Making Decisions Based on the Preferences of Multiple Agents

Vincent Conitzer
Departments of Computer Science and Economics
Duke University
Durham, NC, USA
conitzer@cs.duke.edu

People often have to reach a joint decision even though they have conflicting preferences over the alternatives. Examples range from the mundane—such as allocating chores among the members of a household—to the sublime—such as electing a government and thereby charting the course for a country. The joint decision can be reached by an informal negotiating process or by a carefully specified protocol. Philosophers, mathematicians, political scientists, economists, and others have studied the merits of various protocols for centuries. More recently, especially over the course of the past decade or so, computer scientists have also become deeply involved in this study. The perhaps surprising arrival of computer scientists on this scene is due to a variety of reasons, including the following.

1. Computer networks provide a new platform for communicating preferences. Examples include auction websites, where preferences are communicated in the form of bids, as well as websites that allow one to rate everything from the quality of a product to the attractiveness of a person.

2. Within computer science itself, there are increasingly many settings where a decision must be made based on the conflicting preferences of multiple parties. Examples include determining whose job gets to run first on a machine, whose network traffic is routed along a particular link, or whose advertisement is shown next to a page of search results.

3. Greater computing power and better algorithms, as well as a more computational mindset in the general public, have made it possible to run computationally demanding protocols that lead to much better outcomes. An example is an auction in which bidders can bid on arbitrary sets of items, rather than just on individual items (I will discuss such auctions in more detail later in this article). Such protocols used to be considered theoretical niceties that could never be run in practice (to the extent that they were conceived of at all), but now they are actually practical.

4. The paradigms of computer science give a different and useful perspective on some of the classic problems in economics and related disciplines. For example, various results in economics prove the existence of an equilibrium, but do not provide an efficient method for reaching such an equilibrium.

In this article, I give a (necessarily incomplete) survey of topics that computer scientists are working on in this domain. I will discuss voting and rank aggregation, task and resource allocation, kidney exchanges, auctions and exchanges, charitable giving, and prediction markets; in addition, I will discuss the problem of agents acting in their own best interest, which cuts across most of these applications. I also intersperse a few opinions and predictions about where future research should and will go.

In the below, the parties whose preferences we are interested in are not always people: they can also be, among other things, robots, software agents, or firms.[1] As is done in both computer science and economics, I will use the term *agent* to generically refer to one of the parties.

1. SETTINGS WITHOUT PAYMENTS

I will discuss a variety of settings, so it is helpful to categorize them somewhat. An important aspect is whether the setting allows agents to make payments to each other (in some currency). For example, in a voting setting, we typically do not imagine money changing hands among voters (unethical behavior aside). On the other hand, in an auction, we naturally expect the winning bidder to pay for her winnings. In this section, I discuss various settings in which no money changes hands; I will discuss settings with payments in the next section.

1.1 Voting and rank aggregation

A natural and very general approach for deciding among multiple alternatives is to *vote* over them. In the general theory of voting, agents can do more than vote for a single alternative: usually, they get to *rank* all the alternatives. For example, if a group of people is deciding where to go for dinner together, one of them may prefer American food to Brazilian, and Brazilian to Chinese. This person's vote can then be expressed as $A \succ B \succ C$.

[1] In artificial intelligence, there is the study of *multiagent systems*, where agents—for example, robots—often need a protocol for coordinating on (say) a joint plan.

Given everyone's vote, which cuisine should be chosen? The answer is far from obvious. We need a *voting rule* that takes as input a collection of votes, and as output returns the winning alternative. A simple rule known as the *plurality* rule chooses the alternative that is ranked first the most often. In this case, the agents do not really need to give a full ranking: it suffices to indicate one's most-preferred alternative, so each agent is in fact just voting for a single alternative.

Another rule is the *anti-plurality* rule, which chooses the alternative that is ranked *last* the *least* often. Now, it suffices for agents to report their last-ranked alternative—they are voting *against* an alternative. Which of these two rules is better? It is hard to say. The former tries to maximize the number of agents that are happy about the choice; the latter tries to minimize the number that are unhappy. Another rule, known as the *Borda* rule, tries to strike a balance: when there are three alternatives, it will give two points to an alternative whenever it is ranked first, one whenever it is ranked second, and zero whenever it is ranked last. Many other rules, most of them not relying on such a points-based scheme, have been proposed; *social choice* theorists analyze the desirable and undesirable properties of these rules.

Rather than just choosing a winning alternative, most of these rules can also be used to find an *aggregate ranking* of all the alternatives. For example, we can sort the alternatives by their Borda score, thereby deciding not only on the "best" alternative but also on the second-best, *etc.* There are numerous applications of this that are relevant to computer scientists: as an illustrative example, one can pose the same query to multiple search engines, and combine the resulting rankings of pages into an aggregate ranking.

One particularly nice rule for such settings is the *Kemeny* rule, which finds an aggregate ranking of the alternatives that "minimally disagrees" with the input rankings. More precisely, we say that a disagreement occurs whenever the aggregate ranking ranks one alternative above another, but one of the voters ranks the latter alternative above the former. The Kemeny rule produces a ranking that minimizes the total number of such disagreements (summed over both voters and pairs of alternatives).

The Kemeny rule has a number of desirable properties. For one, if we assume that there exists an unobserved "correct" ranking of the alternatives (reflecting their true quality), and each voter produces an estimate of this correct ranking according to a particular noisy process, then the Kemeny rule produces the maximum likelihood estimate of the correct ranking [40].

Unfortunately, finding the Kemeny rule's output ranking is computationally intractable (formally, NP-hard) [3]! Nevertheless, there are algorithms that can usually solve the problem in practice [8]. As an example, in Duke's computer science department, we used the Kemeny rule to find an aggregate ranking of our top Ph.D. applicants (based on the rankings of the individual admissions committee members); using the CPLEX solver, we found the Kemeny ranking for over a hundred applicants in under a minute.

While enabling the use of computationally demanding voting rules such as the Kemeny rule is valuable, I believe that, in the near future, computer scientists (specifically, the *computational social choice* community) will make much larger contributions to the theory and practice of voting. Real-world organizations often need to make not just a single decision, but rather decisions on a number of interrelated issues. In our dining example, the agents need to decide not only on a restaurant, but also on the time of the dinner; and an agent's preferred restaurant may depend on the time of the dinner. For example, an agent may prefer not to start a heavy Brazilian steakhouse meal shortly before going to bed.

In some sense, the "correct" way of handling this is to make the alternatives combinations of a time and a cuisine, so that an agent can say: "I prefer an early Brazilian meal to a late Chinese meal to..." However, this straightforward approach rapidly becomes impractical as more issues are combined, because the number of alternatives undergoes a combinatorial explosion. Ideally, the agents would have an expressive language in which they can naturally and concisely represent their preferences. One good language for representing such preferences is that of CP-nets [4] (which bear some resemblance to Bayesian networks). A CP-net allows a voter to specify that her preferences for one issue depend on the decisions on some other issues—for example, "If we are eating early, I prefer Brazilian; otherwise, I prefer Chinese." Given a language, we need to design new voting rules that can operate on preferences represented in this language, as well as algorithms for running these rules.

While such *combinatorial voting* [20, 38] is in its infancy, it is easy to see its potential value by considering how *ad hoc* the methods are that we use today for these types of situations. For example, members of Congress must vote on bills that address many different issues, and would often prefer to express preferences on individual issues. Unfortunately, voting on the individual issues separately can easily lead to undesirable results, because there is no guarantee that the issues are resolved in a consistent way. For instance, in the dining example, it may happen that most agents, in general, prefer to eat at a Brazilian steakhouse; and that, in general, most agents prefer to eat late; but most agents do not want to eat at a Brazilian steakhouse late at night. If they vote on the issues separately, the result may well be a late dinner at a Brazilian steakhouse. This is why the language for expressing preferences needs to allow the agents to specify some interactions among the issues.

1.2 Allocating tasks and resources

A voting scheme allows an agent to submit arbitrary preferences over the alternatives. While this generality is nice, in many settings, it is not needed, because we can safely make some assumptions about agents' preferences. Let us consider again the example of allocating chores in a household. One alternative might be: "Alice will vacuum and take out the trash, and Bob will do the dishes." It seems safe to assume that Bob will prefer this alternative to the alternative: "Alice will take out the trash, and Bob will vacuum and do the dishes," since the latter alternative gives Bob an additional task. On the other hand, if we are allocating desirable resources instead of cumbersome tasks, then presumably more is preferred to less. For example, if the agents jointly own a car, an alternative might be: "Alice gets to use the car on Friday, and Bob gets to use it on Saturday and Sunday," which Bob presumably prefers to the alternative "Alice gets to use the car on Friday and Saturday, and Bob gets to use it on Sunday." Here, the use of the car on a particular day is a "resource." These assumptions about preferences—receiving more tasks or fewer resources is never preferred—are com-

monly referred to as *monotonicity* assumptions.

Another reasonable assumption about preferences is that an agent only cares about which tasks or resources are allocated to her. For example, Alice is likely to be indifferent between "Alice gets the car on Friday, Bob on Saturday, and Carol on Sunday" and "Alice gets the car on Friday, Bob on Saturday and Sunday, and Carol never." In economics, the assumption that an agent, given her own resources and tasks, does not care about how the remaining resources and tasks are allocated to the other agents is known as the *no-externalities* assumption. It is not always completely accurate—Alice may dislike the alternative where Carol never gets the car slightly more, for example because Carol will ask Alice to run errands for her in that case—but it is usually assumed.

Reasonable assumptions such as the above allow us to get away from the full generality of the voting model, and make decisions in a way that is more specific to task and resource allocation. Incidentally, there are many applications of task and resource allocation within computer science. For example, we may allocate time on a supercomputer (or other computing resources) instead of time with a car. Also, instead of allocating the chores of a household to its inhabitants, we may allocate jobs to machines.

So, how should we allocate tasks and resources? By far the most common approach to this is to assume that the agents can make or receive *payments* in some currency, which leads us to *auction and exchange mechanisms*. I will discuss such mechanisms in more detail later on, but for now, I first consider methods that do not require payments. These methods will generally try to find an allocation that is "fair" in some sense.

One fairness criterion is *envy-freeness*: we should find an allocation such that every agent prefers her bundle (that is, the tasks or resources allocated to her) to each other agent's bundle. When resources are not divisible, an envy-free allocation is not always possible, and deciding whether one exists is NP-hard [22]. Moreover, one can argue that envy-freeness alone is not sufficient: even if an allocation is envy-free, it is possible that reallocating the tasks or resources can make *everyone* better off, in which case we say that the original allocation is not *Pareto efficient*. For example, consider a situation where one agent owns two left shoes, and another agent owns two right shoes. Neither agent envies the other's situation, but both agents can be made better off by trading a left shoe for a right shoe. Pareto efficiency is generally considered to be of paramount importance. There has been work characterizing the computational complexity of finding an allocation that is both envy-free and Pareto efficient [5].

In a context where every resource is initially owned by one of the agents, it makes sense to use an *exchange*—even if, for some reason, payments are not possible. I discuss an example of an exchange without payments, a *kidney exchange*, next.

1.3 Kidney exchanges

In most exchanges, the participants can make payments to each other, which facilitates trade. I will discuss such exchanges shortly. However, there are some exchanges in which no payments can be made, so that only items change hands. These are known as *barter exchanges*. An example is a *kidney exchange* [27].

Buying and selling kidneys is illegal in most countries; however, this is not the case for *swapping* kidneys. As an example, suppose a patient is in need of a kidney transplant, and there is a donor who is willing to give up her kidney for this particular patient, but unfortunately they are not compatible. There may be a second patient-donor pair in the same situation; moreover, it may be the case that the second patient is compatible with the first donor, and the first patient is compatible with the second donor. In this case, it is beneficial for the two patient-donor pairs to swap their donors' kidneys.

It is helpful to think of each patient-donor pair as a single agent, so that each agent has a kidney and needs a(nother) kidney. This makes it easier to see that more complex trades can be beneficial: agent 1's kidney can go to agent 2, agent 2's kidney to agent 3, and agent 3's kidney to agent 1—this is known as a cycle of length 3. Of course, we can also have cycles of length 4, *etc.*—but it is preferable to not have very long cycles (all the operations in a cycle have to be performed simultaneously so that nobody will back out, which poses a logistical problem for long cycles; also, if last-minute testing discovers an incompatibility in the cycle, the entire cycle collapses).

Kidney exchanges are now a reality, and computer scientists are involved in them [1]. Specifically, they have started working on the computational problem of clearing the exchange: the input describes which patients are compatible with which of the donors' kidneys, and the output specifies which cycles will be used. Using matching algorithms, the problem can be solved in polynomial time if there are no restrictions on how long cycles can be, or if only cycles of length two are allowed. However, if the maximum cycle length is three or more, then the problem is NP-hard. Nevertheless, in practice, large problems can be solved to optimality, using optimization techniques including column generation and branch-and-price search [1].

2. SETTINGS WITH PAYMENTS

We now move on to settings where agents can make or receive payments. Payments are useful because they allow us to quantify agents' preferences. Informally, agents now need to put their money where their mouths are. Payments also allow us to transfer happiness (*utility*) from one agent to another.

2.1 Auctions and exchanges

In many problems that require us to decide on an allocation of tasks or resources, it makes sense to also determine payments that some agents should make to other agents. Returning to our example of allocating chores, imagine that the inhabitants are roommates who each pay a share of the rent, and we end up assigning a disproportionate number of chores to one of the roommates. It seems fair that this roommate should pay a smaller share of the rent, which effectively represents a monetary transfer to this roommate from the others. This arrangement may well be to everyone's benefit, for example, if this roommate is unemployed and has plenty of time for completing chores but little money to spend on rent.

Once we start to consider payments in the allocation of tasks and resources, we are quickly drawn into *auction theory*. (A brief article on auctions and computer science recently appeared in the Communications of the ACM [35].)