Chapter 1

Introduction

Imagine you are a new dean at a large university, and your first task is allocating seats in courses to students for the coming semester[1]. Each student has applied for some subset of the courses available, and each course has a limit on the number of students it can accept. In addition, there are scheduling conflicts between courses, a limited number of seats per course, and the students have hard limits on the number of courses they can take in a semester. As a good dean, you wish to do this fairly, so that the students are happy with their allocations, and do not feel disfavored compared to the other students, and efficiently, so that every student gets enough courses to make the expected progress on their grade. How would you go about solving this problem? Over lunch, you discuss your quandary with a colleague from the computer science department who immediately assuages your fears by informing you that your problem is in fact an instance of indivisible fair allocation with matroid-rank valuations, a well-studied problem for which several algorithms exist! Eager to learn more, you ask your colleague to expound upon her cryptic remark.

What is fair allocation?

Fair allocation is the problem of fairly partitioning a set of resources (in this case, the courses) among agents (the students) with different preferences or valuations

[1] This example is due to Benabbou et al [6].

over these resources. This has been a hot topic of interest since antiquity (a 2000-year old allocation strategy from the Talmud is given in [3]), and remains so today. The mathematical study of fair allocation started with Steinhaus as late as 1948 [26], and for decades the focus was largely on the *divisible* case, in which the resources can be divided into arbitrary small pieces. In the divisible case, fair allocations always exist, and can be computed efficiently [2]. In the dean's scenario, however, the course seats are *indivisible goods*, and so this is a case of indivisible fair allocation. In such a setting, fairness is not always achievable; consider for example allocating a course with one seat between two students who both applied for it. Lucky for the dean, this is a problem which in the last couple of decades has garnered the attention of computer scientists, who have brought algorithmic techniques to the field to great effect.

Generally speaking, an allocation is measured against two justice criteria: *fairness* and *efficiency*. Fairness has to do with the degree to which agents feel that the allocation favors other agents compared to themselves. One common way to describe the *fairness* of an allocation is with the concept of *envy-freeness*, which requires that no agent values another agent's received bundle of resources higher than their own. In the trivial example above, the only envy-free allocation is the one in which no student receives the seat; while this is technically speaking fair, it is highly inefficient. Efficiency deals with maximizing some notion of resource utilization, or, equivalently, reducing waste. The perfectly fair allocation in which no one receives anything is rarely desirable for reasons of efficiency. Conversely, while the allocation in which one agent receives everything is highly efficient (in terms of one notion of efficiency, utilitarian social welfare, which is defined as the sum of agent bundle values), it is obviously unfair. The task of the fair allocation algorithm, then, is to find some balance between these criteria.

How do matroids enter into this?

What the intrepid colleague from the computer science department noticed about the dean's problem, was that it was well-structured, in fact it is a textbook example of matroid-rank valuations in practice. There are good theoretical and practical reasons for the interest in matroid rank functions in the context of fair allocation. Theoretically, they are a well-studied class of submodular functions, which are the set functions that exhibit diminishing marginal returns, meaning that the marginal value for a single item decreases as the size of the input set increases, and are in fact exactly the set of submodular functions with binary

marginals [25, Chapter 39]. In practice, this means that matroid rank functions have desirable properties for modeling user preferences in a setting such as the dean's allocation scenario. The binary marginals allow us to model each student's either willingness (value of 1) or unwillingness (value of 0) to enroll in a given course. The diminishing returns property allow us to model what are known in economics terms as supplementary goods and fixed demand. A student might be interested in two similar courses, but not wish to enroll in both, so given one, the marginal value of the other drops to 0 (the courses are supplementary goods). A student has limited time and energy, and so for each course seat received, the marginal value of the other courses can only decrease – after some threshold is reached in the number of enrolled courses, all remaining courses have value 0 (there is a fixed demand for courses).

Matroids have been extensively studied for close to a century, and generalize concepts from a variety of different mathematical fields. A number of interesting algorithms have been developed for fair allocation with matroid-rank valuations [4, 5, 6, 7, 27] that make use of deep results from matroid theory in their analysis, and deliver well on a range of justice criteria which might be computationally intractable to achieve under general valuations.

What does this thesis contribute?

Perhaps because matroids are so well-understood and pleasant to work with theoretically, there is a dearth of tooling available for generating and working with them programmatically. In an effort to fill this gap, this thesis proposes *Matroids.jl*, a library for the Julia programming language [8], which extends the existing Allocations.jl library [19] with the functionality required to enable the empirical study of matroidal fair allocation algorithms.

This thesis describes the work that has been done to design and build a working, proof-of-concept version of Matroids.jl. It is structured as follows. In the next chapter, I establish the concepts from matroid theory and fair allocation necessary to follow the rest of the thesis. Chapter 3 describes the design of the API and the implementation of various classic matroid algorithms that have found use in fair allocation algorithms, along with methods for checking the fairness of an allocation with matroid-rank valuations, and other fair allocation sundries. In Chapter 4, I use this API to implement Viswanathan and Zick's Yankee Swap algorithm [27] and some other algorithms for matroid-rank-valued fair allocation. Chapter 5 details how Matroids.jl implements the random generation of a range of matroid types. Of particular interest here is Knuth's classic

method for the erection of arbitrary matroids [21], the successful implementation of which was a significant sub-goal of the project. Chapter 6 gives some experimental results for the algorithms over different matroid types. Finally, Chapter 7 gives a summary discussion on the limitiations of Matroids.jl and suggests a few possible avenues of future work.