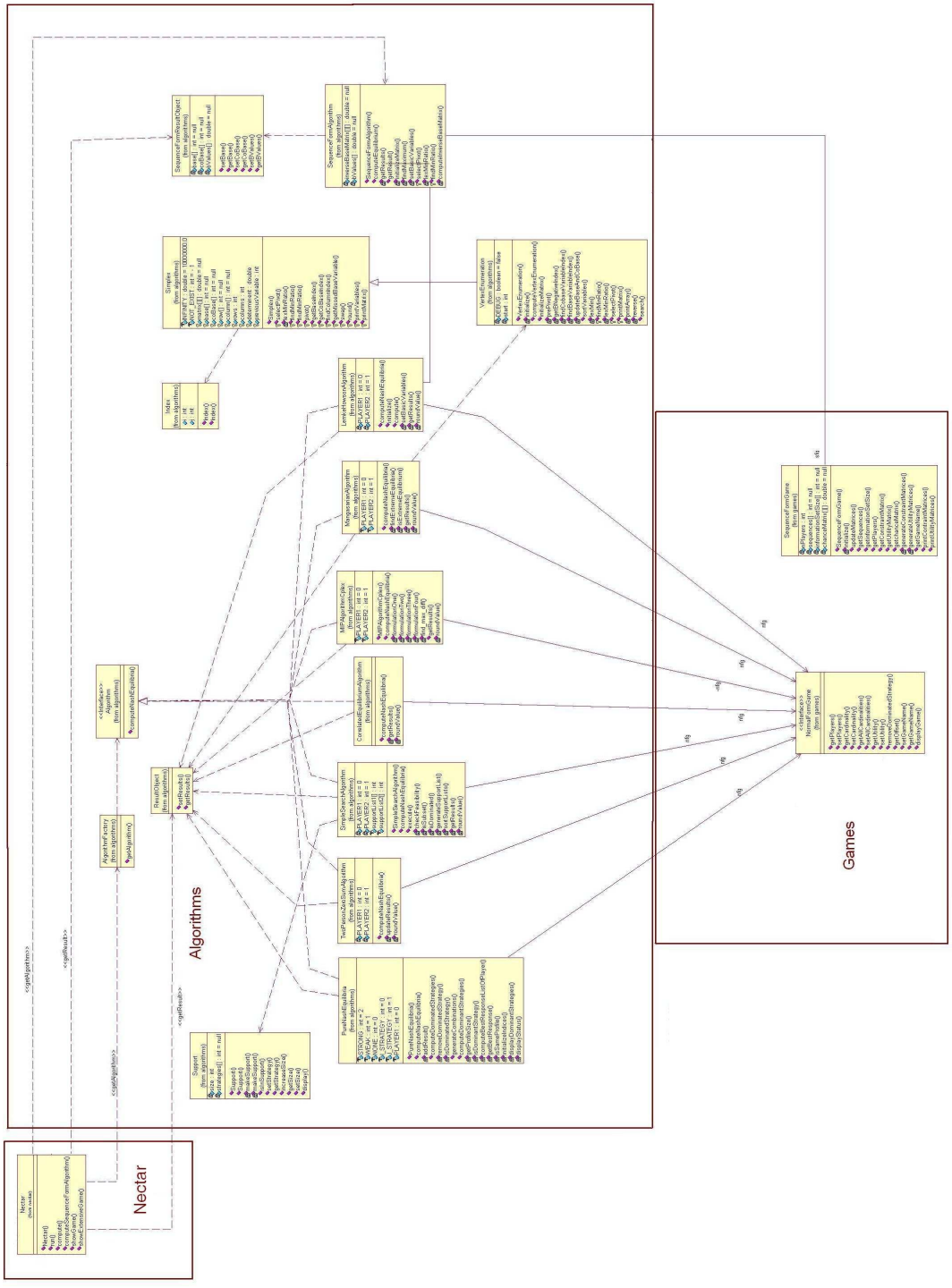

Figure 5.6: Class diagram of Algorithms module of NECTAR

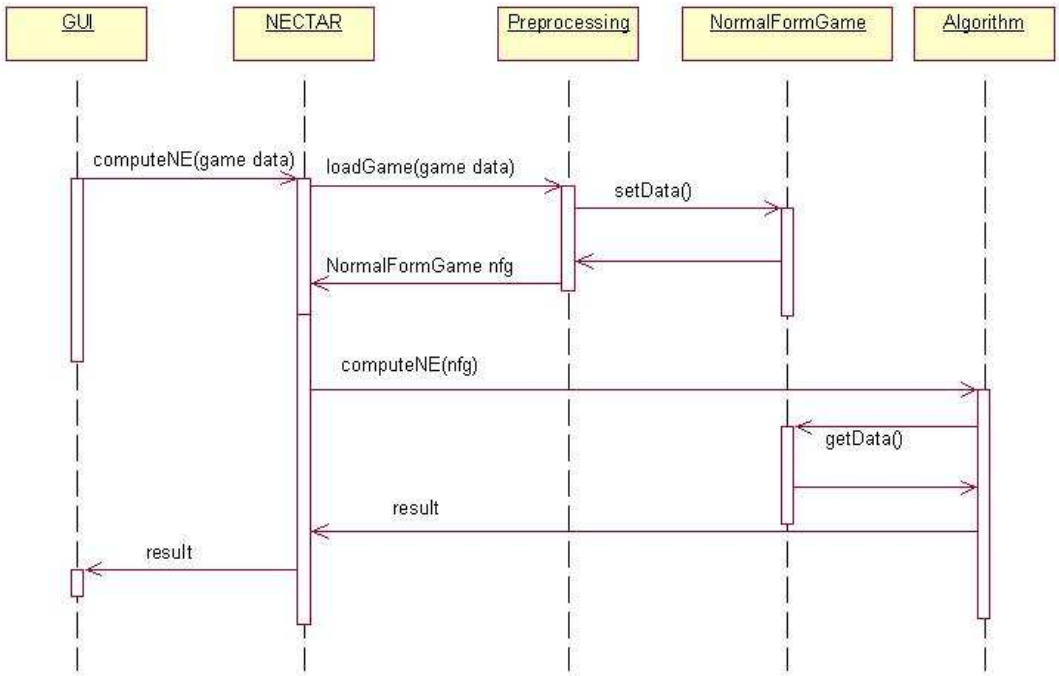

Figure 5.7: Sequence diagram for computing Nash Equilibria

5.4.1 Factory Method Pattern

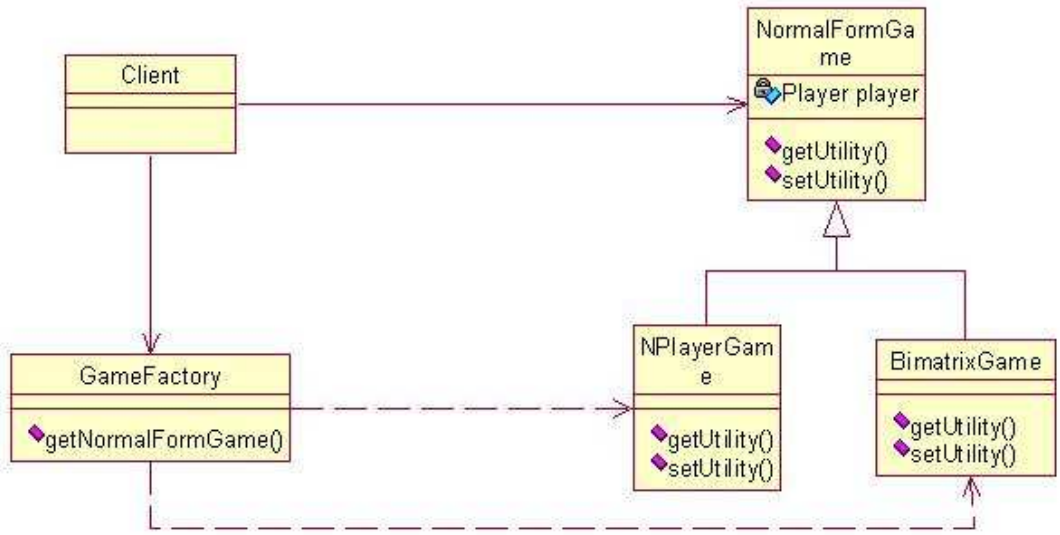

Figure 5.8: Factory Method Pattern

The *BimatrixGame* object represents two-player game and *NPlayerGame* object represents n-player game in normal form. The abstract class for these two classes is the *NormalFormGame* as shown in Figure 5.8. The *factory method* design pattern has *GameFactory* object which generates *NormalFormGame* objects by taking information about the number of players. Depending upon the number of players it instantiates corresponding concrete *NormalFormGame* object and returns to the calling object. This calling object does not know the actual concrete object, it knows only the abstract class interface. By using this pattern, we can easily adapt new type of concrete form of *NormalFormGame* classes without changing any other modules provided that the new concrete classes have same interface.

Similarly, the *algorithms* module is also designed with factory method design pattern. Here the client, sends the information to *AlgorithmFactory* which dynamically instantiates the corresponding algorithm. The client knows only general *Algorithm* interface, he does not know which algorithm actually he is running.

5.4.2 Command and Mediator Patterns

We are using Command and Mediator patterns in the GUI Module. In GUI, the interactions between the visual controls are pretty complex, even in simple applications. Each visual object needs to know about two or more other objects, leading to a quite tangled relationship. This

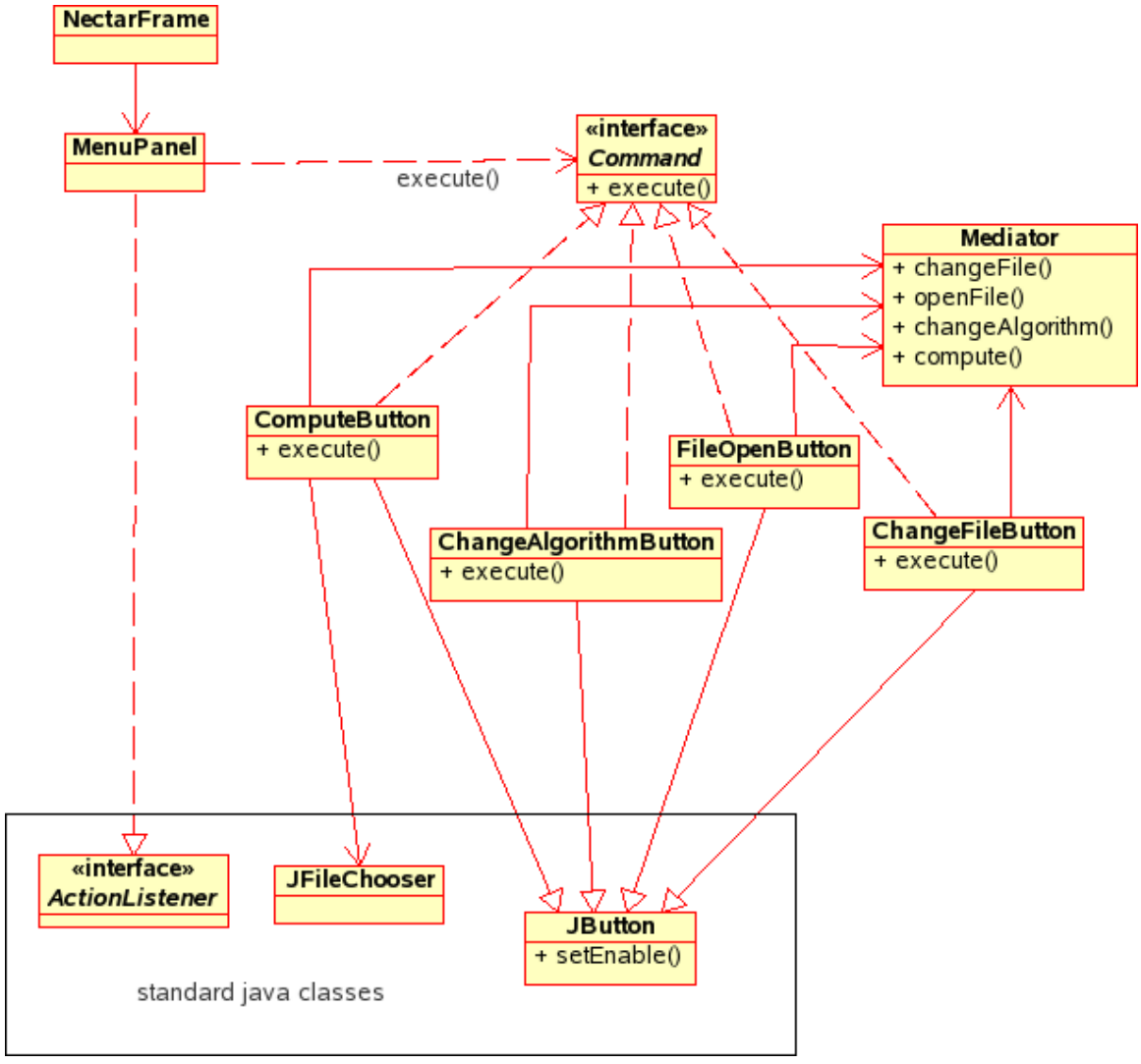

Figure 5.9: Command and Mediator Patterns

problem is addressed by using Mediator design pattern.

The display window contains many objects like *Buttons, Lists and MenuItems etc.*. When one object changes its state, it needs to change the states of some other objects. If the window contains many such objects, it is very difficult when we have to modify any one of these objects or to adapt a new object into the window. When Mediator pattern is used, each object knows only the Mediator object and registers with the Mediator object as shown in Figure 5.9. By the presence of this pattern, the communication complexity among these objects is translated into the single Mediator object. So, when any object changes its state, it intimates to the Mediator object, which in turn updates the other objects.

When the user selects any object on the window, the program receives an ActionEvent, which it must trap by subclassing, the actionPerformed event. When there are many objects, the actionPerformed code gets pretty unwieldy. This really seems a little inelegant. In such cases, Command pattern assures that every object receives its own commands directly. In Command object approach, every object receives its own commands directly. A Command object always has an *Execute()* method that is called when an action occurs on that object.

One important purpose of the Command pattern is to keep the program and user interface objects completely independent from the actions that they initiate.In other words, these program objects should be completely independent from each other and should not have to know how other objects work. The user interface receives a command and tells a Command object to carry out whatever duties it has been instructed to do. The UI does not need to know what tasks will be executed.

A combination of Command, Mediator and Memento patterns in GUI makes implementation very easy. Here the Memento object can be used to store the state information especially for *undo* operations.

5.4.3 Singleton and Facade Patterns

All the GUI components interacts with the Mediator object as described in the previous section. Here all GUI components must interact with single and same instance of the Mediator object. This can be achieved by the use of singleton design pattern. Here, the Mediator object is implemented with singleton design pattern as shown in Figure 5.10.

The Nectar module acts like a Facade here. The GUI module knows only Nectar module and does not know the interfaces of other modules. Because of this loose coupling, we can change internal modules easily without affecting the GUI module. This can be achieved only with facade design pattern. Here, the Nectar object is implemented with facade design pattern as shown in Figure 5.10.

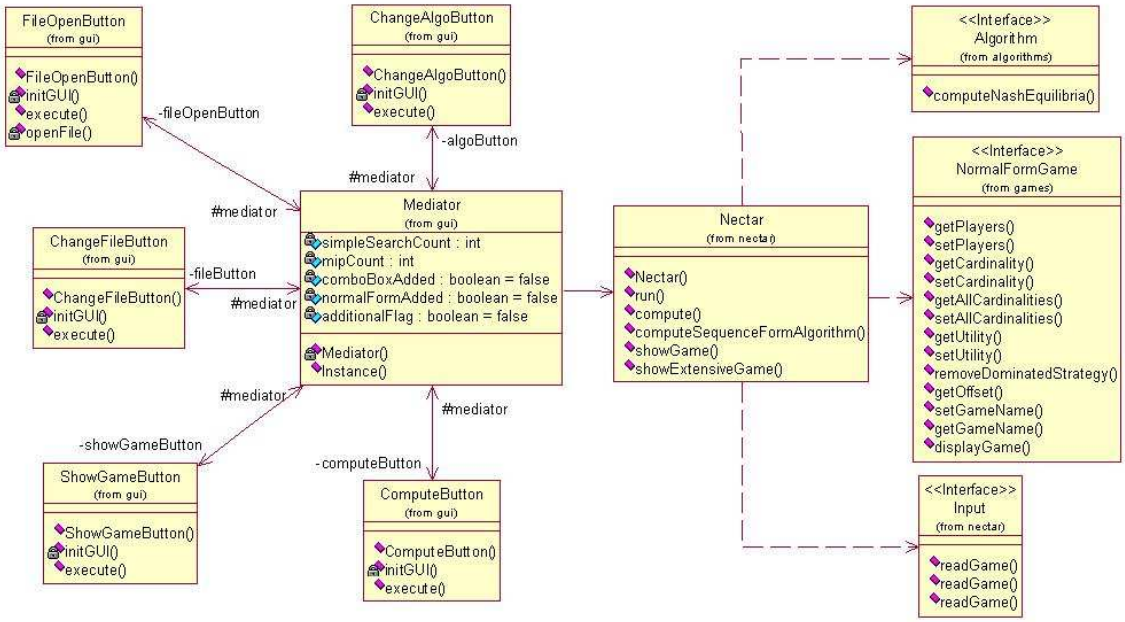

Figure 5.10: Singleton and Facade Patterns

5.4.4 Adapter and AbstractFactory Patterns

Some of the algorithms which are implemented for computing Nash equilibria needs to solve a set of linear equations. This can be achieved by a linear programming solver. With the help of the Adapter pattern the end user can port any linear programming solver(Adaptee) with suitable Adaptor. Currently, we ported ILOG's CPLEX linear programming solver with cplex-adapter as shown in Figure 5.11. But here, we need a set of adapters for a particular specified adaptee(solver) as shown in Figure 5.11. This can be achieved by applying Abstract factory design pattern. With the help of Abstract factory design pattern we can port a specific solver with the set of related adapters.

5.5 Implementation Platform

We are implementing NECTAR in Java, because it is a fully object-oriented, platform independent, and architecture-neutral language. We are using swing for graphical user interface. we are using Ilog's CPLEX software tool for solving the set of linear equations. With the help of Adapter pattern, we provided facility in such a way that the user can use any other software for solving the set of linear equations with a suitable adapter.

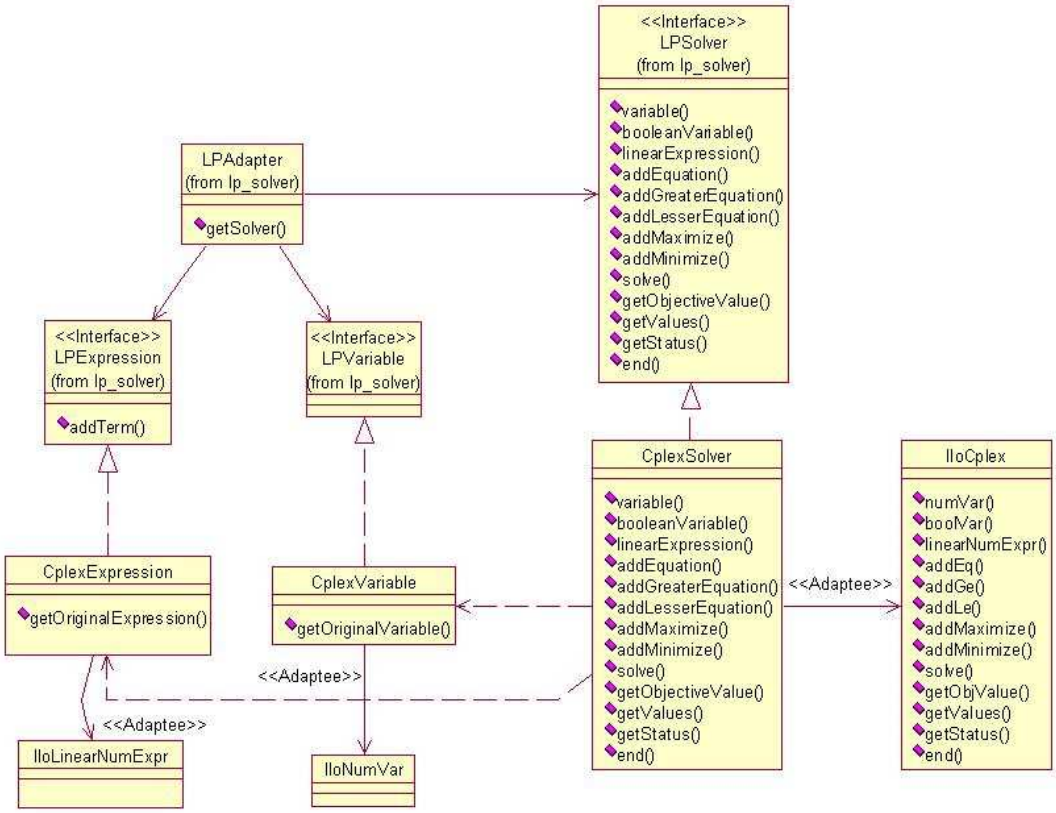

Figure 5.11: Adapter and Abstract Factory Patterns

5.6 Algorithms Implemented

We have implemented the following algorithms.

- Pure Nash equilibria computation for n-person games.
- Nash equilibrium for two person zero-sum games [25].
- Lemke-Howson algorithm for two-person games [17].
- Simple search method for two-person games [27].
- Mixed integer programming method for two-person games [30].
- Mangasarian's algorithm for computing all extreme equilibria for two-person games [18].
- Correlated equilibrium for n-person games [23].
- Efficient computation of equlibrium points for two-person extensive form games [15]

We have tested these algorithms on various representative of games and obtained results. We have implemented conversion algorithms such as extensive-form to normal-form [23] and extensive-form to sequence-form [34] and to find dominated strategies, dominant strategies, reduced normal-form game and whether the given extensive-form game has perfect recall or not *etc.*. We provided good GUI interface to end user for the above mentioned algorithms.

5.7 Data Structures

We implemented different type of data structures for the NECTAR. The main data structures are the game representations: The n-player strategic game can be represented by an n-dimensional array. We implemented an n-dimensional array using single dimension array. Due to the encapsulation and abstract nature of Java, other objects which comminicates with n-dimensional array object do not know the implementation details. Similary, for n-person extensive form game can be represented by the general game tree. We implemented this game tree and provide necessary interface to outside environment.

Chapter 6

Conclusions and Future Work

This chapter provides a summary of contributions of this thesis and outlines several directions for future research.

6.1 Summary of the Thesis

In this section, we summarize the contributions of the thesis, chapter-by-chapter.

- Chapter 1 was devoted to building a proper context for the thesis. We started with frequently asked questions about the NECTAR software tool. Then we gave the motivation and contributions for the NECTAR.
- Chapter 2 presented a comprehensive overview of the fundamentals and essential concepts of game-theorey and various representations of games. Then we described the concept of Nash equilibrium and provided a naive approach for finding a mixed strategy Nash equilibrium for a given game.
- Chapter 3 presented a review of relavent work done earlier for the computation Nash equilibrium points. Then we described the some of the computational complexity issues for the computation of Nash equilibrium points.
- Chapter 4 presented the set of algorithms which were implemented for NECTAR project. We described each algorithm briefly one by one.
- Chapter 5 presented the architecture of the NECTAR and described each module with relavent class diagrams. Then we described a typical control flow of NECTAR and we described the various design patterns which were implemented for the NECTAR.

6.2 Directions for Future Work

We implemented various algorithms for NECTAR which were described in chapter 4. Right now, we are looking to implement the global Newton method algorithm for computing equilibrium points of a n-person game. In future, we are planning to implement the iterated polymatrix approximation method for n-person game [12]. We also intend to implement any new algorithms that will appear in the literature in the coming months. We would like to explore developing our own algorithms driven by specific requirements. We are making the NECTAR software tool with high modularity in order to add new algorithms in future easily.

In future, we are planning to provide the facility such that the user can access the NECTAR through web and do computations. We are also planning to implement the algorithms for computing equilibrium points for cooperative games and equilibrium refinements such as stable equilibria, perfect equilibria etc..

Bibliography

[1] *ILOG - CPLEX 9.0 User's Manual*, October 2003.

[2] Gamut user guide, version 1.0.1, 2004. http://gamut.stanford.edu/userdoc.pdf.

[3] V. Aggarwal. On a generation of all equilibrium points for bimatrix games through the Lemke-Howson algorithm. *Mathematical Programming*, 4:233–234, 1973.

[4] Robert J. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1(1):67–96, 1974.

[5] D. Avis. A revised implementation of the reverse search vertex enumeration algorithm. Technical report, School of Computer Science, McGill University, Canada, 1999.

[6] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Computational Geometry*, 8(3):295–313, 1992.

[7] V. Conitzer and T. Sandholm. Complexity results about Nash equilibria. Technical report, School of Computer Science Carnegie-Mellon University, 5000, Forbes Avenue, Pittsburgh, PA15213, USA, May 2002.

[8] James W. Cooper. *The Design Patterns - Java Companion.* Addison Wesley Design Patterns Series, 1998.

[9] J. Dickhaut and T. Kaplan. A program for finding Nash equilibria. *The Mathematica Journal*, 1(4):87–93, 1991.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison Wesley, 1995.

[11] S. Govindan and R. Wilson. A global Newton method to compute Nash equilibria. *Journal of Economic Theory*, 110(1):65–86, 2003.

[12] S. Govindan and R. Wilson. Computing Nash equilibria by iterated polymatrix approximation. *Journal of Economic Dynamics and Control*, 28(7):1229–1241, 2004.

[13] Mukti Jain. Implementation of algorithms for computing Nash equilibria in noncooperative games, M.E. Project Report, June 2005. Department of Computer Science, Indian Inistitute of Science, Bangalore.

[14] E. Kohlberg and J.F. Mertens. On the strategic stability of equilibria. In *Econometrica 54*, pages 1003–1037, 1986.

[15] D. Koller, N. Megiddo, and B. von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

[16] G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[17] C.E. Lemke and J.T. Howson, Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[18] O.L. Mangasarian. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):778–780, December 1964.

[19] R. McKelvey and A. Mclennan. *Handbook of computational Economics. Vol. I.* Elsevier, 1996.

[20] R. McKelvey, A. McLennan, and T. Turocy. Gambit: Software tools for game theory. http://econweb.tamu.edu/gambit.

[21] N. Megiddo and H.C. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

[22] K.G. Murthy. *Linear Complementarity, Linear and Nonlinear Programming.* Internet Edition, The University of Michigan, Ann Arbor, 1999.

[23] R.B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, Cambridge, Massachusetts, 1997.

[24] J. Nash. Non - cooperative games. *The Annals of Mathematics*, 54(2):286–295, September 1951.

[25] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 2004.

[26] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceeding of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 880–887, 2004.

[27] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a Nash equilibrium. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 664–669, 2004.

[28] J. Rosenbaum. The computational complexity of Nash equilibria, 2002.

[29] J. Rosenmuller. On a generalization of the Lemke-Howson algorithm to noncooperative n-person games. *SIAM Journal of Applied Mathematics*, 21(1):73–79, 1971.

[30] T. Sandholm, A. Gilpin, and V. Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 495–501, 2005.

[31] R. Savani and B. von Stengel. Exponentially many steps for finding a Nash equilibrium in bimatrix game. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 258–267, 2005.

[32] L.S. Shapley. In *Mathematical Programming Study 1: Pivoting and Extensions*, pages 175–189, 1974.

[33] G. Van der Laan, J.J. Talman, and L. Van der Heyden. Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. *Mathematics of Operations Research*, 12(3):377–397, August 1987.

[34] B. von Stengel. Efficient computation of behaviour strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.

[35] B. Von Stengel. Computing equilibria for two-person games. Technical report, London School of Economics, ETH Zentrum, CH-8092, Zurich, Switzerland, 1999.