Encyclopedia of Algorithms
DOI 10.1007/978-3-642-27848-8_787-1

Reducing Bayesian Mechanism Design to Algorithm Design

Yang Cai*[a], Constantinos Daskalakis[b] and Matthew Weinberg[c]
[a]Computer Science, McGill University, Montreal, QC, Canada
[b]EECS, Massachusetts Institute of Technology, Cambridge, MA, USA
[c]Computer Science, Princeton University, Princeton, NJ, USA

Keywords Mechanism design • Job scheduling • Fair allocation • Revenue maximization • Equivalence of separation and optimization

Years and Authors of Summarized Original Work

STOC2012; Cai, Daskalakis, Weinberg
FOCS2012; Cai, Daskalakis, Weinberg
SODA2013; Cai, Daskalakis, Weinberg
FOCS2013; Cai, Daskalakis, Weinberg
SODA2015; Daskalakis, Weinberg

Problem Definition

The goal is to design algorithms that succeed in models where input is reported by strategic agents (henceforth referred to as *strategic input*), as opposed to standard models where the input is directly given (henceforth referred to as *honest input*). For example, consider a resource allocation problem where a single user has m jobs to process on n self-interested machines. Each machine i can process job j in time t_{ij}, and this is privately known only to the machine. Each machine reports some processing times $\hat{t}_{ij}$ to the user, who then runs some algorithm to determine where to process the jobs. Good approximation algorithms are known when machines are honest (i.e., $\hat{t}_{ij} = t_{ij}$ for all i, j) if the user's goal is to minimize the *makespan*, the time elapsed until all jobs are completed, going back to seminal work of Lenstra, Shmoys, and Tardos [13]. However, such algorithms do not account for the strategic nature of the machines, which may want to minimize their own work: why would they report honestly their processing time for each job if they can elicit a more favorable schedule by lying? To accommodate such challenges, new algorithmic tools must be developed that draw inspiration from Game Theory.

Requiring solutions that are robust against potential strategic manipulation potentially increases the computational difficulty of whatever problem is at hand. The discussed works provide a framework with which to design such solutions (henceforth called *mechanisms*) and address the following important question.

Question 1. How much (computationally) more difficult is mechanism design than algorithm design?

*E-mail: cai@cs.mcgill.ca

DOI 10.1007/978-3-642-27848-8_787-1

Using this framework, we resolve this question with an answer of "not at all" for several important problems including job scheduling and fair allocation. Another application of our framework provides efficient algorithms and structural characterization results for multi-item revenue-optimal auction design, a central open problem in mathematical economics.

Model

Environment

1. Set $\mathcal{F}$ of feasible outcomes. Interpret $\mathcal{F}$ as the set of all (feasible) allocations of jobs to machines, allocations of items to bidders, etc.
2. n agents who all care about which outcome is chosen.

Strategic Agents

1. Each agent i has a value $t_i(x)$ for each outcome $x \in \mathcal{F}$. t_i induces a function from $\mathcal{F} \to \mathbb{R}$ and is called the agent's *type*.
2. Each t_i is drawn *independently* from some distribution $\mathcal{D}_i$ of finite support.
3. Agent i knows t_i; all other agents and the designer know only $\mathcal{D}_i$.
4. Agents are *quasi-linear* and *risk neutral*. That is, the utility of an agent of type t for a randomized outcome (distribution over outcomes) $X \in \Delta(\mathcal{F})$, when he is charged price p, is $\mathbb{E}_{x \leftarrow X}[t(x)] - p$.
5. Agents behave in a way that maximizes utility, taking into consideration beliefs about the behavior of other agents.

Designer

1. Designs an *allocation rule* A and *price rule* P. A takes as input a type profile $(t_1, \ldots, t_n)$ and outputs (possibly randomly) an outcome $A(\mathbf{t}) \in \mathcal{F}$. P takes as input a type profile and outputs (possibly randomly) a price vector $P(\mathbf{t})$. The pair (A, P) is called a (direct) *mechanism*. Note that it is without loss of generality to consider only the design of direct mechanisms by the revelation principle [14].
2. Announces A and P to agents. Invites agents to report a type. When $\mathbf{t}$ is reported, selects the outcome $A(\mathbf{t})$ and charges agent i price $P_i(\mathbf{t})$.
3. Has some objective function $\mathcal{O}$ to optimize. $\mathcal{O}$ may depend on the agents' types, the outcome selected, and the prices charged, so we write $\mathcal{O}(\mathbf{t}, x, \mathbf{P})$. Examples include:

   - Social welfare: $\mathcal{O}(\mathbf{t}, x, \mathbf{P}) = \sum_i t_i(x)$.
   - Revenue: $\mathcal{O}(\mathbf{t}, x, \mathbf{P}) = \sum_i P_i(\mathbf{t})$.
   - Makespan: $\mathcal{O}(\mathbf{t}, x, \mathbf{P}) = \max_i\{-t_i(x)\}$ (In job scheduling, agents' values from allocations are nonpositive, since they have cost for processing jobs. An agent's cost for allocation x is then $-t_i(x)$.).
   - Fairness: $\mathcal{O}(\mathbf{t}, x, \mathbf{P}) = \min_i\{t_i(x)\}$.

DOI 10.1007/978-3-642-27848-8_787-1

Game Theoretic Definitions

1. The *interim rule* of a mechanism is a function that takes as input an agent i and type t_i and outputs the distribution of allocations and prices that agent i sees when reporting type t_i over the randomness of the mechanism and the other agents' types, assuming they tell the truth. So the interim allocation rule $(\boldsymbol{\pi}, \mathbf{p})$ of the mechanism (A, P) satisfies:

$$\Pr[x \leftarrow \pi_i(t_i)] = \mathbb{E}_{\mathbf{t}_{-i} \leftarrow \mathcal{D}_{-i}} \left[\Pr[A(t_i; \mathbf{t}_{-i}) = x]\right].$$

$$\Pr[p \leftarrow p_i(t_i)] = \mathbb{E}_{\mathbf{t}_{-i} \leftarrow \mathcal{D}_{-i}} \left[\Pr[P_i(t_i; \mathbf{t}_{-i}) = p]\right].$$

2. A mechanism is *Bayesian Incentive Compatible (BIC)* if every agent receives at least as much utility by reporting their true type as any other type (assuming other agents report truthfully). Formally, $t_i(\pi_i(t_i)) - p_i(t_i) \geq t_i(\pi_i(t_i')) - p_i(t_i')$ for all i, t_i, t_i' (We use the shorthand $t_i(\pi_i(t_i'))$ to denote the expected value of t_i for the random allocation drawn from $\pi_i(t_i')$. Formally, $t_i(\pi_i(t_i')) = \mathbb{E}_{x \leftarrow \pi_i(t_i')}[t_i(x)]$.). A commonly used relaxation of BIC is called *ϵ-Bayesian Incentive Compatible (ϵ-BIC)*. A mechanism is ϵ-BIC if every agent derives at most ϵ less utility by reporting their true type comparing to any other type (assuming other agents report truthfully). Formally, $t_i(\pi_i(t_i)) - p_i(t_i) \geq t_i(\pi_i(t_i')) - p_i(t_i') - \epsilon$ for all i, t_i, t_i'.
3. A mechanism is *individually rational (IR)* if every agent has nonnegative expected utility by participating in the mechanism (assuming other agents report truthfully). Formally, $t_i(\pi_i(t_i)) - p_i(t_i) \geq 0$ for all i, t_i.

Bayesian Mechanism Design (BMeD)

Here we describe formally the mechanism design problem we study. BMeD is parameterized by a set of feasible outcomes $\mathcal{F}$, objective function $\mathcal{O}$, and set of possible types $\mathcal{V}$. Both $\mathcal{V}$ and $\mathcal{F}$ can be discrete or continuous. We assume that every element $v \in \mathcal{V}$ and $x \in \mathcal{F}$ can be represented by a finite bit string $\langle v \rangle$ and $\langle x \rangle$. $\mathcal{V}$ and $\mathcal{F}$ also specify how those bit strings are interpreted. For instance, $\mathcal{V}$ might be the class of all submodular functions, and the bit strings used to represent them may be interpreted as indexing a black-box value oracle. Or $\mathcal{V}$ might be the class of all subadditive functions, and the bit strings used to represent them may be interpreted as an explicit circuit. Or $\mathcal{V}$ could be the class of all additive functions, and the bit strings used to represent them may be interpreted as a vector containing values for each item. So we are parameterizing our problems both by the actual classes $\mathcal{V}$ and $\mathcal{F}$ but also by how elements of these classes are represented. Now, we are ready to formally discuss the problem BMeD$(\mathcal{F}, \mathcal{V}, \mathcal{O})$.

BMeD$(\mathcal{F}, \mathcal{V}, \mathcal{O})$:

INPUT*: For each agent $i \in [n]$, a discrete distribution $\mathcal{D}_i$ over types in $\mathcal{V}$, described* explicitly *by listing the support of $\mathcal{D}_i$ and the corresponding probabilities.*

OUTPUT*: A BIC, IR mechanism.*

GOAL*: Find the mechanism that optimizes $\mathcal{O}$ in expectation, with respect to all BIC, IR mechanisms (when n bidders with types drawn from $\times_i \mathcal{D}_i$ report truthfully).*

APPROXIMATION*: A mechanism is said to be an (ϵ, α)-approximation to BMeD if it outputs an ϵ-BIC mechanism whose expected value of $\mathcal{O}$ (when n bidders with types drawn from $\times_i \mathcal{D}_i$ report truthfully) is at least $\alpha\mathrm{OPT} - \epsilon$ (or at most $\alpha\mathrm{OPT} + \epsilon$ for minimization problems).*

Encyclopedia of Algorithms
DOI 10.1007/978-3-642-27848-8_787-1

Generalized Objective Optimization Problem (GOOP)

Here we describe formally the algorithmic problem we show has strong connections to BMeD. GOOP is parameterized by a set of feasible outcomes $\mathcal{F}$, objective function $\mathcal{O}$, and set of possible types $\mathcal{V}$. We therefore formally discuss the problem GOOP$(\mathcal{F}, \mathcal{V}, \mathcal{O})$. Below, $\mathcal{V}^{\times}$ denotes the closure of $\mathcal{V}$ under linear combinations. Functions in $\mathcal{V}^{\times}$ are represented by a finite list of elements of $\mathcal{V}$, along with (possibly negative) scalar multipliers.

GOOP$(\mathcal{F}, \mathcal{V}, \mathcal{O})$:

INPUT*: For each agent* $i \in [n]$*, a type* $g_i \in \mathcal{V}$*, multiplier* $m_i \in \mathbb{R}$*, and cost function* $f_i \in \mathcal{V}^{\times}$*. Additionally, an indicator bit* b *(The indicator bit* b *is included so that the optimization of just* $\sum_i f_i(x)$ *(without price multipliers or* $\mathcal{O}$*) is formally a special case of* GOOP$(\mathcal{F}, \mathcal{V}, \mathcal{O})$*.).*
OUTPUT*: An allocation* $x \in \mathcal{F}$*, and price vector* $\mathbf{p} \in \mathbb{R}^n$*.*
GOAL*: Find* $\arg\max_{x \in \mathcal{F}, \mathbf{p}} \{b \cdot \mathcal{O}(\mathbf{g}, x, \mathbf{p}) + \sum_i m_i p_i + \sum_i f_i(x)\}$ *(or* $\arg\min$*, if* $\mathcal{O}$ *is a minimization objective like makespan).*
APPROXIMATION*:* $(x, \mathbf{p})$ *is said to be an* (α, β)*-approximation to GOOP if* $\beta \cdot b \cdot \mathcal{O}(\mathbf{g}, x, \mathbf{p}) + \sum_i m_i p_i + \sum_i f_i(x)$ *is at least/most* $\alpha \cdot OPT$*. Note that a* $(\alpha, 1)$*-approximation is the standard notion of an* α*-approximation. Allowing* $\beta \neq 1$ *boosts/discounts the value of* $\mathcal{O}$ *(the objective) before comparing to* $\alpha \cdot OPT$*. Note also that allowing* $\beta \neq 1$ *provides no benefit if* $b = 0$*.*

Key Results

We provide a poly-time black-box reduction from BMeD$(\mathcal{F}, \mathcal{V}, \mathcal{O})$ to GOOP$(\mathcal{F}, \mathcal{V}, \mathcal{O})$. That is, we provide a reduction from Bayesian mechanism design to traditional algorithm design.

Theorem 1. *Let* G *be an* (α, β)*-approximation algorithm for* GOOP$(\mathcal{F},\mathcal{V},\mathcal{O})$*. Then for all* $\epsilon > 0$*, there is an* $(\epsilon, \alpha/\beta)$*-approximation algorithm for* BMeD$(\mathcal{F},\mathcal{V},\mathcal{O})$*. If* ℓ *is the length of the input to a* BMeD$(\mathcal{F},\mathcal{V},\mathcal{O})$ *instance, the algorithm succeeds with probability* $1 - \exp(-\text{poly}(\ell, 1/\epsilon))$*, makes* $\text{poly}(\ell, 1/\epsilon)$ *black-box calls to* G *on inputs of size* $\text{poly}(\ell, 1/\epsilon)$*, and terminates in time* $\text{poly}(\ell, 1/\epsilon)$ *(times the running time of each oracle call to* G*).*

This reduction is developed in a recent series of papers by the authors [4–7, 9]. The possibility of failure and additive error is due to a sampling procedure in the reduction. In addition to the computational aspect provided in Theorem 1, our reduction also has a structural aspect. Namely, we provide a characterization of the optimal mechanism in Bayesian settings.

Theorem 2. *For all objectives* $\mathcal{O}$*, feasibility constraints* $\mathcal{F}$*, set of possible types* $\mathcal{V}$*, and inputs* $\mathcal{D}$ *to* BMeD$(\mathcal{F}, \mathcal{V}, \mathcal{O})$*, the optimal mechanism is a distribution over generalized objective maximizers. Formally, there exists a joint distribution* Δ *over an indicator bit* b^{δ} *and mappings* $(f_1^{\delta}, \ldots, f_n^{\delta})$*, where each* f_i^{δ} *maps types* t_i *to multipliers* $m_i^{\delta}(t_i) \in \mathbb{R}$ *and cost functions* $\phi_i^{\delta}(t_i) \in \mathcal{V}^{\times}$*, such that the optimal mechanism first samples* $(b^{\delta}, \mathbf{f}^{\delta})$ *from* Δ *then maps the type profile* $\mathbf{t}$ *to the allocation and price vector* $(x(\mathbf{t}), \mathbf{p}(\mathbf{t})) = \arg\max_{x \in \mathcal{F}, \mathbf{p}} \{b^{\delta} \cdot \mathcal{O}(\mathbf{t}, x, \mathbf{p}) + \sum_i m_i^{\delta}(t_i) p_i + \sum_i \phi_i^{\delta}(t_i)(x)\}$.

Perhaps the most interesting case of Theorem 2 is when the objective is revenue. In this case, we may interpret the cost functions $\phi_i^{\delta} \in \mathcal{V}^{\times}$ as the *virtual valuation function* of bidder i. By virtual

DOI 10.1007/978-3-642-27848-8_787-1

valuations, we do *not* mean Myerson's specific virtual valuation functions [14], which aren't even defined for multi-item instances. Instead we simply mean *some* virtual valuation functions that may or may not be the same as the types/valuations reported by the agents. We include this and other applications of Theorems 1 and 2 below.

Applications

In this section, we apply Theorem 1 to the objectives of revenue, makespan, and fairness.

Revenue Maximization: We apply Theorem 1 to reduce the BMeD problem of optimizing revenue in multi-item settings to GOOP. In [7], it is shown that for this case, one need only consider instances of GOOP with $b = m_1 = \ldots = m_n = 0$, so the GOOP instances that must be solved require just optimization of the cost function (which we call *virtual welfare* for this application). We obtain the following computational and structural results on optimal auction design in general multi-item settings, addressing a long-standing open question following Myerson's seminal work on single-item auctions [14].

Theorem 3 (Revenue Maximization, Computational). *Let G be an α-approximation algorithm for maximizing virtual welfare over $\mathcal{F}$ when all virtual types are from $\mathcal{V}^{\times}$. Then for all $\epsilon > 0$, there is an (ϵ, α)-approximation algorithm for the problem* $\text{BMeD}(\mathcal{F}, \mathcal{V}, \text{REVENUE})$ *that makes polynomially many black-box calls to G. If ℓ is the length of the input to a* $\text{BMeD}(\mathcal{F}, \mathcal{V}, \text{REVENUE})$ *instance, the algorithm succeeds with probability* $1 - \exp(-\text{poly}(\ell, 1/\epsilon))$*, makes* $\text{poly}(\ell, 1/\epsilon)$ *black-box calls to G on inputs of size* $\text{poly}(\ell, 1/\epsilon)$*, and terminates in time* $\text{poly}(\ell, 1/\epsilon)$ *(times the running time of each oracle call to G).*

Theorem 4 (Revenue Maximization, Structural). *In any multi-item setting with arbitrary feasibility constraints and possible agent types, the allocation rule of the revenue-optimal auction is a distribution over virtual welfare maximizers. Formally, there exists a distribution Δ over mappings $(\phi_1, \ldots, \phi_n)$, where each ϕ_i maps types t_i to cost functions $f_i \in \mathcal{V}^{\times}$, such that the allocation rule for the optimal mechanism first samples $\boldsymbol{\phi}$ from Δ then maps type profile $\mathbf{t}$ to the allocation* $\arg\max_{x \in \mathcal{F}} \{\sum_i \phi_i(t_i)(x)\}$.

We further consider the following important special case: There are m items for sale to n buyers. Any allocation of items to buyers is feasible (that is, each item can be awarded to at most one buyer), so we can denote the set of feasible allocations as $\mathcal{F} = [n+1]^m$. Furthermore, each buyer i has a value v_{ij} for item j and is *additive* across items, meaning that their value for a set S of items is $\sum_{j \in S} v_{ij}$. So we can denote the set of possible types as $\mathbb{R}_+^m$ (and have types represented as such).

Theorem 5 (Revenue Maximization for Additive Buyers, Computational). *There is a poly-time algorithm for* $\text{GOOP}([n+1]^m, \mathbb{R}_+^m, \text{REVENUE})$*. Therefore, there is a poly-time algorithm for* $\text{BMeD}([n+1]^m, \mathbb{R}_+^m, \text{REVENUE})$ *(In this special case, no sampling is required in the reduction, so the theorem holds even for $\epsilon = 0$. Formally, this is a $(0, 1)$-approximation (an exact algorithm). See [4] for details.).*

DOI 10.1007/978-3-642-27848-8_787-1

Theorem 6 (Revenue Maximization for Additive Buyers, Structural). *In any multi-item setting with n additive buyers and m items for sale, the allocation rule of the revenue-optimal auction is a distribution over virtual welfare maximizers. Formally, there exists a distribution Δ over mappings $(\phi_1, \ldots, \phi_n)$, where each ϕ_i maps types t_i to cost functions $f_i \in \mathbb{R}^m$, such that the allocation rule for the optimal mechanism first samples $\boldsymbol{\phi}$ from Δ then awards every item j to a buyer in* $\arg\max_i \{\phi_{ij}(\mathbf{v}_i)\}$ *if their virtual value for item j is nonnegative and does not allocate the item otherwise.*

Job Scheduling on Unrelated Machines: The problem of job scheduling on unrelated machines consists of m jobs and n machines, with machine i able to process job j in time t_{ij}. The goal is to find a schedule (that assigns each job to exactly one machine) minimizing the *makespan*. Specifically, if S_i are the jobs assigned to machine i, the makespan is $\max_i \{\sum_{j \in S_i} t_{ij}\}$. As a mechanism design problem, one considers the machines to be strategic agents who know their processing time for each job (but the designer and other machines do not). In the language of BMeD, we can denote the feasibility constraints as $[n]^m$, the set of possible types as $\mathbb{R}_+^m$, and the objective as MAKESPAN. Theorem 1 reduces BMeD$([n]^m, \mathbb{R}_+^m,$ MAKESPAN$)$ to GOOP$([n]^m, \mathbb{R}_+^m,$ MAKESPAN$)$. It is shown in [7] that for objectives that don't depend on the prices charged at all (called "allocation-only"), only instances of GOOP with $m_i = 0 \ \forall i$ need be considered. It is further shown in [9] that GOOP$([n]^m, \mathbb{R}_+^m,$ MAKESPAN$)$ can be interpreted as a job scheduling problem with costs. Specifically, GOOP$([n]^m, \mathbb{R}_+^m,$ MAKESPAN$)$ takes as input a processing time $t_{ij} \geq 0$, and monetary cost $c_{ij} \in \mathbb{R}$ for all machines i and jobs j. The goal is to find a schedule that minimizes the makespan plus cost. Formally, partition the jobs into disjoint sets S_i to minimize $\max_i \{\sum_{j \in S_i} t_{ij}\} + \sum_i \sum_j c_{ij}$. While it is NP-hard to approximate GOOP$([n]^m, \mathbb{R}_+^m,$ MAKESPAN$)$ within any finite factor, a result of Shmoys and Tardos from the early 1990s obtains a polynomial time $(1, 1/2)$-approximation algorithm [15]. In combination with Theorem 1, this yields the following theorem:

Theorem 7 (Job Scheduling on Unrelated Machines). *For all $\epsilon > 0$, there is a poly-time $(\epsilon, 2)$-approximation algorithm for* BMeD$([n]^m, \mathbb{R}_+^m,$ MAKESPAN*). If ℓ is the length of the input to a* BMeD$([n]^m, \mathbb{R}_+^m,$MAKESPAN*) instance, the algorithm succeeds with probability* $1 - \exp(-\text{poly}(\ell, 1/\epsilon))$ *and terminates in time* $\text{poly}(\ell, 1/\epsilon)$.

Fair Allocation of Indivisible Goods: The problem of fairly allocating indivisible goods consists of m indivisible goods and n children, with child i receiving value v_{ij} for good j. The goal is to find an allocation of goods (that assigns each good to at most one child) maximizing the *fairness*. Specifically, if S_i are the goods allocated to child i, the fairness is $\min_i \{\sum_{j \in S_i} v_{ij}\}$. As a mechanism design problem, one considers the children to be strategic agents who know their own value for each good (but the designer and other children do not). In the language of BMeD, we can denote the feasibility constraints as $[n+1]^m$, the set of possible types as $\mathbb{R}_+^m$, and the objective as FAIRNESS. Theorem 1 reduces BMeD$([n+1]^m, \mathbb{R}_+^m,$ FAIRNESS$)$ to GOOP$([n+1]^m, \mathbb{R}_+^m,$ FAIRNESS$)$, which can be interpreted as a fair allocation problem with costs (again, because FAIRNESS is allocation only) [7,9]. Specifically, GOOP$([n+1]^m, \mathbb{R}_+^m,$ FAIRNESS$)$ takes as input a value $v_{ij} \geq 0$ and monetary cost $c_{ij} \in \mathbb{R}$ for all children i and goods j. The goal is to find an allocation that maximizes the fairness minus cost. Formally, allocate the goods into disjoint sets S_i to maximize $\min_i \{\sum_{j \in S_i} v_{ij}\} - \sum_i \sum_j c_{ij}$. While it is NP-hard to approximate GOOP$([n+1]^m, \mathbb{R}_+^m,$ FAIRNESS$)$ within any finite factor, we develop poly-time $(1, m-n+1)$-

DOI 10.1007/978-3-642-27848-8_787-1

and $(1/2, \tilde{O}(\sqrt{n}))$-approximation algorithms for fair allocation with costs, based on algorithms of Bezáková and Dani [2] and Asadpour and Saberi [1] for fair allocation (without costs).

Theorem 8 (Fair Allocation of Indivisible Goods). *There are poly-time* $(1, m - n + 1)$- *and* $(1/2, \tilde{O}(\sqrt{n}))$*-approximation algorithms for* GOOP$([n + 1]^m, \mathbb{R}_+^m,$ FAIRNESS*). Therefore, for all* $\epsilon > 0$*, there is a* $(\epsilon, \min\{\tilde{O}(\sqrt{n}), m - n + 1\})$*-approximation algorithm for* BMeD$([n + 1]^m, \mathbb{R}_+^m,$ FAIRNESS*). If* ℓ *is the length of the input to a* BMeD$([n+1]^m, \mathbb{R}_+^m,$ FAIRNESS*) instance, the algorithm succeeds with probability* $1 - \exp(-\text{poly}(\ell, 1/\epsilon))$ *and terminates in time* $\text{poly}(\ell, 1/\epsilon)$.

Tools for Convex Optimization

We prove Theorems 1 and 2 by solving a linear program over the space of possible interim allocation rules and generalizations of interim allocation rules that we do not discuss here. In doing so, we also develop new tools applicable for general convex optimization that we discuss here. We omit full details of the approach and refer the reader to a series of papers by the authors [5–7, 9] for specifics of the linear program solved and why it addresses BMeD. Seminal works of Khachiyan [12], Grötschel, Lovász, and Schrijver [10], and Karp and Papadimitriou [11] study the problems of optimization and separation over a close, convex region $P \subseteq \mathbb{R}^d$ (Below, we denote by $\alpha P = \{\alpha \mathbf{x} | \mathbf{x} \in P\}$. Also, for simplicity of exposition, we only consider P that contain the origin, so that $\alpha P \subseteq P$ for all $\alpha \leq 1$, but our results extend to all closed, convex P. See [9] for our most general results.). Formally, these problems are:

Optimize(P):

INPUT*: A direction* $\mathbf{c} \in \mathbb{R}^d$.
OUTPUT*: A point* $\mathbf{x} \in P$.
GOAL*: Find* $\mathbf{x}^* \in \arg\max_{\mathbf{x} \in P}\{\mathbf{c} \cdot \mathbf{x}\}$.

Separate(P):

INPUT*: A point* $\mathbf{x} \in \mathbb{R}^d$.
OUTPUT*: "Yes," or a direction* $\mathbf{c} \in \mathbb{R}^d$.
GOAL*: If* $\mathbf{x} \in P$*, output "yes." Otherwise, output any* $\mathbf{c}$ *such that* $\mathbf{c} \cdot \mathbf{x} > \max_{\mathbf{y} \in P}\{\mathbf{c} \cdot \mathbf{y}\}$.

Khachiyan's Ellipsoid algorithm shows that if one can solve the problem Separate(P) in time poly(d), then one can also solve Optimize(P) in time poly(d). Grötschel, Lovász, and Schrijver and independently Karp and Papadimitriou show that the other direction holds as well: if one can solve Optimize(P) in time poly(d), then one can also solve Separate(P) in time poly(d). This is colloquially called "the equivalence of separation and optimization." While separation as a means for optimization has obvious uses, optimization as a means for separation is more subtle. Still, numerous applications exist (including our results) and we refer the reader to [10, 11] for several others, including the first poly-time algorithm for submodular minimization.

In order to provide our guarantees with respect to approximation, we develop further the equivalence of separation and optimization to accommodate approximation. Specifically, consider the following problems, further parameterized by some $\alpha < 1$:

α-Optimize(P):

INPUT*: A direction* $\mathbf{c} \in \mathbb{R}^d$.

DOI 10.1007/978-3-642-27848-8_787-1

OUTPUT: *A point* $\mathbf{x} \in P$.
GOAL: *Find* $\mathbf{x}$ *satisfying* $\mathbf{c} \cdot \mathbf{x} \geq \alpha \max_{\mathbf{y} \in P}\{\mathbf{c} \cdot \mathbf{y}\}$.

α-Separate(P):

INPUT: *A point* $\mathbf{x} \in \mathbb{R}^d$.
OUTPUT: *"Yes" and a proof that* $\mathbf{x} \in P$*, or a direction* $\mathbf{c} \in \mathbb{R}^d$ *(For formal details on exactly what constitutes a proof, we refer the reader to [6, 7, 9]. Roughly speaking,* $\mathbf{x}$ *is written as a convex combination of points* known *to be in* P*.).*
GOAL: *If* $\mathbf{x} \in \alpha P$*, output "yes" and a proof that* $\mathbf{x} \in P$*. If* $\mathbf{x} \notin P$*, output a direction* $\mathbf{c}$ *such that* $\mathbf{c} \cdot \mathbf{x} > \alpha \max_{\mathbf{y} \in P}\{\mathbf{c} \cdot \mathbf{y}\}$*. If* $\mathbf{x} \in P \backslash \alpha P$*, either is acceptable.*

Theorem 9 (Approximate Equivalence of Separation and Optimization). *For all* $\alpha \leq 1$*, the problems* α-Optimize(P) *and* α-Separate(P) *are computationally equivalent. That is, if one can solve one in time* poly(d)*, one can solve the other in time* poly(d) *as well.*

We also extend these results to accommodate bi-criterion approximation, via the problems below, further parameterized by some $\beta > 1$ and subset $S \subseteq [d]$ of coordinates (Below, when we write $(\beta \mathbf{x}_S, \mathbf{x}_{-S})$, we mean to take $\mathbf{x}$ and multiply each $x_i, i \in S$ by β.).
(α, β, S)-Optimize(P):

INPUT: *A direction* $\mathbf{c} \in \mathbb{R}^d$.
OUTPUT: *A point* $\mathbf{x} \in P$.
GOAL: *Find* $\mathbf{x}$ *satisfying* $\mathbf{c} \cdot (\beta \mathbf{x}_S, \mathbf{x}_{-S}) \geq \alpha \max_{\mathbf{y} \in P}\{\mathbf{c} \cdot \mathbf{y}\}$.

(α, β, S)-Separate(P):

INPUT: *A point* $\mathbf{x} \in \mathbb{R}^d$.
OUTPUT: *"Yes" and a proof that* $\mathbf{x} \in P$*, or a direction* $\mathbf{c} \in \mathbb{R}^d$.
GOAL: *If* $(\beta \mathbf{x}_S, \mathbf{x}_{-S}) \in \alpha P$*, output "yes" and a proof that* $\mathbf{x} \in P$*. If* $\mathbf{x} \notin P$*, output a direction* $\mathbf{c}$ *such that* $\mathbf{c} \cdot (\beta \mathbf{x}_S, \mathbf{x}_{-S}) > \alpha \max_{\mathbf{y} \in P}\{\mathbf{c} \cdot \mathbf{y}\}$*. If* $(\beta \mathbf{x}_S, \mathbf{x}_{-S}) \notin \alpha P$ *and* $\mathbf{x} \in P$*, either is acceptable (An astute reader might worry that for some* α, β, S, P*, the problem* (α, β, S)*-Separate*(P) *is impossible, due to the existence of an* $\mathbf{x} \notin P$ *such that* $(\beta \mathbf{x}_S, \mathbf{x}_{-S}) \in \alpha P$*. For some* α, β, S, P*, this is indeed the case, but we show that* (α, β, S)*-Optimize*(P) *is impossible in these cases as well.).*

Theorem 10 (Bi-Criterion Approximate Equivalence of Separation and Optimization). *For all* $\alpha \leq 1, \beta \geq 1, S \subseteq [d]$*, the problems* (α, β, S)-Optimize(P) *and* (α, β, S)-Separate(P) *are computationally equivalent. That is, if one can solve one in time* poly(d)*, one can solve the other in time* poly(d) *as well.*

More formal statements and how we apply these theorems to yield our main result can be found in [9]. Finally, the theorems hold for minimization as well as maximization and without the restriction that P contains the origin (but the theorem statements are more technical).

Open Problems

Our work provides a novel computational framework for solving Bayesian mechanism design problems. We have applied our framework to solve several specific important problems, such

DOI 10.1007/978-3-642-27848-8_787-1

as computing revenue-optimal auctions in multi-item settings and approximately optimal BIC mechanisms for job scheduling, but numerous important settings and objectives remain unresolved. Theorem 1 provides a concrete approach for tackling such problems, via the design of (α, β)-approximations for the purely algorithmic Generalized Objective Optimization Problem. Therefore, one important direction following our work is to apply our framework to novel settings and design algorithms for the resulting GOOP instances.

Recommended Reading

1. Asadpour A, Saberi A (2007) An approximation algorithm for max-min fair allocation of indivisible goods. In: The 39th annual ACM symposium on theory of computing (STOC), San Diego
2. Bezáková I, Dani V (2005) Allocating indivisible goods. SIGecom Exch 5(3):11–18
3. Cai Y, Daskalakis C (2011) Extreme-value theorems for optimal multidimensional pricing. In: The 52nd annual IEEE symposium on foundations of computer science (FOCS), Palm Springs
4. Cai Y, Daskalakis C, Matthew Weinberg S (2012) An algorithmic characterization of multi-dimensional mechanisms. In: The 44th annual ACM symposium on theory of computing (STOC), New York
5. Cai Y, Daskalakis C, Matthew Weinberg S (2012) Optimal multi-dimensional mechanism design: reducing revenue to welfare maximization. In: The 53rd annual IEEE symposium on foundations of computer science (FOCS), New Brunswick
6. Cai Y, Daskalakis C, Matthew Weinberg S (2013) Reducing revenue to welfare maximization: approximation algorithms and other generalizations. In: The 24th annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans
7. Cai Y, Daskalakis C, Matthew Weinberg S (2013) Understanding incentives: mechanism design becomes algorithm design. In: The 54th annual IEEE symposium on foundations of computer science (FOCS), Berkeley
8. Daskalakis C, Matthew Weinberg S (2012) Symmetries and optimal multi-dimensional mechanism design. In: The 13th ACM conference on electronic commerce (EC), Valencia
9. Daskalakis C, Matthew Weinberg S (2015) Bayesian truthful mechanisms for job scheduling from bi-criterion approximation algorithms. In: The 26th annual ACM-SIAM symposium on discrete algorithms (SODA), San Diego
10. Grötschel M, Lovász L, Schrijver A (1981) The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1(2):169–197
11. Karp RM, Papadimitriou CH (1980) On linear characterizations of combinatorial optimization problems. In: The 21st annual symposium on foundations of computer science (FOCS), Syracuse
12. Khachiyan LG (1979) A polynomial algorithm in linear programming. Sov Math Dokl 20(1):191–194
13. Lenstra JK, Shmoys DB, Tardos É (1990) Approximation algorithms for scheduling unrelated parallel machines. Math Program 46(1–3):259–271
14. Myerson RB (1981) Optimal auction design. Math Oper Res 6(1):58–73
15. Shmoys DB, Tardos É (1993) Scheduling unrelated machines with costs. In: The 4th symposium on discrete algorithms (SODA), Austin