Automated Mechanism Design: Motivations, Foundations, Formulations and More

Prabhat Nagarajan

February 2016
Supervisor: Dr. Michael Albert
CS 370 - Undergraduate Reading and Research

Motivation

Traditionally, mechanisms have been designed manually. That is, the designer identifies the characteristics that she would like in her mechanism, and then mathematically designs the mechanism. This had led to a number of standard well-known mechanisms that perform quite well. However, there are a number of disadvantages to these mechanisms. Many mechanisms (e.g. the VCG Mechanism) maximize social welfare, and do not maximize the designer's objective if she is self-interested. The mechanisms that do maximize the designer's objective apply only to restricted settings. Even amongst these however, they apply to maximizing payments, without consideration for any other preferences the mechanism designer may have.

However, *automated mechanism design* (AMD) allows a mechanism to be created for a particular setting. There are a number of advantages that AMD provides, that are absent from traditional mechanism design. AMD can be used in settings other than those of the classes of problems studied in traditional mechanism design. It makes it possible to avoid the impossibility results. It delivers better mechanisms, as it is tailored to a specific setting, whereas a standard mechanism may deliver satisfying but suboptimal results in this setting. It also shifts the computation required for mechanism design from humans to computers.

Foundations

It so happens that we can design optimal randomized mechanisms in polynomial time using linear programming, and with mixed integer programming we can design optimal deterministic mechanisms.

Definition. (From text) *In an **automated mechanism design setting**, we are given*
A finite set of outcomes O;
A finite set of n agents;
For each agent i,

- *a finite set of **types**: set of types Θ_i,*
- *a probability distribution γ_i over Θ_i (in the case of correlated types, there is a single joint distribution γ over $\Theta_1 \times ... \times \Theta_n$),*
- *a utility function $u_i : \Theta_i \times O \to \mathbb{R}$;*

An objective function whose expectation the designer wishes to maximize.

For example, the set of outcomes could represent who gets the items. Note that the set of types must be finite, and not continuous as can be the case with other mechanism design settings. One can take a continuous type space and discretize it to satisfy the properties of an automated mechanism design setting. The probability distribution γ_i gives the probabilities agent i's types. The utility function simply shows an agent's utility given his type and the outcome of the mechanism.

The designer can have many possible objective functions, including objective functions where the mechanism designer is either **benevolent** or **self-interested**. A benevolent mechanism designer pursues the agents' collective happiness, whereas a self-interested designer seeks to maximize the objective function:

$$g(o) + \sum_{i=1}^{n} \pi_i,$$

where g maps an outcome to a value, a value that measures the designer's preference over the outcomes, and where the summation is the sum of the payments π_i made by the agents.

What we have just defined is an automated mechanism design *setting*, however, we now define *automated mechanism design*.

Definition. (From text) **Automated Mechanism Design (AMD):** *We are given an automated mechanism design setting, an IR notion (ex interim, ex post, or none), and a solution concept (Dominant strategies or Bayes-Nash equilibrium). Also, we are told whether payments are possible, and whether randomization is possible. Finally, we are given a target value G. We are asked whether there exists a mechanism of the specified type that satisfies both the IR notion and the solution concept, and gives an expected value of at least G for the objective.*

Mechanism Design Setting: An Example

We present divorce settlement as an example for an AMD setting. The agents in this situation are the two individuals in the marriage. The arbitor is the mechanism designer that seeks to allocate a painting that they had mutually owned. The four possible outcomes are: $O = \{$the husband gets the painting, the wife gets the painting, the painting remains in joint ownership and is hung in a museum, the painting is burned$\}$. The types for each of the agents are $\Theta_{man} = \{high, low\}$ and $\Theta_{woman} = \{high, low\}$, where *high* represents a strong desire for the painting, and *low* represents not caring for the painting too much. The distribution over the types is $\gamma_{man} = (0.8, 0.2)$ and $\gamma_{woman} = (0.8, 0.2)$, where 0.8 corresponds to the probability of having the type *high*, and 0.2 being the probability of having the type *low* . The man and the woman share a utility function defined as follows:
$u(low,$ get the painting$) = 2$
$u(low,$ other gets the painting$) = 0$
$u(low,$ joint ownership$) = 1$
$u(low,$ burn the painting$) = -10$
$u(high,$ get the painting$) = 100$
$u(high,$ other gets the painting$) = 0$
$u(high,$ joint ownership$) = 50$
$u(high,$ burn the painting$) = -10$
The objective function is to maximize social welfare, meaning the this mechanism designer is benevolent. That is: $maximize \sum_{i=1}^{n} u_i(\theta, o)$.
We have constructed a setting that satisfies the definition of an automated mechanism design setting.

Complexity Results

We have many results showing the hardness of automating the design of deterministic mechanisms. While we omit the proofs of the following statements, the general idea behind each proof is to create a reduction from well-known NP-Complete problems to the deterministic AMD problem, and then to show membership of the AMD problem in the complexity class NP.

Theorem. *The AMD problem for designing deterministic mechanisms without payments is NP-complete.*

Theorem. *The AMD problem for designing deterministic mechanisms with payments in NP-complete.*

Theorem. *The AMD problem for designing deterministic mechanisms is NP-complete.*

Linear and Mixed Integer Programming Approaches

The problem of designing an optimal *randomized* mechanism can be represented as a linear program. The size of this linear program is exponential only in the number of agents. That is, if the number of types or outcomes vary, the linear program will not grow exponentially in size. Thus, if the number of agents is held constant, the size of the linear program is polynomial, and since linear programs can be solved in polynomial time, the problem of designing the optimal randomized mechanism with a constant number of agents is in the complexity class P.

Theorem. *(From text) With a constant number of agents, the optimal randomized mechanism can be found in polynomial time using linear programming, both with and without payments, both for ex post and ex interim IR, and both for implementation in dominant strategies and for implementation in Bayes-Nash equilibrium —even if the types are correlated (that is, an agent's type tells him something about the other agents' types.)*

Proof. Let $T = max_i\{|\Theta_i|\}$, that is, let T be the cardinality of the largest set of types of any agent. The variables in the linear program are the probabilities and the payments. The probabilities are $p(\theta_1, \theta_2, ..., \theta_n)(o)$, the probability of the n types and the outcome o occurring. There can be at most $T^n|O|$ probabilities, as there are n agents with at most T types each, and there are $|O|$ outcomes. The payments are $\pi_i(\theta_1, \theta_2, ..., \theta_n)$, which constitutes at most nT^n variables, as there n agents, again with at most T types, so there are T^n possible tuples of types, so there are nT^n possible payments. As one can see, the number of variables is exponential in n, which is why we fix the number of agents.

The IR constraints are:
For *ex post* IR,

- For all $i \in \{1, 2, ..., n\}$ (for all agents) and for every $(\theta_1, \theta_2, ..., \theta_n) \in \Theta_1 \times \Theta_2 \times ... \times \Theta_n$ (for every possible tuple of types), we add

$$(\sum_{o \in O}(p(\theta_1, \theta_2, ..., \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, ..., \theta_n) \geq 0.$$

  Essentially this is saying that for all agents, the sum of the probabilities of all possible type-tuples with an outcome, multiplied by the utility received by the agent, subtracting the payments the agent has to make, must be greater than 0. Note that at most nT^n constraints are added to the linear program since for each of the n agents, there are at most T^n possible type-tuples.

For *ex interim* IR,

- For every $i \in \{1, 2, ..., n\}$, for every $\theta_i \in \Theta_i$, we add

$$\sum_{\theta_1, ..., \theta_n} \gamma(\theta_1, ..., \theta_n|\theta_i)((\sum_{o \in O}(p(\theta_1, \theta_2, ..., \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, ..., \theta_n)) \geq 0.$$

  This adds at most nT constraints because we have at most T type-tuples, and we compute the sum for each of n agents. Essentially this is individual rationality with an agent's type and beliefs over the other agents' types.

For the solution concept constraint of dominant strategies,

- For every $i \in \{1, 2, ..., n\}$, for every $(\theta_1, \theta_2, ..., \theta_i, ..., \theta_n) \in \Theta_1 \times \Theta_2 \times ... \times \Theta_n$, and for every alternative type report $\hat{\theta_i} \in \Theta_i$, we add the constraint

$$\sum_{o \in O}(p(\theta_1, \theta_2, ..., \theta_i, ...\theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, ..., \theta_i, ..., \theta_n) \geq$$

$$\sum_{o \in O}(p(\theta_1, \theta_2, ..., \hat{\theta_i}, ...\theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, ..., \hat{\theta_i}, ..., \theta_n).$$

  This adds at most nT^{n+1} constraints. For every agent n, and for every T^n type-tuples, we have at most T alternative type reports, leading to nT^{n+1} possible additional constraints. Essentially what this constraint is stating is that the payoff of reporting type θ_i is weakly dominant to reporting other types.

For the solution concept constraint of Bayes-Nash equilibrium,

- For every $i \in \{1, 2, ..., n\}$, for every $\theta_i \in \Theta_i$, and for every alternative type report $\hat{\theta}_i \in \Theta_i$, we add the constraint

$$\sum_{\theta_1,...,\theta_n} \gamma(\theta_1, ..., \theta_n|\theta_i)((\sum_{o \in O}(p(\theta_1, \theta_2, ..., \theta_i, ..., \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, ..., \theta_i, ..., \theta_n)) \geq$$

$$\sum_{\theta_1,...,\theta_n} \gamma(\theta_1, ..., \theta_n|\theta_i)((\sum_{o \in O}(p(\theta_1, \theta_2, ..., \hat{\theta}_i, ..., \theta_n))(o)u(\theta_i, o)) - \pi_i(\theta_1, \theta_2, ..., \hat{\theta}_i, ..., \theta_n)).$$

  This adds at most nT^2 constraints, because for every agent, there are at most T types and at most T alternative type reports, yielding nT^2. This constraint is standard BNE, given an agent's belief over the other agents' types, the agent is a least as well off with her current type as opposed to reporting a different type.

Lastly, we have the objective function of the linear program:

$$\sum_{\theta_1,...,\theta_n} \gamma(\theta_1, ..., \theta_n)((\sum_{o \in O}(p(\theta_1, \theta_2, ..., \theta_i, ..., \theta_n))(o)g(o) + \sum_{i=1}^{n} \pi_i(\theta_1, \theta_2, ...\theta_n)).$$

This is essentially saying the linear program should generate a mechanism that maximizes a combination of the mechanism designer's preference over the outcome (the function g) and the payments received from the agents. □

In the proof above, if we force all probability variables to have a value of either 0 or 1, making our linear program a mixed integer program, we will obtain the optimal deterministic mechanism.

Scalability Results

While we omit the specific numbers and details of the scalability experiments of AMD, we will discuss the runtime as we increase the size of the inputs, specifically on randomized mechanisms. Runtime increases sharply as the number of agents increases. Runtime increases with the number of outcomes, but not sharply. When the number of types per agent increases, the runtime increases fairly sharply. Additionally, implementing dominant strategies is more difficult than BNE. Allowing for payments in mechanisms that maximize social welfare actually reduces runtime. It should be noted that at present, automating the design of mechanisms does not scale well for larger inputs. However, there are still many practical situations with small numbers of inputs(e.g. *thin auctions*).

Multi-Issue Automated Mechanism Design

While we do not go into depth, there is a notion of multi-issue automated mechanism design. Agents may have to make simultaneous decisions on multiple *issues*. The outcome space is then represented as a cross product of the outcomes spaces of each individual issue. While one may think that solving the multi-issue AMD problem can be solved by solving the problem for each issue, in general, this will not yield the optimal mechanism.

Conclusion

This overview is a limited exposure to automated mechanism design. This omits many important details such as the proofs of NP-completeness, empirical results of automatically designed mechanisms, many applications, searches over subsets of outcomes, and more. However, this overview captures the core of automated mechanism design, with the definition of an AMD setting and the definition of AMD itself. It explores the benefits of automated mechanism design over traditional mechanism design. It provides an example of an AMD setting, provides a linear programming approach to designing randomized mechanisms, and explores the practical limitations of automated mechanism design.