Speed up Automated Mechanism Design by Sampling Worst-Case Profiles: An Application to Competitive VCG Redistribution Mechanism for Public Project Problem

Mingyu Guo[1(✉)] and Hong Shen[1,2]

[1] School of Computer Science, University of Adelaide, Adelaide, Australia
{mingyu.guo,hong.shen}@adelaide.edu.au

[2] School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China

Abstract. Computationally Feasible Automated Mechanism Design (CFAMD) combines manual mechanism design and optimization.

In CFAMD, we focus on a parameterized family of strategy-proof mechanisms, and then optimize within the family by adjusting the parameters. This transforms mechanism design (functional optimization) into value optimization, as we only need to optimize over the parameters.

Under CFAMD, given a mechanism (characterized by a list of parameters), we need to be able to efficiently evaluate the mechanism's performance. Otherwise, parameter optimization is computationally impractical when the number of parameters is large.

We propose a new technique for speeding up CFAMD for worst-case objectives. Our technique builds up a set of worst-case type profiles, with which we can efficiently approximate a mechanism's worst-case performance. The new technique allows us to apply CFAMD to cases where mechanism performance evaluation is computationally expensive.

We demonstrate the effectiveness of our approach by applying it to the design of competitive VCG redistribution mechanism for public project problem. This is a well studied mechanism design problem. Several competitive mechanisms have already been proposed. With our new technique, we are able to achieve better competitive ratios than previous results.

Keywords: Automated mechanism design · VCG redistribution mechanisms · Dominant strategy implementation · Groves mechanisms · Public good provision

1 Introduction

1.1 Automated Mechanism Design

Automated Mechanism Design (AMD) [1] studies how to use computational techniques to design new mechanisms. AMD is a core topic of algorithmic game theory and has attracted significant attention from both computer science and economics community. It has also helped deliver many successes on a variety of mechanism design topics.

AMD can incorporate different computational approaches. One naive form of AMD simply discretizes the type space, and solves the mechanism design problem as a linear program (LP) or as a mixed-integer program (MIP). Under this naive approach, mechanism properties are enforced by creating inequalities for *every possible type profile*. For example, let us consider a single-item auction with 3 agents, where every agent's type space is $[0, 1]$. We can discretize the type space into $D = \{0, 0.1, \ldots, 0.9, 1\}$. The set of all possible type profiles is then

$$\{(x, y, z) | (x, y, z) \in D^3\}$$

For every type profile, we use $o_{i,x,y,z}$ to denote agent i's chance for winning the item when the type profile is (x, y, z). We use $p_{i,x,y,z}$ to denote agent i's payment when the type profile is (x, y, z). The $o_{i,x,y,z}$ and the $p_{i,x,y,z}$ are the variables of the optimization model. The $o_{i,x,y,z}$ may be continuous or integer variables, which depends on the problem setting. With the above characterization, many mechanism properties can then be expressed using *exponential number*[1] of linear inequalities. For example, the *non-deficit* property states that the agents' total payment should always be nonnegative, which is then

$$\forall (x, y, z) \in D^3, \sum_i p_{i,x,y,z} \geq 0$$

The *strategy-proofness* property states that under the mechanism, an agent can never benefit by misreporting. For agent 1, this is then

$$\forall x' \in D, (x, y, z) \in D^3, x o_{1,x,y,z} - p_{1,x,y,z} \geq x o_{1,x',y,z} - p_{1,x',y,z}$$

Often, the above naive approach is only realistic for tiny problem instances, as dealing with exponential number of constraints is too expensive. Nevertheless, even in this naive form, AMD can be useful, as good mechanisms discovered for tiny instances can help provide insight for more general cases. *Our paper is not based on the aforementioned naive approach, we take on a computationally feasible approach as detailed below.*

1.2 Computationally Feasible Automated Mechanism Design

One particular approach of AMD is Computationally Feasible Automated Mechanism Design (CFAMD) [5], which reduces the computational cost of AMD by combining computation and manual mechanism design. CFAMD works as follows:

– Manually select a parameterized family of strategy-proof mechanisms. Every mechanism inside the family is characterized by t parameters. We use $M(p_1, p_2, \ldots, p_t)$ to denote the mechanism characterized by the parameters p_1 to p_t.

[1] In terms of the number of agents.

- Given a setting, the task of optimizing over all mechanisms is often an impossible task, as we often cannot even characterize all feasible mechanisms.[2] Under CFAMD, we focus on the parameterized family and optimize by adjusting the parameters. This is computationally much easier as we are no longer dealing with a functional optimization problem. We only need to solve a value optimization problem where we optimize over the parameters.
- If the parameterized family is selected to be general enough, then the *locally* optimal mechanism within the parameterized family can perform close to (or is competitive against) the *globally* optimal mechanism over all possible mechanisms.
- There is often a trade-off between mechanism performance[3] and computational cost. It is easy to see that a more general parameterized family leads to better performance for the resulting mechanism. Meanwhile, it is also often the case that a more general parameterized family makes the optimization process computationally more expensive.

Guo and Conitzer [5] summarized many successful applications of CFAMD, including Likhodedov and Sandholm [12,13] for revenue-maximizing combinatorial auctions, Guo and Conitzer [4,8] for optimal VCG redistribution mechanisms for multi-unit auctions, and Guo and Conitzer [7] for mechanism design without payments.

One crucial presumption of CFAMD is that given the p_i, we are able to evaluate the performance of the corresponding mechanism $M(p_1, p_2, \ldots, p_t)$. Without this presumption, we have no clue on how to adjust the parameters. Furthermore, in practise, we need to be able to evaluate the performance of a mechanism *fast*. For example, if every mechanism is characterized by 20 parameters and we need $O(n^5)$ time complexity to evaluate one mechanism (characterized by one set of parameters), then parameter optimization is too expensive to be practical.

We use PE to denote the task of performance evaluation: given a set of parameters, evaluate the performance of the corresponding mechanism. To apply CFAMD, we need to pick a parameterized family of mechanisms that is general enough for achieving reasonable mechanism performance. Meanwhile, the family needs to be restrictive enough to ensure that PE is efficient.

1.3 Speed up CFAMD by Sampling Worst-Case Profiles

This paper proposes a new technique for speeding up CFAMD. We will demonstrate the effectiveness of our new technique by applying it to the design of competitive VCG redistribution mechanisms for the public project problem. VCG

[2] For example, when it comes to revenue-maximizing combinatorial auction design, we do not have an easy-to-work-with characterization of all combinatorial auctions that are strategy-proof and individually rational.

[3] By performance, we mean how well the mechanism performs with respect to the mechanism design objective. For example, if our objective is to maximize the expected revenue, then a mechanism's performance is the expected revenue under it.

redistribution mechanism for public project problem was first studied in Guo *et al.* [9]. Naroditskiy *et al.* [18] and Guo [3] later studied competitive VCG redistribution mechanism for public project problem. Our new technique is able to achieve more competitive mechanisms than previously proposed competitive mechanisms.

Below we summarize our new technique. We focus on mechanism design with worst-case objectives (*e.g.*, maximizing competitive ratios).

– Our new technique aims to speed up CFAMD. We still need to manually select a parameterized family of strategy-proof mechanisms.
  Like under CFAMD, given a set of parameters, we need to be able to evaluate the performance of the corresponding mechanism, using a calculation process called PE. In this paper, performance of a mechanism refers to its competitive ratio (to be formally defined later).
  Under classic CFAMD, PE needs to be fast because we have to frequently call it whenever we change the mechanism parameters. For example, if we use a simple hill-climbing algorithm for adjusting the mechanism parameters, then we need to call PE to re-evaluate the mechanism performance every time we move the parameters.
  Under our new technique, PE is allowed to be computationally expensive, as we only need to call it infrequently. It should be noted that by allowing PE to be expensive, we are essentially allowing the parameterized mechanism family to be more general than before, which then means that we are able to achieve better mechanism objectives.
– Under CFAMD, we optimize over the mechanism parameters and solve for the optimal mechanism (the optimal set of parameters). For example, we may start from some parameter values, and run a hill-climbing process to obtain a final set of parameter values.
  We observe that we do not need to calculate the *exact* performance of the "intermediate" mechanisms during the optimization process. It is much faster to rely on *performance estimation* during optimization.
  For worst-case objectives (*e.g.*, maximizing competitive ratios), to calculate a mechanism's exact worst-case performance, we either need to go over all possible type profiles, which gets very expensive (there are usually exponential number of type profiles), or we need to rely on complex analysis to pinpoint the worst cases, which is not always achievable.[4]
  One the other hand, it is much easier to *estimate* a mechanism's worst-case performance instead: we focus our attention to a set of *profile samples*, by assuming that the mechanism's worst-case performance (among all type profiles) is close to the worst-case (among only the profile samples).
– The larger the set of profile samples gets, the more accurate our estimation becomes. On the other hand, the larger the set gets, the more expensive is the estimation process. We want to keep the set small, to ensure fast computation. As a result, we need to carefully select a small set of "useful" profile samples that makes the estimation accurate.

[4] Even if it is achievable, the overall process may still be computationally expensive.

We start from an initial set of profile samples S. The initial value of S is not important. We can randomly generate a small set of type profiles to be S.
Based on S, we can estimate a mechanism's worst-case performance (by assuming that the worst-case among S is close to the worst-case among all type profiles). We denote the estimation process by PE(S). Using PE(S), we solve for the optimal $M(p_1, p_2, \ldots, p_t)$ that has the best worst-case performance considering only type profiles in S. Let $M(p_1, p_2, \ldots, p_t)$'s worst-case performance be α_S. We keep in mind that α_S is obtained via estimation, by only considering type profiles among S. This optimization process is easier because we resort to performance estimation instead of exact performance evaluation. We then run exact performance evaluation (PE) on the resulting mechanism $M(p_1, p_2, \ldots, p_t)$ to calculate its exact worst-case performance α and the worst-case type profile θ.
If α and α_S are very close, then we stop our algorithm and consider α to be the resulting performance. The reason is that α_S is obtained by only considering the profile samples, so α_S can serve as an upperbound on the actual worst-case performance. If α is close to the upperbound, then it is already accurate.
α_S may also be quite off from α, because α_S was obtained via estimation, and in our estimation, we failed to consider θ. That is, the type profile θ is an important extreme point that we should consider. We keep a record of the best α achieved and denote it as α^*. We add θ into S and repeat the parameter optimization until we cannot improve α^* any further.

Our new technique speeds up the original CFAMD approach for two reasons. One reason already mentioned above is that, for the most part, we replaced the expensive PE process by the cheaper PE(S) process. Another equally import reason is that if we focus on type profiles from a small set S, it is also much easier to solve for the optimal mechanism parameters. For the mechanism design problem studied in this paper, without the new technique, parameter optimization is an unstructured multi-dimensional optimization problem, which requires methods such as hill-climbing. With the new technique, we can use a linear program to help obtain the optimal parameters. (To optimize a list of parameters, LP is much easier than less-structured methods such as hill-climbing.)

2 Model Description

For the rest of this paper, we focus on designing competitive VCG redistribution mechanism for public project problem. With the help of our new technique, we are able to discover mechanisms with better competitive ratios compared to previous results.

2.1 VCG Redistribution Mechanism for Public Project Problem

The public project problem is a classic mechanism design model [14–16]. In this model, n agents need to decide whether or not to build a public project (*e.g.*, a new airport). There are only two outcomes: *build* or *not build*. The cost of the project is 1. If the agents decide to build, then everyone benefits from the project. We use θ_i $(0 \leq \theta_i \leq 1)$ to denote agent i's valuation for the project (if it is built).[5] If the agents decide not to build, then everyone saves her share of the cost, which equals $1/n$. In summary, for agent i, her valuation is either θ_i (if the project is built) or $1/n$ (if the project is not built).

Ideally, an efficient decision for the public project problem is that we should build if and only if the agents' total valuation for the project exceeds the total cost (*e.g.*, $\sum \theta_i \geq 1$). Of course, in order to calculate the agents' total valuation, we need the agents to truthfully reveal their private types. For the above reasoning, we focus on mechanisms that are efficient and strategy-proof. It turns out that for the public project problem, a mechanism is efficient and strategy-proof if and only if it is a Groves mechanism [11].

For the public project problem, Groves mechanisms[6] work as follows:

- Build if and only if $\sum_i \theta_i \geq 1$.
- Agent i receives $\sum_{j \neq i} \theta_j - h(\theta_{-i})$ (payment) if the decision is to build. Agent i receives $(n-1)/n - h(\theta_{-i})$ if the decision is not to build. Here, h is an arbitrary function and θ_{-i} refers to the types from the agents other than i herself.

A Groves mechanism is characterized by the function h. Every Groves mechanism is efficient and strategy-proof. Another mechanism property that we wish to enforce in this paper is the non-deficit property. That is, we do not require external funding to run the mechanism. A Groves mechanism is non-deficit if and only if the total payment received is non-positive, that is:

If the decision is to build ($\sum_i \theta_i \geq 1$), then

$$\sum_i \sum_{j \neq i} \theta_j - \sum_i h(\theta_{-i}) = (n-1) \sum_i \theta_i - \sum_i h(\theta_{-i}) \leq 0$$

If the decision is not to build ($\sum_i \theta_i \leq 1$), then

$$\sum_i (n-1)/n - \sum_i h(\theta_{-i}) \; = \; (n-1) - \sum_i h(\theta_{-i}) \leq 0$$

Combining the above, we have that a Groves mechanism is non-deficit if and only if for all type profiles

$$(n-1) \max\{\sum_i \theta_i, 1\} \leq \sum_i h(\theta_{-i})$$

[5] Naroditskiy *et al.* [18] proved that it is without loss of generality to assume that every agent's type is bounded above by the project cost, as any competitive mechanism can be easily generalized to cases without this constraint, and still keeps the same competitive ratio.

[6] For our setting, it is without loss of generality to focus on anonymous mechanisms [9].

We adopt the notation from Naroditskiy *et al.* [18]. Given a type profile θ, we use $S(\theta)$ to denote $\max\{\sum_i \theta_i, 1\}$. The non-deficit property is then

$$\forall\theta, (n-1)S(\theta) \leq \sum_i h(\theta_{-i})$$

Under the Groves mechanism, agent i's utility equals $\sum_i \theta_i - h(\theta_{-i})$ if the decision is to build, and her utility equals $1 - h(\theta_{-i})$ if the decision is not to build. That is, agent i's utility equals $S(\theta) - h(\theta_{-i})$. The social welfare (total utility considering payments) is then

$$nS(\theta) - \sum_i h(\theta_{-i})$$

Non-deficit Groves mechanisms are sometimes also called the VCG redistribution mechanisms. In this paper, we focus on non-deficit mechanisms, so we will use the two terms ("VCG redistribution mechanisms" and "Groves mechanisms") interchangeably.[7]

Even though every VCG redistribution mechanism is efficient, the agents' social welfare under a VCG redistribution mechanism can be very low due to the high payments. For example, the VCG mechanism (*aka*, the Clarke mechanism) itself is a VCG redistribution mechanism, characterized by $h(\theta_{-i}) = \max\{\sum_{j\neq i} \theta_j, (n-1)/n\}$. Let us consider the type profile $(1, 0, \ldots, 0)$. The social welfare under this type profile equals

$$nS(\theta) - \sum_i h(\theta_{-i}) \;=\; n - (n-1)/n - \sum_{i>1} 1 \;=\; 1/n$$

That is, the social welfare approaches 0 as n approaches infinity. In this paper, we aim to find VCG redistribution mechanisms that maximize social welfare for the public project problem.

2.2 Competitive VCG Redistribution Mechanism for Public Project Problem

VCG redistribution mechanisms have been widely studied for various resource allocation settings, such as multi-unit auctions and heterogeneous-item auctions [2,4,6,8,9,17]. They have also been studied for the public project problem [3,9,10,18]. Among the previous studies, Naroditskiy *et al.* [18] and Guo [3]

[7] The name "VCG redistribution mechanisms" emphasizes on the fact that a non-deficit Groves mechanism can be interpreted as a two step process, where we first allocate and charge payments according to the VCG mechanism (*aka*, the Clarke mechanism), and then we redistribute the VCG payments back to the agents. An agent's redistribution does not depend on her own bid, which ensures that the redistribution amount does not affect an agent's incentives. We also need to ensure that the agents' total redistribution received is never more than the total VCG payment collected, which is to ensure the non-deficit property.

are the closest related to this paper. Naroditskiy *et al.* [18] first studied "competitive" VCG redistribution mechanisms for public project problem, by borrowing the definition of competitive VCG redistribution mechanism from Moulin [17]. Moulin [17] observes that the notion of maximizing social welfare is not a well-defined one, as social welfare depends on the agents' type profile. A mechanism may achieve high social welfare for some type profiles, but not for the others. This leads to the study of competitive VCG redistribution mechanisms.

Definition 1 *(Due to Moulin* [17]*).* A VCG redistribution mechanism is competitive if its achieved social welfare guarantees a constant fraction of the first-best social welfare, no matter what the actual type profile is. The constant fraction is called the mechanism's *competitive ratio.*

The first-best social welfare is defined as the optimal social welfare assuming that the agents act unselfishly.

Under the public project problem, if the agents act unselfishly, then the first-best social welfare is achieved by the following mechanism:[8]

- Build the project if and only if the agents' total valuation exceeds the total cost.
- The agents don't have to pay any payments because no payments are needed to enforce strategy-proofness for unselfish agents.

As a result, the first-best social welfare under a type profile θ is simply $S(\theta)$. A VCG redistribution mechanism characterized by function h is α-competitive if and only if

$$\forall \theta, nS(\theta) - \sum_i h(\theta_{-i}) \geq \alpha S(\theta)$$

Combining the non-deficit constraint, we have

$$\forall \theta, (n-1)S(\theta) \leq \sum_i h(\theta_{-i}) \leq (n-\alpha)S(\theta) \tag{1}$$

We focus on mechanisms that are strategy-proof, efficient, and non-deficit, which implies that we are focusing on the VCG redistribution mechanisms. Since every VCG redistribution mechanism is characterized by a h function, our problem is then to design h that satisfies Inequality 1 and maximizes the competitive ratio α.

Naroditskiy *et al.* [18] derived the optimal VCG redistribution mechanism for the case of 3 agents. The optimal mechanism is characterized by the following h function, and the optimal competitive ratio is $\alpha = 2/3$.

$$h(\theta_{-i}) = \frac{5}{6}T(\theta_{-i}, 1) + \frac{2}{3}T(\theta_{-i}, \frac{1}{2}) - \frac{1}{3}T(\theta_{-i}^1, \frac{1}{2}) - \frac{1}{3}$$

[8] It should be noted that this mechanism is only used as a benchmark. The competitive ratio can be interpreted as the ratio between "best social welfare for selfish agents" and "best social welfare for unselfish agents".

Here, $T(\theta_{-i}, b)$ is defined as the maximum between "the sum of the types among θ_{-i}" and constant b. $T(\theta^1_{-i}, b)$ is defined as the maximum between "the highest type among θ_{-i}" and constant b.

Unfortunately, the technique used for deriving the above does not generalize to cases with more than three agents. For more than three agents, Naroditskiy *et al.* [18] proposed a *conjectured* upper bound on the optimal competitive ratio. Naroditskiy *et al.* [18] also proposed heuristic-based mechanisms. These mechanisms were *conjectured* to be competitive for *4 to 6 agents*.

Guo [3] proposed a VCG redistribution mechanism that is proven to be competitive. When n approaches infinity, the competitive ratio is at least

$$\lim_{n\to\infty}(1 - U(n) + L(n)) = 0.102$$

where

$$U(n) = \frac{1}{n-1} + \frac{n-1}{4n} + \frac{4(n+1)^3}{27n(n-1)^2}$$

$$L(n) = \min\{\frac{1}{n-1} - \frac{1}{n} - \frac{(n-1)^2}{4n^2}, \frac{1}{n-1} + \frac{1}{2n} - \frac{1}{2}\} - \frac{n-2}{n(n-1)}$$

It should be noted that the above bound is only useful for large n. When $n = 3$, $1 - U(n) + L(n)$ is negative. Nevertheless, for small n, the mechanism proposed by Guo [3] can be evaluated numerically to show that they have positive competitive ratios.

Using our new technique, we find another optimal mechanism for the case of three agents. That is, we discover that the optimal mechanism is not unique when $n = 3$. For $n > 3$, we found competitive VCG redistribution mechanisms with better competitive ratios. We leave the details to Sect. 4.

3 Technical Details

In this section, we present all technique details for solving for competitive VCG redistribution mechanisms for the public project problem. We will make use of our new technique for speeding up CFAMD by sampling worst-case type profiles.

3.1 Parameterized Family

We recall that for three agents, Naroditskiy *et al.* [18] showed that the following mechanism has the optimal competitive ratio:

$$h(\theta_{-i}) = \frac{5}{6}T(\theta_{-i}, 1) + \frac{2}{3}T(\theta_{-i}, \frac{1}{2}) - \frac{1}{3}T(\theta^1_{-i}, \frac{1}{2}) - \frac{1}{3}$$

$T(\theta_{-i}, b)$ is defined as the maximum between "the sum of the types among θ_{-i}" and constant b. $T(\theta^1_{-i}, b)$ is defined as the maximum between "the highest type among θ_{-i}" and constant b.

We introduce a slightly modified notation $T(\theta^a_{-i}, b)$, which is defined as the maximum between "the sum of the a highest types among θ_{-i}" and constant b. Here, a is an integer between 1 and $n-1$. Obviously, $T(\theta_{-i}, b)$ is just $T(\theta^{n-1}_{-i}, b)$.

We introduce the following parameterized family of mechanisms:

$$h(\theta_{-i}) = \sum_{t=1}^{k} c_t T(\theta^{a_t}_{-i}, b_t) + c_0$$

As shown above, a mechanism is characterized by $3k+1$ parameters: the c_t, the a_t, and the b_t. The a_t are integers between 0 and $n-1$. The b_t are nonnegative. It is obvious to see that the aforementioned optimal mechanism for three agents belongs to this family. As will be demonstrated in Sect. 4, we are able to identify mechanisms with good competitive ratios by focusing on this family. The family also satisfies the presumption that given a mechanism inside the family, we are able to evaluate its competitive ratio.

3.2 Performance Evaluation

In this subsection, we discuss how to evaluate the performance of a mechanism, given its parameters.

Let M be the mechanism to be evaluated. Let M's parameters be as follows

$$h(\theta_{-i}) = \sum_{t=1}^{k} c_t T(\theta^{a_t}_{-i}, b_t) + c_0$$

According to Inequality 1, h must satisfy

$$\forall \theta, (n-1)S(\theta) \leq \sum_i h(\theta_{-i}) \leq (n-\alpha)S(\theta)$$

We define value Δ_L as follows:

$$\Delta_L = \max_\theta \left((n-1)S(\theta) - \sum_i h(\theta_{-i}) \right) \tag{2}$$

Δ_L is M's maximum deficit. If $\Delta_L > 0$, then it means that sometimes M incurs deficit. To remedy this, we can increase the constant payment term c_0 by Δ_L/n and focus on the new mechanism instead. Similarly, if $\Delta_L < 0$, then it means that there is always positive surplus. To maximize social welfare, we could decrease c_0 by $|\Delta_L|/n$. That is, it never hurts to replace c_0 by $c_0 + \frac{\Delta_L}{n}$. So for any M, before evaluating its competitive ratio, we first calculate Δ_L and then update the constant term of h. After the update, the maximum deficit is exactly 0. That is, after the update, the mechanism is sure to be non-deficit.

Of course, evaluating Δ_L is not an easy task. We use Θ to denote the set of all type profiles. Without loss of generality, we assume $1 \geq \theta_1 \geq \theta_2 \geq \ldots \geq \theta_n \geq 0$.

$$\Theta = \{(\theta_1, \theta_2, \ldots, \theta_n) | 1 \geq \theta_1 \geq \theta_2 \geq \ldots \geq \theta_n \geq 0\}$$

Define

$$\theta^L = \arg\max_{\theta \in \Theta} \left((n-1)S(\theta) - \sum_i h(\theta_{-i}) \right)$$

Define B to be whether or not the project is built under θ^L. That is, B is True if $\sum_i \theta_i^L \geq 1$. B is False otherwise.

Besides the constant term c_0, h contains k terms. For agent i, the t-th term is $c_t T(\theta_{-i}^{a_t}, b_t)$. We observe that $\theta_{-i}^{a_t}$ is nondecreasing in i. Define G_t to be the following:

$$G_t = \left| \{ i | T(\theta_{-i}^{a_t}, b_t) = b_t, i \in \{1, 2, \dots, n\}\} \right|$$

G_t is an integer. For i from 1 to G_t, $T(\theta_{-i}^{a_t}, b_t)$ is simply b_t. For i from $G_t + 1$ to n, $T(\theta_{-i}^{a_t}, b_t)$ is just the sum of the a_t highest types among θ_{-i}.

Given θ^L, it is easy to calculate $(B, G_1, G_2, \dots, G_k)$. On the other hand, given $(B, G_1, G_2, \dots, G_k)$, we can also calculate θ^L using the following linear program.

Variables: $1 \geq \theta_1 \geq \theta_2 \geq \dots \geq \theta_n \geq 0$
Maximize: $(n-1)S(\theta) - \sum_i h(\theta_{-i})$
Subject to:
If B is True, then $\sum_i \theta_i \geq 1$.
If B is False, then $\sum_i \theta_i \leq 1$.
For all t, $\sum \theta_{-G_t}^{a_t} \leq b_t$ and $\sum \theta_{-(G_t+1)}^{a_t} \geq b_t$

The above is a linear program with n variables. The objective function involves S and h. Since we assume $(B, G_1, G_2, \dots, G_k)$ is known. Both S and h can be rewritten as linear functions. The constraints are for ensuring that the assumed values of $(B, G_1, G_2, \dots, G_k)$ are not violated. There are $2k+1$ constraints, besides the constraints on the range and relative order of the θ_i.

We showed how to calculate θ^L given $(B, G_1, G_2, \dots, G_k)$. But how do we calculate θ^L without the values of B and the G_t?

B can take two possible values. G_t can take $n+1$ possible values (0 to n). The total possible values for $(B, G_1, G_2, \dots, G_k)$ is $2(n+1)^k$. We can go over all possible values of $(B, G_1, G_2, \dots, G_k)$ in order to find θ^L. Solving one linear program takes $O(poly(n))$ complexity. We need to solve $O(n^k)$ linear programs. Altogether, the complexity is $O(n^k poly(n))$, which is feasible when k is small.

We then define function Δ_R to be as follows:

$$\Delta_R(\alpha) = \max_{\theta} \left(\sum_i h(\theta_{-i}) - (n-\alpha)S(\theta) \right) \tag{3}$$

Given a mechanism M (updated according to the calculation of Δ_L), if $\Delta_R(\alpha) \leq 0$, then we can claim that M's competitive ratio is at least α. Since $\Delta_R(\alpha)$ is a monotone function and $0 \leq \alpha \leq 1$, we can use binary search to solve the equation $\Delta_R(\alpha) = 0$. Binary search only needs ten iterations to ensure an error of at most one tenth of one percent ($\frac{1}{2^{10}}$). We can reapply our method for calculating Δ_L to the calculation of $\Delta_R(\alpha)$. Suppose we only run ten iterations of binary search, the complexity of this step is still $O(n^k poly(n))$. We use θ^R to denote the worst-case type profile when we stop the algorithm for calculating $\Delta_R(\alpha)$.

In summary, given a mechanism, we are able to calculate its competitive ratio. The time complexity is $O(n^k poly(n))$. If k is slightly large, then the process becomes very expensive. Fortunately, as mentioned in Subsect. 1.3, we only need to call this process infrequently during the whole algorithm. As a side product of the optimization process, we identified two worst-case type profiles θ^L and θ^R. In the next subsection, we describe how we can use these worst-case type profiles to estimate mechanism performance, which is for speeding up CFAMD.

3.3 Performance Estimation and Parameter Optimization

Due to the fact that PE (evaluating a mechanism's competitive ratio) takes $O(n^k poly(n))$ complexity, we resort to estimation when optimizing over the parameters, to avoid calling PE frequently.

To optimize within the parameterized family, we break down our whole task into two subtasks.

- Task 1: Choose k terms of form $T(\theta_{-i}^{a_t}, b_t)$. Each term is characterized by an integer a_t and a nonnegative real value b_t.
- Task 2: Choose k coefficients (the c_t) for combining these terms, and also choose c_0.

We first focus on Task 2. We assume we have already completed Task 1. That is, given k terms, we want to set the c_t only. This again can be done via a linear program.

As mentioned in Subsect. 1.3, we initialize our algorithm with a set of profile samples. We will use these profile samples to optimize over the mechanism parameters. Once we find an optimal mechanism, which is based estimation, we evaluate its exact competitive ratio using the expensive process described in the previous subsection. If the estimated competitive ratio is quite off from the actual competitive ratio, then we add the corresponding worst-case type profiles (θ^L and θ^R) into our samples. Overtime, we have better and better estimations (and as a result, better and better competitive ratios).

For our problem, we start with only $n+1$ type profiles:

$$S = \{(\underbrace{1, \ldots, 1}_{x}, 0, \ldots, 0) | x \in \{0, 1, \ldots, n\}\}$$

Given a set of profile samples S and k terms, to set the c_i, we use the following linear program:

Variables: $c_0, \ldots, c_k$ and α
Maximize: α
Subject to:
For every $\theta \in S$,

$$(n-1)S(\theta) \leq \sum_i \left(\sum_{t=1}^{k} c_t T(\theta_{-i}^{a_t}, b_t) + c_0 \right) \leq (n-\alpha)S(\theta)$$

The above is a linear program because for every $\theta \in S$, $S(\theta)$ is a constant. For all i, all t, all $\theta \in S$, $T(\theta_{-i}^{a_t}, b_t)$ is also a constant. The above linear program involves only $k+1$ variables, but it has a lot of constraints if S is large. To keep the size of S small, when we add type profile θ_{new} to S, we remove all profiles in S that are close to θ_{new}. Two type profiles' difference is defined as $Diff(\theta, \theta') = \sum_i |\theta_i - \theta'_i|$. We will show in Sect. 4 that in our experiments, the size of S is usually only a few hundred before the algorithm terminates. So it is generally quite cheap to solve the above linear program.

Above we showed how to carry out Task 2 given the result of Task 1. Next, we go back to Task 1. We approach Task 1 as follows:

- **Expansion and Consolidation:** We randomly generate $2k$ terms and use the above linear program to set the optimal coefficients $(c_1, c_2, \ldots, c_{2k})$. We sort the c_i according to their absolute values. We drop terms with low absolute values. The rationale is that if $|c_t|$ is small, then the t-th term is not playing an important role in our mechanism. We repeat this process: expand the list of terms by adding randomly generated ones and then drop those less important terms determined by the linear program. The end result is a list of k "important" terms.
- **Hill Climbing:** After the above expansion and consolidation process, we end up with k terms. We perform "final touch" by running hill climbing on these terms' parameters (namely, the a_t and the b_t).

4 Numerical Results

For three agents, using our technique, besides the optimal mechanism described in Naroditskiy *et al.* [18]. We identified another optimal mechanism:

$$h(\theta_{-i}) = T(\theta_{-i}, \frac{2}{3}) + \frac{1}{2}T(\theta_{-i}, 1) - \frac{1}{2}T(\theta_{-i}^1, \frac{2}{3}) - \frac{1}{6}$$

For four or more agents, we compare our mechanism to the following previous results:

- The heuristic-based SBR mechanism [18]: This mechanism's competitive ratio was numerically calculated for four to six agents. The authors discretized an agent's type space into $\{0, 1/N, 2/N, \ldots, 1\}$, and examined all type profiles

in order to calculate a mechanism's competitive ratio. For at most six agents, the authors were able to push N large enough for the resulting competitive ratio to be numerically stable.
– The ABR[9] mechanism [3]: This mechanism's competitive ratio was proven to be above 0.102 when n goes to infinity. For small n, we can numerically calculate its competitive ratio.
– The *conjectured* optimal upper bound on a mechanism's competitive ratio, proposed in [18]: It should be noted that the upper bound has been proven to be valid. The authors conjectured that it is strict.

We present the competitive ratios of different mechanisms in the following table. It should be noted that for our mechanism, the competitive ratios presented are the ratios we have achieved after running our algorithm for a reasonable amount of time. When presenting our mechanisms, the integer inside the parenthesis is the number of samples in S when the algorithm first identifies the first two significant digits of the competitive ratio.[10] In our simulation, we picked $k = 5$.

n	SBR	ABR	Our mechanism	Upper bound
3	0.333	0.334	0.667 (38)	0.667
4	0.354	0.459	0.600 (603)	0.666
5	0.360	0.402	0.545 (298)	0.714
6	0.394	0.386	0.497 (444)	0.868
7	n too large	0.360	0.465 (514)	0.748
8	n too large	0.352	0.444 (276)	0.755
9	n too large	0.339	0.422 (621)	0.772
10	n too large	0.336	0.405 (184)	0.882

5 Conclusion

We proposed a new technique for speeding up Computationally Feasible Automated Mechanism Design (CFAMD) for worst-case objectives. We demonstrated the effectiveness of our approach by applying it to the design of competitive VCG redistribution mechanism for public project problem. With our new technique, we achieved better competitive ratios than previous results.

[9] Guo [3] proposed only one mechanism, which is based on averaging the VCG payments. Therefore, we call the proposed mechanism average-based redistribution (ABR) mechanism.

[10] For example, on a i7-4770 desktop, for $n = 3$, it takes a few seconds to obtain a mechanism with competitive ratio at least 0.66, but it takes a lot longer to push for more significant digits.

Acknowledgment. This work is supported by Research Initiative Grant of Sun Yat-Sen University under Project 985 and Australian Research Council Discovery Project DP150104871.

References

1. Conitzer, V., Sandholm, T.: Complexity of mechanism design. In: Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI), Edmonton, Canada, pp. 103–110 (2002)
2. Gujar, S., Narahari, Y.: Redistribution mechanisms for assignment of heterogeneous objects. J. Artif. Intell. Res. **41**, 131–154 (2011)
3. Guo, M.: Competitive VCG redistribution mechanism for public project problem. In: Baldoni, M., Chopra, A.K., Son, T.C., Hirayama, K., Torroni, P. (eds.) PRIMA 2016. LNCS, vol. 9862, pp. 279–294. Springer, Cham (2016). doi:10.1007/978-3-319-44832-9_17
4. Guo, M., Conitzer, V.: Worst-case optimal redistribution of VCG payments in multi-unit auctions. Games Econ. Behav. **67**(1), 69–98 (2009)
5. Guo, M., Conitzer, V.: Computationally feasible automated mechanism design: General approach and case studies. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), Atlanta, GA, USA, pp. 1676–1679 (2010). NECTAR track
6. Guo, M., Conitzer, V.: Optimal-in-expectation redistribution mechanisms. Artif. Intell. **174**(5–6), 363–381 (2010)
7. Guo, M., Conitzer, V.: Strategy-proof allocation of multiple items between two agents without payments or priors. In: Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Toronto, Canada, pp. 881–888 (2010)
8. Guo, M., Conitzer, V.: Better redistribution with inefficient allocation in multi-unit auctions. Artif. Intell. **216**, 287–308 (2014)
9. Guo, M., Markakis, E., Apt, K.R., Conitzer, V.: Undominated groves mechanisms. J. Artif. Intell. Res. **46**, 129–163 (2013)
10. Guo, M., Naroditskiy, V., Conitzer, V., Greenwald, A., Jennings, N.R.: Budget-balanced and nearly efficient randomized mechanisms: Public goods and beyond. In: Proceedings of the Seventh Workshop on Internet and Network Economics (WINE), Singapore (2011)
11. Holmström, B.: Groves' scheme on restricted domains. Econometrica **47**(5), 1137–1144 (1979)
12. Likhodedov, A., Sandholm, T.: Methods for boosting revenue in combinatorial auctions. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), San Jose, CA, USA, pp. 232–237 (2004)
13. Likhodedov, A., Sandholm, T.: Approximating revenue-maximizing combinatorial auctions. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), Pittsburgh, PA, USA (2005)
14. Mas-Colell, A., Whinston, M., Green, J.R.: Microeconomic Theory. Oxford University Press, New York (1995)
15. Moore, J.: General Equilibrium and Welfare Economics: An Introduction. Springer, Heidelberg (2006). doi:10.1007/978-3-540-32223-8
16. Moulin, H.: Axioms of Cooperative Decision Making. Cambridge University Press, Cambridge (1988)

17. Moulin, H.: Almost budget-balanced VCG mechanisms to assign multiple objects. J. Econ. Theor. **144**(1), 96–119 (2009)
18. Naroditskiy, V., Guo, M., Dufton, L., Polukarov, M., Jennings, N.R.: Redistribution of VCG payments in public project problems. In: Proceedings of the Eighth Workshop on Internet and Network Economics (WINE), Liverpool (2012)