1

Introduction and Examples

This book provides an introduction to the topics of game theory and market design, with a focus on how AI and optimization methods can be used to understand these problems, as well as enable them in practical settings. The book covers several application areas for these ideas, where each area will have real-life applications that have been deployed. A common theme underlying the areas covered by the book is that for each area, one or more of the real applications are enabled by AI and optimization. Firstly, we will repeatedly see that economic solution concepts often have some underlying convex or mixed-integer formulation of the problem, that allows us to analyze the problem via optimization theory, as well as enabling algorithms via optimization techniques. Secondly, the book uses the concept of *online learning* (also known as no-regret learning; we will use these terms interchangeably) as a unifying theme for enabling algorithms and analysis across many of the economic topics that we cover. Thirdly, applications such as poker require scaling at a level where standard optimization and online learning methods are not enough. In those settings, AI methods such as abstraction or machine learning are often used. For example, in sequential games such as poker, we may have a game that is way too large to even fit in memory. In that case, machine learning may be used to generate some coarse-grained representation of the problem. This coarse-grained representation is then typically what we solve with optimization methods. The following subsections give examples of the types of ideas and applications the book will cover.

1.1 Game Theory

The first pillar of the course will be *game theory*. In classical optimization, we have some form of objective function that we try to minimize or maximize,

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

say $\max_{x \in X} f(x)$, where X is a convex set of possible choices, and f is some concave function. For example, perhaps we are thinking of X as a set of prices that a retailer can set for a given item, and $f(x)$ tells us the revenue that the retailer gets when setting the price x.

In game theory, on the other hand, we study settings where multiple individuals make choices, and the outcome depends on the choices of all the individuals. Suppose that we have two retailers, each choosing prices x_1 and x_2 respectively. Now, suppose that f_1 is a function that tells us the revenue received by retailer 1 in this setup. Since consumers will potentially compare the prices x_1 and x_2, we should expect f_1 to depend on both x_1 and x_2, so we let $f_1(x_1, x_2)$ be the revenue for retailer 1 generated under prices x_1 and x_2. Now we can again try to think of the optimization problem that retailer 1 wishes to solve; first let us assume that x_2 was already chosen and retailer 1 knows its value, in that case they want to solve $\max_{x_1 \in X} f_1(x_1, x_2)$. However, we could similarly argue that retailer 2 should choose their price x_2 based on the price x_1 chosen by retailer 1. Now we have a problem, because we cannot talk about optimally choosing either of the two prices in isolation, and instead we need a way to reason about how they might be chosen in a way that depends on each other. Game theory provides a formal way to reason about this type of situation. For example, the famous *Nash equilibrium*, which we will introduce below, specifies that we should find a pair x_1, x_2 such that they are mutually optimal with respect to each other. Another solution concept we will see is the *Stackelberg equilibrium*, where one retailer is assumed to go first, while anticipating the optimization problem being solved by the second retailer. From now on we will refer to each individual optimizer in a problem either as a *player* or an *agent*.

1.1.1 Nash Equilibrium

One of the most important ideas in game theory is the famous Nash equilibrium. A Nash equilibrium is a specification of an action for each player (or a probability distribution over actions) such that it is a steady state of the game, in the sense that no player wishes to change their probability distribution over actions, given the strategy of every other player. This is best illustrated with an example. Table 1.1 gives the payoffs for the game of rock-paper-scissors (RPS), specified as a *bimatrix* of payoffs. When specified as a bimatrix, the interpretation of the game is as follows. The set of actions for Player 1 is the rows of the matrix, and the set of actions for Player 2 is the columns of the matrix. Each entry in the bimatrix is a pair of payoffs, where the first value is the payoff to Player 1 and the second value is the payoff to Player 2. For example, if Player 1 chooses Paper (the second row) and Player 2 chooses Rock (the first column),

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

	Rock	Paper	Scissors
Rock	0,0	−1,1	1,−1
Paper	1,−1	0,0	−1,1
Scissors	−1,1	1,−1	0,0

Table 1.1 *The payoff matrix for Rock, Paper, Scissors.*

we get the outcome $(1, -1)$. In this outcome, Player 1 receives a payoff of 1 and Player 2 receives a payoff of -1. The goal for each player is to maximize their own payoff. A *pure* Nash equilibrium is then a pair of actions (i.e., a row and a column) such that each player is choosing a payoff-maximizing action given the choice of the other player. A *mixed-strategy* Nash equilibrium (also referred to simply as a Nash equilibrium) is a probability distribution for each player such that they maximize their own payoff given the probability distribution of the other player.

Here is an example of something that is *not* a Nash equilibrium: Player 1 always plays rock, and Player 2 always plays scissors. In this case, Player 2 is not playing optimally given the strategy of Player 1, since they could improve their payoff from -1 to 1 by switching to deterministically playing paper. In fact, this argument works for any pair of deterministic strategies, and so we see that there is no pure Nash equilibrium. Instead, RPS is an example of a game where we need randomization in order to arrive at a Nash equilibrium. Now each player gets to choose a probability distribution over their actions instead (e.g., a distribution over rows for Player 1). The value that a given player receives under a pair of mixed strategies is their expected payoff given the randomized strategies. In RPS, it's easy to see that the unique Nash equilibrium is for each player to play each action with probability $\frac{1}{3}$. Given this distribution, there is no other action that either player can switch to and improve their utility. This is what we call a (mixed-strategy) Nash equilibrium.

The famous result of John Nash from 1951 is that *every* game has a Nash equilibrium, once we allow for mixed strategies. Stated specifically for bimatrix games, the result is:

Theorem 1.1 *Every bimatrix game has a (potentially mixed-strategy) Nash equilibrium.*

In Chapter 2 we will see that Nash's result is broader than this. It guarantees existence for n-player games with a finite set of actions for each player, as long as we allow for mixed strategies. In Chapter 10 we will see a proof of this result, and an extension to a broader class of games.

The attentive reader may have noticed a certain redundancy in our bimatrix

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

representation of the RPS game: the payoffs are all in the form $(1, -1)$, $(0, 0)$, and $(-1, 1)$. In all three cases, the payoff of Player 1 is exactly -1 times the payoff of Player 2. From here, we can deduce that the players have completely opposite preferences: when one player wins, the other loses. More generally, games where the sum of the two players' payoffs equal zero are called *zero-sum games*. In a two-player zero-sum game, each player can equivalently reason about minimizing the utility of the other player, rather than maximizing their own utility. Because of this special structure, zero-sum games can be represented by a single payoff matrix $A \in \mathbb{R}^{n \times m}$, where entry A_{ij} is the payoff to Player 2 when Player 1 takes the action with index i and Player 2 takes the action with index j. In this formulation, Player 1 then wishes to minimize their expected payoff over A. We will see later that this allows us to write the problem in the following form:

$$\min_{x \in \Delta^n} \max_{y \in \Delta^m} x^\top A y,$$

where $\Delta^n = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i = 1, x \geq 0\}$ is the probability simplex over n actions, and Δ^m is the probability simplex of m actions. Problems of this form are variously known as *matrix games*, two-player zero-sum games, or more broadly as *bilinear saddle-point problems*. The key here is that we can now represent the outcome that the players respectively want to minimize and maximize as a single objective, thereby allowing us to write the problem as a nested minimization and maximization problem (we will see later that this ordering is not important, and it can equivalently be written as a nested maximization and minimization problem). Zero-sum matrix games are very special: they can be solved in polynomial time with a linear program (LP) whose size is linear in the matrix size.

RPS is, of course, a rather trivial example of a game. A more exciting application of zero-sum games is to use it to compute an optimal strategy for two-player poker (AKA heads-up poker). In fact, this was the foundation for many recent "superhuman AI for poker" results, as we shall discuss later. In order to model poker games we will need a more expressive game class called *extensive-form games* (EFGs). These games are played on trees, where players may sometimes have groups of nodes, called *information sets*, that they cannot distinguish among. An example is shown in Figure 1.1.

EFGs can also be represented as a bilinear saddle-point problem:

$$\min_{x \in X} \max_{y \in Y} x^\top A y,$$

where X, Y are no longer probability simplexes, but more general convex polytopes that encode the sequential decision spaces of each player. This is called

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

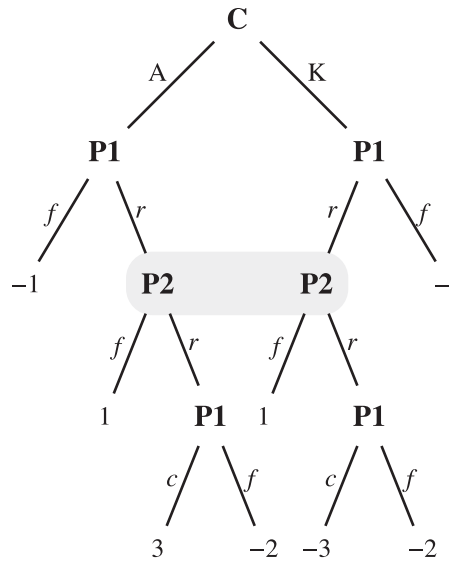


Figure 1.1 A poker game where P1 is dealt Ace or King. "r," "f," and "c" stands for raise, fold, and check respectively. Leaf values denote P1 payoffs. The shaded area denotes an information set: P2 does not know which of these nodes they are at, and must thus use the same strategy in both.

the *sequence-form* representation, covered in Chapter 8. Like matrix games, zero-sum EFGs can be solved in polynomial time with linear programming, with an LP whose size is linear in the game tree.

It turns out that in many practical scenarios, the LP for solving a zero-sum game ends up being far too large to solve. This is especially true for EFGs, where the game tree can quickly become extremely large because the LP size grows exponentially in the depth of the game. Instead, iterative methods are used in practice. What is meant by iterative methods here is the class of algorithms that build a sequence of strategies $x_0, x_1, \ldots, x_T, y_0, y_1, \ldots, y_T$ using only some form of oracle access to Ay and $A^\top x$ (this is different from writing down A explicitly!). Typically, in such iterative methods the average of the sequence of strategies $\bar{x}_T = \frac{1}{T}\sum_{t\in[T]} x_t, \bar{y}_T = \frac{1}{T}\sum_{t\in[T]} y_t$ converges to a Nash equilibrium. The reason these methods are preferred is twofold. First, by never writing down A explicitly we save a lot of memory (now we just need enough memory to store the much smaller x, y strategy vectors). Secondly, they avoid the expensive matrix inversions involved in the typical algorithms for solving LPs such as the simplex algorithm and interior-point methods.

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

The algorithmic techniques we will learn for Nash equilibrium computation are largely centered around iterative methods. First, we will do a quick introduction to online learning and online convex optimization. We will learn about two classes of algorithms: (1) Methods that converge to an equilibrium at a rate of $O(1/\sqrt{T})$. These roughly correspond to saddle-point variants of gradient-descent-like methods. (2) Methods that converge to the solution at a rate of $O(1/T)$. These roughly correspond to saddle-point variants of accelerated gradient methods. Then we will also look at the practical performance of these algorithms. Here we will see that the following quote is very much true:

In theory, theory and practice are the same. In practice, they are not.

In particular, the preferred method in practice is the CFR$^+$ algorithm and later variations, all of which have a theoretical convergence rate of $O(1/\sqrt{T})$. The methods that converge at a rate of $O(1/T)$ in theory are actually slower than CFR$^+$ for most real games!

Being able to compute an approximate Nash equilibrium with iterative methods is only one part of how superhuman AIs were created for poker. In addition, abstraction and deep learning methods were used to create a small enough game that can be solved with iterative methods. We will also cover how these methods are used.

Killer applications of zero-sum games include poker (as we saw), other recreational two-player games, and generative-adversarial networks. Other applications that are, as yet, less proven to be effective in practice are robust sequential decision-making (the adversary represents uncertainty), security scenarios where we assume the world is adversarial, and defense applications.

1.1.2 Stackelberg Equilibrium

A second game-theoretic solution concept that has had extensive application in practice is what's called a *Stackelberg equilibrium*. We will primarily study Stackelberg equilibrium in the context of what is called *security games*.

Imagine the following scenario: we are designing the patrol schedule for national park rangers that try to prevent poaching of endangered wildlife in the park (such as rhinos, which are poached for their horns). There are 20 different watering holes that the rhinos frequent. We have five teams of guards that can patrol watering holes. How can we effectively combat poaching? If we come up with a fixed patrol schedule, then the poachers can observe us for a few days and learn our schedule exactly. Afterwards they can strike at a waterhole that is guaranteed to be empty at some particular time. Thus, we need to design a schedule that is unpredictable, but which also accounts for the fact that some

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

watering holes are more frequented by rhinos (and are thus of higher value), travel constraints, and so forth.

In the security games literature, the most popular solution concept for this kind of setting is the Stackelberg equilibrium. In a Stackelberg equilibrium, we assume that we, as the leader (e.g., the park rangers), get to commit to our (possibly randomized) strategy first. Then, the follower observes our strategy and best responds. This turns out to yield the same solution concept as Nash equilibrium in zero-sum games, but in general games it leads to a different solution concept.

However, if we want to help the park rangers design their schedules, then we will need to be able to compute Stackelberg equilibria of the resulting game model. Again, we will see that optimization is one of the fundamental pillars of the field of security games research. A unique feature of security games is that the strategy space of the leader is typically some combinatorial polytope (e.g., a restriction on the *transportation polytope*), and the problem of computing a Stackelberg equilibrium is intimately related to optimization over the underlying polytope of the defender. Because of this combinatorial nature, security games often end up being much harder to solve than zero-sum Nash equilibrium. Therefore, our focus in the security games section will be on combinatorial approaches to this problem, such as mixed-integer programming and decomposition.

Killer applications of Stackelberg games are mainly in the realm of security. They have been applied in infrastructure security (airports, coast guard, air marshals), to protect wildlife, and to combat fare evasion. A nascent literature is also emerging in cybersecurity. Outside the world of security, Stackelberg games are also used to model things like first mover advantage in the business world.

1.2 Market Design

The second pillar of the book will be *market design*. In market design we are typically concerned with how to design the rules of the game, and how to do that in order to achieve "good" outcomes. Thus, game theory is a key tool in market design, since it will be our way of understanding what outcomes we may expect to arise, given a set of market rules.

Market design is a huge area, and so it has many killer applications. The ones we will see in this course include Internet ad auctions and how to fairly assign course seats to students. There are many others, such as how to price and assign drivers to passengers at Lyft/Uber, how to assign NYC kids to schools, how to

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

	Course A	Course B
Student 1	5	5
Student 2	2	8

Table 1.2 *Example course matching valuations.*

enable nationwide kidney exchanges, and how to allocate the radio spectrum efficiently.

Imagine that we are designing a mechanism for managing course enrollment. How should we decide which students get to take which courses? Certain courses are likely to have significantly more demand for seats than there are sets (e.g., machine learning courses were heavily overdemanded in the 2010s). Overall, we would like the system to somehow be *efficient*, but what does that mean? We would also like the system to be *fair*, but it's not entirely clear what that means either.

At a loss for ideas, we come up with the following solution: we will just have a sign up system where students can sign up until a course fills up. Once a course fills up, we put other students on a waitlist that we clear on a first-in first-out basis as seats become available. The reader may find this system familiar, as it is used at many universities to manage course enrollment. Is this a good system? Well, let's look at a simple example: we will have two students and two courses, each course having one seat. Students are allowed to take at most one course. Let's say that each student values the courses as in Table 1.2:

Student 1 arrives first, is indifferent between the courses, and arbitrarily signs up for course B. Then Student 2 arrives and signs up for A. The total *welfare* of this assignment is $5 + 2 = 7$. This does not seem to be an efficient use of resources: we can improve our solution by swapping the courses, since Student 1 gets the same utility as before, and Student 2 improves their utility. This is what's called a *Pareto-improving* allocation because each student is at least as well off as before, and at least one student is strictly better off. One desideratum for efficiency is that no such improvement should be possible; an allocation with this property is called *Pareto efficient*.

Let's look at another example. Now we have two students and four courses, where each student takes two courses. See Table 1.3 for the valuations. Again, courses have only 1 seat. Now say that Student 1 shows up first, and signs up for A and B. Then Student 2 shows up and signs up for C and D. Call this assignment A_1. Here we get that A_1 is Pareto efficient, as there is no way to improve the utility of Student 2 without hurting the utility of Student 1. But it does not seem like a fair assignment. A fairer solution would be that each

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

	Course A	Course B	Course C	Course D
Student 1	10	10	1	1
Student 2	10	10	1	1

Table 1.3 *Example course matching valuations.*

student gets a course with value 10 and a course with value 1, for a total utility of 11 each. Let A_2 be such an assignment. One way to look at this improvement is through the notion of *envy*: each student should like their own course schedule at least as well as that of any other student. Under A_1 Student 2 envies Student 1 by a value of 18, whereas under A_2 no student envies the other. An allocation where no student envies another student is called *envy-free*. Fairness turns out to be a complicated idea, and we will see later that there are several appealing notions that we may wish to strive for.

Instead of first come, first serve, we can use ideas from market design to get a better mechanism. The solution that we will learn about is based on a fake-money market: we give every student some fixed budget of fake currency (aka funny money). Then, we treat the assignment problem as a market problem under the assigned budgets, and ask for what is called a *market equilibrium*. Briefly, a market equilibrium is a set of prices, one for each item, and an allocation of items to buyers. The allocation must be such that every item is fully allocated (or has a price of zero), and every buyer is getting an assignment that maximizes their utility over all the possible assignments they could afford given the prices and their budget. Given such a market equilibrium, we then take the allocation from the equilibrium, throw away the prices (the money was fake anyway!), and use that to perform our course allocation. This turns out to have a number of attractive fairness and efficiency properties. Course-selection mechanisms based on this idea are deployed at several business schools, such as Columbia Business School, the Wharton School at University of Pennsylvania, Rotman School of Management at University of Toronto, and Tuck School of Business at Dartmouth.

Of course, if we want to implement this protocol, we need to be able to compute a market equilibrium. This turns out to be a rich research area: in the case of what is called a *Fisher market*, where each agent i has a linear utility function $v_i \in \mathbb{R}^m_{\geq 0}$ over the m items in the market, and the items are divisible,

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press

there is a beautiful convex program that results in a market equilibrium:[1]

$$\begin{aligned} \max_{x \geq 0} \quad & \sum_i B_i \log(v_i^\top x_i) \\ \text{s.t.} \quad & \sum_{i \in [n]} x_{ij} \leq 1, \forall j \in [m]. \end{aligned}$$

Here x_{ij} is how much buyer i is allocated of item j. Notice that we are simply maximizing the budget-weighted logarithmic utilities, with no prices! It turns out that the prices are the dual variables on the supply constraints. We will see some nice applications of convex duality and Fenchel conjugates in deriving this relationship. We will also see that this class of markets have a relationship to the types of auction systems that are used at large internet companies such as Google and Meta for allocating ad slots.

In the case of markets such as those for course seats, the problem is computationally harder and requires combinatorial optimization. Current methods use a mixture of MIP and local search.

[1] Note the implicit domain constraint that $v_i^\top x_i > 0$ for each i. It is usually assumed without loss of generality that each agent i has strictly positive value for at least one item.

https://doi.org/10.1017/9781009711265.003 Published online by Cambridge University Press