Mechanism design for resource allocation – with applications to centralized multi-commodity routing

Qipeng Liu  
Institute of interdisciplinary information sciences  
Tsinghua University, China  
lqp1831@gmail.com

Yicheng Liu  
Institute of interdisciplinary information sciences  
Tsinghua University, China  
61c@live.cn

Pingzhong Tang  
Institute of interdisciplinary information sciences  
Tsinghua University, China  
kenshinping@gmail.com

ABSTRACT

We formulate and study the algorithmic mechanism design problem for a general class of resource allocation settings, where the center redistributes the private resources brought by individuals. Money transfer is forbidden. Distinct from the standard literature, which assumes the amount of resources brought by an individual to be public information, we consider this amount as an agent’s private, possibly multidimensional type. Our goal is to design truthful mechanisms that achieve two objectives: maxmin and Pareto efficiency.

For each objective, we provide a reduction that converts any optimal algorithm into a strategy-proof mechanism that achieves the same objective. Our reductions do not inspect the input algorithms but only query these algorithms as oracles.

Applying the reductions, we produce strategy-proof mechanisms in a non-trivial application: network route allocation. Our models and results in the application are valuable on their own rights.

Keywords
mechanism design, strategyproof, resource allocation, network routing

1. INTRODUCTION

One of the most important problems at the intersection of economics and computation is algorithmic mechanism design, which dates back to the seminal work of Nisan and Ronen [20]. The basic problem asks:

Given an algorithmic optimization problem, is it possible to efficiently produce a truthful mechanism that (approximately) achieves the optimal value of the original problem?

Over the past decade, there has been a number of breakthroughs regarding this problem in settings where money transfer is allowed, including the rich literature on truthful welfare-maximizing mechanism design [22, 9, 8], and Bayesian incentive compatible (BIC) mechanism design [14]. Recently, the same problem has been investigated under the context of revenue optimal mechanism design [6, 7].

Distinct from the above literature, we study algorithmic mechanism design without money [21, Chapter 10]. In particular, we study a general class of resource allocation problems, where each agent brings a certain amount of resources and the mechanism distributes these resources to achieve certain objectives. It is important to note that our setting differs from the standard resource allocation literature (such as one-sided matching, hedonic games, etc) in that each agent’s type is the amount of resources she brings, rather than her preference over allocations. Our goal is to design strategy-proof mechanisms that achieve two objectives: maxmin and Pareto efficiency.

Our framework is rich enough to encompass, or at least heavily intersect with, a variety of applications, such as cloud resource allocation [13, 21], facility location [26], fair division [25] and network route allocation [this paper].

Main results

We make the following contributions.

1. Black-box reductions

For any resource allocation problem, we provide two reductions, one for maxmin and the other for Pareto efficiency, that automatically convert any optimal algorithm into a strategy-proof mechanism that achieves the same objective. Our reductions do not make use of any algorithm but only assume black-box access to the algorithm.

• For the maxmin objective, we construct a polynomial time algorithm that, for any input, generates the optimal group-strategy-proof mechanism by calling the optimal algorithm only once.

• For Pareto efficiency, we show that, if there is an algorithm that serially optimizes the utility profile, the algorithm per se is strategy-proof.

2. Application

Simple reductions as they may seem, their generalities are demonstrated by a complex, practical application: network route allocation.

Network route allocation. In this application, each node downloads a file from a server node located somewhere in the network. It can download via a direct route to the sever (it can do so without joining the mechanism, imposing an IR constraints to the design problem) or, by joining the mechanism, download via certain indirect routes that pass through other nodes. Each node has a certain private capacity (resource) that specifies the maximum limit of flow that can pass through it. Upon receiving the reports of private capacities, the mechanism returns a multi-commodity flow on the network. Our model encompasses both client-server networks and peer-to-peer networks [23, Chapter 4]. Applying our reductions, one can obtain IR and strategy-proof mechanisms that achieve either maxmin or efficiency.

Related work

In the literature of mechanism design without money, a popular line of work concerns how to locate a facility \[17\] [20]. They study a mechanism design problem where agents are located on the real line and the mechanism select the location of a facility, where each agent’s cost is its distance to the facility. The objectives are maximize as well as efficiency. They give tight bounds on ratio of the optimal strategy-proof mechanism over the optimal algorithm without incentive constraints. This line of work has been extended to a number of variations, such as there are two facilities \[14\], or the utility function is not linear \[11\], or the objective is the least square \[10\].

Resource allocation is also of central importance to the multiagent system community. It has been widely discussed in the application of smart grid \[23\] [27] and has been of the main application scenario of security games (see \[39\]).

Our resource allocation notion is related to the classic bilateral trade setting \[19\], resource exchange setting \[8\], as well as the recently coined reallocation settings \[4\]. A key distinction is, as mentioned, our setting treats the amount of contributed resources as private type while this amount is public information in their settings and they all treat valuation functions as types. Furthermore, their objectives are focused on characterization of efficient truthful mechanisms while our objective is to reduce mechanism design to algorithm design.

The second blackbox reduction in this paper, i.e., the one for the Pareto efficiency objective, makes use of algorithms that serially maximizes the utility profiles. This idea naturally relates to our reduction to the (randomly) serial dictatorship literature. See \[13\] [10] [16] [2], which are known to be strategyproof and Pareto efficient. Note again that their strategyproofness is defined with respect to truthful announcement of preferences, rather than contributed resources. Another distinction is that the serial dictatorship literature is concerned with allocation of indivisible items while our objective is to reduce mechanism design to algorithm design.

We now formulate the resource allocation problem. An environment specifies the parameters for the mechanism designer to operate.

**Definition 1.** An environment is a tuple \((\mathcal{N}, \mathcal{S}, \mathcal{P}, \mathcal{O}, \mathcal{U})\), where

- \(\mathcal{N}\) denotes the set of \(n\) agents,
- \(\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n\), where each \(\mathcal{S}_i\) is the private type set of agent \(i\),
- \(\mathcal{P}\) is a set of public information and resources shared by all agents,
- \(\mathcal{O}\) is the set of outcomes.

By revelation principle \[18\], one can without loss restrict attentions to the set of direct-revelation mechanisms, which can be regarded as functions that maps agents’ reported type profile and public info. into an outcome.

**Definition 2.** Given an environment, a deterministic mechanism is a function \(\mathcal{M}: \mathcal{S} \times \mathcal{P} \rightarrow \mathcal{O}\).

Each agent \(i\) in the environment comes with a private amount of resources \(s_i \in \mathcal{S}_i\) and is asked by the mechanism to report this quantity. Upon receiving all inputs, the mechanism returns an outcome, i.e., an allocation of resources.

For example, consider a variant of the dominant resource problem defined in \[13\], \(\mathcal{S}_i = \mathbb{R}^{+}\) whose the \(j\)-th element in the vector is the amount of \(j\)-th resource that agent \(i\) owns. Agent \(i\) needs \(c_{i,1}\) units of the first type of resource and \(c_{i,2}\) units of the second and \(\cdots\) \(c_{i,m}\) units of the \(m\)-th in order to conduct 1 unit of job task. Thus, \(\mathcal{P}\) is the set of all possible such \(\{(c_{1,1}, c_{1,2}, \cdots, c_{i,m})\}_{i=1}^{n}\) and \(\mathcal{O}\) is the set of all possible distributions of resources.

A resource allocation environment imposes certain feasibility constraints on any mechanism defined on it. For example, no mechanism shall allocate 3 CPU units to an agent if there are only 2 units of CPU within the society.

**Definition 3.** For any mechanism input \(s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2, \cdots, s_n \in \mathcal{S}_n, p \in \mathcal{P}\), define \(\text{FEA}(s_1, s_2, \cdots, s_n, p) \subseteq \mathcal{O}\) as a set of feasible outcomes under \((s_1, s_2, \cdots, s_n, p)\).

Consider again the dominant resource problem, \(o \in \text{FEA}(s_1, \cdots, s_n, p)\) if and only if the resource allocation prescribed by \(o\) is feasible \[1\] under the input.

We consider environments and feasibility constraints that satisfy the following resource monotone property.

**Definition 4.** For an environment \((\mathcal{N}, \mathcal{S}, \mathcal{P}, \mathcal{O})\), a feasibility constraint function \(\text{FEA}\) is resource monotone if \(\forall i\), there is a partial order \(\preceq_i\) on \(\mathcal{S}_i\) such that given any input profile \(s_1, s_2, \cdots, s_n\) and \(p \in \mathcal{P}\), for any \(s'_i \preceq_i s_i\), we have \(\text{FEA}(s_1, \cdots, s'_i, \cdots, s_n, p) \subseteq \text{FEA}(s_1, \cdots, s_i, \cdots, s_n, p)\).

Intuitively, resource monotonicity states that the larger amount of resource one contributes, the larger is the set of feasible allocations. For a mechanism defined on an environment satisfying the resource monotonicity, we call it a resource allocation mechanism.

Every player has a utility function \(u_i : \mathcal{O} \times \mathcal{P} \rightarrow \mathbb{R}\). In this paper, we consider a type of \(u_i\) that must only be related to public information and outcome. This means that the agent’s utility does not depend on her private resources. In addition, we assume that there exists an empty outcome \(o^*\) that for any other outcome \(o\), \(u_i(o^*, p) \leq u_i(o, p)\).

The above definition of utility is not uncommon in the resource allocation domain. Consider again the dominant resource problem, the number of total units of computation player \(i\) can conduct only depends on the allocation outcome and his \(c\)-vector \((c_{i,1}, c_{i,2}, \cdots, c_{i,m})\) and the empty outcome is the allocation where no one gets any resource.

The goal of resource allocation mechanism design is to optimize a certain real-valued function \(w : \mathcal{O} \rightarrow \mathbb{R}\) for equilibrium outcomes. Here \(w\) can be different functions according to different application scenarios. For example, in some cases \(w\) denotes social welfare, i.e., \(w(o) = \sum_{i=1}^{n} u_i(o, p)\); while in some other cases, \(w\) denotes the minimum utility among all agents’, i.e., \(w(o) = \min u_i(o, p)\).

The solution concept in this paper is the dominant strategy equilibrium.

\[1\] Feasibility means does not over-allocate any type of resource.
Definition 5. A mechanism is dominant-strategy truthful, (aka. strategy-proof) if for any \( s_1, \ldots, s_n, p \) and any \( s'_i \leq i \), then
\[
u_i(M(s_1, \ldots, s_n, p), p) \geq u_i(M(s_1, \ldots, s'_i, \ldots, s_n, p), p)
\]

Definition 6. A mechanism is group strategy-proof if for any \( s_1, \ldots, s_n, p \) and any \( s'_1, \ldots, s'_n \) where each \( s'_i \leq i \), then
\[
u_i(M(s_1, \ldots, s_n, p), p) \geq u_i(M(s'_1, \ldots, s'_n, p), p)
\]

It is easy to see that the definition above subsumes (is stronger than) another definition of group strategyproofness that no subset of agents could jointly deviate so that the resulting outcome is better off for anyone in this subset.

Our goal in this paper is to provide a tool to design strategy-proof mechanisms that optimize certain objectives. In particular, given an algorithm that optimizes a certain objective, we use this algorithm as a black-box and return a strategy-proof mechanism that optimizes the same objective.

3. MAX-MIN

In this section, we consider the problem of designing a strategy-proof mechanism that maximizes the minimal utility among all agents, i.e., \( w(o) = \min_{i \in N} u_i(o, p) \). We show that, given an algorithm that computes the arg max \( w(o) \), we can construct a strategy-proof mechanism with output \( o' \) such that \( w(o') = w(o) \) for each input.

Definition 7. An environment is continuous if for any \( (s_1, \ldots, s_n, p), \) let \( \sigma_w = \arg \max_w \omega \) and for any \( u'_i \leq u_i(\sigma_w, p) \), there must exist some outcome \( o' \in FEAS(s_1, s_2, \ldots, s_i, \ldots, s_n, p) \) such that \( u(s'_i, p) = u'_i \) and for any \( j \neq i \), \( u(s_j, p) = u_j(\sigma_w, p) \).

Theorem 1. If there exists an algorithm \( A \) that optimizes \( w(o) = \min_{i \in N} u_i(o, p) \) for some continuous, resource monotone environment, one can efficiently construct a strategy-proof mechanism \( M \) that optimizes \( w(o) \).

This theorem states that for the resource allocation problems under consideration, designing strategy-proof mechanism with max-min objective is no harder than the corresponding algorithm design problem.

We prove the following stronger theorem instead.

Theorem 2. If there exists an algorithm \( A \) that optimizes \( w(o) = \min_{i \in N} u_i(o, p) \) for some continuous, resource monotone environment, one can efficiently construct a group-strategy-proof mechanism \( M \) that optimizes \( w(o) \).

Given an algorithm \( A \), we can construct \( M \) as follows:

In other words, we find an outcome \( o_0 \) that brings down all agents’ utilities to \( u^* \), the maximin value computed by \( A \).

Proof. First, let us analyze the time complexity of the mechanism. Denote \( TIME(A) \) as the time complexity of algorithm \( A \) and \( TIME(F) \) as an upper bound of the time complexity to find such new outcome. So the time complexity is \( O(TIME(A) + n TIME(F)) \). The time of computing the above algorithm can lead to infeasible allocations, which can be easily detected and punished.

Mechanism 1 A group strategy-proof mechanism via \( A \)

1. on inputs \( s_1, s_2, \ldots, s_n, p \), run \( A(s_1, \ldots, s_n, p) \) to get the outcome \( o_0 \);
2. let \( u^* = \min_{i \in N} u_i(o_0, p) \);
3. for \( i = 1, 2, \ldots, n \):
   find \( o_i \) such that for all \( j \neq i \), \( u_i(o_j, p) = u_i(o_{i-1}, p) \) and \( u_i(o_i, p) = u^* \)
4. outputs \( o_0 \).

\( o_i \) is often small enough, for example in our application, so that \( M \) has time complexity \( O(TIME(A)) \).

Now we verify that \( M \) really outputs a feasible solution which optimizes \( w(o) \). The outcome \( o_0 \) is feasible because in each step \( o_i \) is feasible by the continuity of the environment. Also, it is straightforward that \( w(o_0) = w(o_0) \).

Finally, we prove that \( M \) is group-strategy-proof. Consider any input \( s_1, s_2, \ldots, s_n, p \) and everyone reports \( s'_i \) where \( s'_i \leq i \):

\[
u_i(M(s_1, s_2, \ldots, s_n, p), p) \geq \max_{o \in FEAS(s_1, s_2, \ldots, s_n, p)} w(o) \]
\[
u_i(M(s'_1, s_2, \ldots, s'_n, p), p) \geq \max_{o \in FEAS(s'_1, s_2, \ldots, s'_n, p)} w(o) = u_i(M(s'_1, s'_2, \ldots, s'_n, p), p)
\]

So \( M \) is a group-strategy-proof mechanism that optimizes \( w(o) \).

Applying the same technique, one can produce a strategy-proof mechanism to optimize \( w(o) = \min_{i \in N} f_i(u_i(o, p)) \) with \( n \) strictly increasing functions \( f_1, f_2, \ldots, f_n \). With this extension, we can optimize some interesting objectives subject to the individual rationality constraints. We now formally define individual rationality.

4. MAX-MIN SUBJECT TO INDIVIDUAL RATIONALITY

In this section, we consider the setting where if a player does not join the mechanism, he will get a utility \( r_i(p) \) that only depends on the public information \( p \). In the previous setting, every player’s \( r_i(p) = -\infty \); in other words, she always wants to join the mechanism in the previous setting.

Definition 8. A mechanism is individual rational if on any inputs \( s_1, s_2, \ldots, s_n, p \), the mechanism outputs \( o \) such that for any \( i \in N \), \( u_i(o, p) \geq r_i(p) \).

4.1 Replacing \( u_i \) by \( f_i(u_i) \)

So the problem now is to design a strategy-proof and individual rational mechanism that achieves certain objective. This can be achieved by the same method as Theorem 2.
Theorem 3. If there exists an algorithm \( A \) that computes \( w(o) = \min f_i(u_i(o,p)) \) for a continuous environment with strictly increasing functions \( f_1, f_2, \cdots, f_n \), one can construct a strategy-proof mechanism \( M \) that computes \( w(o) \).

The reduction is identical to that in theorem 1 thus the proof is omitted.

By this theorem, we can design individual rational and strategy-proof mechanism that optimizes the following:

Corollary 1. Let \( f_i(x) = x - r_i(p) \), one obtains an individual rational and strategy-proof mechanism that optimizes the minimal utility gain \( f_i \).

Corollary 2. If \( r_i(p) > 0 \) for any \( i \in \mathcal{N} \), let \( f_i(x) = x/r_i(p) \), one obtains an individually rational and strategy-proof mechanism that optimizes minimal increasing rate.

4.2 Optimizing \( \min_i u_i(o,p) \) subject to IR

A further question is, can we optimize \( \min u_i(o,p) \) instead of \( w(o) = \min f_i(u_i(o,p)) \), subject to IR? We answer this affirmatively by modifying the previous method to produce an IR and SP mechanism that optimizes \( \min u_i(o,p) \).

Theorem 4. If there exists an algorithm \( A \) that computes \( w(o) = \min u_i(o,p) \) for any continuous, resource monotone environment, such that \( u_i(o,p) \geq r_i(p) \), then we can construct a strategy-proof and individual rational mechanism \( M \) that computes \( w(o) \).

5. PARETO EFFICIENCY

In this section, we discuss the objective of implementing a serially optimal outcome. We still use the same setting except that we now restrict agent’s \( i \) type space \( S_i \) to \( \mathbb{R} \).

Definition 9. An outcome \( o \) is called serially optimal under \( s_1, \cdots, s_n, p \) if the utility profile \( (u_1(o,p), \cdots, u_n(o,p)) \) is the lexicographically largest one among all possible utility profiles; formally, for any \( o' \in \text{FEA}(s_1, \cdots, s_n, p) \), either the utility profile of \( o \) equals to that of \( o' \) or there exists some \( j \in \mathcal{N} \) such that for all \( k < j \), \( u_k(o,p) = u_k(o',p) \) and \( u_j(o,p) > u_j(o',p) \).

Clearly, serial optimality implies Pareto efficiency, so we focus on the former. Our problem now becomes, given any list of input, compute a strategy-proof and serially optimal outcome. Before we start, we need the following property.

Definition 10. The utility functions are monotone if for any \( i \in \mathcal{N} \), any inputs \( s_1, \cdots, s_n, p \) and \( 0 < \delta \leq u_i(o,p) - r_i(p) \), there exists an \( \varepsilon > 0 \) such that for any \( 0 < \varepsilon' < \varepsilon \), there exists an outcome \( o' \in \text{FEA}(s_1, \cdots, s_i-\varepsilon', \cdots, s_n, p) \) such that for any \( j \neq i \), \( u_j(o',p) = u_j(o,p) \) and \( u_i(o',p) \geq u_i(o,p) - \delta \).

Theorem 5. If there exists an algorithm \( A \) that serially optimizes \( w(o) = (u_1(o,p), u_2(o,p), \cdots, u_n(o,p)) \) for monotone utility functions, resource monotone environment and \( u_i(o,p) \geq r_i(p) \), and for each player \( i \), her utility, which is a function of \( s_1, u_i(A(s_1, \cdots, s_n, p), p) \) is continuous, the algorithm itself is an IR and SP mechanism that computes a serially optimal outcome.

Proof. We denote \( x_i(\varepsilon) = u_i(A(s_1, \cdots, s_i-\varepsilon, \cdots, s_n, p), p) \) for \( 0 \leq \varepsilon \leq s_i \), for fixed inputs \( s_1, \cdots, s_n, p \) (we call it (1)).

\( x_i(\varepsilon) \) is a continuous function of \( s_i \). First we prove that there exists an \( \varepsilon > 0 \). \( A \) outputs an outcome \( o' \) such that \( u_i(o',p) \leq u_i(o,p) \) under \( s_1, \cdots, s_i-\varepsilon', \cdots, s_n, p \) (we call it (2)) for any \( 0 < \varepsilon' \leq \varepsilon \).

We prove it by contradiction. An important observation is that any feasible solution under (2) is also feasible under (1) by the resource monotone environment condition, in other words,

\[ \text{FEA}(s_1, \cdots, s_i-\varepsilon', \cdots, s_n, p) \subseteq \text{FEA}(s_1, \cdots, s_i, \cdots, s_n, p). \]

If \((u_j(o',p))_{j=1}^n \) has the property that \( u_i(o',p) > u_i(o,p) \), then there must exist \( k < i \) such that \( u_k(o',p) = u_k(o,p) \) for all \( k < j \) and \( u_k(o',p) < u_k(o,p) \) (because a solution in (2) is a solution in (1) and \((u_j(o,p))_{j=1}^n \) is the serially optimal). However we can construct a larger solution for (2): by the monotone utility function condition, we know that there exists an outcome \( o'\) such that \( u_j(o',p) = u_j(o,p) \) for all \( j \neq i \) and \( u_i(o',p) > u_i(o,p) - \delta \) for some \( \delta \) and \( \varepsilon \) which is a contradiction that the algorithm find a serially optimal solution.

Next we prove that \( u_i(\varepsilon) \) is a decreasing function. Let \( g_i(\varepsilon) = x_i(\varepsilon) - r_i(p) \). Because the algorithm is IR, \( g_i(\varepsilon) \geq 0 \) for all feasible \( \varepsilon \). For any \( \varepsilon \) such that \( g_i(\varepsilon) > 0 \), there exists an \( \delta > 0 \) such that \( g_i(\varepsilon) \geq g_i(\varepsilon') \) for any \( \varepsilon \leq \varepsilon' \leq \varepsilon + \delta \) (by the monotone utility function) so it is locally monotonically decreasing. When \( g_i(\varepsilon) = 0 \), for all feasible \( \varepsilon' > \varepsilon \), \( g_i(\varepsilon') = 0 \). If \( g_i(\varepsilon') > 0 \), we can find a first extreme point \( \varepsilon \in (\varepsilon, \varepsilon') \). Then for any small \( \delta > 0 \), \( g_i(\varepsilon' - \delta) < g_i(\varepsilon') \) which conflicts with the utility function monotone.

So \( g_i(\varepsilon) \) is a monotonically decreasing function when \( g_i(\varepsilon) > 0 \). And if \( g_i(\varepsilon') = 0 \), then \( g_i(\varepsilon') \) for \( \varepsilon' > \varepsilon \) will always be 0 .

We can conclude that \( g_i(\cdot) \) is a monotonically decreasing function. Since \( x_i(\cdot) = g_i(\cdot) + r_i(p) \), \( x_i(\cdot) \) is also a monotonically decreasing function which means player \( i \) will never get more profit by reporting a lower \( s_i \leq s_i \).

Theorem states that for monotone utility functions and resource environment, mechanism design is as easy as algorithm design.

Serial optimization by contribution

In order to do serial optimization, one needs to pre-specify a (static) ordering on \( \mathcal{N} \), resulting in unfairness for agents who have low rank. In this section, we show that, without sacrificing strategyproofness, this ordering can be extended
to depend on the input. In particular, this ordering can be more one contributes, the higher priority she gets.

Formally, define "order envy-freeness" as follows:

**Definition 11.** Consider the class of serially optimized mechanisms, a mechanism is order envy-free (OEF) with respect to \((l_i)_{i=1}^n\) \((l_i\) is a weight for agent \(i\)) if for any two agents \(i, j \in \mathcal{N}\), such that \(l_is_i > l_js_j\), \(i\) is ranked before \(j\) in the serial ordering.

Order envy-freeness states that an agent gets a higher priority for optimization than the ones with lower contributions. In particular, if set \(l_i = 1 \forall i\), the order of optimization is same as the ranking on reports.

**Theorem 6.** If there exists an algorithm \(A\) that serially optimizes \(w(o,q) = (u_{q_1}(o,p), u_{q_2}(o,p), \ldots, u_{q_n}(o,p))\) \((\text{where } q \text{ is an index on agents such that } l_is_i > l_js_j \text{ for any } i < j\) for monotone utility functions, resource monotone environment and \(u_{i}(o,p) \geq r_i(p)\), then the algorithm itself is an IR, SP and OEF (w.r.t \((l_i)_{i=1}^n\)) mechanism that compute a serially optimal outcome.

To achieve order envy-freeness, we define a slightly different reduction as follows. Given reported \(s_i\), \(s_j\), we generate an order \(q_i\) such that for any \(i, l_is_i > l_js_j\), and optimizes each \(s_i\) according to the order \(\{q_i\}\). In other words, we define the order according to their contributions: the more you share, the earlier you get served. Technically speaking, this modified mechanism does not belong to the class of mechanisms we have proposed since the ordering of optimization now explicitly depends on the report. However, it is easy to check that this modification does not affect IR, PE, SP and guarantee OEF as a plus. The mechanism is listed below. The proof follows a similar argument to the static case, which we do not repeat.

Mechanism 2 An IR, SP, PE and OEF mechanism

1. on inputs \(s_1, s_2, \ldots, s_n, p, \text{ and } l_1, l_2, \ldots, l_n,\)
2. compute the order \(q_i\) by any sorting algorithm,
3. run \(A(s_{q_1}, \ldots, s_{q_n}, p, q)\) to get the outcome \(o\).
4. output \(o\)

6. APPLICATION: NETWORK ROUTE ALLOCATION

Starting from this section, we show the generality of our methodology by applying it to two realistic scenarios. The two applications are highly nontrivial and valuable on their own rights.

We first consider an interesting route allocation problem in multiple-commodity networks. For the network structures under consideration, there are several vertices known as users. Each user demands a file of certain size stored on a server. The users form a directed graph. Each user has a capacity, which denotes the maximum (traffic) flow that can go through that vertex. We assume this vertex capacity is private information of the user. Between each user and each server, there is an arc constrained by certain capacity, denoting the maximal flow that can go through the arc. Given such a network, a user can download her target file via any route to the destination server\(^3\). Given the reported vertex capacities, a route allocation mechanism allocates a route (or multiple routes, both of which we consider) and feasible flow within the route for each user.

It is not hard to see that our formulation encompasses route allocation problems in both client-server based and peer-to-peer based (simply treat a server in our model as a peer that never downloads) networks. In practice, the same route allocation problem has been witnessed by Xun-You Inc\(^4\) an online gaming platform that aims to resolve the congestions on networks consisting of subnetworks by multiple Internet Service Providers. The same problems have also been witnessed by route optimization among multiple express companies, each of which specializes in some geographic region.

A user’s utility is the negation of its time delay, given by

\[
\text{utility} = \frac{\text{allocated flow}}{\text{file size}}
\]

This notion of utility is widely used in the evaluation of performance in a network system or an operating system. More importantly, it does not depend on the private information of users, i.e., the private capacity of each vertex, allowing our framework to be applicable.

We will use the above reduction theorems to design a strategy-proof mechanism that achieves the minmax objective (see [20]), i.e., to minimize the maximum delay among all users. We also consider the objective of Pareto efficiency.

6.1 Setting

Formally, in our setting, there are \(n\) users \(P = \{p_1, p_2, \ldots, p_m\}\) and \(m\) servers \(Q = \{q_1, q_2, \ldots, q_n\}\) \((p_i \text{ is called user } i \text{ and } q_i \text{ is called server } j)\). For a user \(p_i\), her desired file is stored on server \(d_i\), i.e., \(q_i\). The size of the file is \(c_i\) units. If the amount of her available bandwidth is \(x\) \((x\) units of flow per second\), she can finish her task in \(c_i/x\).

The connections between users form a directed graph \(G = (V, E)\), where \(V\) is the set of vertex in the graph, \(E\) is the set of edges. For a vertex in \(V\), it denotes a user in \(P\). A pair of users can connect with each other. For an edge \((u, v) \in E\), we assume there is not limitation in the edge \((u, v)\), in other words, arbitrarily large flow can pass through this edge. This is with out loss of generality — all results carry over to the setting where each edge has a capacity.

Also, each user \(p_i\) can connect to a subset of servers. For a pair of user \(p_i\) and server \(q_j\), there is an edge between them with bandwidth \(b_{i,j}\). That is, this edge can transmit at most \(b_{i,j}\) units of flow per second. When \(b_{i,j} = 0\), there is no connection between \(p_i\) and \(q_j\).

Each user has a bandwidth limitation locally. The total amount of flow she can download and share with others per second is at most \(v_i\). In reality, \(v_i\) is the limitation that \(v_i\) for each user, there exists a direct route via which the user can download its file without even join the mechanism. By joining the mechanism, however, the user can download the file via any indirect route that pass through other users who also join the mechanism by sharing their bandwidths. This outside option imposes an IR constraint for the design problem.

\[\text{http://www.xunyou.com}\]

\(^3\)We use "bandwidth" and "flow" interchangeably.
can depend on user i’s hardware or software. A user $p_i$ can
download her own file either by the edge directly between $p_i$
and $q_{d_i}$, or indirectly from some $p_j$’s where $p_j$ downloads it
from the edge $p_j$ and $q_{d_i}$.

Consider the illustrating example in Figure 2. The left
nodes denote connections between users and the right nodes
denote connections between users and servers. In this example,
user $p_3$ could download its file through the edge $(p_3, p_1)$,
through the path $(p_3, p_1), (p_1, p_2)$ or the path $(p_3, p_1), (p_1, p_2), (p_2, q_3)$.

We assume that $G, \{b_{i,j}\}_{i=1\ldots n}, \{(d_{i}, c_{i})\}_{i=1\ldots n}$ are pub-
lic information that is shared by all users and the mechanism
designer, while $\{v_{i}\}_{i=1\ldots n}$ is private information known only
to each user $i$. Given a reported profile of $(v_1, v_2, \ldots, v_n)$,
a mechanism returns a flow assignment $x_i$ for each $i$ (a mul-
tiple-route assignment).

Thus, we can define the environment $\{N, S, P, O, u\}$ of
this problem:

- $N$ denotes the set of $n$ agents,
- $S = S_1 \times S_2 \times \cdots \times S_n$ each $S_i = \mathbb{R}$ represents the local
  bandwidth limitation $v_i$,
- $P$ represents servers $Q$, the whole graph structures
  $G = (V, E)$, $\{(d_{i}, c_{i})\}_{i=1\ldots n}$ and all information about
  capacities on edges,
- $O$ is the set of all the possible assignments,
- $u_i(o, p) = \frac{-x_i}{c_i}$ where $x_i$ is i’s total amount of flow in
  the assignment $o$.

Also we need to define individual rationality $r_i(\cdot)$: for any
agent $i \in N$, the flow she receives by a mechanism is greater
then or equal to her direct route bandwidth $b_i d_i$, if she does
not join the mechanism. Formally, $r_i(p) = b_i d_i$.

6.2 Max-Min objective subject to individual rationality

**Lemma 1.** The above environment is resource monotone and
continuous.

**Proof.** For any feasible assignment in $FEA(s_1, s_2, \cdots, s_n, p)$,
it doesn’t violate any constraint in $FEA(s_1, \cdots, s_i + \epsilon, \cdots, s_n, p)$,
implying monotonicity.

For any $u_i' \leq u_i$, we can simply decrease $x_i$ to $x_i' =
-c_i / u_i' \leq -c_i / u_i = x_i$ by cancelling $x_i - x_i'$ units of flow
in the assignment. So it is continuous.

By theorem 3, we only need to give an algorithm to opti-
mize min $u_i(o)$. Because the multiple-commodity flow problem
can be written in linear program, so we can get the
algorithm by binary search and linear programming:

**Mechanism 3** An algorithm to minimize the maximal cost

binary search $D \in \left[0, \max_{i \in N} \frac{c_i}{b_i d_i} \right]$ if the following
LP is feasible under certain $D$, decrease $D$; otherwise,
increase $D$:

maximize : 0 (No objective here, i.e., a linear-
feasibility program.)
subject to
1. $x_i = x_{i,1} + x_{i,2} + \cdots + x_{i,n}$ for all $i = 1, \cdots, n$;
2. $x_i = x_{i,i} + \sum e' f_{i,e'}$ ($e'$ going into $i$) for all
   $i = 1, \cdots, n$;
3. $\sum e' f_{i,e'} = \sum e' f_{i,e'} + x_{i,j}$ ($e'$ going into $j$
   and $e''$ going out of $j$) for all $i \neq j$;
4. $x_i \geq b_i d_i$ for all $i$;
5. $x_i \geq c_i / D$ for all $i$
6. $\sum x_{i,j} + \sum e' f_{j,e'} \leq v_i$ ($e'$ going into of $i$);
7. $\sum k x_{k,i} \leq b, j$ (where $k$ have the following
   property: $d_k = j$), for all $i, j$;
8. all variables are non-negative.

1. In the first equation, $x_i$ denotes the final flow user $i$
can get, and $x_{i,j}$ denotes the flow user $i$ gets from the edge
   $(p_k, q_{d_i})$. $x_{i,j}$ is the flow on the direct edge from
   the user $i$ to the server.

2. Let $f_{i,e}$ denote the final flow in edge $e$ when transport-
ing the file $c_i$ between $d_i$ and $p_d$. So, for the second
   equation, $x_i$ is the sum of all $f_{i,e'}$ (ingoing flow) and
   $x_{i,j}$.

3. The third one shows that in each vertex, the total in-
going flow equals the total outgoing flow, which is known
as the “flow conservation” constraints. Note that when
$i = j$, the equation does not make sense, since it is
impossible for user $i$ to transport her target flow to
others.

4. The fourth constraints ensures individual rationality.

5. The fifth means the upper bound of time each user
   needs is at most $D$.

6. The sixth inequalities $\sum_j x_{i,j} + \sum e' f_{j,e'} \leq v_i$ are the
   'capacity constraints' on vertices.

7. The sixth inequalities $\sum_k x_{k,i} \leq b_{i,j}$ are the 'capacity
   constraints' on edges.

8. The final constraints ensures all variables being non-
   negative.

With Algorithm 6.2, we obtain a SP, IR and Maxmin
mechanism (Algorithm 6.2) by theorem 4.
Mechanism 4 A strategy-proof minmax Mechanism

1. Binary search the minimized maximal downloading time \( D' \) among the solution of the LP (listed above)
2. For all \( p_i \in P \)
   (a) if \( \frac{x_i}{b_i} < D \), set \( x_i' \leftarrow b_i \)
   (b) else set \( x_i' \leftarrow \frac{x_i'}{D} \)
3. For all \( p_i \in P \), add constraint \( x_i = x_i' \)
4. Solve the new LP to compute the assignment.

6.3 Pareto efficiency subject to individual rationality

By theorem 3 we know that if there exists an algorithm to find a serially optimal outcome \( o \) for monotone utility function, resource monotone environment and IR, it is a strategy-proof mechanism. Algorithm 6.3 gives the outcome we want. Almost all constraints are the same as what we listed in the previous section except that the seventh constraint ensures that the final solution \((u_1(o,p), \ldots, u_n(o,p))\) is the serially optimal one among all the outcomes.

Mechanism 5 An IR, PE and SP mechanism

Given \( G, \{b_{i,j}\}, \{(d_i,c_i)\}, \{v_i\} \), repeat the following procedure \( n = |V| \) times:

\[
\text{maximize: } x_i \text{ (the } i\text{-th time)}
\]

subject to

1. \( x_i = x_{i,1} + x_{i,2} + \cdots + x_{i,n} \) for all \( i = 1, \ldots, n \);
2. \( x_i = x_{i,i} + \sum_{j \neq i} x_{i,j} (e' \text{ going into } i) \) for all \( i = 1, \ldots, n \);
3. \( \sum_{j \neq i} x_{j,i} - \sum_{j \neq i} x_{i,j} (e' \text{ going out of } j) \) for all \( i \neq j \);
4. \( x_i \geq b_i \) for all \( i \);
5. \( \sum_{j} x_{j,i} + \sum_{j \neq i} x_{i,j} (e' \text{ going into } i) \) ;
6. \( \sum_{j} x_{j,i} \leq b_{i,j} \) (where \( k \) have the following property: \( d_k = j \), for all \( i, j \);
7. for all \( j \leq i - 1 \), set \( x_i = x_i^* \) where \( x_i^* \)’s are the \( x_i \)’s optimized so far;
8. all variables are non-negative;

The lemma states that, it is without loss of generality to restrict to allocations where each user exhausts all its bandwidth.

**Proof.** First we prove that all bandwidth in \((i, d_i)\) will always be used in some solution. Because of the inequalities \( x_i \geq b_i \), the final bandwidth \( x_i \) is at least \( b_i \). If the bandwidth in the edge \( (p_i, q_{i,d_i}) \) is not fully exhausted (let \( b' \) be the remaining bandwidth in that edge), we can remove \( b_i - b' \) bandwidth from other paths and add it into the edge \((i, d_i)\). This operation does not violate any constraint. So we can repeat this operation and finally all \((i, d_i)\) is fully exhausted.

After having such a solution, we can modify some flow assignment so that all bandwidth in \((i, d_i)\) will always be used by \( i \) in some solutions. Now every edge \((p_i, q_{i,d_i})\) is fully used, we can use exchange techniques to let \((p_i, q_{i,d_i})\) is only used by \( i \). If part of \((p_i, q_{i,d_i})\) is used by other user \( j \), there must be a path from \( p_i \) to \( p_k \) and then from \( p_k \) to \( q_{i,d_i} \) with bandwidth \( \varepsilon \). We can give this path to user \( j \) (attached to the path from \( p_j \) to \( p_i \)) and give \( \varepsilon \) units bandwidth in \((p_i, q_{i,d_i})\) to \( p_i \). Repeating this operations, we finally let all bandwidth in \((p_i, q_{i,d_i})\) be used only by user \( i \).

We now prove that the utility functions are monotone:

**Proof.** For any \( i \in N \) and any inputs \( s_1, \ldots, s_n, p, i \) will get \( u_i(o(p)) = -\frac{\varepsilon_i}{d_i} \) from the best outcome \( o \). For any \( 0 < \delta \leq -\frac{\varepsilon_i}{d_i} < \frac{\varepsilon_i}{d_i} \), we can construct an outcome \( o' \) with utility profile \( \{u_1(o(p)), \ldots, u_i(o(p) - \delta, \ldots, u_n(o(p))\} \) from a solution for \( o \). A simple observation is that \( x_i' = \frac{u_i(o(p)) - \delta}{\varepsilon_i}. \) So one straightforward method is to decrease \( x_i \) to \( x_i'. \) Let \( \varepsilon = x_i - x_i' \). Because of lemma 3 there is a solution \( \{x_j, k\}, \{f_{j,k}\} \) to get the same utility profile of \( o \) such that \( x_{j,k} = b_{j,d_j} \) and \( \sum_{j=1}^n x_{j,k} = x_j \) for all \( j \). Now we construct the following outcome \( o' \):

- \( x_{i,k} = x_{i,k} - \varepsilon, \) and for other \( x_{i,k} = x_{j,k}; \)
- \( x_i = x_i - \varepsilon, \) and for other \( x_j = x_j; \)
- \( f_{j,k} = \) for all possible \( j, e; \)

Now we check whether the solution violates any constraint. For the first two inequalities, they still hold. For the third and sixth one, they aren’t related to any new variables so it still holds. Because \( 0 < \delta \leq -\frac{\varepsilon_i}{d_i} + \frac{\varepsilon_i}{d_i} \), we find that \( x_i' = \frac{u_i(o(p)) - \delta}{\varepsilon_i} \geq b_{i,d_i}. \)

For the fifth one,

\[
\sum_{j} x_{i,j}' + \sum_{j \neq i} f_{j,i}' \leq \sum_{j \neq i} (x_{i,j} + f_{j,i}) \leq \sum_{j \neq i} x_{i,j} + \sum_{j \neq i} f_{j,i}' \leq \sum_{j \neq i} v_{i,j}' = v_i',
\]

Then, we will show that \( x_{i,1}(\varepsilon), \ldots, x_n(\varepsilon) \) is continuous functions of variable \( \varepsilon. \) We can use the following lemma 4; for the function \( x_1(\varepsilon) \), there exists a closed interval \([\alpha, \beta]\), such that the LP is infeasible for all \( \varepsilon \notin [\alpha, \beta] \); and in the interval \([\alpha, \beta]\), \( x_1(\varepsilon) \leq R \) and it is a continuous convex piece-wise linear function. So for \( x_1(\varepsilon) \), it is a continuous function.

Now using induction, if we have proven that \( x_{i,1}(\varepsilon), \ldots, x_n(\varepsilon) \) is continuous functions, then \( x_{k+1}(\varepsilon) = x_{k+1}(\varepsilon), x_1(\varepsilon), \ldots, 

\]

Lemma 2. \( \{u_i\}_{i=1}^n \) are monotone utility functions under the environment.

To prove the lemma we need to prove the following lemma.

Lemma 3. For any possible IR solution \( \{x_1, x_2, \ldots, x_n\}, \{x_{i,j}\}, \{f_{j,i}\}, \{f_{i,e}\}, \{x_{i,j}'\}, \{x_i'\} \) with \( x_i' = b_{i,d_i} \).
Applying the reductions, one can systematically produce strategy-proof mechanisms in a non-trivial way. Specifically, we provided a reduction that converts a 

**THEOREM 7.** There exists an IR and SP mechanism which minimizes maximal cost/delay and an IR, OEF and SP mechanisms which gives a Pareto efficient solution for the route allocation problem.

## 7. CONCLUSION AND FUTHER WORK

In this paper, we studied truthful mechanism design for a general class of resource allocation settings, where the center redistributes the private resources brought by individuals. We designed truthful mechanisms that achieve two objectives: maximin and Pareto efficiency. For each objective, we provided a reduction that converts any optimal algorithm into a strategy-proof mechanism that achieves the same objective. Applying the reductions, one can systematically produce strategy-proof mechanisms in a non-trivial application: network route allocation.

There are a number of exciting research directions to extend this work. First of all, the mechanism for the maximin objective is wasteful in the sense that all the players are brought down to a utility level that equals the maximin. Is there another strategy proof mechanism that achieves maximin but yields better social welfare? Second, one can consider the same set of objectives under more general settings where randomization is allowed. Last but not least, we are also interested in finding more applications to our general algorithmic mechanism design framework.

**REFERENCES**


