Chapter 1

Introduction

There are many important settings in which multiple parties (or *agents*) must jointly make a decision—to choose one *outcome* from a space of many possible outcomes—based on the individuals' preferences (and potentially other privately held information). For example, individual persons may need to decide how to allocate various tasks and resources among themselves; firms may need to decide on how to structure the supply chain to efficiently bring a product to the consumer; the governments of nations may need to decide on what form, if any, an international treaty will take; *etc.* Such *preference aggregation settings* are pervasive in human (and perhaps even animal) life, and have long been studied, especially in economics and political science. In recent years, computer science has joined the set of fields with significant interest in preference aggregation. As computer systems become increasingly interconnected, more and more problems come to the forefront that are fundamentally about selecting good outcomes in the face of conflicting preferences. Examples include scheduling multiple users' jobs; routing network traffic for multiple users; choosing which companies' advertisements to display on a webpage; ranking the webpages of multiple authors in response to a search query; *etc.* In addition, software agents have the potential to aid humans in, or potentially even take over, some preference aggregation tasks, such as trading items over the Internet. The interest of computer scientists in preference aggregation is driven in part by such applications; however, another driver is the fact that computationally nontrivial problems must be solved in almost all complex preference aggregation settings, so that computer scientists can contribute even to settings that are not otherwise related to computer science.

To aggregate their preferences effectively, the agents need to use some *protocol* that will elicit the agents' preferences (and other pertinent information) from them, and select the outcome. There are two great challenges in designing a good protocol. First, a good protocol should, when provided with accurate information, arrive at an outcome that is considered good for the agents. Finding such a good outcome may require significant communication and computation, especially when the outcome and preference spaces are combinatorial in nature. Second, when the agents are self-interested, they will misreport their information to the protocol (also known as *manipulating*) when it is beneficial for them to do so. Thus, a good protocol should take this strategic behavior into account, and select outcomes in such a way that a good outcome is reached even though the agents are strategic (for instance, by making sure the agents have no incentive to misreport). The theory of *mechanism design* studies how to make protocols strategy-proof in this sense.

Computation plays a significant role in both the execution and the design of good protocols. This role is clearly apparent in some cases—for instance, selecting a good outcome, even with all the agents' true information in hand, may require the solution of a computationally hard optimization problem. However, computation can also have more subtle roles in the design of protocols—for instance, when anticipating the extent of an agent's strategic manipulation of the mechanism, it can be helpful to have a good assessment of the agent's computational limits in manipulating.

1.1 A hierarchy of uses for computation in preference aggregation

I propose the following hierarchy (Figure 1.1) for categorizing various needs for computational tools in preference aggregation settings. Computational tools may improve the aggregation of the agents' preferences over outcomes through:

(1) The straightforward **optimization** of the outcome (choosing the outcome that maximizes the sum of the agents' utilities, also known as the *social welfare*, or some other objective), given all the agents' preferences and other information, in potentially complex allocation or other outcome selection tasks—without consideration of strategic behavior.

(2) Enabling the use of a **mechanism** for the selection of the outcome, where the mechanism is designed so that a good outcome will be chosen in spite of strategic behavior by the agents.

(3a) Rather than enabling a known general mechanism, computing a *custom* mechanism on the fly for the setting at hand (**automated mechanism design**).

(3b) Making use of the agents' computational limits, or **bounded rationality**, in the mechanism design process to reach better outcomes.

(4) Generalizing all of the previous tools to enable **automated mechanism design for bounded agents**.

Section 1.2 introduces the nodes of the hierarchy using an example application setting. Section 1.3 discusses the meaning and use of the hierarchy itself (that is, the relationships represented by the directed edges). Section 1.4 discusses research directions in preference aggregation that are orthogonal to this hierarchy. Finally, Section 1.5 gives an outline of the remainder of the dissertation.

1.2 The hierarchy's nodes illustrated by example

In this section, I will introduce the nodes of the hierarchy using the example of a *combinatorial auction*. Combinatorial auctions have recently become a popular research topic (*e.g.* [Rothkopf *et al.*, 1998; Sandholm, 2002a; Nisan, 2000; Sandholm *et al.*, 2005c; Gonen and Lehmann, 2000; Nisan and Ronen, 2000; Lehmann *et al.*, 2002; Bartal *et al.*, 2003; Lavi *et al.*, 2003; Yokoo, 2003; Parkes,

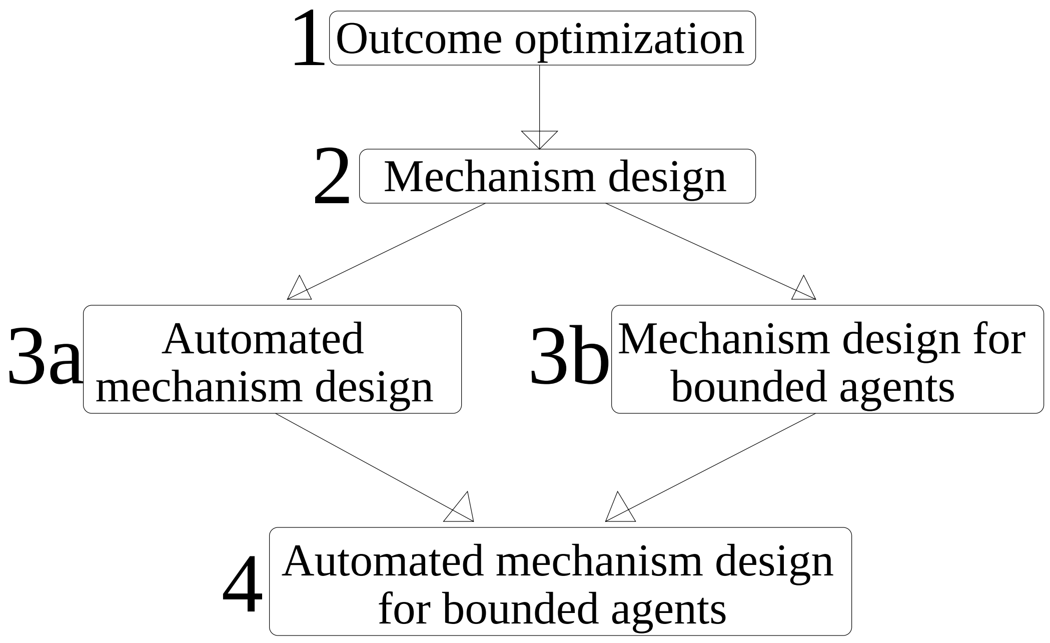
1 Outcome optimization
2 Mechanism design
3a Automated mechanism design
3b Mechanism design for bounded agents
4 Automated mechanism design for bounded agents

Figure 1.1: The hierarchy of uses for computational tools in preference aggregation.

1999a; Wurman and Wellman, 2000; Ausubel and Milgrom, 2002]). They are arguably the canonical example of a preference aggregation domain that is expressive enough to introduce a variety of nontrivial computational issues, and they have increasing practical importance. (In fact, some of the results to be presented in this dissertation strictly concern combinatorial auctions.) Nevertheless, these issues occur in many other important preference aggregation domains, and combinatorial auctions here only serve as an (important) illustrative example.

In a combinatorial auction, a seller has multiple items for sale, and bidders can bid on *bundles* (subsets) of items. For example, suppose Sally has an apple and an orange for sale. Al would enjoy eating the apple and places a bid of \$2 on the apple (more precisely, the bundle consisting of the apple alone). The orange by itself is worth nothing to Al. However, he would prefer having both the apple and the orange, to make a fruit salad; and so, Al bids \$3 on the bundle of the apple and the orange together. Meanwhile, Bill is only interested in the orange, and bids \$2 on the orange. Thus the bids can be represented as follows: $v_A(\{\text{apple}\}) = 2$, $v_A(\{\text{apple, orange}\}) = 3$, $v_B((\{\text{orange}\}) = 2$. (For the purposes of this example, I will assume that all bids are submitted simultaneously—a so-called *sealed-bid* auction. This contrasts with, for instance, the more commonly known *English auction* for a single item in which bidders can continue to improve their bids.) We are now ready to start introducing the nodes of the hierarchy using this example.

1.2.1 Node (1): Outcome optimization

In our example, who should win which items? Giving Al both items will generate a value of \$3, but giving the Al the apple and Bill the orange will generate a value of \$2 + \$2 = \$4. So, given the information that we have, the latter is the most sensible outcome of the auction. This problem of determining which of the bids win is known as the *winner determination* or *clearing* problem. In spite of its apparent simplicity, once the numbers of items and bids become large, this problem becomes computationally hard: it is NP-complete [Rothkopf *et al.*, 1998], even to approximate [Sandholm, 2002a].

The clearing problem is a good example of a problem at the shallowest node **(1)** in the hierarchy: the straightforward **optimization** of the outcome, without consideration of strategic behavior. At this node, we assume that we have a full characterization of each agent's preferences over the outcomes (in our example, this characterization is given by the bids), and we take this characterization at face value. That is, we do not worry about whether the estimates of the agents' preferences that we have are inaccurate (for instance, perhaps the agents reported their preferences to us directly, and they may have misrepresented them to effect a better outcome for themselves). This approach is appropriate when, for instance, we obtained our estimates of the agents' preferences from a disinterested source (not the agents themselves). The optimization algorithm may also be used as a subroutine in a mechanism that does take strategic behavior into consideration—we will discuss this in more detail later.

1.2.2 Node (2): Mechanism design

The ability to solve the clearing problem well is necessary for reaching good outcomes in combinatorial auctions, but, as we will see in this subsection, it is not sufficient. So far, we have not yet discussed what the bidders in the auction should pay. Let us suppose that the auction is a *first-price* auction, in which bidders simply pay the values of their winning bids. In first-price auctions, bidders typically do not bid their true valuations, because if they did, they would gain nothing from having their bids accepted. Rather, bidders will bid lower than their true valuations, to have a chance of making some profit in the auction. To decide how much lower they will bid, they will take into account their perceived chances of winning for each amount that they may bid. In our example, perhaps Al's true marginal value for having the orange (in addition to the apple) was much higher than \$3 - \$2 = \$1; perhaps it was \$10, but Al (in retrospect, incorrectly) thought it was extremely unlikely that Bill would place a bid higher than \$1. Meanwhile, perhaps Bill's true value for the orange was \$9, but Bill (in retrospect, correctly) thought that Al would think that bidding \$3 on the bundle of the apple and orange would in all likelihood be enough to win that bundle, and therefore Bill chose to bid only \$2. Now, even though we solved the clearing problem optimally with respect to the bids provided, a number of things still went wrong. First, from the perspective of economic efficiency, the orange *should* have ended up with Al rather than Bill, because Al's marginal value \$10 for it is greater than Bill's marginal value \$9 for it. Second, Sally received a total revenue of only \$4, whereas it would seem that there was enough competition on the orange alone for her to receive at least \$9.

These problems are the result of the auction format (a sealed-bid first-price auction). For example, an *ascending-price* auction in which bidders can continue to raise their bids may have been more successful here. (Various research has been devoted to the implementation of ascending-price or iterative auctions and exchanges in multi-item settings [Parkes, 1999a; Wurman and Wellman, 2000; Ausubel and Milgrom, 2002; Parkes *et al.*, 2005]. Such protocols are themselves special cases of a more general framework in which the auctioneer is enhanced by *elicitor software* that incrementally and adaptively elicits the bidders' preferences using specific queries until enough information has been elicited to determine the final allocation and payments[Conen and Sandholm, 2001].) An ascending-price auction, however, can run into a variety of other problems in other settings. Yet another approach may be to apply a type of *second-price* auction. In a second-price (or *Vickrey*) auction for one item, the winner pays the value of the second highest bid. (The generalization of the second-price auction to multiple items is known as the *VCG mechanism* [Vickrey, 1961; Clarke, 1971; Groves, 1973].) (Generalizations of) second-price auctions have the nice property that bidders are motivated to bid their true value for an item.

Different formats for auctions (or other preference aggregation domains) are known as *mechanisms*, and the process of constructing them is known as *mechanism design*. The mechanism specifies—for each combination of preferences (or *types*) that may be reported—what the outcome should be, as well as additional variables, such as payments to be made. Over the past several decades, economists have put together a limited library of mechanisms that motivate agents to report their preferences (*i.e.*, bid) truthfully, pursue some objective under this constraint (for instance, social welfare), and that can be applied in settings with varying levels of generality. For example, mechanisms have been discovered that, under some assumptions, will motivate agents to report truthfully in *any* preference aggregation setting, while still always choosing the socially optimal outcome. These are the VCG mechanism mentioned previously, and the dAGVA [d'Aspremont and Gérard-Varet, 1979; Arrow, 1979] mechanism.

Even when these general mechanisms are known, applying them to complex combinatorial settings (such as combinatorial auctions) is typically highly nontrivial, sometimes requiring the solution to multiple winner determination problems. For example, to apply the VCG mechanism to a setting, one typically requires the solution to (#bidders+1) outcome optimization problem instances: the original one, as well as each one in which one of the bidders is omitted from the instance. Moreover, using approximation approaches that are effective for the outcome optimization problem *per se* to execute the mechanism can destroy the strategic properties of the mechanism: agents will no longer be motivated to tell the truth [Nisan and Ronen, 2001; Sandholm, 2002b]! Thus, the agents will report their preferences strategically rather than truthfully. Moreover, there is no guarantee that the resulting strategic equilibrium will produce an outcome that is anywhere close to the socially optimal one. The resulting challenge is to design special approximation algorithms that do incent the agents to report truthfully. Or, put differently, the challenge is to design special truthful mechanisms whose outcomes are at least reasonably good, and can be computed in polynomial time. This is known as *algorithmic mechanism design* [Nisan and Ronen, 2001].

Node **(2)** in the hierarchy is concerned with the issues described in this subsection: designing mechanisms that encourage agents to tell the truth in (expressive) preference aggregation settings, as well as designing efficient algorithms for computing the outcomes of these mechanisms, thereby enabling their use.

1.2.3 Node (3a): Automated mechanism design

Depending on her goals, Sally will prefer some mechanisms to others. Which mechanism is the optimal one? To make this more concrete, suppose Sally assesses the situation as follows:

> *I want to maximize my expected revenue, but I want to cap my revenue at 6. As a secondary objective, I want to maximize Al and Bill's social welfare (combined utility), but I do not want Al's utility for the outcome to exceed Bill's by more than 3. I think Bill's utility for the orange alone is probably (80%) 9, but maybe (20%) it is 1. As for Al's utilities, ...*

Et cetera. When Sally looks into the library of general mechanisms, she will make some troubling discoveries. First of all, nobody has ever studied what to do under exactly her idiosyncratic goals (presumably because her goals lack simplicity and elegance). She will likely discover some relevant facts, though:

1. Many people have actually studied the problem of how to maximize expected revenue when selling two items, but there is still no general characterization of how to do this [Avery and Hendershott, 2000; Armstrong, 2000; Vohra, 2001].

2. When two parties are negotiating over an item, it is in general not possible to design a mechanism such that the item always ends up with the party who likes it best without money flowing in or out of the system consisting of the two parties [Myerson and Satterthwaite, 1983]. This is relevant to Sally in cases where she is clearly going to make her revenue cap, from which point on money can only flow between Al and Bill. (Alternatively, money can flow outside of the system consisting of Al, Bill, and Sally, which is not desirable.)

It may seem at this point that Sally is in over her head and should simply choose one of the known mechanisms that she thinks comes at least somewhat close to what she is looking for. But, while her situation may seem overwhelming, Sally does have a leg up on the economists studying these problems. Rather than being concerned with a general characterization for all settings in the domain of combinatorial auctions, Sally is only concerned with her own setting, which involves an apple, an orange, two buyers Al and Bill, Sally's objective, and specific prior probabilities for the buyers' preferences. If she finds the mechanism that is the best for her particular setting, she will not care whether or not (or to what extent) it generalizes to all possible settings. Still, Sally, who wants to get back to growing apples and oranges, may not be patient or qualified enough to solve even this restricted problem by hand. To address this, we introduced the **automated mechanism design** approach [Conitzer and Sandholm, 2002b], in which we solve the mechanism design problem for a given setting by computer, as an optimization problem. This is the topic of node **(3a)**: computing a *custom* mechanism on the fly for the setting at hand.

1.2.4 Node (3b): Mechanism design for bounded agents

It is the nature of mechanism design to assume that agents are strategic in how they interact with the mechanism. However, assuming fully strategic agents implies assuming perfectly rational agents. As a result, traditional mechanism design may often be too conservative in the extent to which it expects the agents to manipulate. To make this concrete, consider a mechanism designer that is considering Sally's mechanism design problem (and suppose that the setting is a little more complex, involving a larger variety of fruits). The designer's analysis may proceed as follows.

> *... In the case where Al bids $3 on the apple, $4 on the apple and the orange, $1 on the banana, ... And Bill bids... Then, it would be very good for the objective I am pursuing if I can give Al the apple only, and charge him $2.50. Unfortunately, if I did this, Al would have an incentive to misreport his preferences as follows: bid $3.50 on the apple, $6 on the banana and the grapefruit, $3 on the tangerine... – because for this bid we already decided on an outcome that would be better for Al. So, I have to do something else...*

(Of course, most of this analysis would not be done explicitly, but rather implicitly in the mathematical analysis—although the optimization-based approach of automated mechanism design would be closer to this in terms of how explicit the analysis is.) This analysis would be overly conservative if Al was too computationally limited to ever discover the beneficial misreport of his preferences "$3.50 on the apple, $6 on the banana and the grapefruit, $3 on the tangerine...", and thus it may leave money (more precisely, objective value) on the table. Ideally, mechanism design should take into account such computational limitations. This is the topic of node **(3b)**: making use of the agents' **bounded rationality** in the mechanism design process to reach better outcomes.

The way this plays out technically is as follows. The classical theory of mechanism design tells us that there is no benefit to using mechanisms in which the agents have incentives to make insincere revelations (that is, *non-truthful mechanisms*, as opposed to truthful mechanisms): if the agents behave strategically, then for every non-truthful mechanism, there is another truthful mechanism that performs just as well. This result is known as the *revelation principle* [Gibbard, 1973; Green and Laffont, 1977; Myerson, 1979, 1981], and the basic idea behind its proof is remarkably simple. Suppose we envelop a non-truthful mechanism with an *interface layer*, to which agents input their preferences. Then, the interface layer interacts with the original mechanism on behalf of each agent, *playing strategically in the agent's best interest* based on the reported preferences. (Compare, for example, proxy agents on eBay [eBay UK, 2004].) The resulting mechanism is truthful: an agent has no incentive to misreport to the interface layer, because the layer will play the agent's part in the original mechanism in the agent's best interest. Moreover, the final outcome of the new, truthful mechanism will be the same, because the layer will play strategically optimally—just as the agent would have. However, in complex settings, the last step in this argument may be incorrect: it is possible that the agent, due to computational limitations (bounded rationality), would not have been able to play optimally in the original, non-truthful mechanism. If this is so, then the outcome for the non-truthful mechanism would have been different—perhaps better. Thus, mechanism design for computationally bounded agents should not focus strictly on truthful mechanisms, but rather try to construct better, non-truthful mechanisms based on models of the computational boundedness of the agents. The advantage of doing so is that strictly better outcomes may be obtained than by any truthful mechanism, because (only) by using a non-truthful mechanism, we can exploit the agents' computational inability to always act in a strategically optimal way.

1.2.5 Node (4): Automated mechanism design for bounded agents

Finally, ideally, Sally would like to maintain the benefits of automated mechanism design while also making use of the agents' bounded rationality. This is certainly nontrivial, because the settings in which the agents' bounded rationality comes into play tend to be the same as those that are too complex for current automated mechanism design techniques to handle. Specifically, bounded rationality becomes relevant mostly in settings where agents can have exponentially many possible preferences (types), and automated mechanism design usually does not scale to very large numbers of types. Nevertheless, towards the end of this dissertation, we will present one approach to automated mechanism design for bounded agents.

1.3 How to use the hierarchy

Whereas the previous section introduced the individual nodes, in this section, I will focus on the meaning and intended use of the hierarchy itself.

1.3.1 Interpretation

I will first discuss how the hierarchy (that is, the relationship between the nodes represented by the directed edges in Figure 1.1) should be interpreted. Each node inherits the computational complexities typically associated with its ancestor nodes. For instance, a nontrivial mechanism may need to be created for a complex allocation task whose outcome optimization problem is already hard. The computational issues that each node introduces thus become closely intertwined with those of nodes at shallower levels. Throughout this dissertation, for clarity, I will try as much as possible to separate out the computational issues specific to the node under consideration from those of the shallower levels, but the reader should bear in mind that some of the most valuable future research will be done in settings where computational issues from multiple levels are nontrivially present simultaneously.

A related point is that in order for the hierarchy to make sense, the underlying setting (for instance, a combinatorial auction) should be assumed *fixed*—rather than assuming that the setting has a level of complexity that is of "appropriate" difficulty to the node in the hierarchy at hand. This precludes misleading arguments such as the following: "Outcome optimization is only interesting in complex combinatorial settings. Automated mechanism design, on the other hand, is nontrivial even in small, flatly represented settings, and should thus be studied there first. Therefore, automated mechanism design does not necessarily address issues associated with outcome optimization." If the setting is fixed beforehand, then either outcome optimization is trivial, and automated mechanism design may be manageable; or outcome optimization is nontrivial, and automated mechanism design is likely to be difficult for current techniques. Nevertheless, in either case, automated mechanism design must deal with the optimization issues presented by the setting.

1.3.2 Complexity of node vs. complexity of setting

Many (but not all) of the computational issues to be discussed in this thesis are in part the result of the complexity of the setting. For example, single-item auctions do not introduce many computational questions because of the simplicity of the setting, whereas structured auctions are a source of numerous difficult computational questions. The complexity of the setting is closely tied to its *representation*. Simple settings can be flatly represented (that is, all possible outcomes and preferences can be explicitly listed), whereas more complex settings require a *structured* representation. For example, in a large combinatorial auction, it is not feasible to simply list all possible allocations and read through them one by one to find the best one. Rather, we have a combinatorial description of the space of possible allocations, and require sophisticated algorithms to search through this space.

The following table shows how the complexity of each node in the hierarchy depends on whether the setting is flat (that is, all possible outcomes and preferences can be explicitly listed) or structured (that is, all possible outcomes and preferences cannot be explicitly listed, but there is a way of representing all of them in a more concise way, as in a combinatorial auction). The entries in the table without special emphasis do not introduce any (additional) questions related to computation. The entries in *italics* do introduce such questions, and the dissertation will address them in detail. The entries in **bold** are mostly beyond the scope of this dissertation and represent important agendas for future research.

Node	**Flat Settings**	**Structured Settings**
(1) Outcome optimization	trivial	*sometimes computationally hard*
(2) Mechanism design	standard setting for classical mechanism design	*need to balance computation, optimality, and incentives*
(3a) Automated mechanism design	*sometimes computationally hard*	*some approaches for special classes* **no known (computationally efficient) general approaches**
(3b) Mechanism design for bounded agents	bounded rationality is irrelevant	*design mechanisms to beneficially place computational burden on agents* **no known general theory**
(4) Automated mechanism design for bounded agents	coincides with automated mechanism design	*incrementally make mechanisms more strategy-proof* **no known general theory**

The complexity of each node vs. the complexity of the setting.

The fact that these issues are harder in structured settings than in flat settings should, of course, not be interpreted to imply that it is a bad idea to find good concise representations of settings of interest by using their inherent structure. The issues described in the table are harder for structured settings than for flat settings *of the same description length*. However, a good structured representation can significantly reduce the problem's description length (and in many cases writing the problem down is not even feasible without some structured representation). Moreover, the special structure of a setting can often be used to help solve the problem.

1.3.3 Are the shallow nodes outdated?

I should emphasize that it is *not* my opinion that research strictly focused on the shallower levels of the hierarchy (such as solving complex outcome optimization problems without taking questions of mechanism design into account) is outdated or pointless, for the following three reasons.

First, sometimes issues deeper down in the hierarchy do not apply to the setting at hand. For instance, if all agents' preferences are commonly known beforehand, there is no need for mechanism design. Or, if we need to design the mechanism ahead of time for a general domain without any knowledge of the instances that will actually arise, automated design cannot contribute much.[1] Finally, we may not want to assume that the agents are in any way computationally restricted, for instance because we know the stakes are high enough that the agents will spare no effort in finding the most beneficial manipulation. Nevertheless, it is not typical that some of these issues can be assumed away at no cost, and thus perhaps the following two reasons are more important.

The second reason is that at least in the short run, some of the research most valuable to the world will consist of finding *new domains* in preference aggregation where computer science techniques can be fruitfully applied. Some very recent examples of new domains include the following:

- Fortnow *et al.* [2003] introduce an expressive framework for trading securities that generate payoffs in certain states of the world (where the states of the world are represented by Boolean formulas).

- Porter [2004] studies mechanism design for the scheduling of different agents' jobs on a processor.

- We [Conitzer and Sandholm, 2004e] study the domain of donations to charitable causes, and introduce an expressive bidding language for arriving at complex contracts over who pays what to which charities—a generalization of so-called "matching offers". (This topic will be discussed in upcoming chapters, Sections 2.3, 3.3, and 5.2.)

New domains such as these are most naturally studied first at the shallower levels of the hierarchy, and later deeper down in the hierarchy. Attempting to immediately study the new domain in the deepest levels of the hierarchy may leave important issues at shallower levels underinvestigated.

Third, better solutions for problems shallower up in the hierarchy can have significant implications for problems deeper down in the hierarchy. For instance, new algorithms for solving outcome optimization problems (such as clearing a combinatorial auction) to optimality may reduce the need for designing mechanisms that implement easily computable approximations. (For example, the VCG mechanism is always sufficient to implement the social welfare maximizing outcome, if optimal outcomes can be computed.) Also, deriving (possibly partial) general characterizations of mechanisms with desirable properties may help automated mechanism design, by reducing the search space to the set of mechanisms consistent with the characterization.

Thus, there is significant motivation for studying problems strictly at the shallower levels of the hierarchy (especially in new domains); but eventually, as this research matures, the benefits of additional research in the shallower levels will decrease, and we should shift to deeper levels of the hierarchy to create greater value for the world.

[1]Though perhaps it could still be used by running it on some random instances in the domain, and attempting to extract general principles from the mechanisms generated in these experiments.

1.4 Orthogonal research directions

While the hierarchy is useful in classifying much of the research on computational aspects of preference aggregation, there are also research directions that are orthogonal to the topics in the hierarchy. Perhaps the most important such direction is that of *preference elicitation*. In preference elicitation, the idea is to not have each agent reveal its entire preferences in one step, but rather to selectively and incrementally query the agents for aspects of their utility functions, in the hope that the outcome can be determined while having the agents reveal only a fraction of their preference information. There are several benefits to reducing the amount of preferences that the agents need to reveal. First, determining one's utility for any specific outcome can be computationally demanding [Sandholm, 1993a, 2000; Parkes, 1999b; Larson and Sandholm, 2001b]. Second, in many settings, the utility functions of agents are exponentially sized objects that are impractical to communicate. Finally, agents may prefer not to reveal their utility information for reasons of privacy or long-term competitiveness [Rothkopf *et al.*, 1990].

As an example, various *ascending* combinatorial auction protocols have been proposed [Parkes, 1999a; Wurman and Wellman, 2000; Ausubel and Milgrom, 2002; de Vries *et al.*, 2003]. Such auctions maintain current prices for the items/bundles, which allows bidders to focus their bidding efforts on bundles on which they are competitive. Conen and Sandholm [2001] have proposed a more general framework where the auctioneer is enhanced by elicitor software that incrementally and adaptively elicits the bidders' preferences using specific queries until enough information has been elicited to determine the final allocation and payments. This framework was experimentally evaluated for general combinatorial auctions by Hudson and Sandholm [2004] (building on work by Conen and Sandholm [2002]), showing that the amount of preference information that needs to be elicited is only a small fraction of the total amount of preference information. Another direction of interest with close ties to computational learning theory is the identification of classes of preferences for which elicitation requires only a polynomial number of queries [Zinkevich *et al.*, 2003; Blum *et al.*, 2004; Lahaie and Parkes, 2004; Santi *et al.*, 2004; Conitzer *et al.*, 2005]. Negative results have also been obtained: for instance, [Nisan and Segal, 2005] shows that in general, obtaining a better approximation than that generated by auctioning off all objects as a bundle requires the exchange of an exponential amount of information.

Preference elicitation is a direction that is orthogonal to the hierarchy because preference elicitation can be separately studied at each node of the hierarchy. For example, one may simply wish to elicit enough information to compute an optimal allocation with respect to revealed preferences (node **(1)**); alternatively, one may wish to elicit enough information to choose outcomes in a manner that discourages untruthful bidding [Conen and Sandholm, 2001; Hyafil and Boutilier, 2006] (node **(2)**); one may even attempt to automatically design the preference elicitation protocol along with (the rest of) the mechanism [Sandholm *et al.*, 2005a] (node **(3a)**); *etc.*

Another orthogonal objective is the following: rather than having all (relevant) information communicated to a center that then computes the outcome, have the agents compute the outcome themselves in a distributed (and possibly privacy-preserving) manner [Parkes and Shneidman, 2004; Brandt and Sandholm, 2004b,a, 2005b,c,a; Izmalkov *et al.*, 2005; Petcu *et al.*, 2006]. Moreover, in real-world implementation of preference aggregation, many other orthogonal issues come up. For example, when aggregating the preferences of humans, human-computer interaction issues must be considered.

In this dissertation, I will not cover research on topics that are orthogonal to the hierarchy. This is only for the purpose of coherence: it is certainly my opinion that such orthogonal topics concern crucial components of practical preference aggregation systems.

1.5 Outline

The rest of this dissertation is layed out as follows. In Chapter 2, we discuss various settings for expressive preference aggregation, including elections, allocations of tasks and resources using combinatorial auctions and exchanges, donations to (charitable) causes, and settings with externalities. Subsequently, in Chapter 3, which corresponds to node **(1)** in the hierarchy, we analyze the complexity of the outcome optimization problem in all of these settings. In Chapter 4, we briefly review some of the basic concepts in (classical) mechanism design, as well as some famous mechanisms and impossibility results. In Chapter 5, which corresponds to node **(2)** in the hierarchy, we present some difficulties and challenges for classical mechanism design in expressive preference aggregation settings. In Chapter 6, which corresponds to node **(3a)** in the hierarchy, we introduce automated mechanism design. In Chapter 7, we review basic concepts from game theory, as well as the basic argument for restricting attention to truthful mechanisms when all agents act in a strategically optimal way (which implies that they have no computational limitations). Chapters 8 and 9 correspond to node **(3b)** in the hierarchy—mechanism design for bounded agents. In Chapter 8, we display a setting in which there is non-truthful mechanism that always does at least as well as the best truthful mechanism, and, in the face of computational boundedness, does strictly better. In this chapter, we also show that in some voting settings, it is computationally hard to find beneficial manipulations (even given the other voters' votes); however, in the final section of the chapter, we show that under some reasonable restrictions on the voting setting, finding a successful manipulation remains easy in most manipulable instances. In Chapter 9 we take a first step towards rebuilding the theory of mechanism design in the face of computational limitations, by examining how hard various game-theoretic solution concepts are to compute—and hence, which concepts can serve as a reasonable basis for a theory of mechanism design for computationally bounded agents. (Being able to compute game-theoretic solutions is of interest for other reasons as well, as will be discussed in that chapter.) Finally, in Chapter 10, we introduce the first approach that combines aspects of both automated mechanism design and mechanism design for bounded agents. Rather than optimizing the entire mechanism in a single step, this approach incrementally identifies opportunities for the agents to successfully manipulate, and changes the mechanism to remove these opportunities.

To keep the dissertation coherent and of a reasonable length, much of the research that I have done as a Ph.D. student is excluded. The excluded material includes:

- A few topics that would have fit in the dissertation quite well but were excluded to keep the length reasonable. Topics excluded from Chapter 3 include work on computing optimal Kemeny rankings in voting [Conitzer *et al.*, 2006], and solving the winner determination problem in combinatorial auctions with k-wise dependent valuations [Conitzer *et al.*, 2005]. Topics excluded from Chapter 9 include a preprocessing technique for computing a Nash equilibrium [Conitzer and Sandholm, 2006g], mixed integer programming techniques for computing Nash equilibria [Sandholm *et al.*, 2005b], and computing the optimal strategy to commit to [Conitzer and Sandholm, 2006c].

- All work on *learning in games* (which could be considered relevant to node **(3b)** in the hierarchy) [Conitzer and Sandholm, 2006a, 2003d, 2004b].

- All work on *preference elicitation* (reducing the communication necessary for choosing the outcome) in combinatorial auctions [Santi *et al.*, 2004; Conitzer *et al.*, 2005], voting [Conitzer and Sandholm, 2002c, 2005b], and mechanism design [Conitzer and Sandholm, 2004c; Sandholm *et al.*, 2005a].

- All work on *cooperative* (also known as *coalitional*) game theory [Conitzer and Sandholm, 2004d; Yokoo *et al.*, 2005; Ohta *et al.*, 2006; Conitzer and Sandholm, 2006b].

- A few other individual topics, including the complexity of metareasoning [Conitzer and Sandholm, 2003f], interpreting voting rules as maximum likelihood estimates of the “correct” outcome [Conitzer and Sandholm, 2005a], and learning algorithms for online principal-agent settings [Conitzer and Garera, 2006].

- Some of the less-relevant results in papers that are otherwise covered.