An Auction-based Course Allocation System

Project Report

Submitted in partial fulfillment of the requirements of
CS F376 Design Project

By

Neel Shah
ID No. 2016A7PS0076P

Under the supervision of:

Dr. Avinash Gautam

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS
April 2019

“It is all very well planning what you will do in six months, what you will do in a year, but it’s no good at all if you don’t have a plan for tomorrow”

Hilary Mantel

Acknowledgements

I would like to thank my supervisor Prof. Avinash Gautam for his guidance and for motivating me to put my best foot forward.

I would also like to thank all those who helped me over the course of this project.

Contents

Chapter 1

Introduction

The discipline of Mechanism Design offers the mathematical tools to create sound models of strategic environments where rational agents can engage in selfish pursuits. This project seeks to develop an auction-based mechanism for course registration in universities.

1.1 Objectives

- To conceptualize a course allocation model for a university.
- To help the system elicit information about students' choice.
- To evaluate the efficiency and effectiveness of the new model.

1.2 Structure and Scope

The project was developed in an incremental manner. There are seven different models, including the current system, the hypothetical best case scenario, four auction mechanisms and an alternative modelling approach. A thorough study of auction theory was carried out to understand its applications and suitability to the problem at hand. A simple auction was then devised to handle the allocation of courses for students of one batch and one discipline. Going further, this simple model was extended to four different models which handle multiple disciplines, elective courses and student bids differently. These models were then simulated to determine their effectiveness. This was characterized by aggregating the amount of deviation in allotment from the student's ideal choices.

Thus the system can handle any number of students from different branches applying for compulsory and elective courses offered by different departments. It takes a section-wise list of courses and students' bids as input and provides a final course allotment as output.

1.3 Boundaries

1.3.1 Scale

The project domain was quite large and involved various design complexities. Therefore the problem was scaled down in terms of size, although the model and code are both scalable and can be built upon without the need for fundamental changes.

1.3.2 Unavailability of data

Since it was not possible to obtain preference and bid data for a large number of students, a smaller number of authentic datapoints were obtained and replicated with the introduction of reasonable variation.

Chapter 2

Setting the Stage

2.1 The Course Registration Problem

At the university level, students studying different disciplines study compulsory subjects from their discipline and elective subjects that they may choose from different disciplines. The class strength and number of teachers is limited, however. Therefore students must be allotted courses and sections based on some priority ordering.

2.2 The Current Model

In the existing system, students are entered into a randomized priority queue. They are assigned randomly generated priority numbers, and are then en-queued in order of these numbers. A student with a numerically lower priority number picks his courses and sections before a student with a numerically higher priority number.

2.3 Strengths of the Current Model

The system is fair and impartial; it assigns randomly generated priorities and prevents factors such as previous academic performance from becoming an impediment.

2.4 Limitations of the Current Model

The randomized queuing does not incentivize academic planning; the planning efforts of a student might be stymied by a numerically higher priority number. Students thus apply for courses based on popularity and word-of-mouth publicity rather than a concrete plan.

2.5 Motivation for using Auction Theory

The expression of choice of courses naturally resembles the action space of the notionally popular auction. The existing system and the proposed model can be easily compared by fitting both into the auction template. Moreover, an auction allows the agents maximum control over their actions. Students are also likely to be motivated to plan their academics better since a certain value is now associated with their bids [4][3][1].

Chapter 3

Policy Decisions

As a design problem, course registration requires the designer to carefully set various parameters related to the underlying model, in this case, the auction.

These decisions are based on the following conditions [5]:

- Vickerey's Condition states that the price paid by a bidder upon making a successful bid must be as independent from the player's own bids as possible; ideally it must depend only on the other agents' bids.
- Milgrom and Weber state that maximum information must be made available to each participant at the time they place their bid.

3.1 Nature of Auction

The units offered for auctioning are individual course sections. However, students value a complete basket of courses and not just individual sections. Thus, this is a combinatorial auction [2] with complementarities; valuations and bids are expressed for groups of items and not individual items.

3.2 Pricing Scheme

The pricing scheme can be designed to elicit honest preferences from bidders. Vickrey second-price auctions are highly efficient in this regard [6]. However, this makes the proceedings of the auction difficult to understand for the bidders. Therefore, elicitation of honest responses was traded in lieu of ease of use. This was eased by the reasonable assumption that students do not profit from misreporting their preference for courses. Thus, the first-price scheme was preferred over the second-price scheme.

3.3 Auction Format

The ideal system would take a number of alternative timetables from students, compute the combination that maximized social welfare, and allot course sections accordingly. Since it considers all possible combinations, the computational complexity is exponential in input size. As a reference, a simulation consisting of 32 students with two alternative timetables each required over five hours to compute. One can very well imagine the difficulty in replicating this task for a batch of a thousand students.

The auction format chosen is a heuristic modification of the best-case. This format consists of multiple rounds, and each bidder submits only one group of course sections per round. At the end of each round, all bidders are notified of the courses they have been allotted successfully, and the number of vacancies in all course sections. Based on this information, they bid in the successive rounds.

A major stumbling block for the current system is the randomness of the order in which students choose their course sections. The issue was addressed as follows: Of the four auction models that have been described here, two distinguish between compulsory and elective courses, whereas the other two do not. Similarly, two models maintain individual bid queues for students whereas the other two maintain a single queue for all students.

Chapter 4

Model Description

4.1 The Existing Model

The existing model is the randomized queue model. The input taken is each student's preferred timetable and the valuation associated with the same. The output is the set of timetables allotted. When a student enters the queue, all requested courses are allotted to him (based on availability) before proceeding to the next student.

4.2 The Best-Case Model

In the best-case model, the input taken is each student's two most-preferred timetables, and valuation vectors associated with the same. The output is the set of allotted timetables. The system considers all possible timetable combinations and returns the set that maximizes global welfare. There is a trade-off between exponential time requirement and exponential memory requirement. In either case, the solution is practically unfeasible.

4.3 The AuctionOne Model

In the AuctionOne model, the input taken is each student's most-preferred timetable, and valuation vectors associated with the same. The output is the set of allotted timetables. The system maintains separate priority queues for each student, wherein timetable objects are ordered in decreasing value of points. At each step, the first element of each queue is put into a separate similar priority queue and all of those are allotted (based on availability). This continues until all student queues have been exhausted.

Strengths

- Attempts to allot each student's high ranking preferences.

Weaknesses

- A student's highest preference may not be highly valued on an absolute scale.
- Does not handle electives separately. Students may end up without desirable compulsory course sections if the course gets filled up as an elective.

4.4 The AuctionTwo Model

In the AuctionTwo model, the input taken is each student's most-preferred timetable, and valuation vectors associated with the same. The output is the set of allotted timetables. The system maintains a single priority queue for each student, wherein timetable objects are ordered in decreasing value of points. Allotments are made serially until the queue is exhausted.

Strengths

- Attempts to order all preferences on an absolute basis.

Weaknesses

- Conservative bidders who bid uniform values across courses may lose out to risk-taking bidders who bid high amounts on particular courses.
- Does not handle electives separately. Students may end up without desirable compulsory course sections if the course gets filled up as an elective.

4.5 The AuctionThree Model

In the AuctionThree model, the input taken is each student's most-preferred timetable, and valuation vectors associated with the same. The output is the set of allotted timetables. The system maintains separate priority queues for each student, wherein timetable objects are first separated based on whether they are compulsory or elective courses. They are then ordered in decreasing value of points, with greater preference being given to compulsory courses. Thus each student's queue consists of an electives priority queue appended to a compulsory courses priority queue. At each step, the first element of each queue is put into a separate similar priority queue and all of those are allotted (based on availability). This continues until all student queues have been exhausted.

Strengths

- Attempts to allot each student's high ranking compulsory course and elective course preferences (separately).

Weaknesses

- A student's highest preference may not be highly valued on an absolute scale.

4.6 The AuctionFour Model

In the AuctionTwo model, the input taken is each student's most-preferred timetable, and valuation vectors associated with the same. The output is the set of allotted timetables. The system maintains a single priority queue for each student, wherein timetable objects are first separated based on whether they are compulsory or elective courses. They are then ordered in decreasing value of points, with greater preference being given to compulsory courses. Thus the queue consists of an electives priority queue appended to a compulsory courses priority queue. Allotments are made serially until the queue is exhausted.

Strengths

- Attempts to order all compulsory course and elective course preferences (in two separate queues) on an absolute basis.

Weaknesses

- Conservative bidders who bid uniform values across courses may lose out to risk-taking bidders who bid high amounts on particular courses.

4.7 The Knapsack Formulation - An Alternate Approach

Instead of modelling the course registration as an auction game, it can also be modelled as a knapsack problem for n agents (students). In order to formulate this problem, we need the following definitions:

n = number of students m = number of courses j = index over students i = index over courses
$H_i = value$ of course i measured by instructors' and students' feedback of previous offerings.
$H_i/k = value$ of course i averaged over class strength k.
$\vec{v_j}$ = valuation vector, describes the student j's preference over courses. $v_j(i)$ = student j's valuation of course i such that

$$\sum_{i=1}^{m} v_j(i) = 1$$

$\vec{p_j} = profit$ vector, where student j derives $p_j(i)$ profit by valuing course i at $v_j(i)$.

$$\vec{p_j}(i) = (H_i/k).\vec{v_j}(i)$$

$\vec{w}$ = weight vector such that $w(i)$ = number of units associated with course i.
W_0 = capacity of knapsack, ie. the limit on the number of units a student can register for.

The problem is now the familiar 0-1 knapsack problem: the knapsack must be filled with objects which have some weight and some profit associated with them. The objective is to maximize profit while not exceeding the capacity of the knapsack.

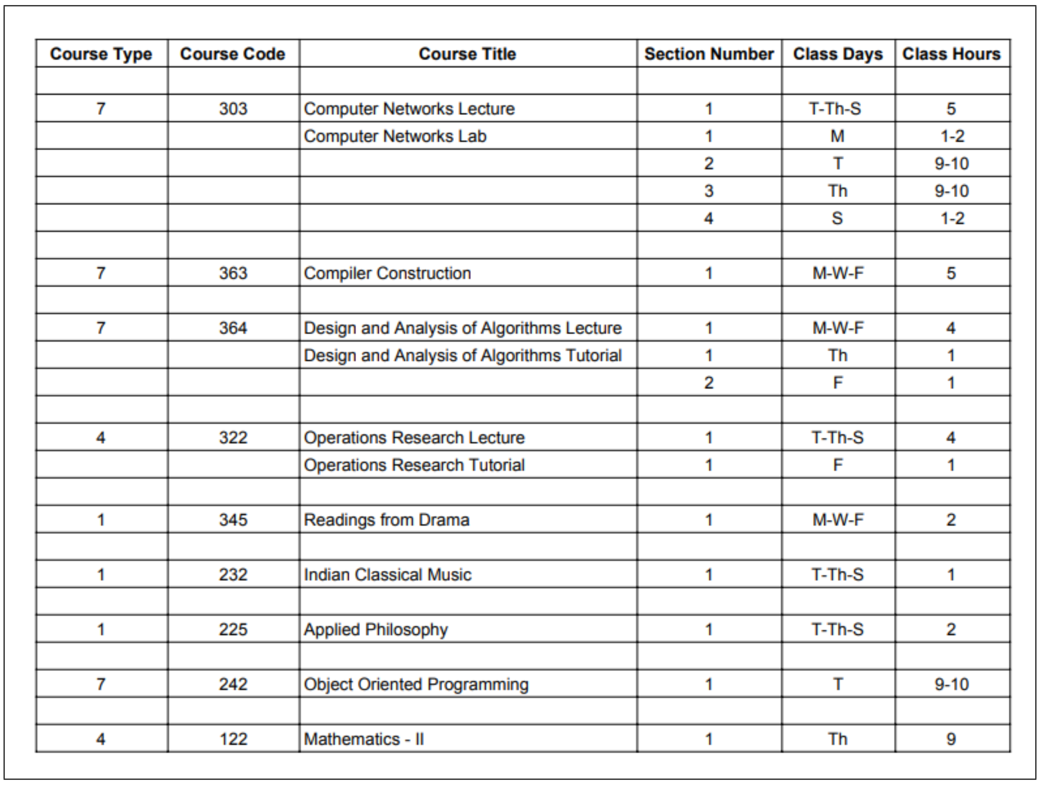
While very efficient algorithms have been developed to solve the knapsack problem, determination of H_i is a very subjective problem. Also we have to incorporate the element of choice through the user-defined valuation vector. Thus it seems intuitive to choose the auction model over this formulation, since the former naturally matches the required action-space. Therefore the knapsack formulation is limited to being a thought experiment.

Chapter 5

Simulation

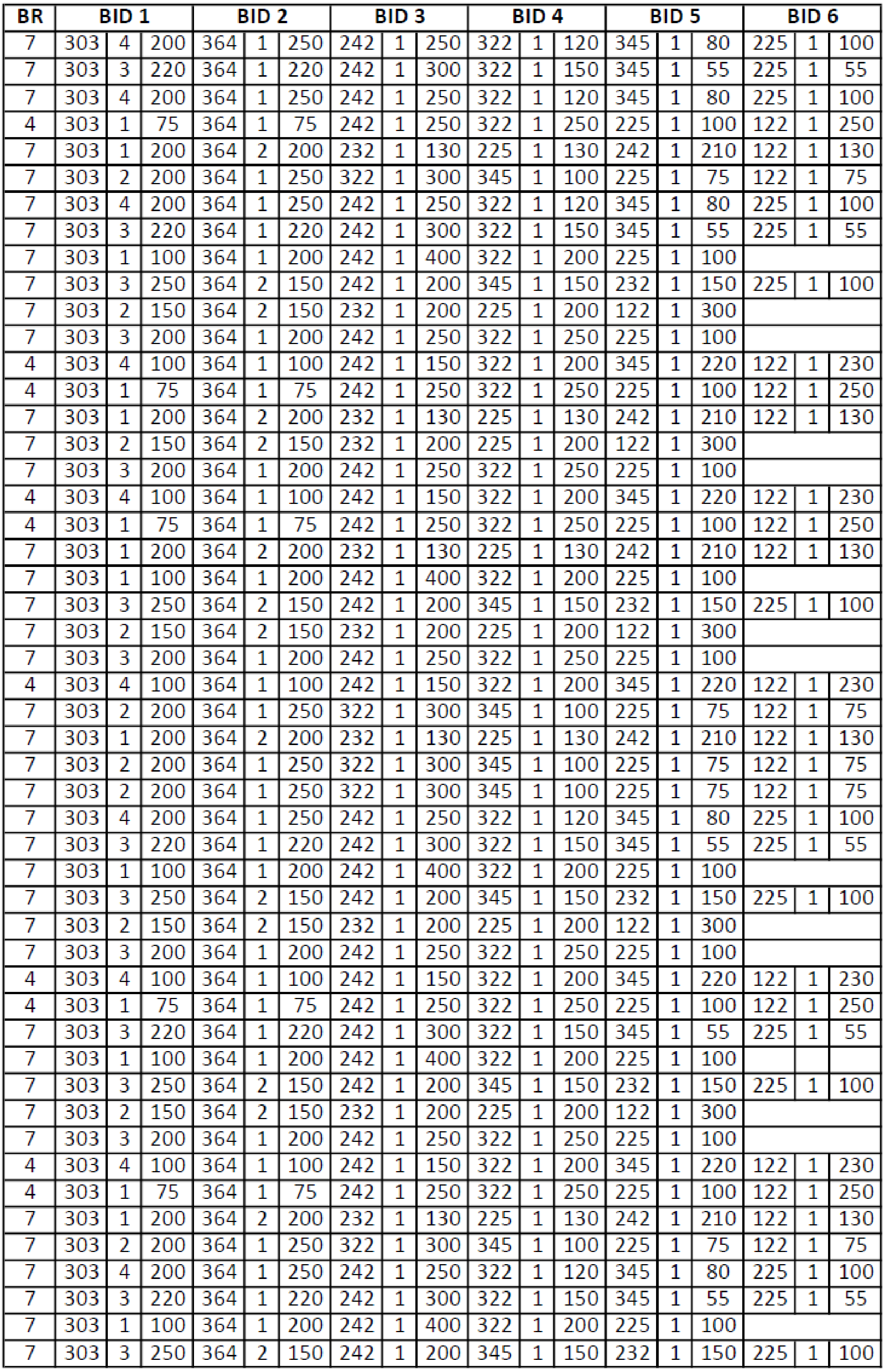
5.1 Data

A sample timetable was prepared consisting of 9 courses offered by three different departments (labelled 1, 4 and 7). Similarly, the 50 students belong to two different disciplines (labelled 4 and 7). Thus all students have to choose compulsory course sections and may also choose elective courses from the other disciplines.



Course Type	Course Code	Course Title	Section Number	Class Days	Class Hours
7	303	Computer Networks Lecture	1	T-Th-S	5
		Computer Networks Lab	1	M	1-2
			2	T	9-10
			3	Th	9-10
			4	S	1-2
7	363	Compiler Construction	1	M-W-F	5
7	364	Design and Analysis of Algorithms Lecture	1	M-W-F	4
		Design and Analysis of Algorithms Tutorial	1	Th	1
			2	F	1
4	322	Operations Research Lecture	1	T-Th-S	4
		Operations Research Tutorial	1	F	1
1	345	Readings from Drama	1	M-W-F	2
1	232	Indian Classical Music	1	T-Th-S	1
1	225	Applied Philosophy	1	T-Th-S	2
7	242	Object Oriented Programming	1	T	9-10
4	122	Mathematics - II	1	Th	9

Figure 5.1: The Master Timetable



BR	BID 1			BID 2			BID 3			BID 4			BID 5			BID 6		
7	303	4	200	364	1	250	242	1	250	322	1	120	345	1	80	225	1	100
7	303	3	220	364	1	220	242	1	300	322	1	150	345	1	55	225	1	55
7	303	4	200	364	1	250	242	1	250	322	1	120	345	1	80	225	1	100
4	303	1	75	364	1	75	242	1	250	322	1	250	225	1	100	122	1	250
7	303	1	200	364	2	200	232	1	130	225	1	130	242	1	210	122	1	130
7	303	2	200	364	1	250	322	1	300	345	1	100	225	1	75	122	1	75
7	303	4	200	364	1	250	242	1	250	322	1	120	345	1	80	225	1	100
7	303	3	220	364	1	220	242	1	300	322	1	150	345	1	55	225	1	55
7	303	1	100	364	1	200	242	1	400	322	1	200	225	1	100			
7	303	3	250	364	2	150	242	1	200	345	1	150	232	1	150	225	1	100
7	303	2	150	364	2	150	232	1	200	225	1	200	122	1	300			
7	303	3	200	364	1	200	242	1	250	322	1	250	225	1	100			
4	303	4	100	364	1	100	242	1	150	322	1	200	345	1	220	122	1	230
4	303	1	75	364	1	75	242	1	250	322	1	250	225	1	100	122	1	250
7	303	1	200	364	2	200	232	1	130	225	1	130	242	1	210	122	1	130
7	303	2	150	364	2	150	232	1	200	225	1	200	122	1	300			
7	303	3	200	364	1	200	242	1	250	322	1	250	225	1	100			
4	303	4	100	364	1	100	242	1	150	322	1	200	345	1	220	122	1	230
4	303	1	75	364	1	75	242	1	250	322	1	250	225	1	100	122	1	250
7	303	1	200	364	2	200	232	1	130	225	1	130	242	1	210	122	1	130
7	303	1	100	364	1	200	242	1	400	322	1	200	225	1	100			
7	303	3	250	364	2	150	242	1	200	345	1	150	232	1	150	225	1	100
7	303	2	150	364	2	150	232	1	200	225	1	200	122	1	300			
7	303	3	200	364	1	200	242	1	250	322	1	250	225	1	100			
4	303	4	100	364	1	100	242	1	150	322	1	200	345	1	220	122	1	230
7	303	2	200	364	1	250	322	1	300	345	1	100	225	1	75	122	1	75
7	303	1	200	364	2	200	232	1	130	225	1	130	242	1	210	122	1	130
7	303	2	200	364	1	250	322	1	300	345	1	100	225	1	75	122	1	75
7	303	2	200	364	1	250	322	1	300	345	1	100	225	1	75	122	1	75
7	303	4	200	364	1	250	242	1	250	322	1	120	345	1	80	225	1	100
7	303	3	220	364	1	220	242	1	300	322	1	150	345	1	55	225	1	55
7	303	1	100	364	1	200	242	1	400	322	1	200	225	1	100			
7	303	3	250	364	2	150	242	1	200	345	1	150	232	1	150	225	1	100
7	303	2	150	364	2	150	232	1	200	225	1	200	122	1	300			
7	303	3	200	364	1	200	242	1	250	322	1	250	225	1	100			
4	303	4	100	364	1	100	242	1	150	322	1	200	345	1	220	122	1	230
4	303	1	75	364	1	75	242	1	250	322	1	250	225	1	100	122	1	250
7	303	3	220	364	1	220	242	1	300	322	1	150	345	1	55	225	1	55
7	303	1	100	364	1	200	242	1	400	322	1	200	225	1	100			
7	303	3	250	364	2	150	242	1	200	345	1	150	232	1	150	225	1	100
7	303	2	150	364	2	150	232	1	200	225	1	200	122	1	300			
7	303	3	200	364	1	200	242	1	250	322	1	250	225	1	100			
4	303	4	100	364	1	100	242	1	150	322	1	200	345	1	220	122	1	230
4	303	1	75	364	1	75	242	1	250	322	1	250	225	1	100	122	1	250
7	303	1	200	364	2	200	232	1	130	225	1	130	242	1	210	122	1	130
7	303	2	200	364	1	250	322	1	300	345	1	100	225	1	75	122	1	75
7	303	4	200	364	1	250	242	1	250	322	1	120	345	1	80	225	1	100
7	303	3	220	364	1	220	242	1	300	322	1	150	345	1	55	225	1	55
7	303	1	100	364	1	200	242	1	400	322	1	200	225	1	100			
7	303	3	250	364	2	150	242	1	200	345	1	150	232	1	150	225	1	100

Figure 5.2: Student Bid Data - The BR column shows branch, Balance shows bidding points available and Bid shows course, section and valuation

5.2 Evaluation Metric

The metric used for determining welfare was defined as

$$welfare = (value\ of\ successful\ bids)/(total\ value\ of\ bids)$$

Global welfare was defined as the average of welfare values of all agents.

$$Global\ welfare = 1/N \sum_{all\ agents} welfare$$

5.3 Results

5.3.1 Existing Model

The statistics for the existing model were as follows:

Time taken (in microseconds)	**Global Welfare**
3896	0.7542

It can also be seen from the following plot that value achieved decreases considerably as PR number falls. This shows that the PR number skews distribution of welfare.

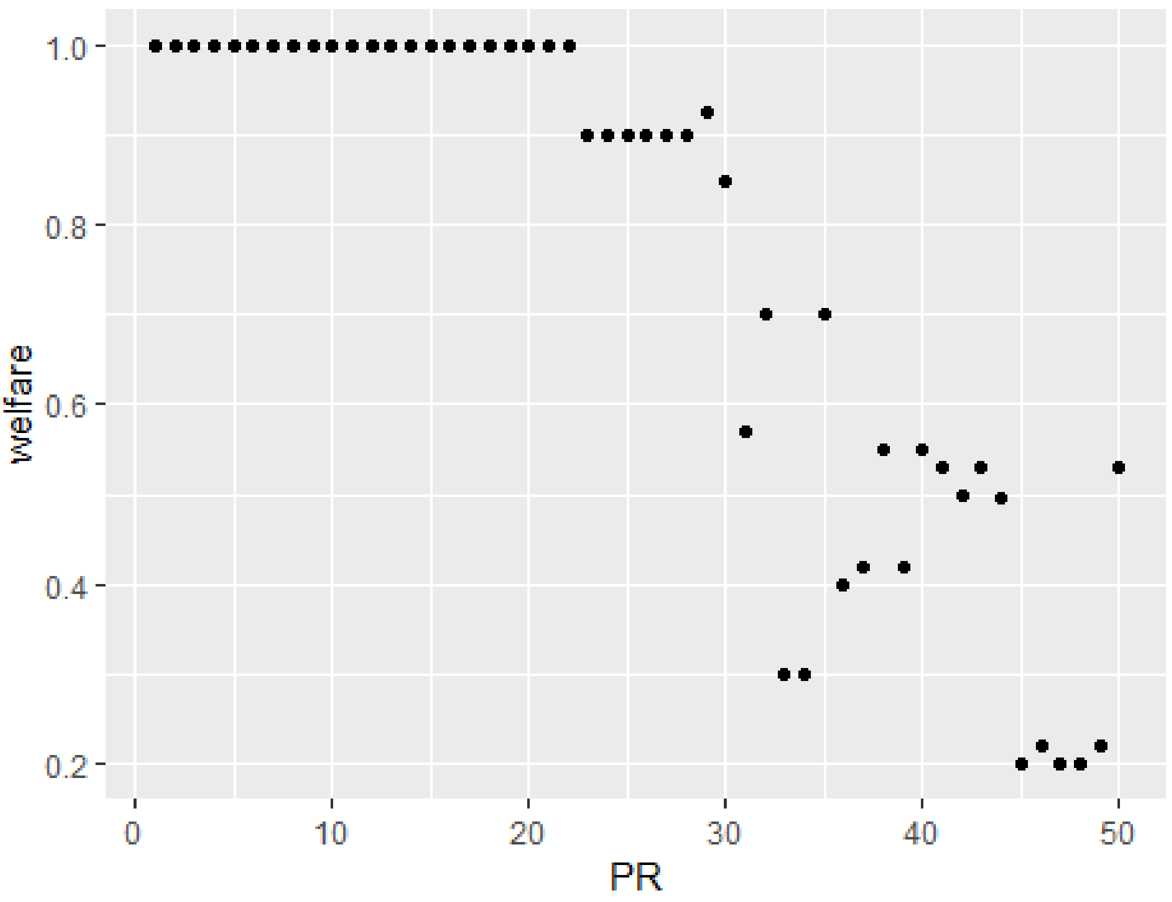


FIGURE 5.3: Variation of Welfare with PR number

5.3.2 Auction Models

The statistics for the auction models were as follows:

Auction Model	**Time taken (in microseconds)**	**Global Welfare**
AuctionOne	2309	0.8140
AuctionTwo	75973	0.8205
AuctionThree	1765	0.8282
AuctionFour	20298	0.8165

Chapter 6

Conclusion

It was shown that the randomized queuing solution for the course registration problem has some serious limitations affecting choice and global welfare. Alternate approaches such as auctions and the knapsack formulation were explored. Auction mechanisms exhibited remarkable closeness to the required action space. Thus an auction game was modelled over the course registration problem. The (hypothetical) best case scenario was described, and heuristic approaches to the same were developed. These approaches were then simulated using software, and their effectiveness was determined.

Appendix A

Software Documentation

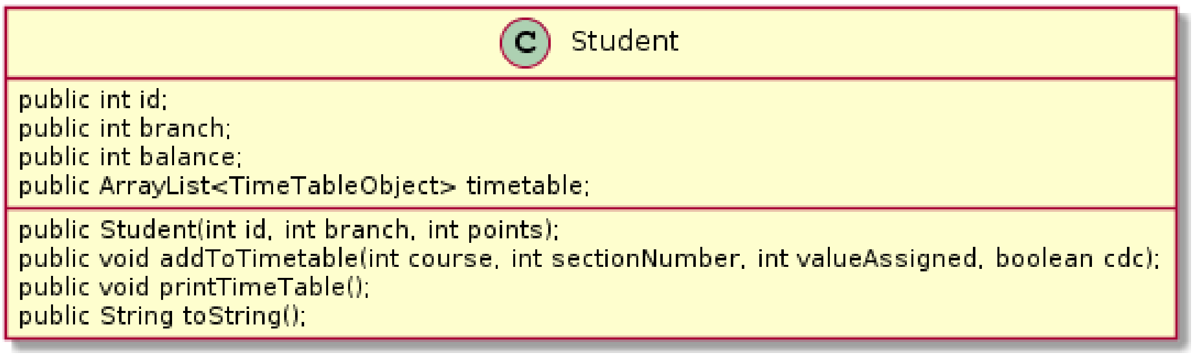
A.1 Common Classes and APIs for Auction Models

The classes *Student*, *Course*, *Section* and *TimeTableObject* are helper classes which provide a modular structure to support the auction mechanism.

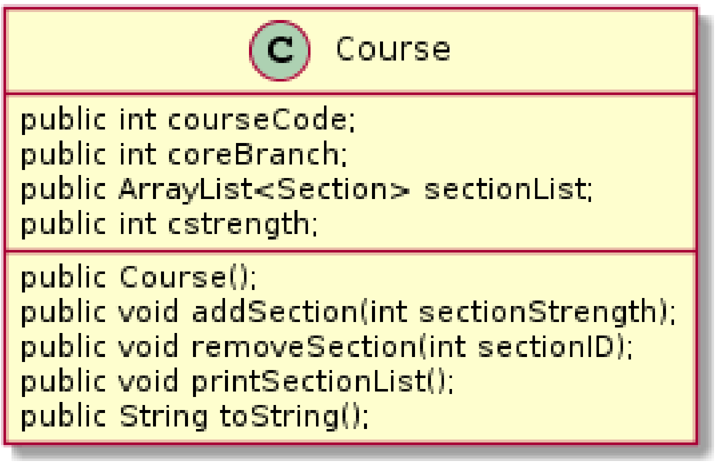
The *ListManip* class provides methods which read the course and student data and organize them into a list of course objects and a list of student objects (populated with timetable preferences) respectively.

An object of the *QueueMaker* class takes the list of students and constructs priority queues of individual bids.

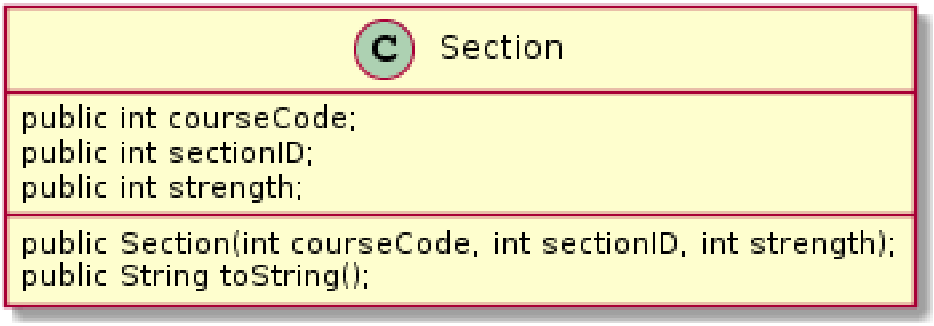
The *SimpleAuction* abstract class specifies the underlying behaviour that each auction class must implement.



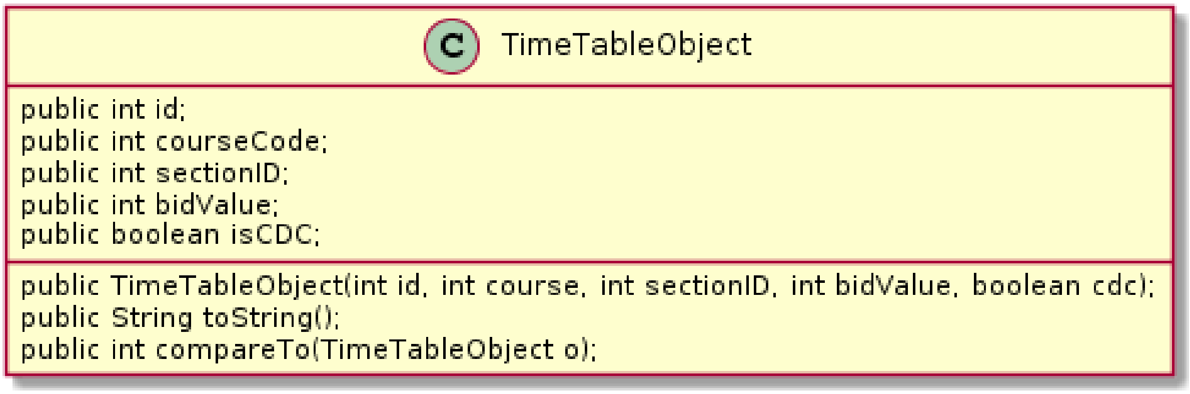
```
public int id;
public int branch;
public int balance;
public ArrayList<TimeTableObject> timetable;
public Student(int id, int branch, int points);
public void addToTimetable(int course, int sectionNumber, int valueAssigned, boolean cdc);
public void printTimeTable();
public String toString();
```



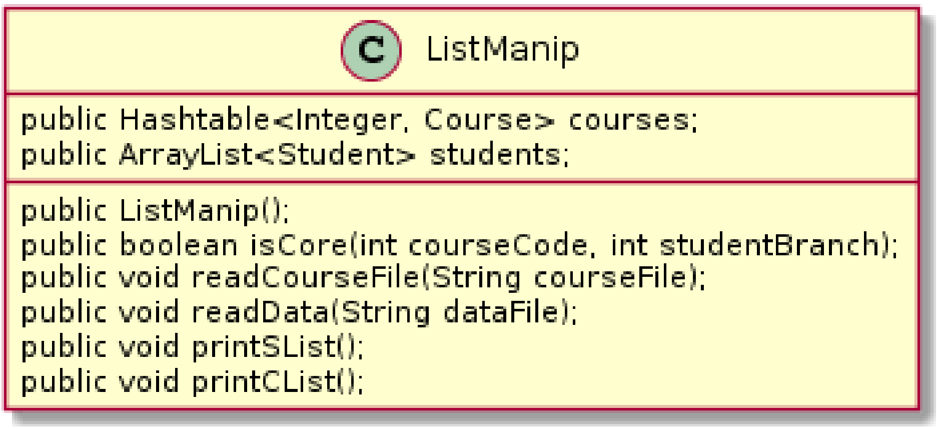
```
Course
public int courseCode;
public int coreBranch;
public ArrayList<Section> sectionList;
public int cstrength;
public Course();
public void addSection(int sectionStrength);
public void removeSection(int sectionID);
public void printSectionList();
public String toString();
```



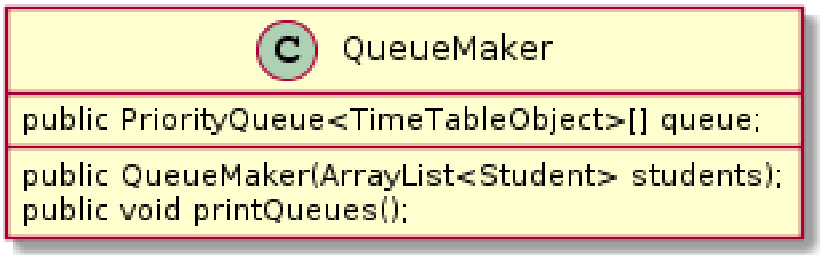
```
Section
public int courseCode;
public int sectionID;
public int strength;
public Section(int courseCode, int sectionID, int strength);
public String toString();
```



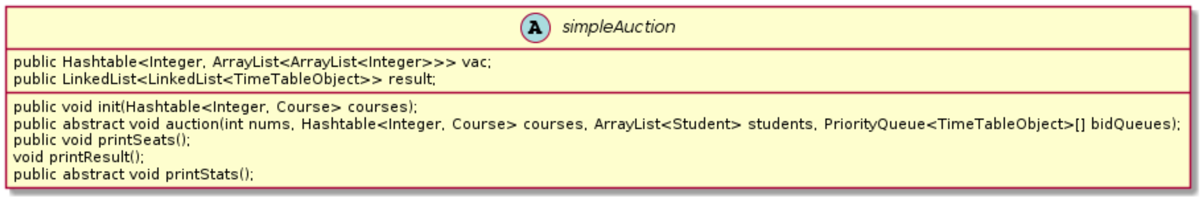
```
TimeTableObject
public int id;
public int courseCode;
public int sectionID;
public int bidValue;
public boolean isCDC;
public TimeTableObject(int id, int course, int sectionID, int bidValue, boolean cdc);
public String toString();
public int compareTo(TimeTableObject o);
```



```
ListManip
public Hashtable<Integer, Course> courses;
public ArrayList<Student> students;
public ListManip();
public boolean isCore(int courseCode, int studentBranch);
public void readCourseFile(String courseFile);
public void readData(String dataFile);
public void printSList();
public void printCList();
```

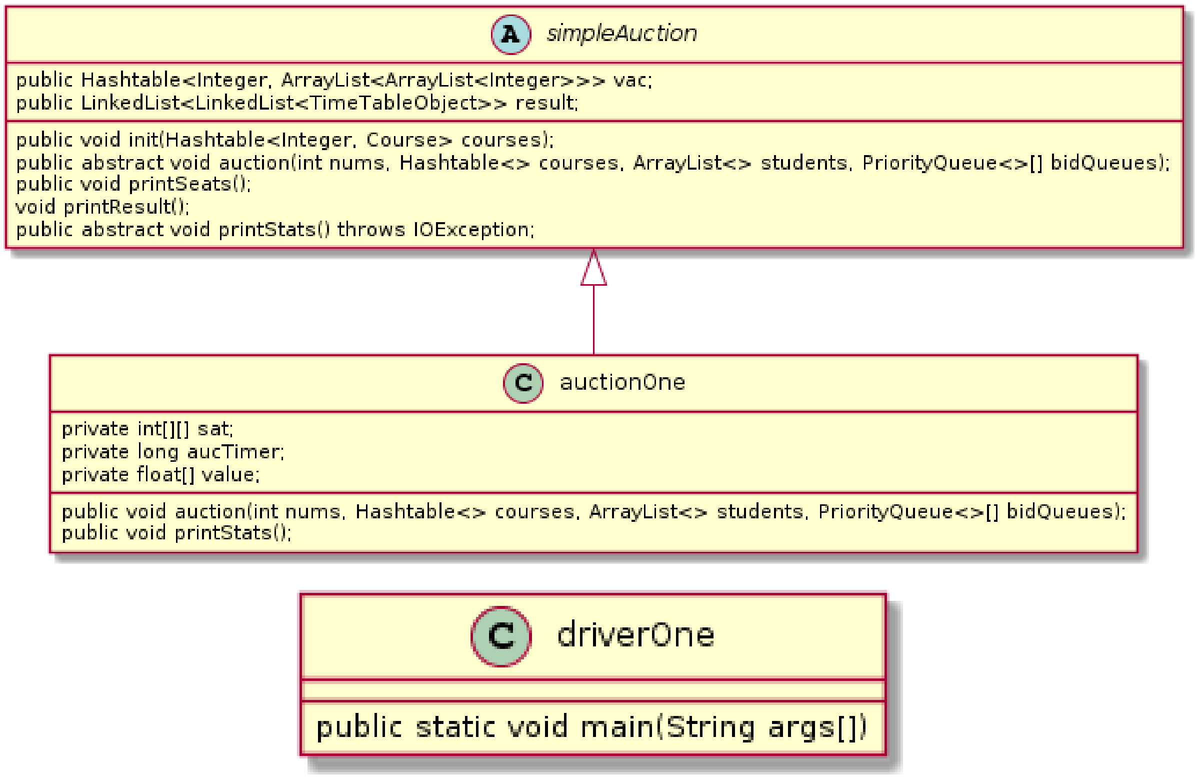


```
public PriorityQueue<TimeTableObject>[] queue;

public QueueMaker(ArrayList<Student> students);
public void printQueues();
```



```
public Hashtable<Integer, ArrayList<ArrayList<Integer>>> vac;
public LinkedList<LinkedList<TimeTableObject>> result;

public void init(Hashtable<Integer, Course> courses);
public abstract void auction(int nums, Hashtable<Integer, Course> courses, ArrayList<Student> students, PriorityQueue<TimeTableObject>[] bidQueues);
public void printSeats();
void printResult();
public abstract void printStats();
```

A.2 AuctionOne

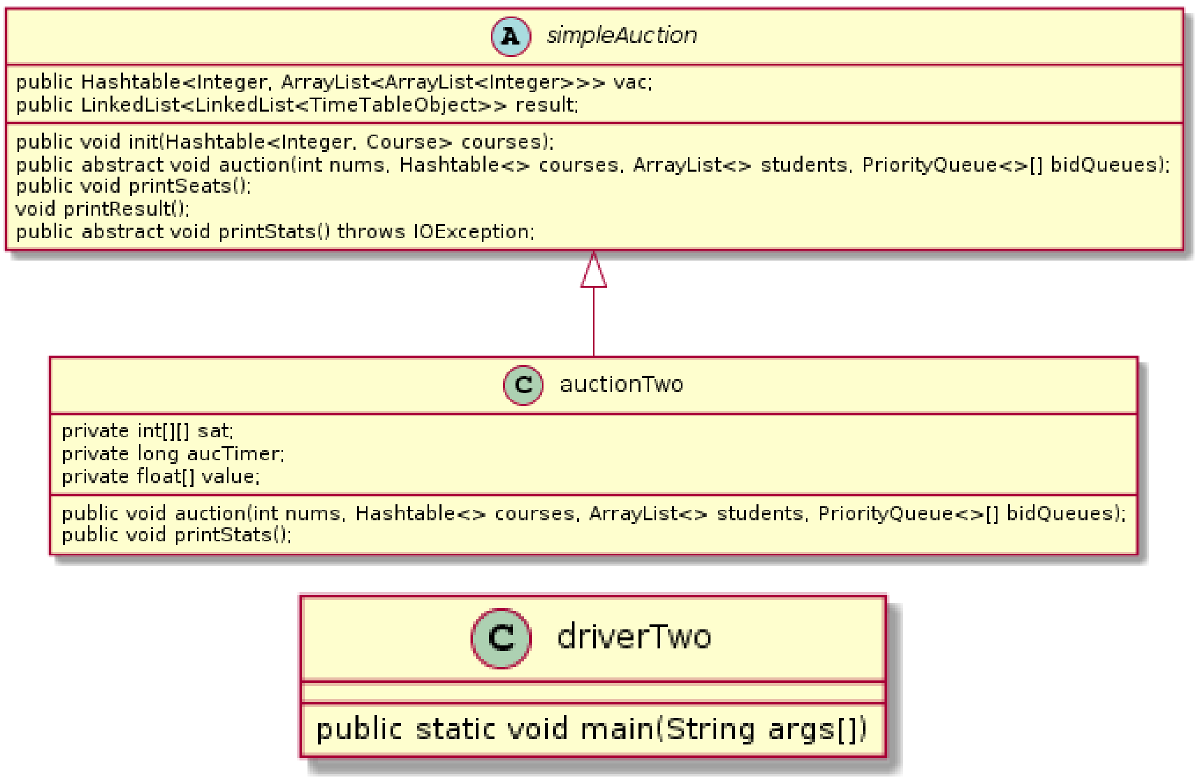


```
public Hashtable<Integer, ArrayList<ArrayList<Integer>>> vac;
public LinkedList<LinkedList<TimeTableObject>> result;

public void init(Hashtable<Integer, Course> courses);
public abstract void auction(int nums, Hashtable<> courses, ArrayList<> students, PriorityQueue<>[] bidQueues);
public void printSeats();
void printResult();
public abstract void printStats() throws IOException;
```

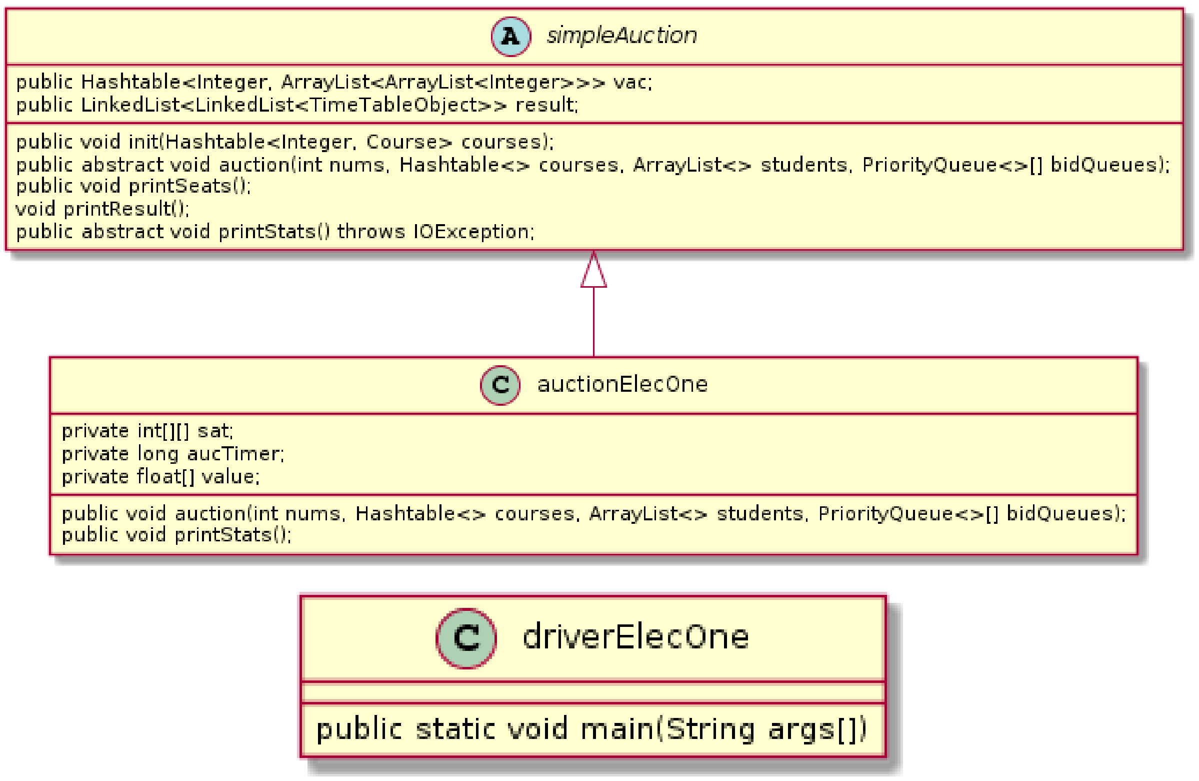
```
private int[][] sat;
private long aucTimer;
private float[] value;

public void auction(int nums, Hashtable<> courses, ArrayList<> students, PriorityQueue<>[] bidQueues);
public void printStats();
```

```
public static void main(String args[])
```

A.3 AuctionTwo

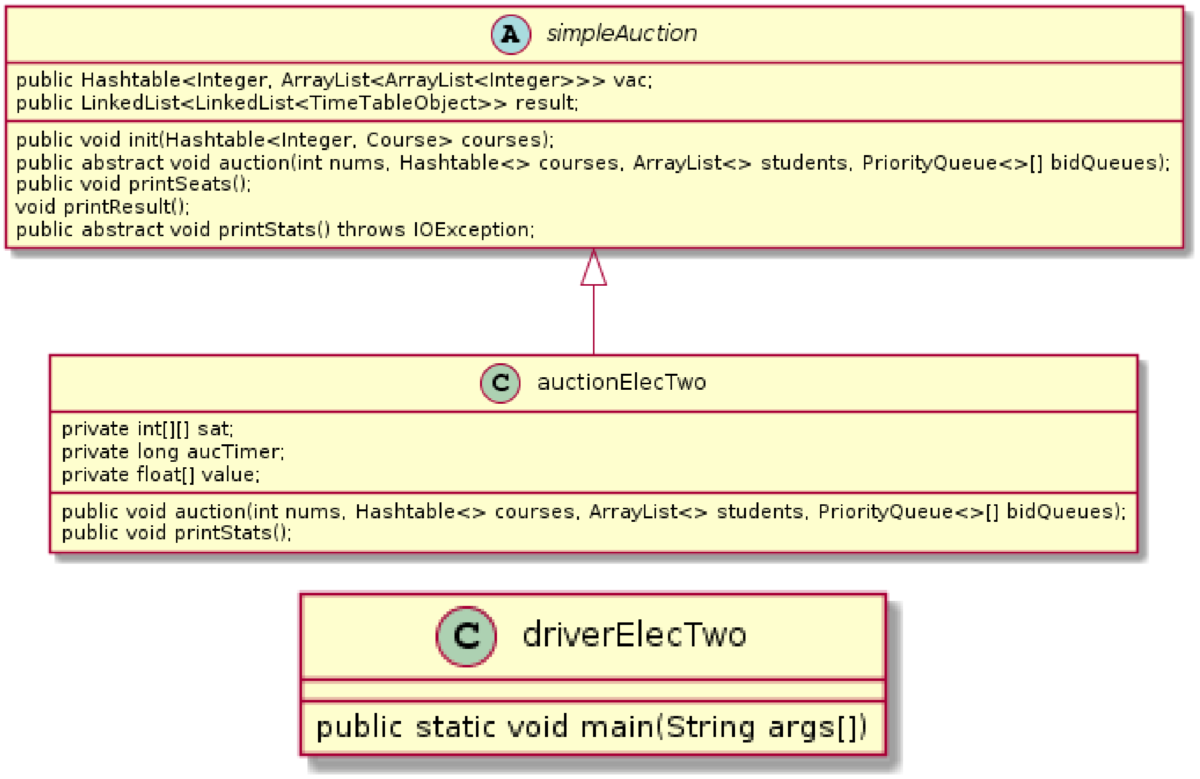

A simpleAuction
public Hashtable<Integer, ArrayList<ArrayList<Integer>>> vac;
public LinkedList<LinkedList<TimeTableObject>> result;
public void init(Hashtable<Integer, Course> courses);
public abstract void auction(int nums, Hashtable<> courses, ArrayList<> students, PriorityQueue<>[] bidQueues);
public void printSeats();
void printResult();
public abstract void printStats() throws IOException;
C auctionTwo
private int[][] sat;
private long aucTimer;
private float[] value;
public void auction(int nums, Hashtable<> courses, ArrayList<> students, PriorityQueue<>[] bidQueues);
public void printStats();
C driverTwo
public static void main(String args[])

A.4 AuctionThree



```
simpleAuction
public Hashtable<Integer, ArrayList<ArrayList<Integer>>> vac;
public LinkedList<LinkedList<TimeTableObject>> result;
public void init(Hashtable<Integer, Course> courses);
public abstract void auction(int nums, Hashtable<> courses, ArrayList<> students, PriorityQueue<>[] bidQueues);
public void printSeats();
void printResult();
public abstract void printStats() throws IOException;

auctionElecOne
private int[][] sat;
private long aucTimer;
private float[] value;
public void auction(int nums, Hashtable<> courses, ArrayList<> students, PriorityQueue<>[] bidQueues);
public void printStats();

driverElecOne
public static void main(String args[])
```

A.5 AuctionFour



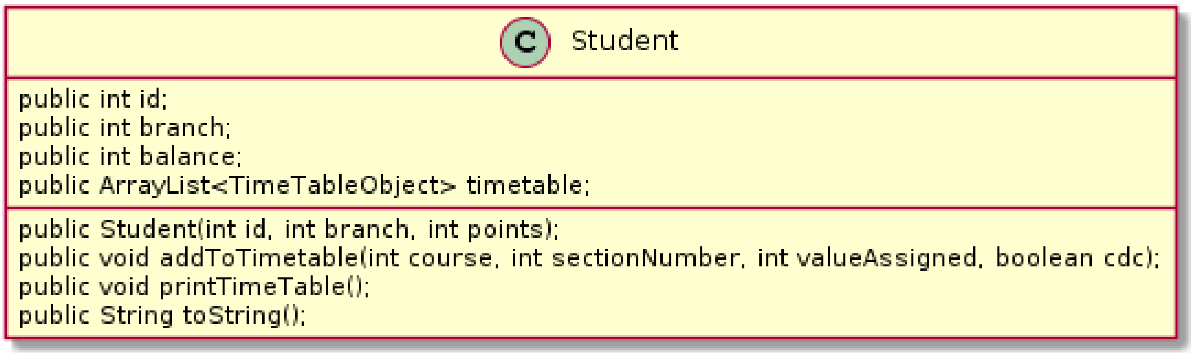
```
public Hashtable<Integer, ArrayList<ArrayList<Integer>>> vac;
public LinkedList<LinkedList<TimeTableObject>> result;

public void init(Hashtable<Integer, Course> courses);
public abstract void auction(int nums, Hashtable<> courses, ArrayList<> students, PriorityQueue<>[] bidQueues);
public void printSeats();
void printResult();
public abstract void printStats() throws IOException;
```

```
private int[][] sat;
private long aucTimer;
private float[] value;

public void auction(int nums, Hashtable<> courses, ArrayList<> students, PriorityQueue<>[] bidQueues);
public void printStats();
```

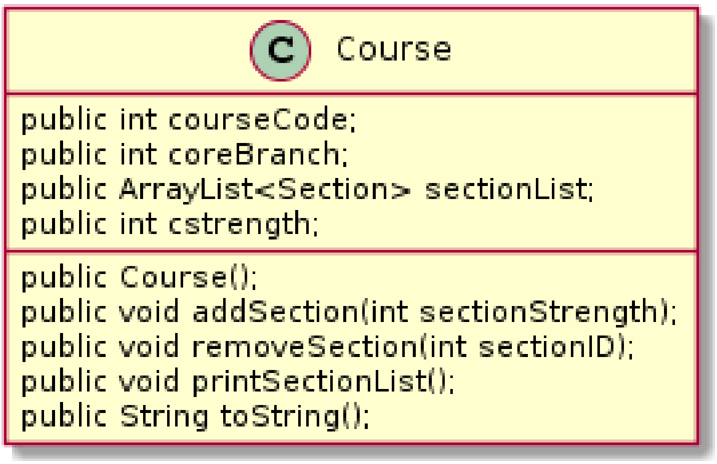
```
public static void main(String args[])
```

A.6 The Existing Model

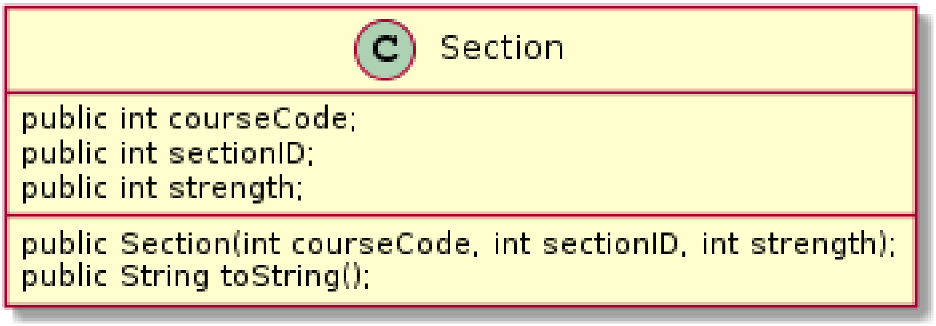
Behaviours of some helper classes such as *ListManip* and *QueueMaker* as well as the classes *prNumber* and *driverPR* are different from those of the four auction models.



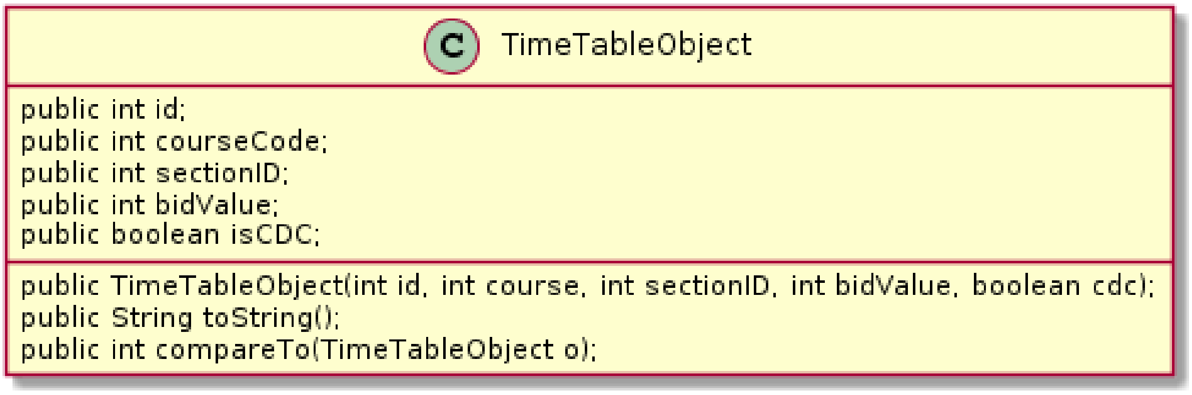
```
public int id;
public int branch;
public int balance;
public ArrayList<TimeTableObject> timetable;

public Student(int id, int branch, int points);
public void addToTimetable(int course, int sectionNumber, int valueAssigned, boolean cdc);
public void printTimeTable();
public String toString();
```



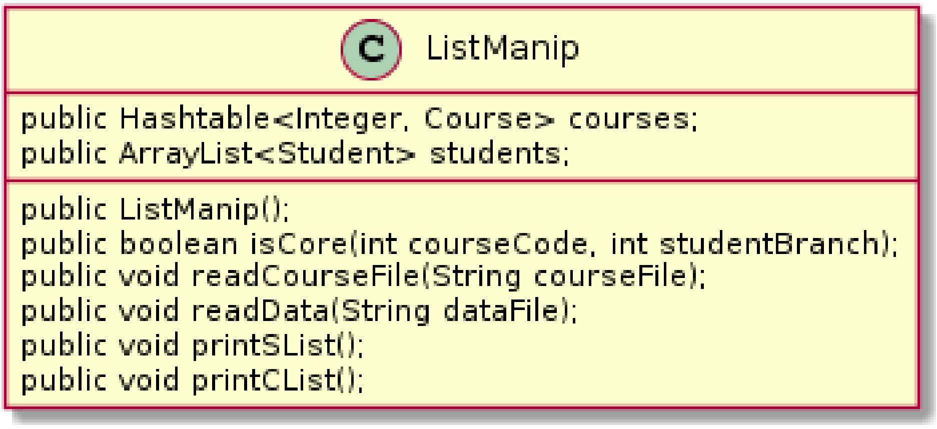
```
Course
public int courseCode;
public int coreBranch;
public ArrayList<Section> sectionList;
public int cstrength;
public Course();
public void addSection(int sectionStrength);
public void removeSection(int sectionID);
public void printSectionList();
public String toString();
```



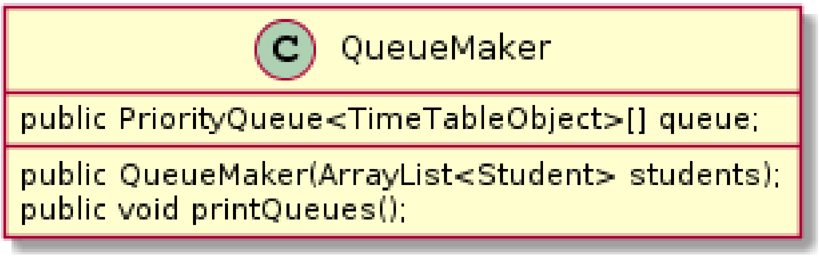
```
Section
public int courseCode;
public int sectionID;
public int strength;
public Section(int courseCode, int sectionID, int strength);
public String toString();
```



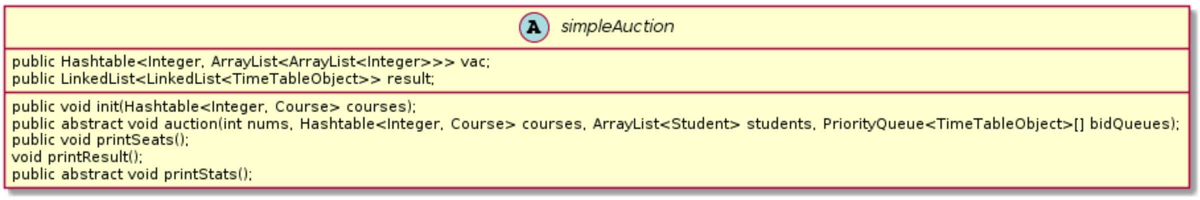
```
TimeTableObject
public int id;
public int courseCode;
public int sectionID;
public int bidValue;
public boolean isCDC;
public TimeTableObject(int id, int course, int sectionID, int bidValue, boolean cdc);
public String toString();
public int compareTo(TimeTableObject o);
```



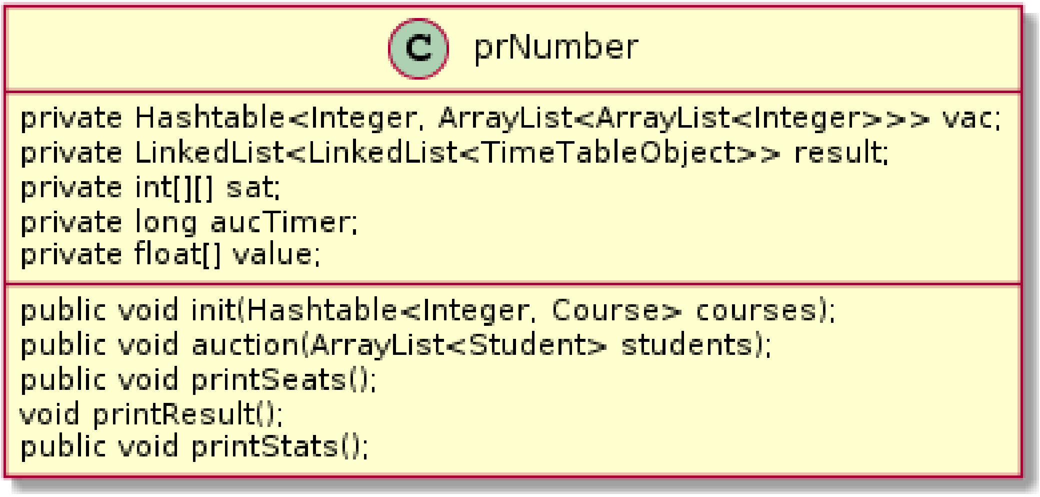
```
ListManip
public Hashtable<Integer, Course> courses;
public ArrayList<Student> students;
public ListManip();
public boolean isCore(int courseCode, int studentBranch);
public void readCourseFile(String courseFile);
public void readData(String dataFile);
public void printSList();
public void printCList();
```



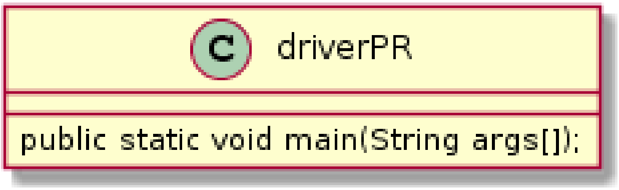
```
public PriorityQueue<TimeTableObject>[] queue;

public QueueMaker(ArrayList<Student> students);
public void printQueues();
```



```
public Hashtable<Integer, ArrayList<ArrayList<Integer>>> vac;
public LinkedList<LinkedList<TimeTableObject>> result;

public void init(Hashtable<Integer, Course> courses);
public abstract void auction(int nums, Hashtable<Integer, Course> courses, ArrayList<Student> students, PriorityQueue<TimeTableObject>[] bidQueues);
public void printSeats();
void printResult();
public abstract void printStats();
```



```
private Hashtable<Integer, ArrayList<ArrayList<Integer>>> vac;
private LinkedList<LinkedList<TimeTableObject>> result;
private int[][] sat;
private long aucTimer;
private float[] value;

public void init(Hashtable<Integer, Course> courses);
public void auction(ArrayList<Student> students);
public void printSeats();
void printResult();
public void printStats();
```



```
public static void main(String args[]);
```

Bibliography

[1] Noam Nisan et al. *Algorithmic Game Theory.* Cambridge University Press, 2007.

[2] Christoph Brunner et al. "Combinatorial Auctioneering". In: 2006.

[3] Klemperer. *Auctions - Theory and Practice.* 2004.

[4] Vijay Krishna. *Auction Theory.* Academic Press, 2002.

[5] Lawrence M. Ausubel. "An Efficient Ascending-Bid Auction for Multiple Objects". In: *American Economic Review* 94 (Dec. 2004), pp. 1452–1475. DOI: `10.1257/0002828043052330`.

[6] William Vickrey. "Counterspeculation, Auctions, and Competitive Sealed Tenders". In: *The Journal of Finance* 16.1 (1961), pp. 8–37.