Computational-Mechanism Design: A Call to Arms

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Published Version</td>
<td>doi:10.1109/MIS.2003.1249168</td>
</tr>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:4101007">http://nrs.harvard.edu/urn-3:HUL.InstRepos:4101007</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
The need is increasing for computer systems that operate a decentralized control regime, are open (individual components can enter and leave at will), and contain several components representing distinct stakeholders with different aims and objectives. Relevant examples include Grid computing, the Semantic Web, pervasive computing, e-commerce, mobile computing, and peer-to-peer (P2P) systems.

For these complex systems, agent-based approaches, which emphasize autonomous actions and flexible interactions, are natural computational models. In designing such multiagent systems (MASs), you must address two fundamental issues. First, you must specify the protocols that govern the interactions. These protocols cover issues such as how the agents’ actions translate into an outcome, the range of actions available to the participants, and whether the interactions occur over steps or are one-shot. Second, given the prevailing protocol, you must define each agent’s strategy (mapping from state history to action). Sometimes, a designer might be able to impose each agent’s protocol and strategy. In such settings, the agents can cooperate to find a good systemwide solution. However, usually this isn’t feasible because the agents represent distinct stakeholders with potentially conflicting goals that seek to maximize their own gains. Consequently, the best a designer can achieve is a noncooperative strategic analysis, in which the designer can impose only the protocol and can’t control which strategies the agents adopt. Against this background, we examine the field of computational-mechanism design. CMD provides an elegant mathematical framework in which to study protocols that give the agents incentive to act and interact in particular ways and that have useful computational properties. It is an emerging field with which MAS designers need to become familiar. Moreover, the relatively new subfield of distributed-computational-mechanism design—which has no trusted center—is attracting considerable interest because it is a better match with the computational MAS model. However, this raises many additional open questions because the agents that must implement the mechanism’s rules are those that stand to benefit from its (mis)operation.

Designing MASs using mechanism design techniques

In contrast with competitive equilibrium theory, where agents respond solely to summary signals (such as prices for different outcomes) about the multiagent problem, we assume CMD agents act in a game-theoretic way, thereby modeling the effect their actions will have on other agents’ actions. This more detailed modeling facilitates the design of predictable systems of interacting agents and has caused MAS designers to start looking at game theory and, more recently, mechanism design. Specifically, MD deals with how to design systems so that certain systemwide properties (for example, efficiency, stability, and fairness) emerge in equilibrium from the constituent components’ interaction. MD is particularly appealing for designing MASs with self-interested agents because it provides methods for simplifying the strategic problems facing agents at design time.

In its native form, MD is a beautiful but brittle endeavor; it models agents as rational and then optimizes with respect to this model. However, the theory’s underlying assumptions can be inappropriate in computational settings because software agents are invariably bounded-rational. Also, the theory focuses on centralized mechanisms engineered so
that agents reveal their complete private information, and the design problem is itself often intractable. Furthermore, it’s assumed that communication between agents and the system is free and faultless and that the system is closed, with a static agent population. It’s also assumed that agents understand the protocols that govern their interactions and abide by them. These assumptions are problematic in MASs because

- Agents don’t have the unbounded computational power that might be required to calculate their preferences for all possible outcomes or calculate equilibrium strategies.
- The mechanism infrastructure in a centralized mechanism might be unable to compute the outcome because the problem might be intractable.
- Communication is not necessarily cost-free and could also be error-prone.
- Most real MASs are dynamic, and the set of agents varies with time owing to the system’s open nature.
- Semantic interoperation between different participants in an open system is difficult to attain, as is a machine-understandable specification of all the associated interaction protocol’s aspects.

CMD seeks to address these limitations and thereby apply MD techniques to computational problems. You might imagine that a new field isn’t required because we can decompose the problem of using MD in a MAS into its economic part (MD) and its computational part (MAS), then attack it in a modular fashion. However, this approach fails to recognize that we must address both economic and computational principles at each stage of design. In fact, often principles from one area can help solve a problem in the other. For example, you could make finding an undesirable equilibrium strategy (an economic problem) an intractable problem (a computational solution) for agents. Similarly, you could make optimal strategies tractable (a computational problem) by designing mechanisms with a simple truth-revealing equilibrium (an economic solution). However, we will not discuss the other side of the interplay between MD and computer systems—namely, using computing techniques to discover good mechanisms.

**MD theory**

MD considers a setting with a set of agents \( N \), each holding private information \( \theta_i \) about its preferences (\( \theta_i \) is also known as the type of the agent \( i \)). The type describes how each agent values all possible outcomes, and it is drawn from each agent’s available type set \( \Theta_i \). Agent \( i \) with type \( \theta_i \) has utility \( u_i(\theta_i, o) \) for outcome \( o \in \Omega \), where \( \Omega \) is the set of possible outcomes. We let the outcome define payments, in addition to decisions in the world, such as task or resource allocations. A mechanism is then a tuple \( M = (\Sigma, g) \) that comprises a strategy space (the set of possible actions) \( \Sigma \) and an outcome rule \( g(\sigma) \in \Omega \) for \( \sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N) \in \Sigma^N \). A strategy \( s_i(\theta_i) \in \Sigma \) defines the actions an agent selects in the mechanism for all possible types \( \theta_i \). The outcome rule takes the actions agents selected and implements a particular outcome. The mechanism, together with the agent types, defines a game. We assume that agents are autonomous and economically rational and that they select a best-response strategy to maximize their expected utility in equilibrium with other agents. This means no agent can benefit from any unilateral deviation. We assume the mechanism can commit to the outcome rule and implement an outcome.

**Mechanism desiderata**

The goal in traditional MD is to design a system in which rational agents interact in a way that leads to equilibriums with desired systemwide properties. These properties are encapsulated in the social choice function (SCF) \( f: \Theta \rightarrow \Omega \), which defines a desired outcome for each possible set of agent types. For example, we might wish to achieve efficiency in the system by maximizing the total utility gained across all agents, in which case

\[
\max_{\Theta \in \Theta^N} \sum_{i \in N} u_i(\theta_i, o) = \max_{\Theta \in \Theta^N} f(\Theta)
\]

You can imagine allocating computational resources to maximize agent utility or dividing tasks to minimize the system’s total cost. In short, the SCF describes the properties that the designer would like MAS outcomes to possess. Given this, the designer’s problem is to provide incentives so that agents choose to act in a way that implements a particular SCF. Here are some typical desired properties (desiderata) of SCFs:

- **Pareto optimality**: Implementing an outcome that is not Pareto-dominated by any other outcome, so no other outcomes make one agent better-off while making other agents worse-off.
- **Maximized social welfare**: Implementing an outcome that maximizes the total utility across agents. This is often called the efficient outcome.
- **Maximized utility to a particular agent**: Maximizing the expected utility to a single agent, typically the center, across all possible mechanisms. A common setting is a revenue-maximizing auction, in which the goal is to design a mechanism that maximizes the auctioneer’s revenue.
- **Budget balance**: The total payment that agents make equals exactly zero (a strict budget balance), so money isn’t injected into or removed from a system. Or, the total payment is nonnegative (a weak budget balance), so the mechanism doesn’t run at a loss. We can also consider an ex ante budget balance, in which the mechanism is balanced on average, and an ex post budget balance, in which the mechanism is balanced at all times, for all instances.
- **Individual rationality**: The SCF gives each agent nonnegative utility in equilibrium. We can consider interim individual rationality, in which an agent has nonnegative utility in expectation given its own type, and ex post individual rationality, in which an agent always has nonnegative utility.

Budget balance is especially important in systems that must be self-sustaining and require no external beneficiary to input money or central authority to collect payments. Yet, budget balance often conflicts with other desiderata, such as efficiency (we return to this later). Individual rationality is important when you can’t force agents to participate in a mechanism.

**Equilibrium solution concepts**

We would like a mechanism \( M = (\Sigma, g) \) to implement SCF \( f \) in equilibrium. We can use several different solution concepts in MD to predict the strategies agents will select. Each solution concept differs in assumptions about agents’ rationality and about the knowledge that agents have about the system’s other agents.

We present the three most useful solution concepts, each successively requiring stronger assumptions about agents (see the “Auctions Illustrating Equilibrium Solution Concepts” sidebar for example auctions).

**Dominant strategy**. Each agent has a best-response strategy no matter what strategy the
An example of a dominant-strategy implementation is the second-price (Vickrey) auction, where the auction clears for the second-highest price. In this case, the dominant strategy is for an agent to truthfully reveal its valuation for the item.

An example of an ex post Nash implementation is the English auction, an ascending-price auction in which the ask price is equal to the current winning bid. A straightforward bidding strategy is to bid at the ask price $p$ whenever $p \leq \nu_i$ for value $\nu_i$. This is an ex post Nash equilibrium—that is, as long as other agents are also straightforward, an agent can do no better, whatever the other agents’ values. However, straightforward bidding is not a dominant strategy equilibrium. Consider another agent that conditions a “crazy” strategy such as “I will bid to $1 million” if the price hits a particular target value. In this case, an agent should submit a jump bid past this target price to prevent this strategy from triggering. Preventing jump bids in the English auction makes straightforward bidding a dominant strategy.

An example of a Bayesian-Nash implementation is the first-price sealed-bid auction. Given a symmetric equilibrium of agent types with values that are identically and independently distributed, $\nu_i = U(0,1)$, the symmetric BNE is for agents to play $s^*_i(\nu_i) = (M - 1)\nu_i / |I|$.

All of these auctions implement the efficient allocation and are revenue-equivalent in equilibrium.

Reference

Centralized CMD, like traditional MD, involves certain computational considerations. We see that all equilibrium solutions in any particular mechanism are interesting in practice. What the revelation principle does provide is a normative effect on the system’s economic properties. The winner-determination problem without that buyer.) CMD challenges

Previously, we highlighted key issues concerned with moving from MD to CMD. Of these, the most important challenge relates to bounded computational power, which presents a problem at both the agent level (preference formulation and strategy selection) and the mechanism level (outcome determination). Let’s consider each challenge in turn.

Preference formulation and communication complexity. Naively, we can ask an agent to list all possible outcomes and provide values according to its type. In simple settings, such as voting for candidates in an election, this is easy and doesn’t require much computation. However, in more complex settings where the agent must decide between bundles of items or formulate its preferences according to multiple criteria, the problem becomes harder. (This occurs commonly when agents’ owners don’t specify preference relationships over all outcomes. Rather, they specify the criteria by which to rate these outcomes—for example, specifying preferences for make, color, and price range on cars rather than for all available cars.) The problem has two components. First, an agent must determine its values. Second, an agent must report its values.

One approach proposed to address the first component is to consider indirect mechanisms, such as iBundle, that let an agent
participate in a mechanism without computing or reporting its complete valuation function. Instead, /Bundle is an ascending-price combinatorial auction in which the agent must compute only its best-response bundle set given prices in each auction round. Another approach is more explicit and aims to formulate queries about agent valuations, bypassing the need for agents to formulate preferences for all outcomes.14

You can mitigate the second problem component by carefully designing bidding languages to let agents communicate their valuations compactly and expressively. This is important because, for example, a naïve flat bidding language in the CAP would require an agent to specify $2^{N-1}$ – 1 bids. However, in many cases a bidder might be interested only in a subset of the combinations and might also have a valuation function with useful structure.

Numerous languages, typically based on logical representations, exist for this purpose.15 Generally speaking, the literature compares the expressiveness and compactness across families of languages. In the context of indirect mechanisms, such as ascending-price auctions, we also need languages to let agents provide partial information about their preferences—for example, “my value for bundle $S_1$ is at least my value for bundle $S_2$.”12

**Strategy selection.** An agent must compute its equilibrium strategy given its information about the mechanism and the other agents in the mechanism. If the agent knows a priori that it has a dominant strategy, then the computational problem is insignificant. However, without this knowledge, the agent must compute an equilibrium strategy. Computing the Nash equilibrium in a game is notoriously for polynomially solvable cases only when the optimal outcomes require bidding that can’t be represented with the imposed structure. Alternatively, we could use the algorithms for polynomially solvable cases only when such a structure is detected in the placed bids.19 However, this structure doesn’t always occur.

The second approach is to use approximations. Because finding an optimal outcome is NP-hard, this approach attempts to find good outcomes that are close to the optimal but much easier to compute.17 The main challenge is to introduce approximations without breaking a mechanism’s incentive properties. For example, we can replace the winner determination algorithm in a VCG mechanism with an approximate one and retain strategyproofness, but only if the algorithm satisfies a property of maximal-in-range such that it makes the right decision across a fixed range of choices independent of the valuations that agents report.17 (Intuitively, if we’re not careful in setting the VCG mechanism, we leave incentives for agents to try to mend the approximations the mechanism introduces through complex game-theoretic reasoning about what other agents will report. This complicates strategy selection.) The literature shows a handful of positive results—for instance, for the case of single-minded bidders that desire at most one bundle20 and for the case of multiunit allocation problems for a slightly relaxed solution concept.21

The third approach is to use an indirect approach. Here, agents interactively compute a mechanism’s outcome. For example, they might provide incremental information about preferences or refine that information in response to aggregate signals (via prices, for example) from the mechanism. The basic idea is to share the computational burden between the agent and the mechanism. So, the key questions are

- What kind of strategy space should be designed for the agents if the agents can’t just reveal their types?
- How much of the burden should shift to the agents?
- Should we assume that all agents have the same computational resources?
- Can we trust the agents to carry out their share of the computational work?

For example, in /Bundle, agents following straightforward strategies will submit bids for bundles at the current auction prices.12 In submitting these bids, the agents are doing some computation (for example, determining bundles with maximal payoff at the current prices). They’re also helping the auctioneer by restricting its winner determination to finding the revenue-maximizing allocation solution across the current bids submitted by agents and not in terms of agent’s complete valuation functions.

**DMD**

So far, we’ve focused on systems with centers for collecting information from agents and determining and implementing an outcome, and a closed set of agents. We’ve also assumed in CMD that the agents have a direct, trusted method to communicate with the center, and that they all can report their actions to the center simultaneously. However, these assumptions don’t always hold in open distributed systems. For example, consider grid computing, in which a set of resources is distributed over wide-area net-
Using Mechanism Design to Promote Truth-Telling in P2P Networks

In current P2P networks such as Gnutella and KaZaa, “free riding”—agents using the system without providing resources toward it—is well-documented. In response, one approach has been to study the economics of tit-for-tat, where agents can only receive resources to the degree that they contribute them. However, this approach is blind to the heterogeneity of local agents that will likely differ in their computational resources and data content and quality. Simply stated, free-riding is not necessarily a bad outcome when performed by agents with little to contribute to the network. A mechanism design approach would instead seek to provide incentives for agents to truthfully report local resources (for example, files available for upload, bandwidth speed, and so on) and compute socially-efficient file-sharing solutions.

References

works that can support large-scale distributed applications. The Grid is predominantly concerned with coordinated resource sharing and problem solving in dynamic, multi-institutional, virtual organizations. As another example, consider P2P systems, which are similar to the Grid but typically have more users with more widely varying capabilities. Grid and P2P systems have distinct stakeholders and a strong need for effective resource allocation. A need clearly exists for economic design and analysis, but there’s no single central agent or controller such as those in classic mechanism design.

At one extreme, we could dispense with MD altogether and simply ask about the “price of anarchy” or the economic cost of just implementing distributed solutions with no carefully designed mechanism. In many cases, we believe that this cost will be too high. So, the challenge remains to design distributed mechanisms that retain normative MD goals (see the sidebar on using MD).

To this end, we explain how to apply DMD while still considering the pertinent problems that must be addressed. In DMD, we consider how to implement an SCF under the constraint that no central agent computes the outcome. On one hand, such a constraint might arise naturally owing to a system’s computational structure. On the other hand, such a distributed system has several advantages over centralized MD (which are vital if MD is to gain a central role in designing MASs with self-interested agents):

- **Robustness.** In a centralized mechanism, the communication channels linking to the center are critical for the system, and failure might incapacitate the entire system’s operation. However, if these channels fail in a decentralized setting, it won’t incapacitate the mechanism, although it could lead to a suboptimal solution.
- **Trustworthiness.** In a distributed mechanism, because no single agent computes the outcome, a higher degree of trust in the mechanism can exist once you address other incentive issues—why agents will choose to perform their roles appropriately, for example. (Trustworthiness in the center is an ever-present problem in a centralized mechanism, which we believe has influenced the limited use of Internet auctions. For example, in the VCG mechanism we discussed, the auctioneer might proclaim a higher-than-actual second price to increase its revenue.)
- **Bottleneck reduction.** In distributed mechanisms, communication must no longer pass through a single point.

In DMD, we imagine distributing a mechanism’s rules across the agents so that agents are asked, for example, to forward messages following a mechanism-specified protocol or to perform computation on the basis of messages received from other agents following a mechanism-specified protocol. Immediately, we see DMD’s intriguing new challenge: the same agents that seek to manipulate a system also run the mechanism. Why, then, wouldn’t an agent choose to deviate from implementing a mechanism’s proposed rules of the game? So, although trusting a single central agent isn’t necessary, we must now have distributed trust—we must trust each agent in a distributed system to implement its piece of the mechanism’s outcome.

A distributed Vickrey auction provides a canonical example. Suppose there’s a center but no trusted communication channel, and agents must forward a bid from their neighbor toward the center. The only effect that forwarding the bid from its neighbor can have on an agent is to increase competition for the item. In such circumstances, a rational agent would choose to simply drop bids from other agents. One proposed approach to this problem is using redundancy and checker nodes to validate decisions on the basis of revealed information, and partitioning so that either no agent computes outcomes that affect its own utility, or agents must perform computation only when it’s in their own interests to be truthful.

Another challenge in developing distributed mechanisms involves reducing the complexity of message passing in the communication network. Consider an interdomain routing problem on the Internet in which individual autonomous systems (or agents) must report their costs for forwarding messages to select the shortest paths between different domains. Incentives matter, but there’s no center, and a DMD approach is required to incite agents to reveal truthful information and support shortest-path selection. However, the system should be able to compute this efficient outcome without overburdening the network with messages just to find it. Although not too robust to agents that manipulate the mechanism’s implementation, a DMD-based solution is proposed for this problem that only modestly increases the routing-table size and convergence time.

Furthermore, the network topology might in itself affect the type of mechanism we can implement in these systems. For example, one study has considered the network com-
plexity of implementing multicast communication on tree-based networks. This might point to adopting a design methodology similar to that in traditional MD, which develops solutions and mechanisms for restricted domains.

DMD also highlights another issue in applying MD in MASs. MD concentrates almost exclusively on methods to handle rational and self-interested agents’ behavior. Yet, most realistic MASs contain various agent behaviors, so this extreme assumption might be too strong. For example, the agents might be obedient (follow the protocol and prescribed strategy), faulty (not function properly or be “irrational” owing to hardware or software failure), strategic (rational, as with the game-theoretic model) or adversarial (“irrational” owing to deviant behavior). A debate already exists about why open P2P systems such as KaZaa work at all given the incentives for users to free-ride. One suggestion is that naive users simply download and run the default client, in which file sharing is enabled. So, in reality we’ll need mechanisms based on assumptions about a mixture of agent behaviors, including obedient, strategic, and faulty agents. Traditional equilibrium solution concepts might be inappropriate in these systems with mixed agent behaviors.

Another related problem that arises with both CMD and DMD is that richer kinds of strategic behavior exist than those assumed in most MD theory. Most of MD focuses on the idea that a single agent might unilaterally seek to manipulate a mechanism’s outcome. In reality, we might expect agents to collude and form groups that act as one “superagent” to benefit from the reduction in competition and greater power the group wields. However, this behavior might harm the system’s overall well-being. (Consider again the Vickrey auction. If the agents having the highest [Agent 1] and second-highest [Agent 2] valuations collude, then these two agents can benefit. For example Agent 1 might offer Agent 2 half the difference between the second-highest and third-highest valuation.) Collusion problems can be more problematic in electronic settings because people may unleash several agents and thus easily adopt multiple identities. Also, they might be bidding across mechanisms, making collusive behavior harder to detect.

Conclusions and general challenges
In addition to the challenges noted earlier, a number of more general issues cut across both CMD and DMD and still require further investigation.

Online mechanisms
Many real MASs are dynamic. Rather than taking a decision for a system of agents once, some mechanisms must take a sequence of decisions and maintain a state (for example, a resource allocation) dynamically over time as agents arrive and leave. For example, consider the problem of Wi-Fi bandwidth allocation in an airport lounge. Clients arrive and leave over time and have heterogeneous valuations for different Wi-Fi connection allocations. An online mechanism is a mechanism that makes a sequence of decisions as the agent types are dynamically revealed as they arrive in the system. The challenge in online MD is to provide appropriate incentive properties so that it’s an equilibrium strategy for an agent to both report its true valuation function for different outcomes and announce its arrival as soon as it enters a system.

Adaptive mechanisms
A second challenge is adaptive-mechanism design, in which the mechanism’s rules change over time. Consider the same Wi-Fi bandwidth allocation problem, but where the mechanism doesn’t start with a good model of the agents’ arrival dynamics or the distribution across agent types. The problem has two components. First, the mechanism must have appropriate incentive properties at any particular instance, even while learning is still taking place and the rules are still sub-optimal with respect to some desiderata. Second, the mechanism’s dynamics must be robust to withstand manipulation from an agent that wishes to misstate its current type to benefit from this action’s dynamic effect on the mechanism’s rules later on.

Money
The presence of money, a common denominator by which every good can be valued, is important in traditional MD. However, in many MASs, such a common denominator doesn’t exist naturally and must often be constructed.

Machine-understandable specifications
Agents act on behalf of their owners. To be effective, owners must communicate their preferences to agents seamlessly. So, the agent’s type should basically mirror its owner’s.

Semantic interoperation
Often, numerous institutions provide the same service that an agent requires. So, the agents must be able to operate in all such institutions simultaneously. This should lead to more efficient outcomes because, in this case, the agents will maximize their utility over the whole space of electronic institutions. However, the ability for agents to effectively understand and communicate in an arbitrary institution is a major challenge facing computer scientists.

Regulation and binding contracts
To prevent a mechanism’s incentive properties from unraveling, the center must be able to credibly commit to the game’s rules. Similarly, an agent’s adherence to the rules laid down by a mechanism can be critical to system functionality. Commitment and trust in the real world are typically built on an intricate set of procedural and legal requirements. This is more difficult in virtual worlds, where legal aspects are less well understood or simply undefined. In MAS systems, additional problems might exist, owing to multiple identities, for example, or because validating the existence (or lack) of particular behaviors can be difficult.

Computational-mechanism design has an important role to play in developing complex distributed systems comprising multiple interacting agents. It offers a powerful suite of tools for analyzing, predicting, and controlling the behavior of self-interested agents. However, several fundamental scientific questions must still be addressed if this nascent field is to reach its full potential and become a significant paradigm for developing such systems.

Acknowledgments
Rajdeep K. Dash receives funding from a BAE Systems studentship and an Overseas Research Scholarship. David Parkes gratefully acknowledges support from National Science Foundation grant IIS-0238147.

References
2. N.R. Jennings and S. Bussmann, “Agent-Based


