The Complexity of Mechanism Design Approximation

Completing the Circle of Reductions Between Mechanism and Algorithm Design

Natalie Collina

May 2019

Advised by Matt Weinberg

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Arts in Computer Science at Princeton University

Abstract

In the field of theoretical computer science we often reason about algorithm design, the process of constructing solutions for problems based upon fixed input. In real-world situations, however, the structure of an algorithm often impacts the way that agents interact with it. Mechanism design describes the problem of constructing algorithms that consider strategic input. Many important problems in fields from auction theory to policy development necessitate mechanisms that guarantee good behavior. Thus, understanding how to solve these problems, and how hard it is to solve them, is a crucial area of research.

In this paper, we strengthen and extend results found in Matthew Weinberg's Algorithms for Strategic Agents [1]. Weinberg provides a framework for transforming mechanism design problems into algorithmic form. His work centers upon a reduction from a mechanism design approximation problem (BMeD) to an algorithmic problem (GOOP). Furthermore, he provides a hardness of approximation result for some valuation classes. However, this reduction was not shown to be tight. Here we complete the circle of reductions between mechanism design and algorithm design that Weinberg worked towards, guaranteeing that the original reduction does not increase the complexity of the original problem. We do so through an approximation-preserving reduction between two related problems, ODP and SADP. In addition, we provide more principled and specific hardness of approximation results for multiple valuation classes.

Acknowledgements

I have reflected countless times over the past two semesters on how lucky I am to have Matt Weinberg as my advisor. He is an encouraging, responsive mentor and a brilliant intellectual resource. I looked forward to our meetings each week, where he would always listen attentively to my ideas and bring fresh ideas of his own. Matt taught me about mechanism design and approximation-preserving reductions, but throughout the year he did something even more important–he thoughtfully and deliberately taught me how to do research. Matt also served as an informational resource and sounding board for my plans for the future. I hope to attend graduate school within the near future, and that sustained interest has much to do with him. Matt Weinberg has been a truly positive force throughout my final year at Princeton, and I could not imagine a better advisor.

The friends I have made throughout my time here have contributed immeasurably to this thesis. Sometimes this contribution has been through explicit conversations about research directions, but often it has been through seemingly unrelated conversations that have challenged my assumptions and expanded my horizons. Through these conversations, I have become a better researcher and thinker. Thank you to Hemani, Kara, Madeleine, Emily, Mrudhula and many others. I want to thank them for their support, their passion, their boldness, and most of all for being genuinely wonderful human beings.

I would be remiss not to thank Scully Co-op. The majority of work for this thesis was completed in the common room outside of the Scully kitchen, and the presence of people such as Lilly, Austin, Will and Debo made the process infinitely more enjoyable. I never expected to find a community at Princeton that brought me so much joy on a day-to-day basis. Thank you to everyone in the Co-op for making Scully 319 my home away from home.

Finally, I would like to thank my family for encouraging my curiosity, for teaching me and for loving me. Thank you to Mom, Dad, Lu, Jared, Catherine, Mark and everyone else. I would not be where I am today without you.

I am immensely grateful for my time here at Princeton. For the past four years, my primary responsibilities have been to learn, explore and create. I can't think of a better gift.

Contents

1 Introduction

1.1 Mechanism Design

A typical algorithmic game theory problem involves deciding on a rational strategy to maximize reward when faced with a set of rules. Mechanism design is sometimes referred to as "reverse game theory," as the goal is to construct a set of rules so that rational agents acting upon these rules will behave in particular ways. Many problems that seem at first glance to be algorithmic actually necessitate a mechanism design solution. Consider the simple case of a vendor selling chocolate bars. The cost to produce each chocolate bar is c, and vendor would like to maximize her profit. Each time a buyer approaches, she asks what his value v is for the item–in other words, how much he would be willing to pay for it. If $v \geq c$, she sells the item at price v. Otherwise she does not sell the item.

While this algorithm seems intuitive, it is predicated on the assumption that the buyer will act honestly. In practice, buyers are interested in maximizing their own payoff. If a buyer has knowledge of the scheme ahead of time and has value $v > c$, his best strategy is to lie and report $v = c$. Assuming all buyers act rationally, the vendor will never make a profit. In order to run a successful business, the vendor must think not only about how to respond to honest input but also how to design a mechanism that incentivizes honest input. This problem of developing a better pricing scheme is an example of mechanism design.

Mechanism design has many real-world applications, including in the field of policy. Thoughtful regulatory policy-making can be thought of as a mechanism design problem. When lawmakers craft regulations, their goal is to encourage a certain group of people to behave in a particular way. For example, when the United Nations explored carbon dioxide regulations in the 1990s, their goal was to create a policy that would minimize emissions by companies. Simply placing a tax on emissions might not have been sufficient–perhaps the companies would still benefit from continuing their current behavior, or perhaps their optimal action would be to lie about their adherence to the regulations. The committee wanted to design a regulatory framework that provably incentivized good behavior. Researchers drew upon mechanism design theory to develop a cap-and-trade mechanism that was a central component of the Kyoto Protocol [2]. Considering the objectives and potential actions of those being regulated is crucial in constructing robust policies, and mechanism design theory provides a framework for exactly this type of reasoning.

In this paper, we consider mechanism design problems through the lens of auction theory. In an auction, bidders have certain preferences over items and are willing to pay different prices for different item combinations. The objective of each of these bidders is to maximize her profit. The auctioneer, who sets the rules for the auction (the mechanism), also has an objective. The most commonly examined mechanism objectives in this context are maximizing welfare (the total value given to bidders) and maximizing profit. The mechanism design problem here is developing a set of auction rules that achieve the auctioneer objective.

Unlike algorithmic problems, mechanism design problems have a multi-step nature: first a mechanism designer is given input describing agent incentives, then they output their mechanism, then the agents provide inputs in response to this mechanism and finally the designer runs the declared mechanism on these inputs. These problems are thus less immediately intuitive than algorithmic problems.

In his 2014 PhD dissertation Algorithms for Strategic Agents (ASA), Professor S. Matthew Weinberg presented a new algorithmic framework for reasoning about mechanism design problems [1]. Weinberg defines the Bayesian Mechanism Design problem (BMeD), which describes mechanism design approximation in a randomized setting. In his paper, he shows an approximation-preserving reduction to an algorithmic problem he calls the Generalized Objective Optimization Problem (GOOP). While BMeD is structured as a mechanism design problem, GOOP is by nature an optimization problem–upon some input, it asks for a maximizing result. His paper provides the first method for viewing approximation mechanism problems as algorithmic problems in a Bayesian setting.

1.2 Goal

The goal of this paper is to extend and strengthen the results of Weinberg's ASA. In his paper, Weinberg provides an approximation-preserving reduction from mechanism design problems to a more straightforward optimization problem. However, the tightness of the reduction presented in the paper was left unresolved, leaving open the possibility that the reduction makes the underlying problem more computationally complex. Consider the case of a friend who asks for our help in solving a mechanism design problem. "Of course," we say, and we happily present the friend with Weinberg's reduction. Thanks to this reduction, our friend's problem is now in an algorithmic form. However, our friend is skeptical. "This might look more like the kind of problem I'm used to solving," they say, "and sure, it will give me the same answer as if I solved the mechanism design problem directly. But how do I know that this reduction did not make my problem harder? What if there was a polynomial-time algorithm to solve the original problem, and there is no polynomial-time algorithm for the new problem?"

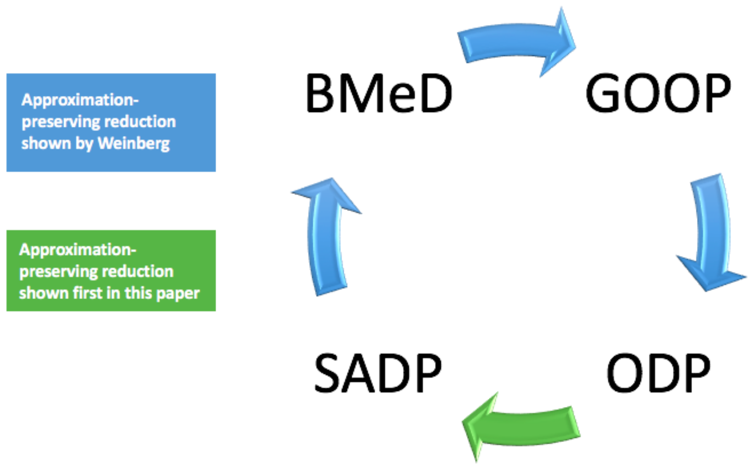
Our friend makes an astute point. Based only upon the results from ASA, we cannot guarantee that the new problem is just as easy as the original. Perhaps we reduced our problem to a harder problem. This can easily be done: for example, one can reduce 2-colorability to 3-colorability by adding a new node with edges to every node in the original graph. This is silly, of course–2-colorability has a linear-time algorithm, while 3-colorability is is NP-complete. This reduction obviously makes the problem much harder. But if 2-colorability was not a well-understood problem, we might have incorrectly thought that this reduction was useful.

In order to guarantee tightness, it is necessary to complete the circle of approximation-preserving reductions between BMeD and GOOP. In pursuit of this goal, ASA provides a reduction from GOOP to a new problem called ODP (Optimize Difference Problem), as well as a reduction from a new problem called SADP (Solve Any Differences Problem) to BMeD. Thus, finding a reduction from ODP to SADP would close the circle. The main result of this paper is a tightness guarantee for Weinberg's reduction through an approximation-preserving, valuation class-preserving reduction from ODP to SADP. This guarantee holds for most valuation classes relevant in the context of mechanism design. Thanks to this result, we can confidently tell our friend that their transformed algorithmic problem is not only easier to reason about than the original problem, but also no harder than the original problem. We conclude that Weinberg's reduction is in fact the "best" approximation-preserving reduction from mechanism design to algorithm design. Beyond comparing the complexities of these two problems, one might ask what the actual complexities of these problems are. In Weinberg's paper, a proof of hardness for BMeD was shown for submodular functions. In this paper we provide proofs of hardness for several additional valuation classes.

1.3 Outline

In section **2**, we will provide necessary background information and terms for a reader to understand our question, methodology and results. This includes explanations of relevant concepts in auction theory and complexity theory, as well as definitions of different valuation classes. In section **3** we delve more deeply into Weinberg's paper and detail various additional sources that have been useful for reaching conclusions about valuation function closures. In section **4** we delineate three potential

Figure 1: Circle of reductions as completed by this paper


Approximation-preserving reduction shown by Weinberg
Approximation-preserving reduction shown first in this paper
BMeD
GOOP
ODP
SADP

approaches to the problem at hand and describe the ultimate approach taken in this paper. In section **5** we provide a thorough investigation of problem restrictions, including an intuition for the importance of each restriction and an example of a potential solution which violates it. In section **6** we introduce three solutions which each preserve some valuation classes, ignoring for the time being C-compatibility. At the end of the section we re-introduce compatibility through slight alterations of previous solutions and show that these alterations do not cause the model to violate restrictions. In section **7** we introduce a hardness result for matroid valuation functions and all classes which contain matroid valuation functions, including GS, submodular and subadditive. Finally, in section **8** we summarize our work and present ideas for future research on this topic. In the appendix we include negative closure results for various reduction models and valuation classes.

2 Background

2.1 Bidders and Auctions

Throughout this paper we will often refer to bidders, items and valuation functions. In an auction framework, we think of items as discrete and distinct things that can be bought and sold. A bidder is an agent who has preferences over these items. Their preferences can be described through some valuation function f. f is a function that takes as input an unordered subset of the total number of available items and outputs some value. If bidder b has valuation function f over a set of items X

and $x \subseteq X$, then if $f(x) = c$, bidder b would be willing to pay up to c for set x.

As we care most about valuation functions that model real-life scenarios, we can imagine rules which typically would govern these valuations: if a bidder would pay 5 dollars for item a, it would be strange for her to be unwilling to pay 5 dollars for the set $\{a, b\}$. Valuation functions that do not value sets at lower values when augmented are called monotone valuations. Furthermore, it is strange to reason about bidders with negative valuation classes in the context of auctions. Finally, it is reasonable to assume that bidders value the empty set at exactly 0. Throughout this paper we consider only valuation functions with these properties. Individual valuation classes have more specific structures, and will be explored in the following section.

2.2 Valuation Function Classes

We aim to provide not simply a closed circle between BMeD and ODP but a closed circle for each function valuation class. The positive result in this paper is a piecewise solution, with separate reduction models for different sets of valuation classes. As we will often make use of the definitions and characteristics for relevant valuation classes, the information will be delineated here:

Additive. The class of additive functions models bidders who value individual items independently of the other items in their set. Formally, a function f is additive iff

$$\forall S, T, S \cap T = \emptyset : f(S \cup T) = f(S) + f(T)$$

Matroid Rank. A function f is matroid rank iff it fulfills the following properties:

1) It is integer-valued on every input

2) It is submodular (see submodular definition below)

3) For any set S and item x,

$$f(S) \leq f(S \cup x) \leq f(S) + 1$$

Budget Additive. Budget additive functions model bidders with additive functions under a budget constraint. Formally, a function f is budget additive iff there $\exists$ a constant c s.t.

$$f(S \cup T) = \mathbf{min}(f(S) + f(T), c)$$

Unit Demand. Unit demand functions model bidders with capacity for a single item. If they are provided with multiple items, their value will be that of their favorite singleton item. Formally, a function f is unit demand iff

$$f(X) = \mathbf{max}_{x \in X} f(x)$$

XOS. A function f is XOS iff there $\exists$ additive functions $a_1, ..., a_m$ such that

$$f(x) = \mathbf{max}_{i \in 1...m}(a_i(x))$$

Matroid Valuation. Matroid valuation functions can be thought of as extensions of matroid rank functions, where a particular item that can be added to the matroid can have some value other than 1. Instead of having value equal to the largest valid subset of the items, a bidder considering items according to a matroid valuation function will have value equal to the most valuable valid subset of the items.

OXS. While XOS functions can be described as taking a unit-demand over additive functions, OXS functions can be described as the addition of unit-demand functions. Consider a bipartite graph G where the nodes on the left-hand side of the graph represent items in an auction. A function f is OXS iff there $\exists$ G s.t. $f(x)$ is equal to the max-weight matching in G using only the left-hand nodes in x.

Gross Substitutes. There are multiple equivalent definitions of gross substitutes, two of which we will detail here. Both will be useful in this paper for proving various positive and negative results.

Definition 1: For some price vector p and set of items x, consider a bidder who chooses an optimal set x^* over these items and their prices. Consider some item $x_i \in x^*$. A bidder's valuation f is GS iff, when the price of some subset of items s is increased and $x_i \notin s$, the new optimal set contains x_i.

Definition 2: Again consider a scenario with a price vector p. Consider the greedy strategy of choosing an optimal set: at each step with current set S, a bidder selects the item x which maximizes $f(S \cup x) - \sum_{s \in S} p(s) - p(x)$. If all such values are negative, the algorithm terminates and the current set is selected. A bidder's valuation is GS iff for any p, the greedy algorithm will terminate on the optimal item set.

This second definition indicates the importance of gross substitute functions. The class exactly describes the set of functions for which max-value sets can be found using a greedy algorithm. GS functions can be thought of as extensions of matroid valuation functions that continue to fulfill this key property. However, the nature of this extension is not well understood. Unlike the other classes listed here, there is currently no known set of operations that can be performed upon a well-formed class of functions to construct the entire set of GS functions [3]. Given the algorithmic significance and lack of understanding of the class, it is both crucial and perhaps uniquely difficult to reason about.

Submodular. Submodular functions model bidders whose marginal values for adding a particular item to a set will not increase if that set is augmented. Formally, a function f is submodular iff for any set X and any items $y, z \neq X$:

$$f(X \cup y \cup z) - f(X \cup y) \leq f(X \cup z) - f(X)$$

Subadditive. Subadditive functions model bidders whose marginal value for adding any item i to their set is no greater than i's singleton value. Formally, a function f is subadditive iff, for all disjoint sets S and T,

$$f(S \cup T) \leq f(S) + f(T)$$

Subadditive functions are the broadest class of functions which we consider in this setting.

2.3 Valuation Class Intersections

All of the classes mentioned above are contained within the class of subadditive functions. In non-subadditive functions, bidders can value two items at 0, but highly value them in combination. It is hard to imagine how we would attempt to optimize functions of this sort. Non-subadditive functions are not widely studied in auction theory because they are both difficult to reason about and unhelpful for modeling realistic scenarios.

Most of the function classes shown above can be described in a strict hierarchy.

$$\textit{Additive} \subsetneq \textit{OXS} \subsetneq \textit{Matroid Valuation} \subsetneq \textit{GS} \subsetneq \textit{XOS} \subsetneq \textit{Submodular} \subsetneq \textit{Subadditive}$$

2.4 Valuation Class Closures

As our circle of reductions must hold for specific valuation classes, we must ensure that the interesting valuation classes are closed under our transformation. A function class V is closed under some operation P iff

$$\forall f \in V : \quad P(f) \in V$$

In other words, P maps functions in V exclusively to functions in V. Note that for $V_1 \subseteq V_2$, $P(V_1) \in V_1 \not\Rightarrow P(V_2) \in V_2$, and $P(V_2) \in V_2 \not\Rightarrow P(V_1) \in V_1$.

Operation P might involve the combination of multiple functions. Consider the addition operation. We can ask whether the addition of two functions $f, g \in V$ is itself $\in V$. For most V, this will be true. For example, if $h = f + g$ and f and g are additive, then h is certainly additive, and the singleton value each x_i is $f(x_i) + g(x_i)$.

2.5 Computational Complexity and Reductions

Reductions. As a refresher, a reduction from problem A to problem B is an efficient protocol that transforms inputs to A into inputs to B in such a way that the solution output by B is the correct solution to A. A successful reduction from A to B signifies that A is no more computationally complex than B. If we would like to prove that A and B have the same computational complexity, we can perform a reduction in both directions. Alternatively, we can reduce A to B, B to some problem C, and C to A. As referenced in the introduction, the goal of this thesis is to close the circle of reductions between BMeD and GOOP. We use the approach of showing a reduction from

BMeD to GOOP to ODP to SADP to BMeD.

Hardness. One key concept in complexity theory that appears in this paper is that of computational hardness. All problems that are at least as computationally complex as NP-complete problems are NP-hard. Practically speaking, large instances of NP-hard problems are infeasible for even the fastest computers to solve in reasonable time. A hardness result for a problem precludes the development of an efficient algorithmic solution (assuming that P $\neq$ NP). This concept will be important in section **7**, when we prove hardness results for several key valuation classes. In our case, the hardness result is exponential. Thus there exists no poly-time algorithm, regardless of open questions in complexity theory.

2.6 ODP and SADP

The crux of this paper involves a reduction from a problem we call ODP to a problem we call SADP. Below, we provide problem definitions.

ODP (Optimize Difference Problem) $\mathrm{ODP}_{V,\alpha}$ takes two valuation functions f and g in valuation class V as input. It returns a set x^* such that

$$f(x^*) - g(x^*) \geq (1-\alpha)\mathbf{max}_{x\in X}(f(x) - g(x))$$

Let us provide an example of an ODP problem, using the subadditive valuation functions f and g delineated below:

Table 1: Valuation function f

Sets of size 1	Value	Sets of size 2	Value
{a}	5	{a,b}	6
{b}	2		

Table 2: Valuation function g

Sets of size 1	Value	Sets of size 2	Value
{a}	2	{a,b}	5
{b}	3		

The optimal solution to ODP will be the item set x^* that maximizes the difference between $f(x)$ and $g(x)$.

Note the distinction between this problem and the problem of allocating items to bidders to maximize the difference: in the second case, we could simply allocate all items to the bidder with valuation f and none to the bidder with valuation g. By contrast, in this case our solution must be a single set that both valuation functions are evaluated over. In this small example we can compute all possible solutions:

$$\begin{aligned} f(\emptyset) - g(\emptyset) &= 0 - 0 = 0 \\ f(a) - g(a) &= 5 - 2 = 3 \\ f(b) - g(b) &= 2 - 3 = -1 \\ f(a,b) - g(a,b) &= 6 - 5 = 1 \end{aligned}$$

The set $\{a\}$ provides the largest value, and therefore it is an optimal solution to ODP. In addition, the set $\{a,b\}$ provides an $\frac{1}{3}$-approximation for the optimal solution. Therefore both $\{a\}$ and $\{a,b\}$ are valid solutions to ODP$_\alpha$ when $\alpha \geq \frac{2}{3}$, while only $\{a\}$ is a valid solution when $\alpha < \frac{2}{3}$. The remaining sets garner non-positive values, and thus will not be valid solutions for any α. In this paper, we assume that f and g are monotone, non-negative, and evaluate to 0 at the empty set.

SADP (Solve Any Differences Problem) SADP is another optimization problem with key similarities to ODP. SADP takes n valuation functions $k_1, ..., k_n$ over a set of items as input and returns a set x' such that, for some i,

$$k_i(x') - k_{i+1}(x') \geq (1-\alpha)\mathbf{max}_{x \in X}(k_i(x) - k_{i+1}(x))$$

Essentially, SADP is given the freedom to solve one of many instances of ODP.

While a typical SADP instance might have multiple valid solutions representing optimizing sets for different $k_i - k_{i-1}$, a solution to ODP is restricted to some set optimizing $f - g$. Intuitively, a valid reduction from ODP to SADP must ensure that the difference between k_i and k_{i+1} looks similar to the difference between f and g for all i.

3 Related Work

3.1 Hardness of Revenue Maximization

Exact mechanism design solutions for Bayesian revenue maximization are known to be hard in many contexts. Hardness results exist even for seemingly simple scenarios, such as when there are a small number of possible bidder types or a small number of items. A 2018 paper by Chen, Diakonikolas, Paparas, Sun and Yannakakis examined the complexity of optimal Bayesian mechanism design for single-bidder scenarios. The problem is similar to that presented in ASA: a seller is given a probability distribution over the types of buyers who will come to their store, and they must decide how to price their items. The paper finds a polynomial-time algorithm for the case where there are only two potential buyer types (cases where support = 2). However, it proves that the case for three or more potential buyer types is NP-hard. In particular, the decision version of the problem with three or more buyer types is NP-complete [4]. Another paper by Chen at. al, On the Complexity of Optimal Lottery Pricing and Randomized Mechanisms, considers the complexity of the lottery problem, a related model in which the single buyer is given a menu with prices, items and probabilities of receiving those items depending on their purchase. They show that this problem is also NP-hard unless the polynomial time hierarchy collapses [5]. Dobzinski, Fu and Kleinberg showed that exact mechanisms for OXS valuations in particular is hard [6]. Daskalakis, Deckelbaum and Tzamos developed a framework using optimal transport theory and duality theory to provide solutions for particular optimization instances, optimal mechanisms in auctions with two items and other specific scenarios [7]. However, it has generally been proven difficult to find optimal mechanisms even when distributions have particular structures.

These hardness results demonstrate the difficulty of optimally solving mechanism design, but do not preclude the potential for an efficient approximation solution for revenue. In the case of many hard problems, there exist approximation algorithms which are significantly more efficient than optimal algorithms. In practice, getting close to the best solution is often just as useful. Dobzinski, Fu and Kleinberg investigated revenue-approximating mechanisms in a single-item scenario, where bidders are drawn from a correlated distribution. The paper showed that in this context there is an approximation mechanism that is poly-time in the size of the support of the bidder type distribution

[6].

3.2 Algorithms for Strategic Agents

Weinberg's 2014 PhD thesis, Algorithms for Strategic Agents, presented a novel framework for reasoning about mechanism design problems in an approximation setting that extend to multiple items. He achieves this by reducing approximation mechanism design problems, as defined through the problem BMeD, to GOOP, an algorithmic problem. [1]. Through this reduction, he is able to provide a hardness of approximation result for mechanism design. Results in ASA are based upon multiple research collaborations between Matt Weinberg, Yang Cai and Constantinos Daskalakis [8] [9] [10]. In particular, their 2013 paper, Understanding Incentives: Mechanism Design becomes Algorithm Design, forms the backbone for Wienberg's hardness result [11].

BMeD. BMeD takes as input a set of types T and a distribution of bidder preferences. This can be thought of as probabilistically describing the nature of the agents interacting with the mechanism. In addition, it takes as input an objective O. BMeD outputs a mechanism which is Bayesian Incentive Compatible (BIC). In other words, if all agents know the distribution of outcomes, their optimal strategy is to report their preferences truthfully. This mechanism approximately maximizes the objective.

GOOP. For a given valuation class V, GOOP takes in a multiplier $w \geq 0$, an objective function O, k valuation functions $g_1, ..., g_k \in V$, and valuation function $f \in V^x$. Here, V^x denotes the closure of V under addition and scalar multiplication. The output of GOOP$_\alpha$ is an outcome X from a space of feasible outcomes F that outputs an X such that

$$(w \cdot O((g_1, ..., g_k), X) + f(X)) \geq (1 - \alpha)(w \cdot O((g_1, ..., g_k), X^*) + f(X^*))$$

where X^* is the maximizing outcome.

Weinberg performed an approximation-preserving reduction from BMeD to GOOP, demonstrating that approximation Bayesian mechanism design can be reformulated as an algorithmic problem.

GOOP to ODP. The intuition of Weinberg's reduction from GOOP to ODP is as follows: in the context of revenue maximization, we do not need to consider O. Then this problem simply becomes that of approximately maximizing $f(X)$. Recall that $f(X)$ is a valuation function in V^x.

Thus, assuming that V is closed under addition and positive scalar multiplication, we can write f as $f_1 - f_2$, where f_1 and f_2 are in V. This optimization problem is exactly $\text{ODP}_{V,\alpha}$.

SADP to BMeD. The reduction provided from SADP to BMeD promises the following: if SADP is D-balanced and has n input functions, an α-approximation in BMeD implies an $(\alpha - \frac{(1-\alpha)D}{n-1})$-approximation to SADP. We will define D-balanced in section **5**.

Submodular Hardness Instance in ASA. In section **8** of ASA, Weinberg provides a hard SADP instance for submodular functions. Through this, he demonstrates that SADP and thus BMeD are hard problems in some instances. However, this instance does not correspond to any ODP instance. Furthermore, this function is not gross substitute or matroid valuation. This paper takes a more principled approach by providing a reduction that maps ODP problems to BMeD problems. We are then able to use this framework to show an exponentially hard ODP instance that can be solved using SADP. Through this, we are able to guarantee that SADP, and therefore BMeD, is not only hard for some instances, but hard for instances that correspond to GOOP solutions–and thus model actual mechanism design problems. Furthermore, our reduction holds for more specific valuation classes.

3.3 Valuation Class Closures

In order to make claims about valuation class closures, we draw upon previous research on the behavior of valuation classes under operations. The behavior of GS functions in particular is an active area of research. Dr. Renato Pas Lemme has performed extensive research on the structure and behavior of the GS class. In his 2018 paper On the Construction of Substitutes, he attempts to generate a constructive description of the class [3]. While this result remains open, he generates many strong conclusions useful for this paper, including some notable limitations of GS functions. In particular, GS functions are not closed under addition. They are not even closed under the addition of matroid rank functions, which are a strict subset of the GS class. The paper also delineates some operations under which GS functions are closed, such as affine transformations and strong quotient sum.

Another Pas Lemme paper, Gross Substitutability: An Algorithmic Survey, proves that GS functions are well-layered [12]. We define well-layered below:

Well-layered: A function f is well-layered iff, when running a greedy algorithm on $f - p$ for any price vector p, the set X_i selected at step t (which will thus have size t) is s.t.

$$f(X_i) - \sum_{\forall i \in X_i} (p_i) = \mathbf{max}_{X_j, |X_j|=t}(f(X_j) - \sum_{\forall j \in X_j} (p_j))$$

In other words, the set selected at step t of the greedy algorithm is the set with the highest payoff of size t. This characteristic of gross substitute functions is central to our positive result for the class in section 6.4.

4 Approach

In proving that SADP is harder than ODP for a given valuation class V, several approaches could be taken. We will list them below in order of increasing power.

1. We could provide a $\text{SADP}_{V,\alpha}$ instance that is NP-hard for any $\alpha \neq 1$. While this does not explicitly prove that $\text{SADP}_{V,\alpha}$ is a harder problem than $\text{ODP}_{V,\alpha}$–perhaps $\text{ODP}_{V,\alpha}$ is NP-hard with greater complexity–it at least implies that $\text{BMeD}_{V,\alpha}$ is NP-hard and therefore that the reduction from $\text{BMeD}_{V,\alpha}$ to $\text{ODP}_{V,\alpha}$ did not transform an easy problem into an NP-hard problem.

2. We could provide an $\text{ODP}_{V,\alpha}$ instance that is NP-hard and then reduce this instance to $\text{SADP}_{V,\alpha}$. As above, this would imply that solving SADP_V, α for valuation class V is NP-hard. $\text{ODP}_{V,\alpha}$ and $\text{SADP}_{V,\alpha}$ relate and demonstrate that the types of instances that are hard for $\text{SADP}_{V,\alpha}$ are typically the types of instances that are hard for $\text{ODP}_{V,\alpha}$. Such a reduction could possibly be easier to construct than the full reduction needed for approach **3**, as perhaps the structure of the particular hard instance would permit a reduction which would be invalid for other inputs in V.

3.1 We could provide a general reduction from $\text{ODP}_{V,\alpha}$ to $\text{SADP}_{V,\alpha}$ that holds for all $f, g \in V$. This result would demonstrate that $\text{SADP}_{V,\alpha}$ is at least as hard as $\text{ODP}_{V,\alpha}$ for each instance, thus fully closing the circle of reductions between GOOP and BMeD.

3.2 After resolving **3.1**, a powerful secondary result would be to find a hard ODP_V, α instance for some V and any $\alpha < 1$. This would indicate not only that the circle of reductions is closed for

V, but also that BMeD and GOOP are NP-hard for V.

As mentioned in the introduction, we take approach **3.1** and **3.2** in this paper. Thus, in describing the problem specifications below, our goal is to find a reduction that holds for all $f, g \in V$.

5 Reduction Requirements

Because of the specificity of this reduction, we must perform a thorough exploration of the restrictions. In this section, we introduce all reduction requirements, justify each, and provide an example of a promising-looking approach that violates this requirement. Through this process we hope to clarify the problem at hand and convince the reader that there are no trivial solutions.

All counterexamples provided in this section will make use of additive functions. While we are not particularly interested in solving mechanism design problems on additive functions themselves–optimizing such functions can be done very quickly–additive functions are contained within almost all valuation classes relevant to this paper. Therefore a counterexample to an approach using additive functions is sufficient to completely invalidate the approach.

Note on 5.6.1 and 5.6.2. Weinberg's reduction is approximation-preserving. Specifically, a $(\frac{(1-\alpha)D}{n} + \alpha)$-approximation to SADP implies a α-approximation to ODP. Here D denotes the construction being D-balanced and n denotes the number of input functions. Note that, in order for this to be useful, we must be able to make n much larger than D. The first two requirements that follow ensure that this is the case.

5.1 Works for Large n

Definition. Our construction must work for any n, where n is the number of input functions to SADP.

Intuition. As mentioned above, the reduction that Weinberg has constructed is not quite approximation preserving. Roughly, we lose a factor of $\frac{D}{n}$ in approximation. However, if we are able to make n very large, this approximation loss becomes negligible.

Violating Construction. An early thought when approaching this problem, one which takes advantage of the clear similarities between SADP and ODP, is to set $n = 2$. Then the input to SADP is exactly (f, g). SADP is required to approximately solve the difference between some adjacent set of functions, but there are only two functions. Thus, when $n = 2$, SADP is equivalent to ODP.

However, as described above, this approach fails because our approximation guarantee depends on having large n. Note that when $\alpha = \frac{1}{2}$, we are guaranteed only a 0-approximation. Therefore this promisingly straightforward approach fails. However, it is worth noting that this approach does work when $\alpha = 1$ (when we need a 1-approximation). Thus there is a trivial reduction when considering the non-approximation setting.

5.2 D-balanced

Definition.

D − balanced :*For a list of functions* $(f_1, ..., f_n)$, *let* X_l^* *denote the allocation that maximizes* $f_l(\cdot) - f_{l+1}(\cdot)$ *for all* $l \in [n]$. *We say that*$(f_1, ..., f_n)$*are* D*-balanced if* $f_n(X_n^*) \le D(f_l(X_l^*) - f_{l+1}(X_l^*))$ *for all* $l \in [n-1]$.

Intuition. Even if we are able to make n large, we are only able to make $\frac{D}{n}$ negligible if D does not depend on n.

Violating Construction. Consider the following input to SADP:

$$f, g, 2g - f, 3g - 2f, ..., (n-1)g - (n-2)f, ng - (n-1)f$$

Note that $h_i - h_{i+1} = f - g$ for all i. This looks promising! It will return the proper x^* even when we increase n. However, it is not D-balanced. Consider the case of two simple additive functions:

Table 3: Valuation function f

Sets of size 1	Value	Sets of size 2	Value
{a}	2	{a,b}	8
{b}	6		

Table 4: Valuation function g

Sets of size 1	Value	Sets of size 2	Value
{a}	1	{a,b}	10
{b}	9		

Here, $x^* = \{a\}$, and

$$\mathbf{max}(h_i(x) - h_{i+1}(x)) = f(x^*) - g(x^*) = 1$$

However, recall that the final function is defined as $ng - (n-1)f$.

$$\mathbf{max}(h_n(x)) = h_n(S) = 10n - 8(n-1) = n + 8$$

Thus we must find a D such that

$$D \geq n + 9$$

This D depends on n, so this construction is not D-balanced.

5.3 Explicitly Approximation-preserving (No addition of constants)

Definition. In addition to having the qualities above, which are necessary to ensure the implicit approximation preservation between ODP and SADP, we also must ensure that the optimization performed by SADP explicitly preserves approximation.

Intuition. Even if an α-approximation to SADP implies a roughly α-approximation to ODP, this is only useful insofar as ODP and SADP agree on what they are approximating. If SADP is generating a $\frac{1}{2}$-approximation for the expression $f(x) - g(x) + 1$, but $f(x^*) - g(x^*) = 1$, a valid output by SADP is $\emptyset$. This garners a value of 1 for SADP, while the max is 2. However $\emptyset$ is a 0-approximation for the original ODP problem. As $f(x)^* - g(x^*)$ can be arbitrarily close to 0, the addition of any constants that do not cancel out in subtraction will invalidate this approximation preservation.

5.4 "Normal-looking"

Definition. Functions must be non-negative, monotone, and valued at 0 on the empty set,

Intuition. If the reason that ODP is a hard problem is because it is difficult for valuations with negative values, this is not particularly interesting. Just as we want to reason about valuation classes individually, we also want to reason only about the functions in these valuation classes which are relevant to mechanism design problems.

Violating Construction. Consider attempting to fix our construction in the D-balanced subsection by subtracting a constant from all h_i. Let $a = ng(S) - (n-1)f(S)$. We could then write

$$f - a, g - a, 2g - f - a, 3g - 2f - a, ..., (n-1)g - (n-2)f - a, ng - (n-1)f - a$$

This function still returns the right x^* with large n, as all the a's cancel out in the subtraction. Furthermore, it appears to be D-balanced as well: the difference between each consecutive function is strictly positive, but the final expression evaluates to at most $ng(S) - (n-1)f(S) - ng(S) + (n-1)f(S) = 0$. Then even for $D = 0$ this is D-balanced.

However, these functions are now comprised of negative values. In addition, the empty set is nonzero. Therefore this approach is also invalid.

5.5 Preserves Valuation Classes

Definition. If we are performing a reduction for some ODP$_{V,\alpha}$, all input functions to SADP must be $\in V$.

Intuition. The original reduction from BMeD to GOOP holds for all valuation classes. We want to ensure that our reduction holds for particular valuation classes as well. The true statement we aim to prove in this paper is: "the problem of solving SADP with inputs $h_1, .., h_n$ in valuation class V is at least as hard as solving ODP with inputs f, g in valuation class V".

Recall our friend who wants to solve a mechanism design problem. Consider the case where she wants to solve BMeD for matroid rank valuation functions. We provide her with the reduction and promise her that the hardest instances of BMeD are no easier than the hardest instances of GOOP. "That's not good enough for me," she replies. "What if all of the hard instances of BMeD involved non-matroid rank valuations? Then this reduction could still make my problem much harder."

Again, our clever friend is exactly right. The problem of solving BMeD certainly varies in hardness for different valuation classes, and thus a reduction that does not preserve valuation class is not a particularly useful result. This paper instead allows us to promise our friend the new algorithmic problem will not be more difficult than solving BMeD on matroid rank functions (along with many other classes).

Violating Construction. Here is yet another approach, which seems to follow all previous rules: Let all functions input to SADP be evaluated over the set of items S', which includes all items that f and g were evaluated over, and n additional items $t_1, ..., t_n$. For any input x' to some h_i, let x denote the subset of x' from the original S.

Let $h_i(x')$ be $f(x)$ if there are i items t in x. Let $h_i(x')$ be $g(x)$ if there are less than i items t in x, and let $h_i(x')$ be 0 if there are more than i items in t.

Consider attempting to maximize $h_i - h_{i+1}$. If we include less than i extra items, then this value is equivalent to $g - g = 0$. If we include exactly i items, it is equivalent to $f - g$. If we include exactly $i + 1$ items, it is equivalent to $0 - f(x)$. If we include more than $i + 1$ items, it is equivalent to $0 - g(x)$. Thus the only way to garner a non-negative output is to include i extra items. After this, the problem is equivalent to $f - g$.

This construction is valid for arbitrary n, and is D-balanced because the final function does not scale with n. It also has no non-negative values and is zero at the empty set. However, this function is strange in a different way: it is certainly not guaranteed to be in the valuation class that f and g belong to. Consider again the following valuation functions:

Table 5: Valuation function f

Sets of size 1	Value	Sets of size 2	Value
{a}	2	{a,b}	8
{b}	6		

Table 6: Valuation function g

Sets of size 1	Value	Sets of size 2	Value
{a}	1	{a,b}	10
{b}	9		

Now let us construct h_1 according to our rules, considering just one additional item t. When t is present, the evaluation will be according to f, and then t is not present the evaluation will be according to g.

Table 7: Valuation function h_1

Sets of size 1	Value	Sets of size 2	Value	Sets of size 3	Value
{a}	1	{a,b}	8	{a,b,t}	9
{b}	9	{a,t}	2		
{t}	0	{b,t}	6		

Despite our original functions being additive, this function is not even subadditive. Observe that $h(a) + h(t) < h(a, t)$. As every valuation class we are exploring in this paper includes additive functions as a subset, and none include non-subadditive functions, this transformation does not preserve valuation class for any relevant classes.

5.6 C-compatible

Definition. ASA defines C-compatibility as a subproperty of cyclic monotonicity. Both definitions as written in the original paper are provided below:

Cyclic Monotonicity. A list of (possibly randomized) allocations $X_1, ..., X_n$ is said to be cyclic monotone with respect to $t_1, ..., t_n$ if the welfare-maximizing matching of types to allocations is to match allocation X_i to type t_i for all i.

C-compatibility. We say that a list of types $(t_1, ..., t_n)$ and a list of (possible randomized) allocations $X_1, ..., X_n$ are compatible if $(X_1, ..., X_n)$ is cyclic monotone with respect to $(t_1, ..., t_n)$, and for any $i < j$, the welfare-maximizing matching of types $t_{i+1}, ..., t_j$ to $X_i, ..., X_{j-1}$ is to match allocation X_l to type t_{l+1} for all l.

C-compatibility will be discussed in greater depth at the end of section **6**.

5.7 Key Takeaway from Restriction Exploration

In the above examples we have struggled to find a construction that fulfills D-balanced and approximation preservation simultaneously. Intuitively, we must somehow generate a long list of functions such that $h_i - h_{i+1}$ behaves similarly to $h_{i-1} - h_i$, but h_1 and h_n are on roughly the same scale. These restrictions provide intuition for our ultimate approaches.

In further sections we will refer back to these restrictions. Therefore we will summarize the conditions here along with their corresponding numbers:

Number	Constraint
1	Large n
2	Small D
3	Approximation-preserving
4	Normal-looking
5	Preserves valuation class
6	C-compatible

6 Reduction Results

Notes on Compatibility While initially presenting our results, we will ignore the restriction of C-compatibility. Compatibility can be achieved in all of our results by making slight adjustments to the models. For the sake of clarity, proofs that these adjustments satisfy C-compatibility and do not violate any other constraints will be withheld until the end of the results section.

6.1 Multi-packet Model

Here we present a key transformation that, while not getting us fully over the finish line, opens up space for creative solutions. This framework, which we will refer to as the multi-packet model, provides the backbone for all three final solutions.

Assume that we have an original input to ODP of two functions f and g over m items, where f and g are within some valuation class V. Let us define n disjoint copies (or "packets") of these items, $S_1, ..., S_n$. Thus for each $x_i \in S$, we will now have $x_{i1}, ..., x_{in}$. Then we can define n valuation functions $h_1, ..., h_n$, where h_i has a separate valuation function over each packet and resulting packet values are additive. Formally:

$$h_i = \sum_{j=1}^{n} h_{ij}(S_j)$$

Here, h_{ij} represents the ith function's valuation of packet j and S_j represents the items from the jth set included. h_{ij} is defined as follows:

$$h_{ij}(\cdot) = f(\cdot) \iff j \geq i$$

$$h_{ij}(\cdot) = g(\cdot) \iff j < i$$

To provide intuition, h_i values each of the packets according to either $f(\cdot)$ or $g(\cdot)$. In particular, $h_1(\cdot)$ values all packets at $f(\cdot)$, and as i increases the $f(\cdot)$ valuation turn into $g(\cdot)$ valuations. Below we provide an example where $n = 5$:

$$h_1(X) = f(X_1) + f(X_2) + f(X_3) + f(X_4) + f(X_5)$$

$$h_2(X) = g(X_1) + f(X_2) + f(X_3) + f(X_4) + f(X_5)$$

$$h_3(X) = g(X_1) + g(X_2) + f(X_3) + f(X_4) + f(X_5)$$

$$h_5(X) = g(X_1) + g(X_2) + g(X_3) + f(X_4) + f(X_5)$$

$$h_5(X) = g(X_1) + g(X_2) + g(X_3) + g(X_4) + f(X_5)$$

$$h_6(X) = g(X_1) + g(X_2) + g(X_3) + g(X_4) + g(X_5)$$

In the multi-packet model, the set of items from S_i in the α-approximation for $h_i - h_{i+1}$ is an α-approximation x^* for $f - g$.

Proof: By construction, h_i and h_{i+1} will have the same valuations on every packet except for

S_i. Assume that $h_i - h_{i+1}$ is being evaluated on some set X. We can split X into two subsets, X_i (items in S_i) and X_a (items in some S_a, $a \neq i$). Because the value of h is the addition of its value over all the packets, we can write

$$h_i(X) - h_{i+1}(X) = h_i(X_i) - h_{i+1}(X_i) + h_i(X_a) - h_{i+1}(X_a)$$

Regardless of the construction of X_a, $h_i(X_a) - h_{i+1}(X_a) = 0$, as h_i and h_{i+1} have the same valuations on all of these packets. Thus:

$$h_i(X) - h_{i+1}(X) = h_i(X_i) - h_{i+1}(X_i)$$

Finally, by our definition of h:

$$h_i(X_i) - h_{i+1}(X_i) = f(X) - g(X)$$

Thus the problem of maximizing $h_i - h_{i+1}$ for any i is exactly equivalent to maximizing $f - g$. A valid solution to SADP could include additional items from other sets, but these items would have no impact on the final value. So we can throw away these items and return X_i, which will have exactly the same approximation ratio on ODP as on SADP. This concludes our proof.

This model seems promising, but by itself it does not actually solve our problem: the final function has max value $n * g(S)$. This value scales with n, while $h_i(x) - h_{i+1}(x)$ does not. Therefore it is not D-balanced. To reach an appropriate solution, we need to place a cap on the size of $h_n(S)$. We are in a better place than before, however: creating multiple item packets gives us more flexibility to implement such constraints.

GS is closed under the addition of valuations over disjoint items.
As most valuation classes are closed under addition, it is clear that the addition of functions in V over disjoint sets of items will certainly remain in V. However, given the unusually brittle nature of the GS class, we will briefly prove this closure for GS.

Consider two GS functions f_1 and f_2 evaluated over disjoint sets S_1 and S_2. Let $h = f_1 + f_2$, where h is evaluated over $S = S_1 \cup S_2$. Assume that for some price vector p, the optimal set is X, which includes item x_i. Because no item adds value to both f_1 and f_2,

$$\mathbf{max}_{X \in S}(h(X) - \sum_{\forall x_i \in X} p_i) = \mathbf{max}_{X_1 \in S_1}(f_1(X_1) - \sum_{\forall x_i \in S_1} p_i) + \mathbf{max}_{X_2 \in S_2}(f_2(X_2) - \sum_{\forall x_i \in S_2} p_i)$$

Now let us increase the price of x_i, and w.l.o.g. assume that $x_i \in S_1$. This will only impact the valuation of f_1. Certainly the optimal set from S_2 will not change. As f_1 is GS, all items in $S_i \neq x_i$ will be in the new optimal set. Therefore h is GS.

6.2 Max Model

6.2.1 Model

Description. Let us provide our first positive result. Recall the functions $h_1, ..., h_n$ defined in section 6.1. Let us redefine them slightly: instead of summing up all h_{ij} for each h_i, we will instead take the maximum over all of them. So our inputs to SADP are now defined as follows:

$$h_i = \max_{j \in 1 \ldots n} h_{ij}(S_j)$$

Below is an example for the $n = 5$ case:

$$k_1(X) = \mathbf{max}(f(X_1), f(X_2), f(X_3), f(X_4), f(X_5))$$

$$k_2(X) = \mathbf{max}(g(X_1), f(X_2), f(X_3), f(X_4), f(X_5))$$

$$k_3(X) = \mathbf{max}(g(X_1), g(X_2), f(X_3), f(X_4), f(X_5))$$

$$k_4(X) = \mathbf{max}(g(X_1), g(X_2), g(X_3), f(X_4), f(X_5))$$

$$k_5(X) = \mathbf{max}(g(X_1), g(X_2), g(X_3), g(X_4), f(X_5))$$

$$k_6(X) = \mathbf{max}(g(X_1), g(X_2), g(X_3), g(X_4), g(X_5))$$

Constraint Satisfaction. We claim that this transformation represents a D-balanced, approximation-preserving reduction that preserves certain valuation classes. Namely, if SADP can find a $(1-\alpha)$-approximate x for the transformed input, then this x is also a $(1-\alpha)$-approximation of ODP(f, g).

Let us consider the expression $k_i - k_{i+1}$ for any i. Let us assume that this was the set of functions that SADP chose to maximize the difference between, and the approximate solution was a set x^{**}. Let us define the optimal solution to ODP as x^*.
By definition:

$$k_i(X^{**}) - k_{i+1}(X^{**}) = \max_{j \in 1 \ldots n} h_{ij}(X_j^{**}) - \max_{j \in 1 \ldots n} h_{(i+1)j}(X_j^{**})$$

If X^{**} is an $(1-\alpha)$-approximation of the difference, then the maximum over all h_{ij} must be h_{ii}. Assume for contradiction that the difference is approximately maximized and $\max_{j \in 1 \ldots n} h_{ij}(X_j^{**}) =$

h_{ik}, where $k \neq i$. Then $\max_{j \in 1 \ldots n} h_{ij}(X_j^{**}) \geq h_{ik}$, as $h_{ik} = h_{(i+1)k}$ for all $k \neq i$. Then $k_i - k_{i+1} \leq 0$. This contradicts our assumption about the difference being strictly positive.

Furthermore, a 1-approximation to SADP is exactly when the max of both the top and bottom sets is evaluating X_i. If any other set from the bottom is chosen, then this will be strictly worse than selecting $g(X_i)$. Given this, the 1-approximation optimization of ODP and SADP are exactly the same problem–though SADP could have a couple extra items..

Given that $\mathbf{max}_{j \in 1 \ldots n} h_{ij}(X_j^{**}) = h_{ii}$, if X^{**} is an $(1-\alpha)$-approximation of SADP, then the subset of X^{**} in c_i is a $(1-\alpha)$-approximation of ODP. Let us define the max set in k_{i+1} as $b(x^{***})$, where b could be f or g. Let us assume for contradiction that this is not true. Then SADP is correctly approximated:

$$(f(x^{**}) - b(x^{***})) \geq (1-\alpha)(f(x^*) - g(x^*))$$

However, ODP is not:

$$(f(x^{**}) - g(x^{**})) < (1-\alpha)(f(x^*) - g(x^*))$$

Because the max function in k_{i+1} is $b(x^{***})$,

$$b(x^{***}) \geq g(x^{**})$$

It follows that

$$(1-\alpha)(f(x^*) - g(x^*)) \leq f(x^{**}) - b(x^{***}) \leq f(x^{**}) - g(x^{**}) < (1-\alpha)(f(x^*) - g(x^*))$$

We have thus derived a contradiction. Thus our construction satisfies constraints 1 2 and 3. All the functions involved are evaluated at 0 at the empty set and are nonzero everywhere (assuming that f and g are of this nature), and thus it also satisfies constraint 4. An exploration of the closed valuation classes (constraint 5) will be performed below. C-compatibility (constraint 6) will again be covered at the end.

6.2.2 Closures

Unit-Demand. Unit demand is defined as the make singleton item over all available items. Trivially, taking the max over unit demand functions is unit demand.

XOS. The XOS function can be thought of as the class of functions constructed from the max of multiple additive functions. If we have some $h = \mathbf{max}(f, g)$ where f and g are XOS, then:

$$h = \mathbf{max}(\mathbf{max}(f_1, f_2, ..., f_x), \mathbf{max}(g_1, g_2, ..., g_y))$$

Where f_i and g_i are the ith additive functions used in the construction of f and g respectively. This can be rewritten as

$$h = \mathbf{max}(f_1, f_2, ..., f_x, g_1, g_2, ..., g_y)$$

Thus h is the max over a set of additive functions, so h is closed under the max operation.

Subadditive. Subadditive functions are defined by the following constraint: if f is subadditive, for every disjoint set of items S, T, $f(S \cup T) \leq f(S) + f(T)$. Proof: Take some function $h = \mathbf{max}(f, g)$, where f and g are subadditive functions. Then

$$\begin{aligned}
h(S \cup T) &= \mathbf{max}(f(S \cup T), g(S \cup T)) \\
&\leq \mathbf{max}(f(S) + f(T), g(S) + g(T)) && \text{(by subadditivity of } f \text{ and } g) \\
&\leq \mathbf{max}(f(S), g(S)) + \mathbf{max}(f(T), g(T)) && \text{(by the definition of max)} \\
&= h(S) + h(T) && \text{(by the definition of } h)
\end{aligned}$$

Thus,

$$h(S \cup T) \leq h(S) + h(T)$$

This fulfills the specifications of subadditivity. As any h that is the max of two subadditive functions is subadditive, subadditive functions are closed under the max operator.

6.3 Min with a Constant Model

6.3.1 Model

Description. While the above model satisfies all constraints, it is not closed for many important valuation classes (see appendix). Here we present another transformation that is valid for a different set of valuation classes.

Similarly to the max model, we will place a limit on the growth of our functions. Again recall the functions $h_1, ..., h_n$ defined in section 6.1. Here we will define our new input functions to SADP

$k_1, ..., k_n$ as $k_i = \mathbf{min}(h_i, f(S))$. Note that $f(S)$ is a constant that can be easily calculated upon input of f and g. Below is an example for the $n = 5$ case:

$$k_1 = \mathbf{min}(f(X_1) + f(X_2) + f(X_3) + f(X_4) + f(X_5), f(S))$$

$$k_2 = \mathbf{min}(g(X_1) + f(X_2) + f(X_3) + f(X_4) + f(X_5), f(S))$$

$$k_3 = \mathbf{min}(g(X_1) + g(X_2) + f(X_3) + f(X_4) + f(X_5), f(S))$$

$$k_4 = \mathbf{min}(g(X_1) + g(X_2) + g(X_3) + f(X_4) + f(X_5), f(S))$$

$$k_5 = \mathbf{min}(g(X_1) + g(X_2) + g(X_3) + g(X_4) + f(X_5), f(S))$$

$$k_6 = \mathbf{min}(g(X_1) + g(X_2) + g(X_3) + g(X_4) + g(X_5), f(S))$$

Constraint Satisfaction. We claim that this transformation represents a D-balanced, approximation-preserving reduction. Namely, if SADP can find a $(1 - \alpha)$-approximate x for the transformed input, then this x is also a $(1 - \alpha)$-approximation of ODP(f, g).

Let us consider the expression $k_i - k_{i+1}$ for any i. Let us assume that this was the set of functions that SADP chose to maximize the difference between, and the approximate solution was a set x^*.

By definition:

$$k_i(x^*) - k_{i+1}(x^*) = \mathbf{min}(h_i(x^*), f(S)) - \mathbf{min}(h_{i+1}(x^*), f(S))$$

$$= min(f(x_i^*) + h_i^{-i}(x)) - g(x_i^*) - h_{i+1}^{-i})(x))$$

By construction, h_i and h_{i+1} will have the same valuations on every packet except for the ith. We can split x^* into two subsets, x_i^* (items in S_i) and x_a^* (items in some S_a, $a \neq i$). Let us consider what could happen to the value of $\mathbf{min}(h_i(x^*), f(S)) - \mathbf{min}(h_{i+1}(x^*), f(S))$ if we remove any item in x_a by considering different cases.

Case 1: $h_i(x) < f(S)$, $h_{i+1}(x) < f(S)$

Here, neither value has met its cap. Thus $k_i - k_{i+1} = h_i - h_{i+1}$. As proven in section 6.1, α-approximation solution to $h_i - h_{i+1}$ is an α-approximation solution to $f - g$, and therefore we are done.

Case 2: $h_i(x) < f(S)$, $h_{i+1}(x) > f(S)$

This case cannot occur. The resulting value would be negative and therefore x would not be an approximation of the optimal solution.

Case 3: $h_i(x) > f(S)$, $h_{i+1}(x) > f(S)$

The resulting value would be zero and therefore x would not be an approximation of the optimal solution. Again, this case cannot occur.

Case 4: $h_i(x) > f(S)$, $h_{i+1}(x) < f(S)$

Here, h_i has met its cap but h_{i+1} has not. Removing some element in x_a would decrease the value of the negative term more than it would decrease the value of the positive term. Therefore removing this item would increase the final value.

Therefore all items in x_a contribute a non-positive value to $k_i(x) - k_{i+1}(x)$. This immediately implies that the x garnering a 1-approximation for this expression is the x that maximizes $k_i(x_i) - k_{i+1}(x_i)$, which is equal to

$$\begin{aligned}
\mathbf{min}(h_{ii}(x), f(S)) - \mathbf{min}(h_{(i+1)i}(x), f(S)) \\
= \mathbf{min}(f(x), f(S)) - \mathbf{min}(g(x), f(S)) \\
= f(x) - g(x)
\end{aligned}$$

Therefore $(k_i(x^*) - k_{i+1}(x^*)) = f(x^*) - g(x^*)$ for $x^* = \mathbf{max}_{x \in X}(k_i(x) - k_{i+1}(x))$

$$\begin{aligned}
&(1-\alpha)(f(x^*) - g(x^*)) \\
&= (1-\alpha)(k_i(x^*) - k_{i+1}(x^*)) \\
&\leq k_i(x) - k_{i+1}(x) && \text{(By the approximation of SADP)}
\end{aligned}$$

Therefore $(1-\alpha)(f(x^*) - g(x^*)) \leq k_i(x) - k_{i+1}(x)$, so x is a $(1-\alpha)$-approximation for ODP. This input is D-balanced for small D. The maximum value for each function is $f(S)$. As this value is a constant and does not scale with n, for large enough n D approaches 1. Our h are simply the addition of multiple functions in V, so assuming closure under addition, h is certainly in V. However, k is the minimum of h and a constant. Therefore this reduction only holds for V that are closed under $\mathbf{min}(f, c)$. This construction is also D-balanced, assuming that $f(x^*) - g(x^*) > 0$.

6.3.2 Closures

Submodular. It is a known result that submodular functions are closed under taking the minimum with a constant.

6.4 Item Cap Model

6.4.1 Model

Description. Similarly to the previous two models, the Item Cap (IC) operation places a limit on the total welfare of the final set. However, now we approach this limit indirectly by constraining the total number of items that our functions can have valuations over. Consider the multi-packet model, but with an item cap that is exactly $|S|$, the size of the original set of items. As with the previous models, we still have sufficient space to maximize $f - g$ between functions, but not a lot more space than that.

Constraint Satisfaction. Assume that functions $h_1, ..., h_n$ are constructed via the item cap transformation from functions f and g. Furthermore, assume that $SADP_{V,0}$ returns the set x^*. Finally, for some h_i and h_{i+1} that SADP chooses to maximize, assume that the max-valued subset of size m is x_1 in h_i and x_2 in h_{i+1}.

Let us define a_i^{-i} as the evaluation of h_i on all chunks but the ith, before the IC operation. Similarly, we will define a_{i+1}^{-i} as the evaluation of h_i on all chunks but the ith, before the IC operation.

Then

$$\begin{aligned} h_i(x^*) - h_{i+1}(x^*) &= IC_m(a_i^{-i} + f_i) - IC_m(a_{i+1}^{-i} + g_i) \\ &= a_i^{-i}(x_1) + f(x_1) - a_{i+1}^{-i}(x_2) - g(x_2) \end{aligned}$$

Note that a_i^{-i} and a_{i+1}^{-i} are equivalent, so we will refer to them as a^{-i} from now on. Consider the allocation x_{OPT}, the allocation maximizing our original ODP problem. As $|x_{OPT}| \leq m$, $h_i(x_{OPT}) - h_{i+1}(x_{OPT}) = f(x_{OPT}) - g(x_{OPT})$. Let us consider possible ways to improve this set, and disprove all cases.

1) If we remove or add any set of items from c_i, by the definition of x_{OPT} this will decrease or not impact the value.

2) Now consider the addition of some set in some other chunk c_j. If some of these items are included in the top m items for g but not for f, then the value for g is increased and the value for f stays the same.

3) If some of these items are included in the top m items for f but not for g, then if we define the set of items h_i is kicking out of c_i as Y and the new items as Z,

$$f(x_{OPT} \backslash Y) + a^{-i}(Z) - g(x_{OPT}) > f(x_{OPT}) - g(x_{OPT})$$

$$f(x_{OPT} \backslash Y) > f(x_{OPT}) - a^{-i}(Z)$$

And as h_{i+1} did not make any substitutions,

$$g(x_{OPT} \backslash Y) \leq g(x_{OPT}) - a^{-i}(Z)$$

Thus

$$f(x_{OPT} \backslash Y) - g(x_{OPT} \backslash Y) > f(x_{OPT}) - g(x_{OPT})$$

This contradiction our assumption about the optimality of x_{OPT}.

4) If some new items are included in the top m items for both f and g, both valuations will have kicked out some items. Then our new payoff is $f(x_{OPT} \backslash Y) - g(x_{OPT} \backslash Z)$. Let us assume for contradiction that this is an improvement. Then:

$$f(x_{OPT} \backslash Y) - g(x_{OPT} \backslash Z) > f(x_{OPT}) - g(x_{OPT})$$

But, by assumption that x_{OPT} is optimal:

$$f(x_{OPT}) - g(x_{OPT}) \geq f(x_{OPT} \backslash Y) - g(x_{OPT} \backslash Y)$$

Thus

$$f(x_{OPT} \backslash Y) - g(x_{OPT} \backslash Z) > f(x_{OPT} \backslash Y) - g(x_{OPT} \backslash Y)$$

$$g(x_{OPT} \backslash Z) < g(x_{OPT} \backslash Y)$$

However, this derives a contradiction: if this were true, then h_{i+1} would have selected the subset $x_{OPT} \backslash Y$ instead. Therefore this case is not possible .

There is no way to construct some input set that gives a solution to SADP$_{V,\alpha}$ which does not give a solution to ODP$_{V,\alpha}$. Thus getting a $(1-\alpha)$-approximation to SADP is thus an equivalent problem to getting a $(1-\alpha)$-approximation to ODP. This takes care of constraint 3. The logic holds for an arbitrary n, so constraint 1 is satisfied. The IC operation will retain the value of 0 on the empty set and produce non-negative functions, and therefore constraint 4 is satisfied. Finally, for constraint 2, D does not scale with n. Because our functions are subadditive, h_n is maximized by selecting c copies of the max-valued singleton item from disjoint sets. Thus, if the maximum singleton value is v, $h_n = cv$. In the case of our reduction $c = m$ where $m = |X|$. Thus

$$\mathbf{max}_{x \in S}(h_n(x)) = mv$$

Neither m nor v depend on n, and therefore constraint 2 is satisfied. Now we will consider the behavior of valuation classes under the IC operation.

6.4.2 Closures

Matroid valuation. Matroid valuations are known to be closed under the operation of limiting the max valid matroid size. This is exactly the IC operation.

GS. As per Pas Lemme, Gross Substitute functions are well-layered [12]. This means that when performing the greedy algorithm with some GS function f and some price vector p, the set constructed at each step is the max-value set of all sets of the same size. Now consider the function $h = IC(f, c)$.

Let us run the greedy algorithm on h with price vector p. If the optimal set in f has size $\leq c$, then $\mathbf{max}(h_p) = \mathbf{max}(f_p)$, and thus the greedy algorithm will find the max-payoff set. If the optimal set in f has size $> c$, then at step c the greedy algorithm will have the set which is the optimal set for f_p of size c. This set certainly has better payoff for h_p than any smaller set encountered via the greedy algorithm, as otherwise the greedy algorithm would have terminated previously. Furthermore, as all of these previously encountered sets had better payoff than all sets of their same size (by the well-layered characteristic), our final set is certainly the optimal. Therefore the greedy algorithm for h on any price vector will return the max-payoff set. This is exactly the definition of GS, so GS is closed under IC.

One important note is that, as GS functions are not closed under addition, the reduction from GOOP to OP (as stated in section 3) is no longer completely sound. This reduction involves adding together functions to form $f - g$, but f and g may no longer be GS. A particular type of GS functions, called tree-concordant functions, are closed under addition [3]. Therefore only if all valuation functions input to GOOP are tree-concordant does the complete circle of reductions certainly hold. Otherwise, the hardness result in section **7** at least holds, as maximizing the difference between two GS functions is certainly no harder than maximizing the difference between many GS functions. Therefore if the former is NP-hard, the latter is NP-hard as well.

Subadditivity. Let us define some function $h = IC(f, c)$ for some constant c and some subadditive function f. Now we can consider the value of $h(S \cup T)$ for any two disjoint sets S and T.

$$h(S \cup T) = IC(f(S \cup T), c)$$

By the subadditivity of f, $f(S \cup T) \leq f(S) + f(T)$. Consider any item cap on these functions. For any such cap, the best set of size c to pick for $f(S \cup T)$ is some X, which includes some subset of T T' and some subset of S S'. Thus

$$IC(f(S \cup T), c) = f(X) = f(T' \cup S')$$

By the subadditivity of f:

$$\leq f(T') + f(S')$$

By construction, we know that $|T|$ and $|S| \leq c$. Thus

$$f(T') + f(S') = IC(f(T'), c) + IC(f(S'), c)$$

This expression is $\leq IC(f(T), c) + IC(f(S), c)$. This is clear because whatever T' and S' are, they could certainly be selected in the IC protocol by their supersets T and S. Thus

$$h(S \cup T) = IC(f(S \cup T), c) \leq f(T') + f(S') \leq IC(f(T), c) + IC(f(S), c) = h(S) + h(T)$$

h satisfies the subadditivity condition, so subadditive functions are closed under IC.

6.5 Reintroduction of C-compatibility

Lemma 1. Any set of functions $h_1 ... h_n$ and corresponding allocations $x_1 ... x_n$ where:

1) Matching x_i to h_i for all i yields the max-weight matching

2) x_i maximizes $h_i - h_{i-1}$ for all i

3) $h_i(x_i) \leq n^a$ for some constant a

4) $h_i(x_{i-1}) - h_i(x_{i-b}) \geq \frac{1}{n^d}$ for some constant d and all $0 \leq b \leq i-2$

is C-compatible for some $C = n^{a+d+1}$.

Proof: We can set $Q_i = 2^{n^{a+d}i}$. Then the welfare difference of allocating x_{i-1} to h_i and allocating any other set to h_i is at least

$$2^{n^{a+d}i}\frac{1}{n^d} = 2^{n^{a+d}i}\frac{1}{2^{log(n)d}} = 2^{n^{a+d}i-dlog(n)}$$

Deciding on this allocation in the worse case will mean that all other matchings have weight 0. In this case, the amount of weight lost is equal to

$$\sum_{j=0}^{i-1} Q_j n^a = \sum_{j=0}^{i-1} Q_j 2^{alog(n)} \leq Q_{i-1}(i-1)2^{alog(n)} = 2^{n^{a+d}(i-1)}(i-1)2^{alog(n)} \leq 2^{n^{a+d}i-n^{a+d}+alog(n)+log(n)}$$

This upper bound on the total value lost by other players when set x_{i-1} is given to player h_i is smaller than the benefit.

Claim:

$$2^{n^{a+d}i-dlog(n)} > 2^{n^{a+d}i-n^{a+d}+alog(n)+log(n)}$$

We can show this by starting with the following inequality, which holds for $n > 2$:

$$\begin{aligned} n^{a+d} &> (a+d+1)log(n) \\ -dlog(n) &> -n^{a+d} + alog(n) + log(n) \\ 2^{-dlog(n)} &> 2^{-n^{a+d}+alog(n)+log(n)} \\ 2^{n^{a+d}i-dlog(n)} &> 2^{n^{a+d}i-n^{a+d}+alog(n)+log(n)} \end{aligned}$$

As the benefit of giving h_i x_{i-1} over any other set is greater than the total possible value incurred in the matching by all other h_{i-b}, the max-weight matching will certainly match h_i to x_{i-1}. Given this, we now have a new weight-maximization problem with functions up to h_{i-1} and sets up to x_{i-2}. Using the same logic as above, we know that the allocation must assign x_{i-2} to h_{i-1}.

Continuing in this manner, we can inductively assign all x_{i-k} to h_{i-k-1} to create a max-weight matching. This construction therefore satisfies compatibility. The maximum value of any multiplier Q_n is $2^{n^{a+d+1}}$, and therefore the construction is n^{a+d+1}-compatible.

Transformation Patterns. The alterations that must be made to the three constructions in order to satisfy C-compatibility are extremely similar. For clarity we have written out each, but note that the intuition for any example holds for all.

6.5.1 Max Alteration

We can transform our constructions into a form that satisfies this lemma. Recall that the original max construction was of the following form:

$$max(f(x_1), f(x_2), ..., f(x_n))$$

$$max(g(x_1), f(x_2), ..., f(x_n))$$

$$max(g(x_1), g(x_2), ..., f(x_n))$$

And so on.

Consider the following transformation:

$$max(f(x_1) + b(x_1), (1 + \frac{1}{n^2})(f(x_2) + b(x_2)), ..., (1 + \frac{n}{n^2})(f(x_n) + b(x_n)))$$

$$max(g(x_1) + b(x_1), (1 + \frac{1}{n^2})(f(x_2) + b(x_2)), ..., (1 + \frac{n}{n^2})(f(x_n) + b(x_n)))$$

$$max(g(x_1) + b(x_1), (1 + \frac{1}{n^2})(g(x_2) + b(x_2)), ..., (1 + \frac{n}{n^2})(f(x_n) + b(x_n)))$$

Here, $b(x) = \frac{|x|}{n}$. Let us first confirm that this construction is still approximation-preserving, D-balanced, and closed under the operations we claimed it was previously. Our final function has scaled only by a factor of $1 + \frac{1}{n}$. In addition, we have scaled each packet valuation in exactly the same manner throughout the functions. Therefore constraints 1 and 2 hold, and it remains the case that an α-approximation of $h_i - h_{i+1}$ is an α-approximation of $h_{i,i}(x) - h_{i+1,i}(x)$.

Furthermore, this difference is equal to

$$max(f(x_1)+b(x_1), (1+\frac{1}{n^2})(f(x_2)+b(x_2)), ..., (1+\frac{n}{n^2})(f(x_n)+b(x_n))) - max(g(x_1)+b(x_1), (1+\frac{1}{n^2})(f(x_2)+b(x_2)), ..., (1 + \frac{n}{n^2})(f(x_n) + b(x_n)))$$

As before, any item allocated to a set that is not the set i will provide no value to $h_i - h_{i+1}$. Thus the same logic from proof **6.2** remains.

Now let us confirm that this construction satisfies our constraints for C-compatible. Let the set of allocations be $x_1^*, x_2^*, ..., x_{n-1}^*, S$, where S is the complete set. $h_i - h_{i-1}$ is maximized exactly at x_i^*, and h_n is maximized at S. Therefore condition **2)** is satisfied. Each of our functions h_i are bounded by $(1+\frac{1}{n})\mathbf{max(f(S), g(S))}$. Thus for large enough n, $h_i(x_i) \leq \mathbf{max(f(S), g(S))} \leq \mathbf{n^1}$. Therefore condition **3)** holds with $a = 1$.

Finally, $h_i(x_{i-1}) = (1+\frac{i-2}{n})(g(x^*) + \frac{|x^*|}{n})$, and $h_i(i-b) \leq (1+\frac{i-3}{n})(g(x^*) + \frac{|x^*|}{n})$. Thus for all b:

$$h_i(x_{i-1}) - h_i(i-b) \geq \frac{g(x^*) + \frac{|x^*|}{n}}{n} \geq \frac{|x^*|}{n^2}$$

And, per our assumption that $|x^*| > 0$,

$$h_i(x_{i-1}) - h_i(i-b) \geq \frac{1}{n^2}$$

This restriction 4 holds with $d = 2$. Thus this construction is $n^{a+d+1} = n^4$-compatible.

6.5.2 Min Alteration

In a similar fashion, recall that the original min construction was of the following form:

$$min(f(x_1) + f(x_2) + ... + f(x_n), f(S))$$

$$min(g(x_1) + f(x_2) + ... + f(x_n), f(S))$$

$$min(g(x_1) + g(x_2) + ... + f(x_n), f(S))$$

And so on. Consider the following transformation:

$$min(f(x_1) + b(x_1) + (1+\frac{1}{n^2})(f(x_2) + b(x_2)) + ... + (1+\frac{n}{n^2})(f(x_n) + b(x_n)), 2f(S))$$

$$min(g(x_1) + b(x_1) + (1+\frac{1}{n^2})(f(x_2) + b(x_2)) + ... + (1+\frac{n}{n^2})(f(x_n) + b(x_n)), 2f(S))$$

$$min(g(x_1) + b(x_1) + (1+\frac{1}{n^2})(g(x_2) + b(x_2)) + ... + (1+\frac{n}{n^2})(f(x_n) + b(x_n)), 2f(S))$$

Here, $b(x) = \frac{|x|}{n}$. Let us first confirm that this construction is still approximation-preserving, D-balanced, and closed under the operations we claimed it was previously. Note that we have increased the ceiling to $2f(S)$, as $f(x^*)$, after scaling, could now garner a slightly higher value than $f(S)$. However, this multiple is always less than 2, so $f(x^*) < 2f(S)$. In addition, we have scaled each packet valuation in exactly the same manner throughout the functions. Therefore it remains the case that an α-approximation of $h_i - h_{i+1}$ is an α-approximation of $h_{i,i}(x) - h_{i+1,i}(x)$.

Furthermore, this difference is equal to

$$(1 + \frac{i-1}{n^2})(f(x_i) + \frac{|x|}{n}) - (1 + \frac{i-1}{n^2})(g(x_i) + \frac{|x|}{n}) = (1 + \frac{i-1}{n^2})(f_{x_i} - g_{x_i})$$

This is simply a scalar multiple of $f - g$. Therefore our approximation guarantee remains. Furthermore, this construction remains D-balanced, as the $\mathbf{max}(h_n) = 2f(S)$ and $\mathbf{max}(h_i - h_{i+1}) \geq f(x^*) - g(x^*)$. Assuming that $f(x^*) = c > 0$, we can write

$$2f(S) \leq D * c$$
$$D \geq \frac{2f(S)}{c}$$

There will always exist some constant c independent of n such that this inequality holds true.

Now let us confirm that this construction satisfies our constraints for C-compatible. Let the set of allocations be $x_1^*, x_2^*, ..., x_{n-1}^*, S$, where S is the complete set. $h_i - h_{i-1}$ is maximized exactly at x_i^*, and h_n is maximized at S. Therefore condition 2) is satisfied. Each of our functions h_i are bounded by $2f(S)$. Thus for large enough n, $h_i(x_i) \leq 2f(S) \leq n^1$. Therefore condition 3) holds with $a = 1$.

Finally, exactly as with the max alteration, $h_i(x_{i-1}) = (1 + \frac{i-2}{n})(g(x^*) + \frac{|x^*|}{n})$, and $h_i(i-b) \leq (1 + \frac{i-3}{n})(g(x^*) + \frac{|x^*|}{n})$. Thus for all b:

$$h_i(x_{i-1}) - h_i(i-b) \geq \frac{g(x^*) + \frac{|x^*|}{n}}{n} \geq \frac{|x^*|}{n^2}$$

And, per our assumption that $|x^*| > 0$,

$$h_i(x_{i-1}) - h_i(i-b) \geq \frac{1}{n^2}$$

This restriction 4 holds with $d = 2$. Thus this construction is $n^{a+d+1} = n^4$-compatible.

6.5.3 IC Alteration

We will transform our functions exactly as above before taking the item cap. The item cap value will remain m. As above, scaling the values of our disjoint sets before adding them together will not impact closure results.[1]

The final construction is equivalent to the final min construction, except for the operation performed upon the functions:

$$IC_m(f(x_1) + b(x_1) + (1 + \frac{1}{n^2})(f(x_2) + b(x_2)) + ... + (1 + \frac{n}{n^2})(f(x_n) + b(x_n)))$$
$$IC_m(g(x_1) + b(x_1) + (1 + \frac{1}{n^2})(f(x_2) + b(x_2)) + ... + (1 + \frac{n}{n^2})(f(x_n) + b(x_n)))$$
$$IC_m(g(x_1) + b(x_1) + (1 + \frac{1}{n^2})(g(x_2) + b(x_2)) + ... + (1 + \frac{n}{n^2})(f(x_n) + b(x_n)))$$

Again, $b(x) = \frac{|x|}{n}$.

All of the valuation classes that we state are closed under IC–subadditive, gross substitutes, unit demand and matroid valuation–are all closed under scalar multiplication and under the addition of valuations over disjoint sets. Therefore the transformation does not impact function closures. Furthermore, the max value of h_n is now only increased by a factor of roughly $\frac{1}{n}$ at most. This value does not increase as n increases, and therefore constraint 2 is still satisfied. The logic still holds for arbitrary n, and therefore constraint 1 is still satisfied. We retain functions of the form specified by constraint 4.

Now the difference between any two consecutive functions is:

$$IC(g(x_1) + b(x_1) + (1 + \frac{1}{n^2})(f(x_2) + b(x_2)) + ... + (1 + \frac{n}{n^2})(f(x_n) + b(x_n))) - IC(g(x_1) +$$
$$b(x_1) + (1 + \frac{1}{n^2})(g(x_2) + b(x_2)) + ... + (1 + \frac{n}{n^2})(f(x_n) + b(x_n)))$$

Furthermore, the proof that this is in fact a valid, explicitly approximation-preserving reduction is analogous to the proof provided in section 6.4. Note that the operations of scaling the functions and adding a constant that cancels out in subtraction do not violate any aspects of the proof.

[1]The key exception to this statement is the class of matroid rank functions. These functions are constrained to integer valuations and increases of 0 or 1 upon the addition of an item. Adjustments to ensure C-compatibility make the functions no longer integer-valued. Thus, even though matroid rank functions are closed under the non-C-compatible construction, the complete construction does not constitute a valid reduction for the class.

6.6 Results Summary

Table 8: Summary of Closure Results

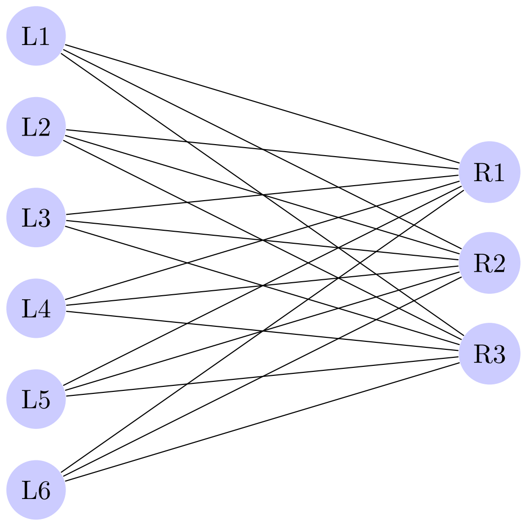
Transformation	Max	Min	Item Cap
Subadditive	Yes	No	Yes
Submodular	No	Yes	No
Gross Substitute	No	No	Yes
XOS	Yes		
Unit Demand	Yes		
Matroid Valuation			Yes
Matroid Rank	No	No	No*
Budget Additive	No	No	No

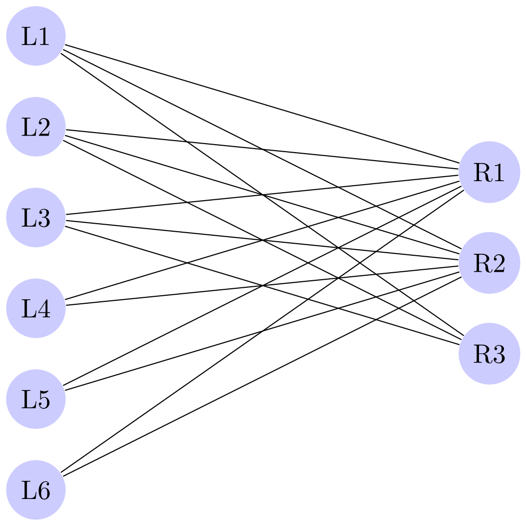
7 Hardness Results

In this section, we provide a hardness result for matroid valuation functions by showing a hard ODP instance. As the class of matroid valuation functions is contained within GS functions, this proof provides the first hardness result for both classes. Furthermore, as our proof is a full reduction from ODP to SADP it provides a more meaningful hardness result for submodular functions than that provided in ASA. Our work proves that SADP under submodular (as well as the other classes provided) is hard in cases we care about–when it is used to solve ODP.

ODP instance. We will show an f and g here that are matroid valuation functions such that approximating the max of $f - g$ is exponential in complexity for any $\alpha \neq 0$. We will provide f and g that are OXS functions, and for clarify we will describe them by their bipartite graphs. Note that after our IC transformation the inputs to SADP are not OXS. However, OXS is a subclass of matroid valuation (and this is a subclass of GS), and thus f and g are also matroid valuation and GS. These classes are closed under IC.

Let us begin by describing f. The function is an OXS function modeled as follows: there will be m left-hand nodes and $\frac{m}{2}$ right-hand nodes. All left-hand nodes will have edges to all right-hand nodes with weights of 1. g will be represented by a similar graph, except an arbitrary set of $\frac{m}{2}$ left-hand nodes will not have edges to the final right-hand node. Below is an instance where $m = 6$:





Achieving any nonzero approximation of $f-g$ is exponentially hard. Let us define $r*$ as the final right-hand node. Let us define $x*$ as the left-hand nodes in g which have no edges to $r*$. For any set of items x where $|x| < \frac{m}{2}$, $f(x) = g(x) = |x|$, because we can match all of the left hand nodes representing the items to right-hand nodes. For any set of items x where $|x| > \frac{m}{2}$, $f(x) = g(x) = \frac{m}{2}$. This is because, given the size of the set there must be at least one item with an edge to $r*$, so we can match these nodes. Then we can match exactly $\frac{m}{2} - 1$ other nodes before we run out of right-hand nodes. For any set of items x where $|x| = \frac{m}{2}$ again $x \neq x*$, $f(x) = g(x) = \frac{m}{2}$. We can match a node not in $x*$ to $r*$ and all of the remaining nodes to other right-hand nodes.

The only remaining case is when $x = x^*$. In this case we can match all nodes in the f graph, gaining a value of $\frac{m}{2}$. But none of these nodes have an edge to $r*$ in the g graph, so the max-weight matching is $\frac{m}{2} - 1$. Thus $f(x*) - g(x*) = 1$, and for all other x $f(x) - g(x) = 0$. Therefore the only set which provides a nonzero approximation of the max is the optimal set itself, $x*$. Furthermore, finding this set is exponentially hard, even if the optimizer discerns that the optimal set must be of size $\frac{m}{2}$. There are $\binom{n}{\frac{m}{2}}$ sets of size $\frac{m}{2}$ to examine, and only one $x*$. All other sets provide exactly the same value, so querying the value of an incorrect set provides no information to the optimizer. Thus finding $x*$ takes 2^{cm} time.

Given our proof that there is a valuation class-preserving, approximation-preserving reduction from ODP to SADP using IC for matroid rank and GS functions, providing this instance is sufficient to prove hardness of approximation.

8 Future Work

In this paper we have provided a framework for transforming ODP instances into SADP instances. The positive results above prove reduction tightness from BMeD to GOOP for many valuation classes. However, some valuation classes remain unsolved. In particular, budget-additive functions and matroid rank functions are both interesting classes that both lack full reductions and hardness results. Furthermore, no current reduction fully closes the circle for each GS instance, as the reduction from GOOP to ODP necessitates closure under addition. All these valuation classes are interesting to reason about, and future work may attempt to extend this framework to draw stronger conclusions about their hardness.

References

[1] S. Matthew Weinberg. *Algorithms for Strategic Agents*. PhD thesis, Massachusetts Institute of Technology, 2014.

[2] United Nations Framework Convention in Climate Change. The kyoto protocol mechanisms, 2010.

[3] Eric Balkanski and Renato Paes Leme. On the construction of substitutes. 2018.

[4] Xi Chen, Ilias Diakonikolas, Dimitris Paparas, Xiaorui Sun, and Mihalis Yannakakis. The complexity of optimal multidimensional pricing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1319–1328, 2014.

[5] Xi Chen, Ilias Diakonikolas, Anthi Orfanou, Dimitris Paparas, Xiaorui Sun, and Mihalis Yannakakis. On the complexity of optimal lottery pricing and randomized mechanisms. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1464–1479, 2015.

[6] Shahar Dobzinski, Hu Fu, and Robert D. Kleinberg. Optimal Auctions with Correlated Bidders are Easy. In *the 43rd ACM Symposium on Theory of Computing (STOC)*, 2011.

[7] Constantinos Daskalakis, Alan Deckelbaum, and Christos Tzamos. Mechanism Design via Optimal Transport. In *The 14th ACM Conference on Electronic Commerce (EC)*, 2013.

[8] Yang Cai, Constantinos Daskalakis, and S. Matthew Weinberg. An Algorithmic Characterization of Multi-Dimensional Mechanisms. In *the 44th Annual ACM Symposium on Theory of Computing (STOC)*, 2012.

[9] Yang Cai, Constantinos Daskalakis, and S. Matthew Weinberg. Optimal Multi-Dimensional Mechanism Design: Reducing Revenue to Welfare Maximization. In *the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2012.

[10] Yang Cai, Constantinos Daskalakis, and S. Matthew Weinberg. Reducing Revenue to Welfare Maximization: Approximation Algorithms and other Generalizations. In *the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2013.

[11] Yang Cai, Constantinos Daskalakis, and S. Matthew Weinberg. Understanding Incentives: Mechanism Design becomes Algorithm Design. In *the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2013.

[12] Renato Paes Leme. Gross substitutability: An algorithmic survey. 106:294–316, 2017.

9 Appendix

Below we present proofs that various valuation classes are not closed under various models. We must present multiple solutions because certain valuation classes are not closed under certain transformations.

9.1 Budget Additive is not closed under Max, Min or IC

In all three of these transformations we consider disjoint sets of items. However, if f and g have different budgets, any addition of these functions will not be budget additive.

Budget additive functions are also not closed under the max operations. Consider any two additive functions f and g. Create two new budget additive functions f' and g' with budgets $b \geq \mathbf{max}(f(S), g(S))$. Here S is the complete set. Then these functions behave exactly as f and g. However, we know that there exist additive functions f and g such that $\mathbf{max}(f, g)$ is not submodular. Therefore there exist budget-additive functions f' and g' such that $\mathbf{max}(f', g')$ is not submodular, and thus not budget additive.

9.2 Submodular and GS are not closed under max

We present two additive functions where, when the max is taken, the result is not submodular. As all additive functions are submodular, it follows that submodular functions are not closed under the max operation. Furthermore, all additive functions are GS, and all GS are submodular. Thus GS is not closed under the max operation.

Consider the following two additive valuation functions over three items:

Table 9: Valuation function f

Sets of size 1	Value	Sets of size 2	Value	Sets of size 3	Value
{a}	4	{a,b}	10	{a,b,c}	10
{b}	6	{a,c}	4		
{c}	0	{b,c}	6		

Table 10: Valuation function g

Sets of size 1	Value	Sets of size 2	Value	Sets of size 3	Value
{a}	0	{a,b}	0	{a,b,c}	4
{b}	0	{a,c}	4		
{c}	4	{b,c}	4		

Now we can consider some function $h = \mathbf{max}(f, g)$.

Table 11: Valuation function h

Sets of size 1	Value	Sets of size 2	Value	Sets of size 3	Value
{a}	4	{a,b}	10	{a,b,c}	10
{b}	6	{a,c}	4		
{c}	4	{b,c}	6		

Note that

$$h(a, b, c) - h(a, c) = 10 - 4 = 6$$

$$h(b, c) - h(c) = 6 - 4 = 2$$

Adding item b to the set garners a larger increase when a is in the set than when a is not present. This violates the restriction of submodularity. h was constructed from taking the max of two additive (and therefore submodular) functions. Therefore the class of submodular functions is not closed under the max operation.

9.3 GS is not closed under min with a constant

Additive functions are a subclass of GS functions. The min operation on an additive function is a budget-additive function. This class is not contained within gross substitutes, and therefore GS is not closed under min with a constant. We show an illustrative example with the following additive function:

Table 12: Valuation function f

Sets of size 1	Value	Sets of size 2	Value	Sets of size 3	Value
{a}	2	{a,b}	4	{a,b,c}	14
{b}	2	{a,c}	12		
{c}	10	{b,c}	12		

Now consider some function $h = \mathbf{min}(f, 3)$:

Table 13: Valuation function h

Sets of size 1	Value	Sets of size 2	Value	Sets of size 3	Value
{a}	2	{a,b}	3	{a,b,c}	3
{b}	2	{a,c}	3		
{c}	3	{b,c}	3		

Consider the price vector $p = \{.5, .5, 1\}$. The two optimal sets are $\{a, b\}$ and $\{c\}$, each with payoff 2. Now consider the vector $q = \{5, .5, 1\}$. All sets including item a now provide a negative payoff. Of the remaining sets, $h(c) - p(c) = 3 - 1 = 2$, $h(b, c) - p(b, c) = 3 - 1.5 = 1.5$, and $h(b) - p(b) = 2 - .5 = 1.5$. Therefore the single optimal set is $\{c\}$.

By increasing the price of a, b was removed from the optimal set. Therefore h is not GS.

9.4 Submodular is not closed under IC

Interestingly, though GS and subadditive functions are closed under IC, the class of submodular functions is not. We prove this by providing an example of a submodular function that is no longer submodular when an item cap is placed on it. We then show how this counterexample can be easily transformed into a case of the addition of two equal functions on disjoint sets where the item cap is the number of items in each set. As a counterexample exists in this specific case, there is no way to use this model to transform submodular functions.

Table 14: Valuation function f

Sets of size 1	Value	Sets of size 2	Value	Sets of size 3	Value	Sets of size 4	Value
{a}	5	{c,d}	10	{a,b,c}	9	{a,b,c,d}	10
{b}	5	{a,d}	9	{a,b,d}	10		
{c}	5	{b,d}	9	{a,c,d}	10		
{d}	5	{a,c}	9	{b,c,d}	10		
		{b,c}	9				
		{a,b}	9				

This valuation function is submodular. Recall the definition of submodularity. For any set of items X and any items $y, z \notin X$:

$$f(X \cup y \cup z) - f(X \cup y) \leq f(X \cup z) - f(X)$$

First consider the case of $X = \emptyset$. For all items y and z:

$$f(y \cup z) - f(y) \leq 5 = f(z) - f(\emptyset)$$

Now consider any set X of size 1. For all items y and z:

$$f(X \cup y \cup z) - f(X \cup y) \leq 1 < 4 \leq f(X \cup z) - f(X)$$

Now consider any set X of size 2. If $z \neq d$, $X \cup y$ cannot be the set $\{a, b, c\}$. In this case, $f(X \cup y) = f(X \cup z) = 10$, so:

$$f(X \cup y \cup z) - f(X \cup y) = 0 \leq f(X \cup z) - f(X)$$

Finally, consider the case where $z = d$. Then $X \neq \{c, d\}$, so $f(X) = 9$. $X \cup y$ must be equal to $\{a, b, c\}$, so $f(X \cup y) = 9$. Furthermore, $X \cup z \neq \{a, b, c\}$, so $f(X \cup z) = 10$.

$$f(X \cup y \cup z) - f(X \cup y) = 10 - 9 = 1 = f(X \cup z) - f(X)$$

The function is also monotone. Note that the only sets (X, Y) where $|X| < |Y|$ but $f(X) > f(Y)$ are $(\{c, d\}, \{a, b, c\})$. In this case $X \not\subseteq Y$, so it does not violate monotonicity. Thus, f represents a monotone submodular function.

Now let us introduce an item cap of size 2. For every valuation function over 3 or more items, the value is the highest-valued subset of size 2. The new valuation function is as follows, with the singular change in bold:

Table 15: Valuation function h

Sets of size 1	Value	Sets of size 2	Value	Sets of size 3	Value	Sets of size 4	Value
{a}	5	{c,d}	10	{a,b,c}	9	{a,b,c,d}	10
{b}	5	{a,d}	9	**{a, b, d}**	**9**		
{c}	5	{b,d}	9	{a,c,d}	10		
{d}	5	{a,c}	9	{b,c,d}	10		
		{b,c}	9				
		{a,b}	9				

All sets that contain items c and d will have value 10, because $f(c,d) = 10$. However, the value of the set a, b, d decreases because there is no possible subset of size 2 to select with value 10. Now the value of $f(a,b,c,d) - f(a,b,d) = 10 - 9 = 1$. However, the value of $f(a,b,c) - f(a,b) = 9 - 9 = 0$. Thus

$$f(a,b,c,d) - f(a,b,d) > f(a,b,c) - f(a,b)$$

If we let $X = \{a,b\}$, $x = c$ and $y = d$, we see that

$$f(X \cup x \cup y) - f(X \cup y) > f(X \cup x) - f(X)$$

Thus contradicting the submodularity definition.

This function is not a counterexample for the GS case, as it is not GS: consider the price vector $p = \{4.8, 10, 4.85, 4.85\}$. A greedy algorithm would select a at first with payoff .2, then terminate. However, the optimal set is c, d, with payoff .3. As the greedy algorithm does not find the optimal payoff set, this valuation function is not GS. As we see in section 6.4, gross substitutes are in fact closed under IC.