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].)

Most people are familiar with the *English auction* format, where a single item (or a single lot of items) is for sale, and bidders call out increasing bids until nobody is willing to place a higher bid. There are many other auction formats, such as the *Dutch auction*, where the price is high initially and bidders stay silent as the price gradually decreases, until a bidder announces that she wants to purchase the item at that price, at which point the auction ends immediately.

Yet another format is the *sealed-bid* format, where bidders write down a bid on a piece of paper, place it in an envelope, and give it to the auctioneer; the auctioneer opens the envelopes and declares the highest bid the winner. Because at this point, we are mostly concerned with how to make a decision based on the agents' preferences, rather than with how these preferences are communicated, it will be easiest for us to think about the sealed-bid format for now.

If we are assigning a task rather than allocating a resource, we can use a *reverse* auction. Here, a bid of \$10 on a task indicates that the bidder wants to *be paid* \$10 for completing the task; in this context, the lowest bid wins.

Generally, in an auction, there is a seller who receives the payment from the winning bidder (or, in a reverse auction, a buyer who makes the payment to the winning bidder). A seller is not always present, however: for example, if the agents are trying to decide who gets the right to drive the car on a particular day, they can hold an auction for this right, but in this case it would be natural for the winning agent's payment to go to the losing agents. Some recent work has been devoted to designing mechanisms for redistributing the auction's revenue to the agents.

The key benefit of using an auction (or reverse auction) is that generally, the resource ends up with the agent who values it the most (or the task ends up with the agent who minds doing it the least); in this case, we say that the auction results in an *efficient* allocation. If an allocation is inefficient, then it is possible to make everyone better off by reallocating some of the tasks/resources, as well as some money. By this argument, efficiency and Pareto efficiency are the same concept in this context.

When there are multiple resources (or tasks) that need to be allocated, one straightforward way of doing this is to hold a separate auction for each resource. However, this approach has a significant downside, which is related to the following observation: how much one of the resources is worth to an agent generally depends on which other resources that agent receives.

For example, if Alice already has the right to drive the car on Friday, then probably having it on Thursday as well is not worth much to her, because she can already run her errands on Friday. In contrast, if she does not have the car on any other day, then having it on Thursday is probably very valuable to her. When having one resource makes having another worth less, then we say that the resources are *substitutes*. On the other hand, Alice may want to go on a two-day trip, in which case having the car on Thursday is worth nothing unless she also has it on Friday. When having one resource makes having another worth more, then we say that the resources are *complements*.

Substitutability and complementarity make it suboptimal to sell the resources in separate auctions, for the following reason. If the auction for the right to use the car on Thursday is run first, in some sense Alice does not know how much she values it, because she does not yet know whether she will win the auctions for the other days. This uncertainty can result in inefficient allocations.

Combinatorial auctions [12] provide a solution. In a (sealed-bid) combinatorial auction, a bidder's bid does not just indicate how much the bidder values each individual item; rather, the bidder expresses a value for every nonempty subset (bundle) of the items. For example, Alice's bid could say: "Having the car on Thursday is worth \$5 to me, having it on Friday is worth \$6, and having it on both Thursday and Friday is worth \$8." Given all this information (for all bidders), an algorithm can search through all possibilities for allocating the items to the bidders, and find the most efficient one—that is, the allocation that maximizes the sum of the agents' valuations.

Similarly, in a *combinatorial reverse auction*, each bidder expresses how much she wants to be compensated for every bundle of tasks that might be assigned to her. Yet another variant is a *combinatorial exchange*, in which agents can take the role of a seller as well as the role of a buyer, and they express combinatorial valuations for these more complex trades. These variants face many of the same issues as combinatorial auctions [12].

Once there are more than a few items in a combinatorial auction, the straightforward approach in which each bidder explicitly states how much every bundle of items is worth to her becomes completely impractical, since there are exponentially many bundles. Instead, we can let bidders use an expressive *bidding language* that allows them to express natural valuation functions concisely (similarly to the CP-nets that I mentioned in the context of combinatorial voting). A simple example is the XOR language, in which a bidder explicitly expresses valuations for some (but generally not all) bundles. For example, if the items for sale are $\{a, b, c\}$, a bidder could bid $(\{a\}, 5)$ XOR $(\{b, c\}, 10)$. This indicates that she values the bundle $\{a\}$ at 5; the bundle $\{b, c\}$ at 10; the bundle $\{a, b\}$ at 5, since it is not explicitly listed, but it contains the bundle $\{a\}$; and the bundle $\{a, b, c\}$ at 10, since the highest-value listed bundle that it contains is $\{b, c\}$ (the use of XOR, rather than OR, indicates that we cannot simply add up the values of the two listed bundles to get 15).

The choice of bidding language affects issues such as the computational complexity of the *winner determination* problem—that is, the problem of finding the efficient allocation of the items, given the bids. Even if each bidder only bids on a single bundle, the combinatorial auction winner determination problem is NP-hard [28] and inapproximable [29]. On the other hand, it is known that under certain conditions on the bids, the winner determination problem can be solved in polynomial time [23]. For example, if bidders bid only on bundles of at most two items, then the winner determination problem can be solved in polynomial time, via matching algorithms. In general, the runtime heavily depends on how the bids are generated: in some cases, it is possible to scale to hundreds of thousands of items and tens of thousands of bids, whereas in other cases, current techniques have trouble scaling beyond tens of items and hundreds of bids [30].

Instead of letting bidders bid only once—that is, requiring them to give all their valuation information at once—it is possible to use an *iterative* (or *preference elicitation*) format, in which bidders repeatedly respond to queries about their valuations [25, 32]. In a single-item setting, this corresponds to the distinction between a sealed-bid auction, in

which each bidder bids only once, and an English auction, in which the auctioneer repeatedly queries the bidders for higher valuations. Using preference elicitation in a combinatorial auction has the potential to greatly decrease the total amount of valuation information that the bidders need to communicate, while still finding the efficient allocation. This leads to the following inherently computational question: how should the procedure for querying the bidders be designed to minimize the required amount of communication?

Combinatorial auctions are more than a theoretical curiosity: they are used in practice in settings where the items display significant complementarities. Prominent examples include auctions for radio spectrum, as well as reverse auctions for strategic sourcing (in which large companies set up contracts with suppliers) [12, 31, 35].

In a context that is perhaps closer to home for most computer scientists, auctions are now also used by the leading search engines to allocate the advertising space on their search results pages. This is another example of an auction with multiple resources for sale: any search performed by a user results in multiple advertisement slots becoming available. These auctions are called *sponsored search auctions*, and they introduce a variety of new issues. For example, in a typical sponsored search auction, an advertiser pays only if the user clicks on her ad, rather than every time that her ad is displayed. The prominent place that sponsored search auctions occupy in the business models of the companies that use them has helped to bring about an explosion of research on them in recent years [19]; a thorough discussion would easily merit its own article.

While auctions and exchanges are the settings with payments that have attracted the most attention from computer scientists, there are numerous other, more specialized applications. Some of these are discussed below.

2.2 Charitable giving

Let us consider a person who is contemplating donating some money (say, \$100) to a charitable cause. It may seem that the potential donor should just evaluate what else she would do with the money, and whether that is worth more to her than to see the charity receive \$100. While this is a reasonable way to proceed, there are other options if there are multiple potential donors.

Suppose that there is a second donor that is making the same decision. Also, let us suppose that each donor concludes that she would slightly prefer spending \$100 on other things to seeing the charity receive \$100. Hence, using the straightforward decision procedure from above, neither donor will give any money. However, it may well be that, even with these preferences, each donor would prefer the outcome where *both* donors give. That is, each donor may prefer the outcome where the charity receives \$200, and she contributes only \$100 of this. This is because, other things being equal, they would like the other donor to give as much money to the charity as possible. (Unlike settings discussed earlier in this article, this is inherently a setting with a type of *externality*: a donor has preferences over what another donor does with her money.) The reason that with the straightforward decision procedure, neither donor gives to the charity, is that neither donor affects the other donor's decision.

In fact, there is a way in which a donor can affect another donor's decision. Suppose that one of the two donors can make a binding *matching offer*, committing to donating the same amount as the other donor. In this case, the other donor has a choice between giving \$100, resulting in a \$200 total contribution to the charity, and giving nothing, resulting in a \$0 total contribution to the charity. Given the preferences that we assumed, the donor will in fact give \$100, thereby forcing the other donor to give \$100 as well. It should be noted that both donors (as well as the charity) prefer this outcome to the outcome that results when they make their decisions separately (which is for both of them to give \$0).

In practice, a matching offer is generally made by a single large donor, offering to match donations by multiple smaller donors. As we just saw, simple matching offers can lead to improved results, but they are still restrictive. What can be done if multiple donors want to make their donations conditional on the others' donations? This type of expressiveness can lead to even better outcomes, but one has to be careful to avoid circularities. For example, consider the case where A will match B's contribution, and B will match A's contribution.

We proposed a system in which each donor can make her donation conditional on the total donated to the charity [10]. In fact, the framework allows for donations to be conditional on the total amounts donated to multiple charities. We also designed algorithms for determining the final outcome based on everyone's offers, which is NP-hard in general but tractable in special cases. We used this system to collect donations for the victims of the Indian Ocean Tsunami, and later for the victims of Hurricane Katrina. While the total amount collected from these events was small (about \$1000), the events gave some insight into how donors use the system. About 75% of the donors made their donations conditional on the total amount collected, suggesting that donors appreciated being able to do so. One interesting observation is that the effectiveness of the system (in terms of how much participants were willing to donate) apparently depended on *whose* donations donors were matching. The Tsunami event was conducted among the participants of a workshop, so that to some extent everyone knew everyone else; in contrast, the Hurricane event was open to anyone. The Tsunami event was more successful, perhaps because the participants knew whose donations they were matching. More recent systems also allow donors to condition on *who* is giving, taking social network structure into account [14]. I believe that this innovation has the potential to make such systems much more successful.

2.3 Prediction markets

The markets that I have considered so far generally produce a tangible outcome, such as an allocation of resources. The participating agents have different preferences over the possible outcomes, and the market is a mechanism for finding the "right" outcome for these preferences. The type of market that I discuss in this section is a little different.

A *prediction market* [37] concerns a particular future event whose outcome is currently uncertain. For example, the event could be an upcoming sports game, or an election. The agents trading in the prediction market generally cannot (significantly) influence the outcome of the event; the goal of the market is merely to *predict* the outcome of the event, based on the collective information and reasoning of the participating agents. Typically, the market prediction

is in the form of a probability: for example, the market's assessment may be that the probability that team A will beat team B is 43%. Prediction markets are quite popular on the Web: examples include the Iowa Electronic Markets as well as Intrade. Each of these runs prediction markets on a variety of events; it appears that the political events (for example, predicting the winner of an election) are the most popular.

A common way to run a prediction market is as follows. We create a *security* that pays out (say) \$1 if team A wins, and \$0 if team A does not win. We then let agents trade these securities. Eventually, this should result in a relatively stable market price: for example, the security may trade at about \$0.43. This can be interpreted to mean that the market (that is, the collection of agents) currently believes that the probability that team A will win is about 43%.

If an agent disagrees with this assessment, then she should buy or sell some of the securities. For example, if an agent believes that the probability is 46% (even after observing the current market price of \$0.43), then she can buy one of the securities at price \$0.43, and her expected payout for this security will be $46\% \cdot \$1 = \$0.46$. As she buys more securities, the market price will eventually go up to \$0.46.

If the agent believes that the probability is 40%, then she should sell some of the securities. If she currently does not own any of the securities, she can either short-sell, so that she effectively owns a negative number of the securities; or, she can buy securities for the complementary outcome(s): for example, if the match between A and B is guaranteed to have a winner, she can buy a security that pays out if B wins. The prices of these securities are related: if the match is guaranteed to have a winner, then the sum of the current prices of the security that pays out \$1 if A wins, and the security that pays out \$1 if B wins, must always be equal to \$1. If it were not, then there would be an opportunity for *arbitrage*: a combination of deals that leads to a risk-free profit. Specifically, if the sum of the current prices is (say) \$0.9, then one can buy both of the securities, and have a guaranteed profit of \$0.1, because one of them must pay out. If the sum is (say) \$1.1, then one can sell both securities, which again will result in a guaranteed profit of \$0.1.

One complication for standard prediction markets is that many real-world events have exponentially many possible outcomes. For example, consider a US presidential election. In a sense, every state (and the District of Columbia) has a separate outcome, so that even with two presidential candidates there are 2^{51} possible outcomes of the election.[2] Of course, we can have a separate market for each of the states, but this will still result in some missed opportunities.

For example, I may believe that with probability 80%, the Democratic candidate will win at least one of Florida, Ohio, and North Carolina. It is not immediately clear how this belief should translate into trading strategies for securities for the individual states. I would much rather simply buy a security that pays out exactly if the Democratic candidate wins at least one of Florida, Ohio, and North Carolina. Now, suppose there is another trader who believes that with probability 30%, the Republican candidate will win all of Florida, Ohio, North Carolina, and Missouri, and would like to buy a security that pays out precisely under these conditions. Ideally, the prediction market could automatically create both of these securities, charge me (say) \$0.79 for mine, and charge the other trader (say) \$0.29 for hers. Both of us will accept these deals; moreover, since at most one of our two securities will pay out, the prediction market is guaranteed a risk-free profit of at least \$0.08. Such *combinatorial prediction markets* have recently started to receive attention [6]. Running such markets requires solving computationally hard problems: for example, determining whether there is a risk-free combination of securities that can be created is generally NP-hard.

[2]Actually, this is slightly inaccurate: the states of Maine and Nebraska do not use a winner-takes-all system, further increasing the number of possible outcomes.

3. STRATEGIC BEHAVIOR: GAME THEORY AND MECHANISM DESIGN

So far, I have focused on allowing agents to communicate their preferences (or, in the case of prediction markets, their beliefs), ideally in an expressive and natural way, and on making good decisions based on what was communicated. I have ignored one key aspect, though: are the agents incentivized to communicate their preferences and beliefs truthfully, or can they benefit from misreporting them?

For example, in an election, an agent's true preferences may be $a \succ b \succ c$. However, if the agent realizes that a has no chance of winning, she may instead choose to vote $b \succ a \succ c$, so as to at least maximize the chances of b winning. Similarly, in an auction, an agent who values the item for sale at \$10 may instead bid only \$5, in the hope of paying less. While such strategic behavior may be beneficial for the agent who engages in it, it generally makes the quality of the overall outcome worse, because now it is chosen based on input that does not reflect the true preferences.

These considerations lead us into *mechanism design*. Informally stated, the goal of mechanism design is to design rules for choosing the outcome that lead to good results even in settings where agents are strategic—that is, an agent will lie about her preferences if this is in her best interest. Mechanism design has been studied primarily (until recently, almost exclusively) in economics.[3] Evaluating the quality of a mechanism is nontrivial: it requires being able to predict how multiple strategic agents will act in each other's presence. *Game theory* provides tools for making such predictions.[4] A nice article about computer science and game theory recently appeared in the Communications of the ACM [34].

The standard approach to mechanism design is simply to ensure that it is never beneficial for an agent to lie about her preferences. A result known as the *revelation principle* suggests that this approach is, from the point of view of strategic behavior, without loss of generality. A mechanism under which it is never beneficial to lie is called *truthful*. Unfortunately, it turns out that in general voting settings, no good truthful mechanisms exist, by a result known as the Gibbard-Satterthwaite impossibility theorem [15, 33].

For settings such as auctions and exchanges, where payments can be made, there are much more positive results. For one, if our goal is to allocate the resources efficiently,

[3]In 2007, Hurwicz, Maskin, and Myerson received the Nobel Prize in Economics for their fundamental work on mechanism design.

[4]Game theory has led to two other Nobel Prizes in Economics: Nash, Selten, and Harsanyi received one in 1994, and Aumann and Schelling in 2005.

there are rules for specifying how much agents should pay that make the mechanism as a whole truthful.

A simple example of such a payment rule is the *second-price sealed-bid* auction for a single item. In this auction, the bidder with the highest bid wins, but only pays the second-highest bid. As a result, the winning bidder's bid does not affect the price she pays; so the only effect that misreporting can possibly have is that she does not win, which would make her worse off. Similarly, the only effect that misreporting can possibly have for a losing bidder is that she ends up winning at a price that is too high for her, which would make her worse off. So, a bidder is always best off reporting her true valuation for the item—that is, the second-price sealed-bid auction is truthful. This scheme can be generalized to combinatorial auctions and exchanges (and other settings), resulting in the class of *Vickrey-Clarke-Groves (VCG) mechanisms* [36, 7, 16].

The issues studied in mechanism design interact with the computational issues that I discussed before in subtle ways. For example, suppose that we want to run a combinatorial auction using a VCG mechanism. Technically, this means that we should always solve the winner determination problem to optimality, that is, find the most efficient allocation—which we know is NP-hard. If we do not always succeed at finding the most efficient allocation, then the resulting mechanism will, in general, not be truthful. A significant amount of research has addressed the problem of designing polynomial-time approximation algorithms that, in combination with the right payment rule, are truthful [21]. More generally, the problem of designing efficient algorithms that can be made truthful is the main topic of *algorithmic mechanism design* [24]. This line of research has also been extended to distributed settings without a trusted center [13].

We can use computers not only to run existing mechanisms, but also to *design* new mechanisms from scratch. That is, for a given setting, we let an algorithm search through the space of all possible truthful mechanisms for an optimal one [9]. This approach is called *automated mechanism design*. Finding an optimal mechanism is computationally much harder than running an existing mechanism, and as a result automated mechanism design has so far been successful only on small instances. Nevertheless, some real instances are in fact small, and even for larger instances, solving a simplified version can give some helpful intuition. Automated mechanism design can also be used to solve some small instances of a general mechanism design problem; then, a human mechanism designer can try to identify a pattern in these small solutions, conjecture the general solution, and prove it analytically. In this way, automated mechanism design can contribute to microeconomic theory. This methodology has recently been used to design mechanisms for redistributing an auction's revenue to the bidders in a truthful way (for example, [17]), and the methodology is starting to be adopted more widely.

It is not always the mechanism designer or the party running the mechanism that faces hard computational problems. Under some mechanisms, it is computationally hard for the *agents* to find the strategically optimal action to take. This is not the case for truthful mechanisms, where strategically optimal behavior simply means telling the truth. However, no reasonable voting rule is truthful in sufficiently general settings (by the Gibbard-Satterthwaite theorem mentioned above). It has been shown that in a variety of voting settings, it is NP-hard to find the strategically optimal vote(s) to cast, even if the other agents' votes are already known (for example, [2, 18, 11]). This is a case where computational hardness can be desirable: it can be argued that if a voter cannot find a way of misreporting her preferences that benefits her, then she will presumably tell the truth. For now, the impact of this type of result is limited by the fact that NP-hardness is a worst-case measure, and it may well be the case that it is easy to find an effective way of misreporting one's preferences *most of the time*.

Another important issue is that the mechanisms from traditional mechanism design mainly guard against a single type of manipulation: misreporting one's preferences. However, mechanisms that are run in highly anonymous environments such as the Internet are vulnerable to other types of manipulation. Specifically, it is often possible for a single agent to pretend to be multiple agents (known as *false-name manipulation* or a *Sybil attack*). The standard mechanisms for guarding against misreporting, such as the VCG mechanisms, are generally not robust to false-name manipulation. A mechanism that is robust to it—that is, under which no agent ever benefits from using multiple identifiers—is said to be *false-name-proof* [39], and a growing body of research attempts to design such mechanisms.

A final direction in mechanism design concerns extending its techniques to dynamic environments, where decisions have to be made over time as additional information enters the system. Recent years have seen rapid progress in generalizing mechanism design techniques from static to dynamic settings [26]. For example, sponsored search auctions are, in principle, a good application domain for such techniques: the demand for, as well as the supply of, advertisement slots next to the results for specific searches changes over time, but allocation decisions must be made now.

4. CONCLUSIONS

In this article, I have considered a number of settings in which a decision needs to be made based on the preferences of multiple agents, as well as mechanisms for reaching the decision. People have been using such mechanisms for millennia, and have studied them formally for centuries (although their game-theoretic analysis has taken place mostly in the last fifty years). Still, computer scientists are fundamentally changing these mechanisms and how they are being used.

Increased computing power and better algorithms enable the use of mechanisms, such as the Kemeny voting rule and combinatorial auctions, that used to be considered impractical. Also, the Internet provides a great platform for these mechanisms: it makes it easy for spatially distributed users to communicate their preferences to the mechanism, and they will generally be forced to communicate them in a precise way (for example, a bidder will have to enter a number on a website rather than vaguely communicating her preferences over the phone), which makes it possible to run the mechanism automatically. I (speculatively) imagine that in the future, more Web-based mechanisms will be oriented around social networking sites such as Facebook and MySpace; the charitable donations work [14] is a good example of how such social network structure can be used. Computer scientists are also encountering mechanism design problems in their own work, for example, when shared computing resources need to be allocated to users. Finally, the paradigms of computer science give a different and useful perspective

on some classic problems in economics.

This article has summarized a number of applications where computer scientists have already become involved in the design of markets and other protocols for making decisions based on the preferences of multiple agents. I anticipate that the number and importance of such applications will grow steeply in the years to come. One major reason for this is that computer scientists and economists interested in market design have grown closer together in recent years, and are now seen working together more often (this is necessitated by high-value applications such as sponsored search auctions). Computer scientists have caught up on many of the key techniques developed in the microeconomics literature. On the other side, economists are becoming increasingly familiar with techniques from modern computer science. This is a very nice example where "computational thinking" is being exported to another discipline (which is certainly not to say that there were no prior instances of economists thinking computationally).

Acknowledgments

This work is supported by NSF award number IIS-0812113, a Research Fellowship from the Alfred P. Sloan Foundation, and a Yahoo! Faculty Research Grant. I thank the CACM reviewers for very detailed and helpful feedback. I also thank Tuomas Sandholm for feedback on the kidney exchange section, and David Pennock for feedback on the prediction markets section. All errors and omissions are my own (though of course I faced constraints on length and number of citations).

5. REFERENCES

[1] D. Abraham, A. Blum, and T. Sandholm. Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 295–304, San Diego, CA, USA, 2007.

[2] J. Bartholdi, III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

[3] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.

[4] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: a tool for representing and reasoning with conditional ceteris paribus statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[5] S. Bouveret and J. Lang. Efficiency and envy-freeness in fair division of indivisible goods: logical representation and complexity. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 935–940, Edinburgh, Scotland, UK, 2005.

[6] Y. Chen, L. Fortnow, E. Nikolova, and D. M. Pennock. Combinatorial betting. *ACM SIGecom Exchanges*, 7(1):61–64, 2007.

[7] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[8] V. Conitzer, A. Davenport, and J. Kalagnanam. Improved bounds for computing Kemeny rankings. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 620–626, Boston, MA, USA, 2006.

[9] V. Conitzer and T. Sandholm. Complexity of mechanism design. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 103–110, Edmonton, Canada, 2002.

[10] V. Conitzer and T. Sandholm. Expressive negotiation over donations to charities. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 51–60, New York, NY, USA, 2004.

[11] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3):Article 14, 1–33, 2007.

[12] P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial Auctions*. MIT Press, 2006.

[13] J. Feigenbaum, M. Schapira, and S. Shenker. Distributed algorithmic mechanism design. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 14. Cambridge University Press, 2007.

[14] A. Ghosh and M. Mahdian. Charity auctions on social networks. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1019–1028, 2008.

[15] A. Gibbard. Manipulation of voting schemes: a general result. *Econometrica*, 41:587–602, 1973.

[16] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[17] M. Guo and V. Conitzer. Worst-case optimal redistribution of VCG payments in multi-unit auctions. *Games and Economic Behavior*, 2009. To appear.

[18] E. Hemaspaandra and L. A. Hemaspaandra. Dichotomy for voting systems. *J. Comput. Syst. Sci.*, 73(1):73–83, 2007.

[19] S. Lahaie, D. M. Pennock, A. Saberi, and R. V. Vohra. Sponsored search auctions. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 28. Cambridge University Press, 2007.

[20] J. Lang. Vote and aggregation in combinatorial domains with structured preferences. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1366–1371, Hyderabad, India, 2007.

[21] D. Lehmann, L. I. O'Callaghan, and Y. Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):577–602, 2002.

[22] R. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 125–131, New York, NY, USA, 2004.

[23] R. Müller. Tractable cases of the winner determination problem. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, chapter 13, pages 319–336. MIT Press, 2006.

[24] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196,

2001.

[25] D. Parkes. Iterative combinatorial auctions. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, chapter 2, pages 41–77. MIT Press, 2006.

[26] D. Parkes. Online mechanisms. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 16. Cambridge University Press, 2007.

[27] A. E. Roth, T. Sonmez, and M. U. Unver. Kidney exchange. *Quarterly Journal of Economics*, 119(2):457–488, 2004.

[28] M. Rothkopf, A. Pekeč, and R. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[29] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, Jan. 2002.

[30] T. Sandholm. Optimal winner determination algorithms. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, chapter 14, pages 337–368. MIT Press, 2006.

[31] T. Sandholm. Expressive commerce and its application to sourcing: How we conducted $35 billion of generalized combinatorial auctions. *AI Magazine*, 28(3):45–58, 2007.

[32] T. Sandholm and C. Boutilier. Preference elicitation in combinatorial auctions. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, chapter 10, pages 233–263. MIT Press, 2006.

[33] M. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.

[34] Y. Shoham. Computer science and game theory. *Communications of the ACM*, 51(8):74–79, 2008.

[35] H. R. Varian. Designing the perfect auction. *Communications of the ACM*, 51(8):9–11, 2008.

[36] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[37] J. Wolfers and E. Zitzewitz. Prediction Markets. *The Journal of Economic Perspectives*, 18(2):107–126, 2004.

[38] L. Xia, V. Conitzer, and J. Lang. Voting on multiattribute domains with cyclic preferential dependencies. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 202–207, Chicago, IL, USA, 2008.

[39] M. Yokoo, Y. Sakurai, and S. Matsubara. The effect of false-name bids in combinatorial auctions: New fraud in Internet auctions. *Games and Economic Behavior*, 46(1):174–188, 2004.

[40] H. P. Young. Optimal voting rules. *Journal of Economic Perspectives*, 9(1):51–64, 1995.