Mechanism Design for Preference Aggregation over Coalitions

Student: Eric Hsu[1] and Advisor: Sheila McIlraith[2]

[1] University of Toronto, eihsu@cs.toronto.edu
[2] University of Toronto, sheila@cs.toronto.edu

Abstract. The aggregation of individuals' preferences into a single group outcome is both well-studied and fundamental within decision theory. Historically, though, a pervasive simplification has been to strip agents of the ability to form coalitions and strategically reveal their intentions. CP techniques can address such possibilities within a restricted social-choice framework that represents mechanisms for aggregating preferences as decision trees; the internal nodes are labeled by various individuals, and the leaves represent outcomes that arise from the flow of individual decisions down the various branches of the tree. Currently, a highly simplified version of the language allows the automatic generation of such trees meeting user-specified properties, using a standard CSP-solver. By further enrichments within a more specialized reasoning framework, the eventual goal is to circumvent negative theoretical theorems concerning preference aggregation, by exploiting the combination of coalition reasoning and automated design.

1 Introduction

Mechanisms implement mappings from sets of preferences gathered from individual agents to a single preference ordering over group outcomes. To be considered fair, a mechanism ought to satisfy a number of properties with technical definitions but intuitive appeal: no single agent should have the power to ignore the wishes of all others (non-dictatorship); if all agents most desire some outcome, that outcome should be chosen by the mechanism (unanimity); and the relative ordering of two alternatives in the final ranking should not change if it stays the same for all of the individual agents (independence to irrelevant alternatives). Unfortunately, classical impossibility results in economics directly extend to the case of preferences; even these basic criteria cannot all be met at once by a single mechanism [1, 2].

Further, the typical setting under which such results hold is already itself compromised in excluding collusion, coalitions, or strategic partnerships. That is, in those limited situations where strategic voting can be eliminated, a voting scheme might be ruined if the parties involved form blocs to pursue common interests. Where equilibria exist, they are typically individual-oriented; no single agent can improve its position by changing strategies, but if two were to do so at the same time then equilibrium would be broken. A related phenomenon is the

assumption of rationality, which fails to model the notion of threats–if agents are willing to do something worse for themselves just to deter others from particular lines of action, and such agents are able communicate this effectively, they can develop an advantage that is not captured within traditional decision theory.

Coalitions and communications complicate theoretical analysis or underlying paradigms of rationality, but in any particular setting they might actually serve to constrain a problem to the point that traditional impossibility results can be circumvented. That is to say, it could be worthwhile to integrate them not only because they are crucial real-world phenomena, but because they might even make specific tasks easier.

In particular, automated mechanism design is a recently-proposed methodology for overcoming some of the impossibility results by designing particular mechanisms on the fly for particular situations [3]. While no perfect mechanism can work in every context, wherever there is information about the players a custom mechanism can be designed for use in that specific setting. While the original emphasis of such research is on auction over continuous domains and uses linear programming, the general idea is very abstract and can perhaps be transformed for discrete systems, via constraint programming.

The general format for doing so follows recent research involving logics over extensive games, which itself originates from Alternating Time (ATL) temporal logics [4, 5]. The remainder of this abstract will informally describe the chosen format, and show how a simplified version has been solved within a standard CSP-solver.

2 Extensive Games for Social Choice

Extensive games are equivalent to other representations of decision functions, such as tables or equations, but they most concisely capture the structure of relationships between the individual parties. That is, an enormous, complete extensive game can represent every possible combination of agent profiles, but a smaller game built from a subset of that tree exploits the structure of a particular situation.

More concretely, an extensive game is a tree where leaf nodes are labeled with outcomes, and the rest of the nodes are labeled with the names of various parties involved in the game. Both kinds of labels may be duplicated throughout the tree. The meaning of labeling a node with an agent's name is that if the game reaches that node, the agent can choose which of its children should form the next step. If the child is another interior node and thus labeled with an agent's name, the meaning is that the new agent gets to choose what happens next. If the chosen child is a leaf, then the game ends with the outcome labeling the chosen leaf. The tree structure, with all its labels, is known to all parties before the game is played.

Figure 1 shows a small example of such a game, in the form of a majority-like voting procedure. The game begins with A choosing which of B and C chooses next. It the ends with B or C choosing one of the three outcomes, x, y, and z. Recall that the structure of the game is known to all of the players.

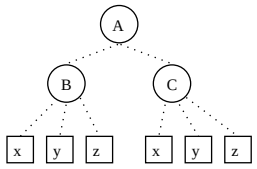

Fig. 1. Simple Voting Procedure.

The voting tree supports the principle of majority rule in that if any two of the parties desire a particular outcome, then that outcome comes to pass. Clearly if A agrees with either B or C, and knows it, then it can choose the node of the partner in agreement. Further, if B and C both want the same outcome, then A is confronted with a lack of options; no matter what it will have to choose someone who will end up choosing the outcomes in question.

In the general case the leaves coming from a final agent node need not cover exactly the set of all outcomes. In the present implementation, this feature has been temporarily omitted, but it is a useful extension for the future. For example, the mechanism in Figure 1 could be modified so that the only outcome nodes descended from B were y and z; then A and B would have the power to choose x or choose y, but agents A and C would each individually have veto power over outcome z.

3 Coalition Properties

Such considerations raise the second main representational issue: the properties that are specified to hold over such games. In general it is possible to make very rich specifications, but in order to most efficiently get a proof-of-concept, the full language has been whittled down to a simple essential core.

3.1 General Case and Simplifications

Communication Properties Recall that the games are played under perfect information; the agents are all informed of the structure of the protocol before voting commences. On the other hand, agents do not necessarily know each others' preferences. One of the points of this project is to study how game properties can change when agents strategically reveal their preferences to others (as "threats" or "promises," etc.) However, at present such communicative properties are left aside in interest of getting things working. What remains is a setting of perfect information over orderings; the design of a mechanism reasons over what will happen when different combinations of agents have shared goals, and when they all know of each others' goals.

Full Preference Orderings Another simplification is that the initial implementation only considers agents' single top choices, rather than allowing them

to hold orderings over the entire space of outcomes. This is like election of most governments, whereby citizens can only support a single candidate, all or nothing.

Propositional Outcomes by Boolean Formulas Finally, the most interesting future effort will be to replace simple outcomes with propositional theories. That is, each leaf node will present an agent with the choice of some complete assignment to a propositional domain. Such models could just be simulated by 2^n simple outcome nodes, where n is the number of propositions, but explicit propositionization would allow the expression of sets of assignments via boolean formulas.

For instance, an agent B might be allowed to decide that $(p \vee q) \wedge \neg r$ holds, rather than a single specific combination of p, q, and r. This would allow more complex reasoning and strategies in decision-making and coalition formation, as parties might not have exactly identical interests but still be able to infer which decisions or partnerships will produce satisfying results.

3.2 Simplified Formalism

For the initial proof-of-concept implementation, the above capabilities are left to future work. Instead, the focus is on a highly restricted subset of the full language.

Specifically, a coalition property is a triple denoted $[\Gamma : x]o$, where Γ is a subset of the agents, and x and o are outcomes (or, in the future, properties of outcomes in the richer case of boolean formulas.) Within the simplified setting, the meaning is that if all of the agents in Γ want the simple outcome x, then outcome o will come to pass.

Typically, and throughout the rest of this abstract, x and o are identical, so preferences can be abbreviated as $\Gamma : x$; when some coalition wants x, they can get x. A final simplification that greatly clarifies presentation is to make the outcomes indistinguishable over coalition properties. That is, where $\Gamma : x$ holds it now holds for any particular outcome, as though the x were a universally quantified variable rather than a specific outcome. In other words, we reason about whether a coalition Γ will be able to get the outcome of its choosing, whatever outcome that may be; this is particularly relevant under the previous simplification where all terminal agent nodes have exactly the full set of outcomes as children.

In summary, the input to the system will be a set of constraints of the form $\Gamma : x$ (or $\neg\Gamma : x$), prescribing that various sets of agents can or cannot control the outcome. To that end, the x is itself sort of superfluous as notation–the point is whether or not Γ is decisive within the game to be designed.

4 Framework

This simplified version of the mechanism design problem has been implemented within the EFC CSP solver developed at the University of Toronto [6].

4.1 Variables and Constraints

The system is constrained to generate complete binary trees of depth parameterized by the user. Such trees can represent more elegant trees of arbitrary branching and balancing. For instance, if A is supposed to choose between P, Q, and R, it could first choose between P and itself, where on choosing itself it would then face the option of either Q or R. Similarly, if the user specifies a greater depth than necessary, the system can generate a tree with many redundant nodes, as is the case for filling out unbalanced regions.

Thus, the basic idea is to provide the system with a blank tree template, and instruct it to fill it out so that it meets the provided constraints. Secondary variables keep track of what outcomes result from the existence of various coalitions.

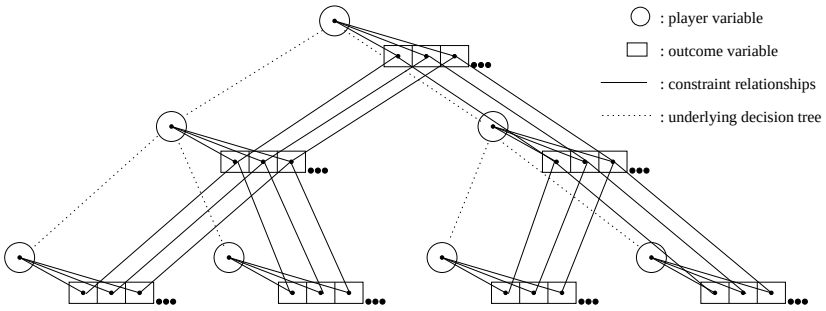

Fig. 2. Structure of Player/Value Variables and Their Constraints.

Figure 2 shows the structure of such variables and the relationships between them. Circles represent variables ranging over agent names (like A, B, or C); they comprise the game tree, and the task of mechanism design is to label them with the appropriate agents to ensure that the desired properties are met. The game tree is depicted by dotted lines, as it is not directly represented within the constraints.

The boxes in the figure represent variables ranging over outcomes (like x, y, or z); they are grouped together with nodes to represent the (potentially unknown) choice that will be made by that player for each coalition mentioned in the supplied constraints. That is, the first box by each node represents the outcome that will be chosen at that node under the assumptions of Coalition Constraint 1, the second box represents Coalition Constraint 2, and so on, up to the number of constraints imposed upon the mechanism design process.

The semantics of the outcome variables work from the bottom up. Restricting our attention to a single coalition constraint $\Gamma : x$, we want to check whether the coalition Γ has control within a particular tree. Thus we are only concerned with one set of boxes from the arrays beside the circles in the figure. Beginning with the bottom-most player nodes, we label the associated outcome nodes based on

whether the chosen players are in Γ. If they are, the outcome is 'x', as they will choose whatever the coalition is trying for. Otherwise, the outcome is '$*$'; the node is outside of coalition control so nobody knows that its player will select.

From there the process continues to the parents of these nodes; given the outcomes proposed by its children, a parent will try to choose 'x' if it is a member of the coalition, but will choose '$*$' when it is not, unless both of its children propose 'x'. Thus the value of a non-leaf outcome node is subject to a constraint with three other variables: that of its player node, and the corresponding outcome variables for the player node's two children.

Finally, the top outcome node reflects the overall outcome for the given coalition within this game; 'x' means that the coalition was able to enforce its will, '$*$' means it was not. Such values are feasible or not for a top outcome depending on whether the coalition constraint is positive or negative, that is, on whether the coalition is supposed to have control or not.

Variable ordering must be tightly controlled in order to ensure that this process occurs in the correct order; propagators ensure that it happens efficiently.

5 Results and Future Work

With the stated, extreme, simplifications, EFC is able to solve the automated mechanism design problem for problems involving about 10 parties, thousands of coalition constraints, and trees of depth up to 10, within seconds. Beyond that, the exponential size of explicit tree representation combines with the combinatorially increasing number of possible labelings to rapidly make the problem intractable.

Future efforts will probably have to use a tighter representation, treating paths as first-class objects, perhaps under some new set of simplifications. This may mean a switch into a less explicit reasoning language, like the situation calculus or temporal logic.

The hope is that such gains will allow the reinstatement of the full language described in Section 3.

References

1. Arrow, K.: Social Choice and Individual Values. John Wiley and Sons (1951)
2. Rossi, F., Venable, K.B., Walsh, T.: Aggregating preferences cannot be fair. Intelligenza Artificiale: Constraints and Agents (2005)
3. Conitzer, V.: Complexity of mechanism design. Proc. of 18th Conference on Uncertainty in Artificial Intelligence (UAI '02), Edmonton, Canada (2002) 103–110
4. van Otterloo, S., van der Hoek, W., Wooldridge, M.: Preferences in game logics. Proc. of 3rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS '04), New York, U.S.A. (2004)
5. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM (2002) 672–713
6. Bacchus, F., Katsirelos, G. (http://www.cs.toronto.edu/~gkatsi/efc/)