Chapter 11

Conclusions and Future Research

This dissertation set out to investigate the role that computation plays in various aspects of preference aggregation, and to use computation to improve the resulting outcomes. In this final chapter, we will review the research contributions of this dissertation, as well as discuss directions for future research.

11.1 Contributions

The following are the main research contributions of this dissertation. (Some minor contributions are omitted.)

- **A hierarchical framework** that categorizes various ways in which computational tools can improve preference aggregation (Chapter 1). This framework provides an organized view of much of the work on computational aspects of preference aggregation (and, in particular, of the research in this dissertation); it also provides a natural guide towards future research (as we will discuss in Section 11.2).

- **New settings for expressive preference aggregation** (Chapter 2). Specifically, we introduced expressive preference aggregation for *donations to (charitable) causes*, which allows agents to make their donations conditional on how much, and to which charities, other agents donate. We also introduced expressive preference aggregation in settings with *externalities*, where each agent controls variables that affect the welfare of the other agents.

- **New techniques for solving outcome optimization problems** in expressive preference aggregation settings (Chapter 3). In the context of *voting*, we introduced a powerful preprocessing technique for computing Slater rankings: a set of similar candidates can be solved recursively and replaced with a single super-candidate in the original problem. For *combinatorial auctions*, we showed that if the items are vertices in a graph, and each bid is on a connected component of the graph, then the winner determination problem can be solved efficiently if the graph is known and has bounded treewidth, or if the graph is a tree (in which case we can discover the graph). For the setting of expressive preference aggregation for *donations to charities*, we showed that the outcome optimization problem is inapproximable,

but we also provided mixed integer programming techniques for solving this problem in general, and exhibited special cases that are easy to solve. Finally, for the setting of expressive preference aggregation in settings with *externalities*, we showed that it is typically NP-hard to find a nontrivial feasible solution, but we also exhibited two special cases in which this is not the case (in one, the feasible solution with the maximal concessions can be found efficiently, and in the other, the social-welfare maximizing solution).

- **An analysis of classical mechanism design techniques** in expressive preference aggregation settings (Chapter 5). We showed that the *VCG mechanism* is extremely vulnerable to collusion, and provides poor revenue guarantees, in combinatorial auctions and exchanges. We also gave conditions that characterize when these vulnerabilities occur, and studied how computationally hard it is to decide if these conditions hold. For aggregating preferences over *donations to charities*, we showed a fundamental impossibility result that precludes the existence of an ideal mechanism (even with only a single charity), but we also gave some positive results that show that negotiating over multiple charities simultaneously can be helpful in designing good mechanisms.

- **Automated mechanism design** (Chapter 6). We defined the basic problem of automated mechanism design and determined its computational complexity for a number of special cases. We introduced a linear programming (mixed integer programming) approach for designing randomized (deterministic) mechanisms in general, and a special-purpose algorithm for a special case. We gave various applications and presented scalability results. Finally, we studied a more concise representation of problem instances, and showed how this changes the complexity of the problem.

- **Mechanisms that make manipulation computationally hard** (Chapter 8). We demonstrated that *the revelation principle fails when agents are computationally bounded*. Specifically, we exhibited a family of settings where a non-truthful mechanism is easier to execute and harder to manipulate than the best truthful mechanism; moreover, the non-truthful mechanism will perform just as well as the optimal truthful mechanism if it is manipulated nonetheless, and otherwise it will perform strictly better. We then showed that in voting, *adding a preround* can make manipulation (by an individual voter) significantly harder—NP-hard, #P hard, or PSPACE-hard, depending on whether the scheduling of the preround precedes, follows, or is interleaved with the voting, respectively. We also showed that coalitional weighted manipulation can be hard even for voting settings *with few candidates*, and that these results also imply hardness of manipulation by an individual if there is uncertainty about the others' votes. Finally, we gave an impossibility result showing that voting rules that are *usually hard to manipulate* do not exist, if we require the rule and the distribution over instances to satisfy some other sensible properties.

- **An analysis of the complexity of computing game-theoretic solutions** (Chapter 9). We showed that the basic computational problems related to *dominance* can be solved efficiently, with a few exceptions: for iterated weak dominance in normal-form games, and dominance by pure strategies in Bayesian games, these questions are NP-complete; and iterated dominance in Bayesian games can even require an exponential number of iterations. We also gave a

single reduction that shows that finding (even approximately) optimal *Nash equilibria* (for various notions of optimal), or Nash equilibria with certain other properties, is NP-complete; and that counting the number of (connected sets of) Nash equilibria is #P-hard. In addition, we showed that determining whether a pure-strategy Bayes-Nash equilibrium exists is NP-complete. Finally, we introduced a new *parameterized definition of strategy eliminability* and showed that it generalizes both strict dominance and Nash equilibrium eliminability (in that these are obtained for some settings of the parameters). We also showed that strategy eliminability is efficiently computable if and only if the parameter settings are close to those corresponding to dominance.

- **A methodology for incrementally making mechanisms more strategy-proof** (Chapter 10). This work can be interpreted as one methodology for automatically designing mechanisms for bounded agents, and as such is a first step towards our long-term goal of bringing automated mechanism design and mechanism design for bounded agents together. The idea of this methodology is to start with a naïve mechanism, such as the one that maximizes social welfare without any incentive compatibility constraints, and subsequently to identify opportunities for manipulation and locally change the mechanism to prevent these. We showed that this approach can generate certain known mechanisms (including non-truthful ones). We introduced algorithms for automatically computing outcomes according to this approach, and argued that the resulting mechanisms are hard to manipulate.

11.2 Future research

The hierarchy introduced in this dissertation provides a natural guide to future research. Typically, a new domain for expressive preference aggregation will initially be studied at the shallow levels of the hierarchy, after which research on the domain will gradually move to deeper levels. For example, the allocation of tasks and resources (using combinatorial auctions and exchanges) was initially studied at the shallowest node (outcome optimization); in recent years, most research on this domain has focused on (algorithmic) mechanism design, the second node in the hierarchy; and most recently, automated mechanism design has started to be applied to these settings. In contrast, domains that have only recently started receiving serious attention, such as expressive negotation in settings with externalities, are still being studied exclusively at the level of outcome optimization. Hence, natural directions for future research include pushing existing domains deeper down the hierarchy, as well as introducing new domains—or formalizing domains that already exist in the real world—and (presumably) studying them at the shallowest levels first. Additionally, in the context of mechanism design for bounded agents (and especially automated mechanism design for bounded agents), it is not yet completely clear how mechanisms should be evaluated. Thus, future research at these nodes will also involve developing a general theory for such evaluation. Domain-specific studies, such as the ones we did on voting, may help in doing so.

Much research also remains to be done on topics orthogonal to the hierarchy, such as preference elicitation and distributed computation of outcomes (see Section 1.4). These topics can be studied at each node in the hierarchy, for any domain. However, typically, doing so requires that research on that domain at that node has already reached a basic level of maturity—specifically, it requires that

we have a good grasp on how outcomes should be chosen, and how these outcomes can be efficiently computed given the preferences of the agents. Because of this, these orthogonal topics will usually not be the first to receive attention, but this is certainly not because they are unimportant.

While the hierarchy proposed in this dissertation provides a high-level guideline to future research, the research contributions at the individual nodes of the hierarchy suggest many more specific open questions and directions. The remainder of this section will lay out some of these more immediately accessible avenues for future research.

11.2.1 Node (1): Outcome optimization

In Section 3.1, we saw how the preprocessing technique of finding and aggregating sets of similar candidates into super-candidates can drastically speed up the search for optimal Slater rankings (which are NP-hard to find). One may ask whether similar techniques can be applied to other hard-to-execute voting rules. We already discussed how the technique can be extended to apply to computing Kemeny rankings, but we do not yet have experimental results on the efficacy of doing so. It would also be interesting to characterize restrictions on the votes that have the effect of making the preprocessing technique sufficient to solve the Slater problem in polynomial time. (One such restriction that we discussed is that of having a hierarchical pairwise election graph, but there may be other restrictions with this property.) Another possibility is to look for entirely different preprocessing techniques, or to try to generalize the similar-candidates technique. Finally, given a good understanding of what makes a voting rule amenable to the use of such techniques, one may use this understanding in the design of new voting rules—by ensuring that these new rules allow for the application of such techniques and can therefore be executed fast.

In Section 3.2, we showed how the winner determination problem in combinatorial auctions can be solved in polynomial time, *if* we know an item graph for the instance that has bounded treewidth; additionally, we saw how to *find* an item *tree* in polynomial time (if it exists). This left us with a very specific open question: can we find item graphs with small treewidth (but treewidth greater than 1) in polynomial time if they exist? For example, can we find an item graph of treewidth 2 in polynomial time (if it exists)? Alternatively, given that an item graph of treewidth k exists, can we find an item graph of treewidth at most (say) $2k$? If we can, then the mere fact that an item graph of bounded treewidth *exists* for the winner determination problem at hand will guarantee that it can be solved in polynomial time (whereas now, we additionally require that we *know* the graph). As another specific open question (perhaps of less significance), we showed that in the case where bids are allowed to contain multiple components, constructing a line item graph is NP-complete when 5 components per bid are allowed; but we left open whether this is so for fewer (say, 4) components. Additional future research directions include comparing the item-graph based algorithms to other winner determination algorithms (*e.g.* using a solver such as CPLEX), as well as *integrating* the item-graph based algorithms into search algorithms (where they can be applied at individual nodes in the search tree).

As for the framework and algorithms for expressive preference aggregation for donations to charities (Sections 2.3 and 3.3), one possible future direction is to build a web-based implementation of these techniques that will allow them to be used in practice. Another direction is to experimentally test the scalability of the mixed integer and linear programming formulations of the clearing problem that we proposed. One can also try to characterize other restrictions on the bids that make the

clearing problem easy to solve. Another possibility is to consider the elicitation problem in this setting, and to design good iterative mechanisms for addressing this problem. Finally, one can consider other bidding languages—for example, languages that allow donations to be conditional on which other donors are donating (and on how much they are donating).

For the outcome optimization problem in settings with externalities (Sections 2.4 and 3.4), we showed mostly negative (NP-hardness) results. Future research can potentially address this in several ways. First, algorithms could be given that require exponential time in the worst case, but run fast in practice; one possibility for this could be the use of mixed integer programming techniques. Another possibility is to try to design approximation algorithms, although this does not look very promising given that even the problem of finding a nontrivial feasible solution is NP-hard in most settings that we studied. Finally, we may try to simplify the problem, possibly by changing or restricting the language in which agents express their preferences.

11.2.2 Node (2): Mechanism design

Our research on vulnerability of the VCG mechanism to collusion and low revenue in combinatorial auctions and exchanges (Section 5.1) suggests a number of future research directions. First, it would be desirable to create new mechanisms that do not share these weaknesses. These mechanisms may be truthful mechanisms—perhaps even other Groves mechanisms—but they may also be non-truthful mechanisms. For example, we saw that (even under strategic behavior by the agents) the first-price mechanism does not run into trouble in some of the instances that caused problems for the VCG mechanism, and it would certainly be interesting to characterize more generally how first-price mechanisms perform in terms of collusion and revenue. As another direction, we only characterized when certain worst-case scenarios can occur under the VCG mechanism—but there are certainly other instances in which bad (albeit not worst-case) outcomes can occur. Providing a more complete characterization that also classifies these instances would give us an even better understanding of the VCG mechanism's vulnerabilities.

Mechanisms for expressive preference aggregation in the setting of donations to charities (Section 5.2) (or, more generally, for expressive preference aggregation in settings with externalities/public goods) still leave much to be desired. In part, this is due to fundamental impossibility results such as the one that we presented. Nevertheless, our results also suggest that such impossibilities can sometimes be mitigated by using a single mechanism to decide on the donations to multiple charities—although it is not yet clear how to do this in the general case. While difficulties for mechanism design in these settings occur even when restricting our attention to the case of quasilinear preferences, it is important that eventually mechanisms will address the case of more general preferences as well. This is especially so because typically, when donating money to a large charity, the marginal benefit to the charity of another dollar remains roughly constant even when donations are large. Thus, the main reason why donors give only limited amounts is that larger donations will prevent them from buying more basic goods for themselves (*e.g.* food, clothing)—that is, as they donate more, the marginal utility of keeping a dollar for themselves becomes larger, and thus their utility is not linear in money. Automated mechanism design may be helpful in creating mechanisms in the face of non-quasilinear utilities.

11.2.3 Node (3a): Automated mechanism design

Automated mechanism design is a relatively new research area, and because of this much remains to be done. Perhaps the most important direction for future research is getting automated mechanism design to scale to larger instances. There are numerous ways in which this can be achieved. First of all, better algorithms for the general problem can be developed. This can be done either by improving the mixed integer/linear programming formulations, or by developing new algorithms altogether. As examples of the former, we have observed (1) that some of the constraints in our formulation imply some of the others (similar observations have been made by others [Gui *et al.*, 2004; Lovejoy, 2006]), and omitting these implied constraints (perhaps surprisingly) significantly decreases the time that CPLEX requires to solve instances; and (2) in settings that are symmetric with respect to the agents, formulations that are much more concise (and easier to solve) can be given. As an example of the latter, in Section 6.7 we introduced a special-purpose algorithm for the case of designing deterministic algorithms without payments for a single agent, and perhaps this algorithm can be extended to apply to the general problem.

On the other hand, rather than address the general problem, one can also choose to focus on specific domains. For example, one can restrict attention to the automated design of combinatorial auction mechanisms, as has been done by Likhodedov and Sandholm [2003, 2004, 2005]. By focusing on such a specific domain, it is possible to make use of theoretical results that characterize (optimal) truthful mechanisms in that domain, which can significantly reduce the space of mechanisms that must be searched. It is also possible to restrict the space of mechanisms under consideration without such a characterization result. For example, Sandholm and Gilpin [2006] restrict attention to sequences of take-it-or-leave-it offers for selling items. Such a restriction will potentially exclude all optimal mechanisms from the search space, but typically there are many reasonable restrictions of the search space that one would not expect to come at too much of a cost. For example, one can require that the lowest k bids never win anything. (Such restrictions, besides improving scalability, also allow us to rule out mechanisms that are intuitively unappealing.) If there is a formal guarantee that optimal mechanisms in the restricted search space always have objective values that are close to those of optimal mechanisms in the unrestricted search space, then an algorithm that identifies an optimal mechanism in the restricted search space constitutes an approximation algorithm for the unrestricted setting.

There are many other important future directions on automated mechanism design besides improving its scalability. New domains in which AMD can be applied continue to be found (for example, recommender systems [Jurca and Faltings, 2006]). One can also use AMD as a tool in traditional mechanism design. For example, by letting the software compute the optimal mechanism for a number of sample instances in the domain of interest, one can gain intuition about what the characterization of the optimal mechanism for the general case should look like. It is also possible to use automated mechanism design software to help disprove general conjectures, by solving randomly sampled instances until a counterexample is found. For example, a conjecture that optimal mechanisms in a certain domain need never use randomization can be disproved by finding an instance in the domain where the optimal randomized mechanism performs strictly better than the optimal deterministic mechanism.

Finally, one can seek to expand the automated mechanism design toolbox, for instance by studying how to model additional solution concepts. For example, one can add constraints to the problem

to prevent collusion from being beneficial. One can also try to modify the Bayes-Nash equilibrium constraints so that truthful reporting remains optimal even if the designer's prior over agents' types is slightly inaccurate. Additionally, it may be possible to create tools for cases where the designer has only partial information about the prior over agents' types. Finally, it would be useful to have techniques for the case where the type space (and perhaps also the outcome space) is continuous rather than discrete. Even if techniques for solving such continuous instances directly remain elusive, it would help to at least know how to discretize them well.

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

In Section 8.1 we showed that there exist settings in which there are non-truthful mechanisms that perform at least as well as any truthful mechanism (and strictly better if agents are computationally bounded), and that are also computationally easier to execute. Future research should investigate whether this result can be generalized to other settings of interest, such as combinatorial auctions. One may also consider non-truthful mechanisms that do not have all three of these properties, *i.e.* non-truthful mechanisms that are not easier to execute, or are not guaranteed to perform strictly better in the face of computational boundedness, or are not guaranteed to perform at least as well when agents are strategic. In the last case, it would be risky to run this non-truthful mechanism instead of the optimal truthful one because the outcome may be worse, and hence it would be good to have some way of assessing this risk, *i.e.* being able to estimate what the odds are that the resulting outcome will be worse (or better), and how much worse (or better) it will be.

There are also various avenues for future research on voting rules that are computationally hard to manipulate (Sections 8.2, 8.3, and 8.4). Most significantly, it is important to see whether the impossibility result that we presented can be circumvented to create a voting rule that is in some sense usually hard to manipulate. We offered a few approaches at the end of Section 8.4 that could potentially create such a rule (without contradicting the impossibility result). Another direction for future research is to create new ways to tweak existing voting rules to make them harder to manipulate. (Elkind and Lipmaa [2005a] have already generalized the technique of adding a preround, showing that voting rules can be *hybridized* with each other to make manipulation harder.) It would be especially interesting to create tweaks that make the manipulation problem hard even with few candidates. Finally, it is important to study in more detail how hard the manipulation problem is when the nonmanipulators' votes are not exactly known, as this is typically the situation that manipulators are confronted with.

Much work also remains to be done on computing game-theoretic solutions (Chapter 9). In Section 9.2 we showed that computing Nash equilibria with certain properties is NP-hard. Are there any interesting properties for which this is not the case? To illustrate this, consider the following computational problem: find a Nash equilibrium with the property that there does not exist another Nash equilibrium in which each player's support is reduced by one strategy (relative to the former equilibrium). An equilibrium with this property can be found by finding any one Nash equilibrium, and then trying each possible way of reducing each player's support by one strategy (there are only polynomially many ways of doing so, and given the supports, finding a Nash equilibrium only requires solving a linear feasibility program). If we are successful in finding such a reduced Nash equilibrium, then we repeat the process with that equilibrium, until we fail. (It should be kept in mind that computing any one Nash equilibrium is still PPAD-complete.) Discovering properties of

this kind can also help address the equilibrium selection problem (if there are multiple equilibria, which one should be played?), since one can argue that it is more natural to play an easy-to-compute equilibrium than to play a hard-to-compute equilibrium.

Various questions also remain on the generalized eliminability criterion introduced in Section 9.3. Are there other special cases (besides the case of small E sets) where it can be computed in polynomial time whether a strategy is eliminable? Are there alternative characterizations of this eliminability criterion (or restricted versions thereof)? Such alternative characterizations may make the criterion easier to understand, and may also lead to new approaches to computing whether a strategy can be eliminated. Perhaps it is also possible to create altogether different eliminability criteria. Such a criterion may be strictly weaker or stronger than the one presented here, so that eliminability in the one sense implies eliminability in the other sense; or the criteria may not be comparable. Another future research direction is to apply the eliminability technique in practice, testing it on random distributions of games such as those generated by GAMUT [Nudelman *et al.*, 2004], and using it to speed up computation of Nash equilibria (possibly on similar distributions of games).

Finally, other directions for future research that can be taken for any one of these solution concepts include computing solutions for restricted classes of games, as well as computing solutions under different game representations (including games of imperfect information, graphical games [Kearns *et al.*, 2001], action-graph games [Bhat and Leyton-Brown, 2004], *etc.*).

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

Automated mechanism design for bounded agents is in its infancy, and future research will likely create new and more comprehensive approaches. However, even the initial approach that we proposed—starting with a naïve mechanism and incrementally making it more strategy-proof—raises many questions. Most significantly, while we have given a general template for the technique, many choices must be made to fully instantiate it. For instance, we must choose the set of manipulations that we try to prevent, the way in which we try to prevent them, and when we terminate the updating process. We gave examples of various instantiations of the technique and the mechanisms that they generate, but ideally we would have a single instantiation of the technique that dominates all others. Even if this is not possible, it would be very helpful if we could provide some guidance as to which instantiation is likely to work best for a given application.

Another avenue for future research is to use this approach to help us create new general mechanisms. Whereas automated mechanism design in the sense of Chapter 6 can only help us conjecture truthful mechanisms, this approach potentially can also help us create new non-truthful mechanisms—for example, new voting rules. (Recall that all truthful voting rules are unsatisfactory by the Gibbard-Satterthwaite impossibility theorem.)

On the matter of designing algorithms for automatically executing the approach, it is important to find algorithms that scale to larger numbers of iterations. Not only will this allow us to design mechanisms with fewer possibilities for beneficial manipulation, but it will also make the remaining beneficial manipulations more difficult to find, because agents must reason through more iterations to find them. (Of course, if the agents have access to the more efficient algorithms as well, this benefit disappears. Nevertheless, if we as designers do not find more efficient algorithms, we run the risk that agents will find these algorithms themselves, and will be able to out-compute us and

find beneficial manipulations.)