Limits on Computationally Efficient VCG-Based Mechanisms for Combinatorial Auctions and Public Projects

Thesis by

David Buchfuhrer

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2011

(Defended May 20, 2011)

Acknowledgements

Thanks to Chris Umans for help and advice in researching and writing up everything in this dissertation. Thanks to Yaron Singer and Michael Schapira for developing the ideas behind showing computational hardness of VCG mechanisms and for working with me on applying these ideas to public projects. Thanks to Shaddin Dughmi for helping me work through different possible approaches to the material in Chapter 5. Thanks to John Ledyard for giving me an economist's perspective on the problems I was studying and helping me evaluate and eliminate several approaches to eliminating the asymmetry described in Chapter 5. Thanks to my committee members John Ledyard, Leonard Schulman, Chris Umans, and Adam Wierman for taking the time to evaluate my thesis proposal and provide helpful comments.

An extended abstract of the material in Chapter 3 was published in [8]. An extended abstract of the material in Chapter 4 was published in [7].

Abstract

A natural goal in designing mechanisms for auctions and public projects is to maximize the social welfare while incentivizing players to bid truthfully. If these are the only concerns, the problem is easily solved by use of the VCG mechanism. Unfortunately, this mechanism is not computationally efficient in general and there are currently no other general methods for designing truthful mechanisms. However, it is possible to design computationally efficient VCG-based mechanisms which approximately maximize the social welfare.

We explore the design space of computationally efficient VCG-based mechanisms under submodular valuations and show that the achievable approximation guarantees are poor, even compared to efficient non-truthful algorithms. Some of these approximation hardness results stem from an asymmetry in the information available to the players versus that available to the mechanism. We develop an alternative *Instance Oracle* model which reduces this asymmetry by allowing the mechanism to access some computational capabilities of the players. By building assumptions about player computation into the model, a more realistic study of mechanism design can be undertaken.

Finally, we give VCG-based mechanisms for some problems in the Instance Oracle model which achieve provably better approximations than the best VCG-based mechanism in the standard model. However, for other problems we give reductions in the Instance Oracle model which prove inapproximability results as strong as those shown in the standard model. These provide more robust hardness results that are not simply artifacts of the asymmetry in the standard model.

Contents

Chapter 1

Introduction

Combinatorial auctions and combinatorial public projects are two important allocation problems in algorithmic mechanism design. The goal of each is to find a suitable allocation. A combinatorial auction consists of a set of m items and n players. An allocation in a combinatorial auction is a partition of the items $S_1, \ldots, S_n$ such that each player gets the set $S_i \subseteq [m]$, and $S_i \cap S_j = \emptyset$ for $i \neq j$. Some items may not be given to any player. A combinatorial public project consists of m items, n players, and a parameter k. An allocation in a combinatorial public project is a subset $S \subseteq [m]$ of the items of size $|S| = k$.

In both cases, each player i has a value function v_i over subsets of the items. Each value function is assumed to be non-negative ($v_i(S) \geq 0$), non-decreasing ($v_i(S) \geq v_i(T)$ for $S \supseteq T$), and normalized ($v_i(\emptyset) = 0$). We also assume that players only care about their own value, and not about the value obtained by others. A mechanism for either of these problems takes in the valuation functions $v_1, \ldots, v_m$ and returns an allocation A and a list of prices $p_1, \ldots p_n$, where player i must pay price p_i. Being rational, each player seeks to maximize his quasi-linear utility function $v_i(A) - p_i$ ($v_i(S_i) - p_i$ for combinatorial auctions or $v_i(S) - p_i$ for combinatorial public projects).

Before studying this problem, it is important to determine the goal which we wish the mechanism to achieve. One natural goal is to maximize the revenue, $\sum_i p_i$. Revenue maximization is important and well-studied [31, 39, 40, 9, 26, 27, 11], but is not the goal we consider. Instead, we look at maximization of the *social welfare*, $\sum_i v_i(A)$.

From a purely computational perspective, the complexity of this problem depends only on how hard it is to find an allocation A maximizing or approximating the maximum value of $\sum_i v_i(A)$. For example, when auctioning a single item, simply allocate it to the player whose value for the item is highest. However, as rational players, every player would attempt to declare infinite value for the item in order to ensure getting it. This is why prices are important. We must find a way to set the prices such that the player with highest value gets the item. One way to achieve this is to set the price paid by the winning player to the second-highest bid, and all other prices to zero. This way, no player benefits by declaring a higher price (as if this causes the player to win, it will result in negative utility) and no player benefits by declaring a lower price (as this does not decrease the payment for the player declaring a lower price). As everyone benefits most from declaring their valuation correctly, we can assume that rational players will make truthful declarations under this mechanism. Therefore, this mechanism results in the player with highest value getting the item.

The second-price mechanism for single item auctions was famously generalized by the Vickrey-Clarke-Groves (VCG) mechanism [42, 10, 25]. In this rather simple scheme, one simply calculates an optimal allocation A, then for each player i, calculate an optimal allocation A_{-i} for all players excluding i. Player i is then charged price $p_i = \sum_{j \neq i} v_j(A_{-i}) - v_j(A)$ equal to the decrease in social welfare of other players due to i's participation. Player i is therefore incentivized to maximize $v_i(A) + \sum_{j \neq i} v_j(A) - \sum_{j \neq i} v_j(A_{-i})$, which is the social welfare, minus a term that does not depend on what player i declares. Thus, each player is incentivized to maximize the social welfare. So a truthful mechanism is always available for maximizing welfare.

Further demonstrating the importance of the VCG mechanism, Roberts showed that for sufficiently general problems, the VCG mechanism is the *only* truthful mechanism [38]. More recent work has also made progress on demonstrating this result for more restricted problems [2, 29, 18, 36]. So the VCG mechanism is not only important because it is a truthful mechanism. It is important because in many cases, it is the only truthful mechanism.

This can be problematic from a computational perspective. When finding an optimal allocation is computationally infeasible, the VCG mechanism cannot be implemented. One might be tempted to implement the VCG mechanism using an approximation algorithm, but the VCG mechanism is only truthful if $\sum_i v_i(A)$ is always maximized by truthful reporting of valuation functions. [34] described the class of approximation algorithms which can be truthfully implemented using the VCG mechanism, which are known as *maximal-in-range* algorithms.

Every algorithm has a range R of possible allocations it can output depending on the input given. For many familiar algorithms, R will be the set of all possible allocations. A maximal-in-range algorithm always outputs an allocation $A = \operatorname{argmax}_{A \in R} \sum_i v_i(A)$ which maximizes social welfare within the range. Maximal-in-range algorithms can be tricky to design, as it can be difficult to find a range which contains a good approximation of the social welfare for any instance and for which the welfare-maximizing allocation can be efficiently found. The restriction to maximal-in-range mechanisms can dramatically worsen the best achievable approximation ratio [14, 36].

1.1 Definitions and Notation

In this section, we formally define the important terms used in this dissertation and provide the definitions of and notation for the problems studied. We use m to denote the number of items in an allocation problem and n to denote the number of players. We begin by defining valuation functions. Valuation functions are central to the problems studied in this dissertation, as they define the preferences of the players.

Definition 1.1 (Valuation Function)**.** *A valuation function v is a function $v : 2^{[m]} \to \mathbb{R}^+ \cup \{0\}$ from subsets of $[m] = \{1, \ldots, m\}$ to the non-negative real numbers. A valuation function is normalized to $v(\emptyset) = 0$ and monotone, so that $v(S \cup T) \geq v(S)$ for all sets S, T.*

In order to achieve the best possible outcome, we wish to maximize the the overall sum of player valuations. We call this sum the social welfare.

Definition 1.2 (Social Welfare)**.** *The social welfare of an allocation is the sum over each player of the player's value for the allocation. If there are n players with valuation functions $v_1, \ldots, v_n$, the social welfare of A is $\sum_i v_i(A)$. The social welfare of a combinatorial auction or combinatorial public project is the maximum social welfare of any allocation, $\max_A \sum_i v_i(A)$.*

The first problem we consider is the combinatorial public project problem.

Definition 1.3 (Combinatorial Public Project)**.** *A combinatorial public project consists of n players $[n]$, m items $[m]$, and a parameter k. Each player i has a valuation function v_i. An allocation A consists a subset S of $[m]$ of size k. The social welfare of A is $\sum_i v_i(S)$.*

We also study combinatorial auctions.

Definition 1.4 (Combinatorial Auction)**.** *A combinatorial auction consists of n players $[n]$ and m items $[m]$. Each player i has a valuation function v_i. An allocation A consists of n subsets of $[m]$, $S_1, \ldots, S_n$ such that $S_i \cap S_j = \emptyset$ for $i \neq j$. The social welfare of A is $\sum_i v_i(S_i)$.*

All valuations which we study are subadditive.

Definition 1.5 (Subadditive)**.** *A valuation function v is subadditive if $v(S \cup T) \leq v(S) + v(T)$ for all sets S, T.*

Most of the functions we study are also submodular.

Definition 1.6 (Submodular)**.** *A valuation function v is submodular if $v(S \cup T) + v(S \cap T) \leq v(S) + v(T)$ for all sets S, T.*

In both auctions and public projects, players wish to maximize their utility.

Definition 1.7 (Utility)**.** *Given a valuation function v and a set of prices $p_1, \ldots, p_m$, the utility of a set S is $v(S) - \sum_{i \in S} p_i$.*

A set which maximizes utility is called a demand set.

Definition 1.8 (Demand Set)**.** *Given a valuation function v and a set of prices $p_1, \ldots, p_m$, a demand set is a set S maximizing the utility, $v(S) - \sum_{i \in S} p_i$.*

Demand sets are not important just because of their relation to utility maximization. They are also necessary to define the class of gross substitutes, which contains several of the valuation classes we study.

Definition 1.9 (Gross Substitutes)**.** *A valuation function v satisfies the gross substitutes property if for any two sets of prices $p_1, \ldots, p_m$ and $q_1, \ldots, q_m$ such that $q_i \geq p_i$ and S is a demand set for $p_1, \ldots, p_m$, that there exists some demand set T for $q_1, \ldots, q_m$ such that $S \cap \{i : p_i = q_i\} \subseteq T$. In other words, the items which were already demanded and did not have their prices raised remain in a demand set.*

We call the process that leads to allocations and prices a mechanism.

Definition 1.10 (Mechanism)**.** *A mechanism M is an algorithm which takes in an instance of an allocation problem and returns an allocation A and a list of prices $p_1, \ldots, p_n$.*

The mechanisms that we are interested in are those in which players are not incentivized to lie about their valuations. These are called truthful mechanisms.

Definition 1.11 (Truthful Mechanism)**.** *A mechanism M is truthful if for any player with value v_i and any values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n$ for players other than i, player i's utility $v_i(A) - p_i$ is maximized by declaring v_i.*

One way to create a maximal-in-range mechanism is through the use of VCG payments.

Definition 1.12 (VCG Payments)**.** *The VCG payment for player i given an allocation algorithm A is defined as follows. Let S be the allocation determined by i given all player values, and S_{-i} be the allocation A determines if we replace v_i with 0. The VCG payment is $p_i = \sum_{j \neq i} v_j(S_i) - v_j(S_{-i})$.*

VCG payments result in a truthful mechanism if the algorithm they are applied to is maximal-in-range.

Definition 1.13 (Maximal-in-Range (MIR))**.** *An allocation algorithm A is maximal-in-range if there exists a set of values C_S for each set S such that for any valuation functions $v_1, \ldots, v_n$, A returns a set S which maximizes $\sum_i v_i(S)$ over the range of A.*

The use of a maximal-in-range algorithm together with VCG payments is the only known general way to truthfully maximize social welfare, and remains an active area of study [16, 18, 29, 36]. We call mechanisms which use maximal-in-range algorithms with VCG payments VCG-based mechanisms. Maximal-in-range algorithms are a special case of affine maximizers.

Definition 1.14 (Affine maximizer)**.** *An allocation algorithm A is an affine maximizer if there exists a set of values C_S for each set S such that for any valuation functions $v_1, \ldots, v_n$, A returns a set S which maximizes $\sum_i v_i(S) + C_S$ over the range of A. Note that maximal-in-range mechanisms are a special case of affine maximizers where $C_S = 0$ for all S.*

1.1.1 Notation

A class of combinatorial auction or combinatorial public project problem can be defined by three important factors. The first is the type of problem, auction, or public project. The second is the class of valuation functions that $v_1, \ldots, v_n$ belong to. Finally, we have the number of players n. Individual problems will also have different numbers of items m and for combinatorial public projects, parameters k. But we classify instances by type of problem, class of valuation functions, and number of players. All three of these factors can affect the difficulty of mechanism design. In order to clearly express what each of these factors are, we use a three-part notation. We will explain this notation using the example PC_2, which we will now explain.

The first part of the notation denotes whether the problem is a combinatorial auction or a combinatorial public project. For a combinatorial auction, we use an A and for a combinatorial public project, we use a P. So in the example PC_2, the problem is a combinatorial public project.

The second part denotes the class of valuation functions allowed. This is simply one or more letters following the A or P. In the example PC_2, the class of valuation functions is the one denoted by C, which refers to capped-additive valuations. We will define capped additive valuations as well as other valuation classes and how they are denoted in Section 1.1.2.

The final part of the notation is an optional subscript. If present, it is the number of players. If not, the number of players is not limited. So PC_2 is the combinatorial public projects problem with 2 capped-additive players and PC is the combinatorial public projects problem with an unbounded number of capped-additive players.

1.1.2 Problem Definitions

In this dissertation, we study problems for which the valuation functions are subadditive. This class of functions (also known as complement-free) is important to the study of combinatorial auctions [15, 22, 23, 30, 44]. In particular, we study valuation functions drawn from a complement-free hierarchy (see Figure 1.1) based on the classes studied in [30, 33]. The classes we study in this hierarchy are defined below. All of the submodular valuation functions have *succinct representations*.

Definition 1.15 (Succinct Representation)**.** *A valuation class has a succinct representation if functions in the class can be uniquely identified by a representation which has size polynomial in the number of items m and in $\log(v_i([m]))$, the log of the maximum value for any set.*

Succinct representation is important if we want to study computational complexity, as computational complexity is based on input size. For example, if the input size for a public project is 2^m, then it is possible to enumerate all $\binom{m}{k} \leq 2^m$ possible allocations in polynomial time in order to find the one which maximizes social welfare.

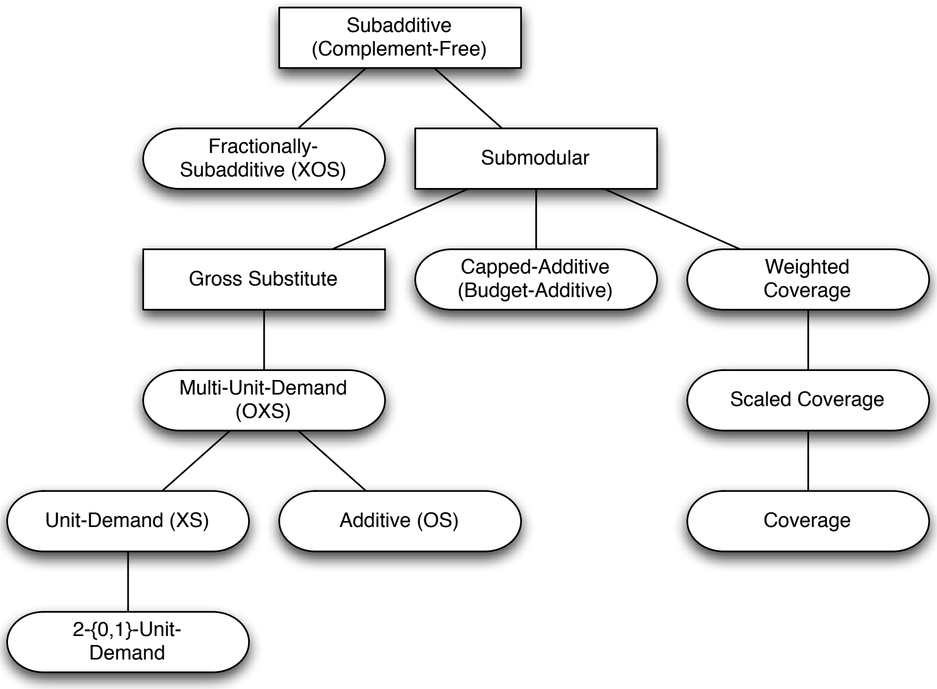

Figure 1.1: The Complement-Free Hierarchy. The valuation classes studied directly in this dissertation are marked by rounded nodes. The names in parentheses are alternative names which are sometimes also used to refer to these classes. We do not use these alternative names in this dissertation.

Definition 1.16 (Additive (A))**.** *An additive valuation v_i has value v_i^j for each item j and the value of a set S is $v_i(S) = \sum_{j \in S} v_i^j$. Additive valuations are not discussed much in this dissertation, as we can show fairly easily that both PA and AA have polynomial-time solutions (see Theorem 1.2).*

Before proving that PA and AA have polynomial-time solutions via Theorem 1.2, we require a helpful lemma.

Lemma 1.1. *There exists an algorithm which for any k, m, finds the k largest or smallest items from an unordered array of length m in $O(m)$ time.*

Proof. We will show how to find the k largest items. To find the k smallest items, simply reverse the comparisons in this process. [3] shows that the kth largest item in an array of length m can be found in $O(m)$ time independent of k. To find the other $k-1$ larger items, simply iterate through the array twice. On the first pass, add any items which are greater than the kth largest item to the list of the kth largest items. This pass may result in a list of length less than k, as there may be some items equal to the kth largest which belong on the list. So on the second pass, add items equal to the kth largest item until the list has k items.

After this process, the list that has been built contains all items greater than the kth largest item, and none less than it. It also has length k, as there must be at least k items greater than or equal to the kth largest item (including itself). Therefore, this list contains the k largest items. Since finding the kth largest item takes $O(m)$ time, as do each of the 2 passes, the total running time of this algorithm is $O(m)$. $\square$

Theorem 1.2. *Both AA and PA can be solved in $O(mn)$ time where m is the number of items and n is the number of players.*

Proof. For AA, the social welfare is simply $\sum v_i^j$ over pairs i, j such that item j is assigned to player i. So this is maximized by assigning each item j to the player i with the maximum value of v_i^j. For each item it only takes $O(n)$ time to find this maximum, for a total of $O(mn)$ time over m items.

For PA, the social welfare of a set S is $\sum_{j\in S}\sum_i v_i^j$. So to maximize the social welfare, simply choose the k items with highest value for $\sum_i v_i^j$. These m sums can be found in $O(n)$ time each, for a total of $O(mn)$ time. By Lemma 1.1, the k highest sums can then be found in $O(m)$ time, for a total of $O(mn)$ time. $\square$

Definition 1.17 (Unit-Demand (U))**.** *A unit-demand valuation v_i has a value v_i^j for each item j and the value of a set S is the maximum value of any item in S, $v_i(S) = \max_{j\in S} v_i^j$. v_i can be succinctly represented by the m values v_i^j.*

Definition 1.18 (ℓ-{0,1}-Unit-Demand (ℓU))**.** *An ℓ-{0,1}-unit-demand valuation v_i is a unit-demand valuation for which v_i^j is either 0 or 1 for each j, and for which $v_i^j = 1$ for at most ℓ values of j.*

We will mostly consider the 2-{0,1}-unit-demand valuation (2U).

Definition 1.19 (Multi-Unit-Demand (MU))**.** *A multi-unit-demand valuation v_i consists of several unit-demand valuations $v_i^{(j)}$. In order to compute the value $v_i(S)$ of a set S, one item is assigned to each unit-demand valuation function, and the values for each function are summed. The value is the maximum possible such sum,*

$$v_i(S) = \max_{\substack{S_1,\dots,S_\ell \\ S_i\subseteq S \\ S_i\cap S_j=\emptyset}} \sum_j v_i^{(j)}(S_j).$$

Lemma 1.3 shows that $v_i(S)$ is computable in polynomial time. Lemma 1.5 shows that any multi-unit-demand function v_i can be succinctly represented by at most m^2 unit-demand valuation functions.

Lemma 1.3. *If v_i is a multi-unit-demand valuation function over m items, it is possible to compute*

$$v_i(S) = \max_{\substack{S_1,\dots,S_\ell \\ S_i\subseteq S \\ S_i\cap S_j=\emptyset}} \sum_{j=1}^{\ell} v_i^{(j)}(S_j)$$

for any set $S \subseteq [m]$ in time polynomial in m and ℓ.

Proof. Given a partition $S_1, \ldots, S_\ell$ of S, the value computed by the partition is $\sum_{j=1}^{\ell} v_i^{(j)}(S_j) = \sum_{j=1}^{\ell} v_i^{(j)}(\{s_j^*\})$ for some s_j^* in each S_j. So in order to maximize the value, we need only to optimally match each valuation function to an item. This can be accomplished via maximum weighted bipartite matching. Create a bipartite graph in which one side has m nodes corresponding to the items, and the other has ℓ nodes corresponding to the functions $v_i^{(1)}, \ldots, v_i^{(\ell)}$. The edge between nodes corresponding to item j and function $v_i^{(j)}$ has weight $v_i^{(j')}(j)$. Thus, a maximum weighted matching maximizes $\sum v_i^{(j')}(j)$ over all matchings of items j to functions $v_i^{(j')}$. As a maximum weighted matching on a bipartite graph with $m + \ell$ nodes can be found in time polynomial in m and ℓ, this completes the proof. □

Computing the value of a set S for a single multi-unit-demand player is equivalent to computing an optimal allocation in an auction with several such players. So Lemma 1.3 implies the following corollary.

Corollary 1.4. *AMU (and therefore also AU) can be solved exactly in polynomial time.*

Proof. Suppose we have n players with multi-unit-demand functions $v_1, \ldots, v_n$, where $v_i(S) = \max_{S_1,\ldots,S_{\ell_i}} \sum_j v_i^{(j)}(S_j)$. We create a single multi-unit-demand function V with unit-demand functions $V^{(i,j)} = v_i^{(j)}$. The value of a set S is the maximum over all partitions

$$S_{(1,1)}, \ldots, S_{(1,m)}, S_{(2,1)}, \ldots, S_{(2,m)}, \ldots, S_{(n,1)}, \ldots, S_{(n,m)}$$

of $\sum_{(i,j)} v_i^{(j)}(S_{(i,j)})$. Note that this is equivalent to giving player i $\bigcup_j S_{(i,j)}$. So the maximum social welfare in the AMU instance is the value of $[m]$ to the single player we have created. Therefore, Lemma 1.3 shows that AMU can be solved exactly in polynomial time. □

Lemma 1.5. *For any multi-unit-demand valuation $v_i(S) = \sum_{j=1}^{\ell} v_i^{(j)}(S)$, there is a set M of size at most m^2 such that*

$$v_i(S) = \max_{\substack{S_1,\ldots,S_\ell \\ S_i \subseteq S \\ S_i \cap S_j = \emptyset}} \sum_{j \in M} v_i^{(j)}(S_j).$$

Furthermore, this set can be found in $O(m\ell)$ time.

Proof. If $\ell \leq m$, this is trivially true. So assume $\ell > m$.

Each item j either contributes to the value or not. If it does, it must be matched to one of the m valuation functions $v_i^{(j)}$ that have maximum value for j. One of these is always available to match with j, as the other $m - 1$ items can only be matched to at most $m - 1$ of these. So for any maximum value matching, we only need to consider the at most m valuation functions per item that they might be matched to it in an optimal matching, for a total of m^2 valuation functions.

Lemma 1.1 shows that we can find the m functions of highest value for item j in $O(\ell)$ time. Since this is repeated m times, once for each item, the total time is $O(m\ell)$. $\square$

Definition 1.20 (Capped-Additive (C))**.** *A capped-additive valuation v_i has value v_i^j for each item j and a value cap c. The value of a set S is the minimum of the sum of all values in S and the value cap, $v_i(S) = \min(\sum_{j\in S} v_i^j, c)$. v_i can be represented succinctly by the m values $v_i^j = v_i(\{j\})$ and the budget cap $c_i = v_i([m])$.*

Definition 1.21 (Coverage (COV))**.** *A coverage valuation v_i has a subset $V_i^j \subseteq U$ of some universe U associated with each item j and the value of a set S is the size of the union of the corresponding sets, $v_i(S) = \left|\bigcup_{j\in S} V_i^j\right|$. v_i can be represented succinctly by the sets V_i^j if $|U|$ has a polynomial bound.*

Definition 1.22 (Scaled Coverage Valuation (SCOV))**.** *A scaled coverage valuation v_i has a subset $V_i^j \subseteq U$ of some universe U associated with each item j as well as a scaling factor α and the value of a set S is α times the size of the union of the corresponding sets, $v_i(S) = \alpha \left|\bigcup_{j\in S} V_i^j\right|$. v_i can be represented succinctly by the sets V_i^j together with the scale factor α if $|U|$ has a polynomial bound.*

Definition 1.23 (Weighted Coverage Valuation (WCOV))**.** *A weighted coverage valuation v_i has a subset $V_i^j \subseteq U$ of some universe U associated with each item j as well as a weight w_u for each $u \in U$. The value of a set S is the size of the sum of w_u over items u covered by sets V_i^j corresponding to items $j \in S$,*

$$v_i(S) = \sum_{u \in \bigcup\limits_{j\in S} V_i^j} w_u.$$

v_i can be represented succinctly by the sets V_i^j together with the weights w_u if $|U|$ has a polynomial bound.

Definition 1.24 (Fractionally-Subadditive (FS))**.** *A fractionally-subadditive valuation v_i has ℓ additive functions $v_i^{(1)}, \ldots, v_i^{(\ell)}$. The value of a set S is the maximum value over the ℓ additive functions, $v_i(S) = \max_j v_i^{(j)}(S)$. v_i is represented by the ℓ additive functions $v_i^{(1)}, \ldots, v_i^{(\ell)}$. Lemma 1.6 shows that arbitrary v_i cannot be represented succinctly regardless of choice of representation. As we are interested in valuations with succinct representations, our results deal with $\ell \in poly(m)$, making our representations succinct.*

Lemma 1.6. *For each $m > 0$ and any choice of representation, there exists a fractionally-subadditive valuation function v_i for $2m$ items with $v_i(S) \leq 2m^2$ which requires at least 2^m bits to represent.*

Proof. We will prove this by showing that there are at least $2^{\binom{2m}{m}}$ fractionally-subadditive valuation functions. As there are only $2^{\binom{2m}{m}} - 1$ strings with fewer than $\binom{2m}{m}$ bits, this proves that $\binom{2m}{m}$ bits are required for at least one of these valuations. The binomial coefficient $\binom{2m}{m}$ is equal to the number of subsets of $[2m]$ of size m and 2^m is the number of subsets of $[m]$. For each subset M of $[m]$, there is a corresponding subset M' of $[2m]$ of size m containing all elements of M, plus items $m+1, \ldots, m+(m-|M|)$. Thus, $\binom{2m}{m} \geq 2^m$, so showing that there are $2^{\binom{2m}{m}}$ different fractionally-subadditive functions over m items proves that one of them requires at least 2^m bits to represent.

The $2^{\binom{2m}{m}}$ valuation functions are defined as follows. Let $\binom{[2m]}{m}$ be the set of subsets of $[2m]$ of size m. For each $\mathcal{M} \subseteq \binom{[2m]}{m}$, let

$$v_{\mathcal{M}}(S) = \begin{cases} (m+1) \cdot |S|, & |S| < m \text{ or } S \in \mathcal{M} \\ m \cdot |S|, & |S| \geq m \text{ and } S \notin \mathcal{M} \end{cases}.$$

Let $\mathcal{M}, \mathcal{M}'$ be distinct subsets of $\binom{[2m]}{m}$. We will show that $v_{\mathcal{M}}$ and $v_{\mathcal{M}'}$ are different functions. As $\mathcal{M}$ and $\mathcal{M}'$ are different, there exists some set S that is in one of them but not the other. Without loss of generality, assume $\mathcal{M}$ contains some set S that $\mathcal{M}'$ does not. By the above definition, $v_{\mathcal{M}}(S) = (m+1) \cdot m$, as all sets in $\mathcal{M}$ have size m. As $S \notin \mathcal{M}'$, $v_{\mathcal{M}'}(S) = m \cdot m$, which is different from $(m+1) \cdot m$ for $m > 0$. Thus $v_{\mathcal{M}} \neq v_{\mathcal{M}'}$ for every $\mathcal{M} \neq \mathcal{M}'$, so there are $2^{\binom{2m}{m}}$ distinct functions $v_{\mathcal{M}}$, one for each set $\mathcal{M} \subseteq \binom{[2m]}{m}$.

To complete the proof, we need only see that for every $\mathcal{M} \subseteq \binom{[2m]}{m}$, $v_{\mathcal{M}}$ is fractionally-subadditive. We construct additive functions $v_{\mathcal{M}}^T$ for each $T \subseteq [2m]$ as follows. For $|T| < m$ or $T \in \mathcal{M}$,

$$v_{\mathcal{M}}^T(\{s\}) = \begin{cases} m+1, & s \in T \\ 0, & \text{otherwise} \end{cases}.$$

For $|T| \geq m$ and $T \notin \mathcal{M}$,

$$v_{\mathcal{M}}^T(\{s\}) = \begin{cases} m, & s \in T \\ 0, & \text{otherwise} \end{cases}.$$

This defines the additive functions $v_{\mathcal{M}}^T(S) = \sum_{s \in S} v_{\mathcal{M}}^T(\{s\})$. Using these, we define the fractionally-subadditive function

$$v_{\mathcal{M}}(S) = \max_T v_{\mathcal{M}}^T(S).$$

For any T, $v_{\mathcal{M}}^T(S) \leq (m+1) \cdot |S|$. If $|S| < m$ or $S \in \mathcal{M}$, $v_{\mathcal{M}}^S(S) = (m+1) \cdot |S|$. So $v_{\mathcal{M}}(S) = (m+1) \cdot |S|$.

For $S > m$, we consider two cases. For $|T| \leq m$, $v_{\mathcal{M}}^T(S) \leq m \cdot (m+1)$, as each $s \in T$ has value at most $m+1$ and $|T| \leq m$. Since $|S| > m$, this is at most $m \cdot |S|$. For $|T| > m$, the value of each item is at most m and there are only $|S|$ items, for a total value of at most $|S| \cdot m$. Since $v_{\mathcal{M}}^S(S) = m \cdot |S|$ achieves this bound, $v_{\mathcal{M}}(S) = m \cdot |S|$.

Finally, we have the case that S is such that $|S| = m, S \notin \mathcal{M}$. For $|T| < m$, the maximum value for any set is $(m+1) \cdot |T| \leq (m+1) \cdot (m-1) = m^2 - 1 < m \cdot |S|$. For $T \in \mathcal{M}$, there are at most

$m-1$ items in both S and T, so the maximum value is again at most $(m+1)\cdot(m-1) < m \cdot |S|$. For $|T| \geq m, T \notin \mathcal{M}$, the value is at most $m \cdot |S|$, which is achieved by $v^S_{\mathcal{M}}(S)$. So $v_{\mathcal{M}}(S) = m \cdot |S|$.

By the above three cases, we have shown that

$$v_{\mathcal{M}}(S) = \begin{cases} (m+1)\cdot|S|, & |S| < m \text{ or } S \in \mathcal{M} \\ m \cdot |S|, & |S| \geq m \text{ and } S \notin \mathcal{M} \end{cases}$$

is fractionally-subadditive, completing the proof. □

1.2 Overview of Results

In Chapters 3 and 4 we show hardness results for VCG-based mechanisms for combinatorial public projects and combinatorial auctions, respectively. These results for the valuation classes studied in this dissertation are summarized in Figure 1.2.

Valuation Class	Problem	Best MIR Approximation Ratio
Additive	PA	1
	AA	1
Unit-Demand	PU_n	1
	P2U	$\sqrt{m}$ [New]
	AU	1
Multi-Unit-Demand	PMU_2	1 [New]
	PMU_3	$\sqrt{m}$ [New]
	AMU	1
Capped-Additive	PC_2	$\sqrt{m}$ [New]
	AC_n	n [New]
	AC	$\min(n, O(\sqrt{m}))$ [New]
Coverage	PCOV_1	$\sqrt{m}$ [New]
Fractionally-Subadditive	PFS_n	1
	PFS	$\sqrt{m}$ [New]

Figure 1.2: The best approximation ratios achievable by a polynomial-time maximal-in-range algorithm assuming that NP does not have polynomial circuits. When n is used as a subscript, it refers to any constant n. A $\sqrt{m}$ upper bound for public projects with subadditive players is shown in [41], so all $\sqrt{m}$ are proven in this dissertation via a lower-bound of $m^{1/2-\epsilon}$ for all $\epsilon > 0$.

For certain cases in Chapter 3, we are able to show a few results about general truthful mechanisms. VCG-based approximation algorithms cannot approximate P2U with a ratio better than $\sqrt{m}$, but we show a truthful 2-approximation. We extend the VCG-based hardness result for PCOV_1 to show that any polynomial-time truthful mechanism for PSCOV_1 cannot approximate the social welfare better than $\sqrt{m}$ unless NP has polynomial circuits.

In Chapter 3, we also show some computational results which do not take truthfulness into account. These results are summarized in Figure 1.3.

Finally, in Chapter 5 we present a new model that builds computational complexity on top of

Valuation Class	Problem	Bounds on Approximation Ratio r
Unit-Demand	PU_n, constant n	$r = 1$
	PU	$r = e/(e-1)$ [New]
Multi-Unit-Demand	PMU_2	$r = 1$ [New]
	PMU_3	1 [New] $< r \leq 3/2$ [New]
	$\mathrm{PMU}_n, 4 \leq n \leq 9$	1 [New] $< r \leq e/(e-1)$
	PMU_{10}	$1+\epsilon$ [New] $< r \leq e/(e-1)$ (no FPTAS)
	PMU	$r = e/(e-1)$ [New]
Capped-Additive	PC_1	$r = 1$
	PC_n, constant $n \geq 2$	$r = 1+\epsilon$ (FPTAS) [New]
	PC	$r = e/(e-1)$ [New]
Fractionally-Subadditive	PFS_n, constant n	$r = 1$
	PFS	$2^{\log^{1-\gamma}(\min(n,m))}$ [New] $\leq r \leq \min(\epsilon \cdot n, \sqrt{m})$ [41]

Figure 1.3: Computational results shown in Chapter 3. Equality refers to matching upper and lower bounds, up to arbitrarily small ϵ factors. All $e/(e-1)$ upper bounds are from [32]. When ϵ is used, it refers to any constant $\epsilon > 0$.

truthfulness in order to reduce the asymmetry in the definitions of truthfulness and computational efficiency. For a mechanism to be truthful, no lie can exist which may benefit a player, even if this lie is hard to compute. So the mechanism is limited by computational efficiency, but the hypothetical players it is designed for are not. This asymmetry is precisely what leads to the result that PSCOV_1 is hard to approximate truthfully, as any truthful mechanism must limit its range so much that even computationally intractable lies do not exist. Under the model we develop in Chapter 5, we are able to show that PCOV_2 is hard to approximate truthfully even when PCOV_1 has a polynomial-time truthful solution. This allows for more robust hardness results that do not depend on assuming that players are more computationally powerful than mechanisms.

Chapter 2

Alternate Approaches

In this chapter, we consider approaches to the study of truthful mechanisms which are not the focus of this dissertation. These approaches are randomized algorithms and communication complexity. Outside of this chapter, our primary interest is the computational complexity of deterministic algorithms.

2.1 Randomized Algorithms

Our hardness results only hold for deterministic mechanisms. Recent results [13, 19, 20] show that randomization can sometimes allow for improved approximation ratios. In particular, [19] shows that there exists a randomized FPTAS for AC_n for any constant n which satisfies a randomized notion of truthfulness. Other results have shown that randomization can be of no help depending on the problem and the version of randomized truthfulness used [7, 12].

A universally truthful mechanism is one which chooses a truthful mechanism from some distribution, then runs it. Universally truthful mechanisms can also be used to improve on the hardness results for PU, PMU, and PC, as we show in Theorem 2.1 and Corollaries 2.3, 2.4, and 2.5 below.

Definition 2.1 (Universally Truthful)**.** *A universally truthful mechanism $\mathcal{M}$ consists of several truthful mechanisms $M_1, \ldots, M_\ell$. $\mathcal{M}$ chooses a mechanism M_i randomly from some distribution and runs it in order to determine an allocation and payments.*

Theorem 2.1. *There exists a universally truthful mechanism for PU which runs in $O(km)$ time and achieves an expected approximation ratio of at most n/k.*

Proof. For each subset $T \subseteq [m], |T| = k$ we define a mechanism $\mathcal{M}_T$. $\mathcal{M}_T$ arbitrarily orders elements of $T = \{t_1, \ldots, t_k\}$, then iteratively builds up sets $S_0, \ldots, S_k$ and allocates S_k. It starts with $S_0 = \emptyset$

and adds items according to

$$S_i = \begin{cases} S_{i-1} \cup \{\text{argmax}_j v_{t_i}^j\}, & \text{argmax}_j v_{t_i}^j \notin S_{i-1} \\ S_{i-1} \cup \{\min_{j \notin S_{i-1}} j\}, & \text{otherwise} \end{cases}$$

where $\text{argmax}_j v_{t_i}^j$ refers to any single item j which maximizes $v_{t_i}^j$. $\mathcal{M}_T$ sets $p_i = 0$ for all i, so no players make payments. The universally truthful mechanism chooses a size k subset T uniformly at random, and uses mechanism $\mathcal{M}_T$.

An item is added at each step, so $|S_k| = k$ and we end up with a valid allocation. To see that this allocation has an expected social welfare of at least k/n, first note that players in T have their values maximized by construction. As we are choosing T uniformly at random, each player has chance k/n to be in the first T, so each player gets at least a k/n fraction of his maximum value in expectation. Thus, the expected approximation ratio is at most n/k.

Now, we need only see that each $\mathcal{M}_T$ is truthful and runs in polynomial time. Players $i \notin T$ are ignored, so they have no incentive to lie. Players in T have their value maximized, so they also have no incentive to lie. Thus, $\mathcal{M}_T$ is truthful.

$\mathcal{M}_T$ runs in time $O(km)$ independent of T. There are k iterations to compute the sets $S_1, \ldots, S_k$. Each iteration only requires finding $\text{argmax}_j v_i^j$ and $\min_{j \notin S_{i-1}} v_i^j$, each of which can be found in $O(m)$ time. Checking whether $\text{argmax}_j v_i^j$ is in S_{i-1} takes $O(|S_{i-1}|) \subseteq O(m)$ time. Thus, each step takes $O(m)$ time. $v_{t_i}(S_{i-1})$ is then compared to $\max_j v_{t_i}^j$, with the corresponding item possibly being added to S_{i-1} to arrive at S_i. As there are only m such values, this step also only requires $O(m)$ time. So each iteration requires only $O(m)$ time, for a total of $O(km)$ time. □

The idea in Theorem 2.1 can be generalized to show that for any valuation function V and any constant ℓ, that if PV_ℓ can be solved exactly, there is a universally truthful n/ℓ approximation for PV.

Theorem 2.2. *Let V be a class of valuation functions. If PV_ℓ can be solved exactly in polynomial time, there exists a polynomial time universally truthful mechanism which approximates the social welfare of PV with a ratio of n/ℓ.*

Proof. Similarly to the proof of Theorem 2.1, we construct a mechanism $\mathcal{M}_T$ for each subset $T \subseteq [n], |T| = \ell$. This mechanism runs an algorithm which exactly maximizes the social welfare for the players in T and ignores all other players. This is different from the mechanism in the proof of Theorem 2.1 in that we find an S which maximizes $\sum_{i \in T} v_i(S)$ rather than maximizing $v_i(S)$ for each $i \in T$. Players outside of T are charged 0 and players in T are charged VCG payments.

Let S^* be an optimal allocation and A_T be the allocation found by $\mathcal{M}_T$. The expected social welfare is

$$\sum_{T\subseteq\binom{[n]}{\ell}} \frac{1}{\binom{n}{\ell}} \sum_{i\in[n]} v_i(A_T).$$

As the players in T maximize their social welfare, $\sum_{i\in[n]} v_i(A_T) \geq \sum_{i\in T} v_i(S^*)$. So the expected social welfare is at least

$$\sum_{T\in\binom{[n]}{\ell}} \frac{1}{\binom{n}{\ell}} \sum_{i\in T} v_i(S^*) = \frac{1}{\binom{n}{\ell}} \sum_{T\in\binom{[n]}{\ell}} \sum_{i\in T} v_i(S^*).$$

Since each player i appears in $\binom{n-1}{\ell-1}$ sets T of size ℓ, the above double summation is equal to $\sum_{i\in[n]} \binom{n-1}{\ell-1} v_i(S^*)$. Note that $\ell\binom{n}{\ell} = n\binom{n-1}{\ell-1}$ (as both count the number of ways to select a committee of size ℓ with a chairperson from a group of n people), so

$$\frac{1}{\binom{n}{\ell}} = \frac{\ell}{n\binom{n-1}{\ell-1}}.$$

Using this identity, we get a total social welfare of at least

$$\begin{aligned}
\frac{1}{\binom{n}{\ell}} \sum_{i\in[n]} \binom{n-1}{\ell-1} v_i(S^*) &= \frac{\binom{n-1}{\ell-1}}{\binom{n}{\ell}} \sum_{i\in[n]} v_i(S^*) \\
&= \frac{\ell\binom{n-1}{\ell-1}}{n\binom{n-1}{\ell-1}} \sum_{i\in[n]} v_i(S^*) \\
&= \frac{\ell}{n} \sum_{i\in[n]} v_i(S^*).
\end{aligned}$$

So this mechanism has an expected value of at least ℓ/n times the maximum social welfare, giving an expected approximation ratio of at most n/ℓ.

Now, we need only to see that this is universally truthful and runs in polynomial time. Each $\mathcal{M}_T$ is truthful for players outside of T because their valuations do not affect the outcome. $\mathcal{M}_T$ runs in polynomial time if PV$_\ell$ can be solved exactly in polynomial time, and truthfulness follows from the use of VCG payments. □

Using Theorem 2.2, we get the following corollaries.

Corollary 2.3. *For any constant ℓ, there exists a polynomial-time universally truthful mechanism for PFS which achieves an expected approximation ratio of at most n/ℓ.*

Proof. This follows from Theorem 2.2 and Theorem 3.21, which shows that PFS$_\ell$ can be solved in polynomial time for any constant ℓ. □

Corollary 2.4. *There exists a polynomial-time universally truthful mechanism for PMU which achieves an expected approximation ratio of at most $n/2$.*

Proof. This follows from Theorem 2.2 and Theorem 3.13, which shows that PMU_2 can be solved in polynomial time. □

Corollary 2.5. *There exists a polynomial-time universally truthful mechanism for PC with an expected approximation ratio of at most n.*

Proof. This follows from Theorem 2.2. We need only see that PC_1 can be solved exactly in polynomial time. Clearly, allocating the k items of highest value to the single player will maximize the social welfare. Thus, PC_1 can be solved in polynomial time, completing the proof. □

2.2 Communication Complexity

Another approach to showing the hardness of mechanism design is communication complexity. Rather than show that a problem is difficult to compute the answer to or that truthfulness combined with computational efficiency leads to inapproximability, a communication complexity approach shows that even attaining enough information to solve the problem requires excessive communication. If a problem requires exponential communication to solve, then it is impossible to find a polynomial-time solution regardless of truthfulness. This approach has been useful for demonstrating hardness of subadditive auctions [17, 28, 35] and, more recently, public projects [12].

Unfortunately, this strategy does not work for problems where the valuations have succinct representations. If each valuation functions can be represented in space polynomial in m, then each player need only communicate polynomial information to the mechanism in order for an exact solution to be found, after which VCG payments allow for an exact truthful solution. We restrict our attention in this dissertation to succinctly represented valuations.

Succinct representation need not be a complete barrier to communication complexity results. Query models limit the communication to the mechanism posing certain questions to the players, then receiving the answer back. The two query types of interest in this dissertation are value queries and demand queries.

Definition 2.2 (Value Query)**.** *A value asks what the value of a set S is to player i. A value query thus consists of a set S. The response to a value query is $v_i(S)$.*

Definition 2.3 (Demand Query)**.** *A demand query asks for a demand set for player i under prices $p_1, \dots, p_m$. A demand query thus consists of a set S and a set of m prices $p_1, \dots, p_m$. The response to a demand query is a set S maximizing $v_i(S) - \sum_{j \in S} p_j$.*

A polynomial number of demand queries are sufficient to compute a value query [4], so demand queries are strictly more powerful.

In the query model, it is possible that even a valuation with succinct representation could require exponential communication, as seen in Lemma 2.6 below.

Lemma 2.6. *PFS_1 requires exponential communication to solve exactly if communication is limited to value queries. Furthermore, this is true even when v_1 is chosen from a set of valuation functions which can be represented in $\tilde{O}(m^2)$ bits.*

Proof. We will build a function by first picking a set $T \subseteq [m]$ of size $|T| = m/2$. We build the fractionally subadditive function out of the following additive functions. For $i = 1, \ldots, m$,

$$v_1^{(i)}(S) = \begin{cases} 1, & i \in S \\ 0, & \text{otherwise} \end{cases}$$

as well as $v_1^{(m+1)}(S) = 2|S|/m$ and $v_1^{(m+2)}(S) = |S \cap T|(2/m + 1/m^2)$. Set $v_1(S) = \max_i v_1^{(i)}(S)$. $v_1^{(1)}, \ldots, v_1^{(m+2)}$ require $\tilde{O}(m)$ bits each, as they each need only have m values, 1 for each item, and each value requires $\tilde{O}(1)$ bits. As there are $O(m)$ functions each requiring $\tilde{O}(m)$ bits, there are a total of $\tilde{O}(m^2)$ bits required to represent v_1.

Claim 2.6.1.

$$v_1(S) = \begin{cases} 1, & 0 < |S| < m/2 \\ 1 + 2/m, & S = T \\ 2|S|/m, & \textit{otherwise} \end{cases} .$$

Proof. First, let $S = \emptyset$. Then $v_1(S) = 0$, as all of the functions that S consists of are additive. This corresponds to the above case $2|S|/m = 2(0)/m = 0$.

If $0 < |S| < m/2$, $v_1^{(i)}(S) \in \{0, 1\}$ for $i \leq m$ and for every $i \in S$, $v_1^{(i)}(S) = 1$. So $v_1(S) \geq 1$. Furthermore,

$$\begin{aligned} v_1^{(m+1)}(S) &= 2|S|/m \\ &< 2(m/2)/m \\ &= 1 \end{aligned}$$

and

$$\begin{aligned} v_1^{(m+2)}(S) &= |S \cap T|(2/m + 1/m^2) \\ &\leq |S|(2/m + 1/m^2) \\ &\leq (m/2 - 1)(2/m + 1/m^2) \\ &= 1 + 1/m - (2/m + 1/m^2) \\ &= 1 - 1/m - 1/m^2 \\ &< 1. \end{aligned}$$

So $v_1(S) = 1$.

Now, consider $S = T$.

$$\begin{aligned} v_1^{(m+2)}(T) &= |T \cap T|(2/m + 1/m^2) \\ &= |T|(2/m + 1/m^2) \\ &= (m/2)(2/m + 1/m^2) \\ &= 1 + 2/m \end{aligned}$$

$v_1^{(i)}(T) \leq 1 < 1 + 2/m$ for $i \leq m$, so these won't be the maximum.

$$\begin{aligned} v_1^{(m+1)}(T) &= \frac{2|T|}{m} \\ &= \frac{2(m/2)}{m} \\ &= 1 \\ &< 1 + 2/m, \end{aligned}$$

so this isn't the maximum either. So $v_1(T) = 1 + 2/m$.

Now, suppose $|S| = m/2, S \neq T$. As $S \neq T$, $|S \cap T| < |S| = m/2$, so $|S \cap T| \leq m/2 - 1$. So

$$\begin{aligned} v_1^{(m+2)}(S) &= |S \cap T|(2/m + 1/m^2) \\ &= (m/2 - 1)(2/m + 1/m^2) \\ &= m/2(2/m + 1/m^2) - (2/m + 1/m^2) \\ &= 1 + 1/m - (2/m + 1/m^2) \\ &< 1 + 1/m - 1/m \\ &= 1 \end{aligned}$$

$v_1^{(m+1)}(S) = 2|S|/m = 1$. For $i \leq m$, $v_1^{(i)}(S) \leq 1$. So $v_1^{(m+1)}(S) = 1 = 2|S|/m$.

Finally, let $|S| > m/2$. $v_1^{(m+1)}(S) = 2|S|/m$. As $|S| > m/2$, $2|S|/m \geq 1 + 2/m$. Note that $|S \cap T| \leq |T| = m/2$, so

$$\begin{aligned} v_1^{(m+2)}(S) &= |S \cap T|(2/m + 1/m^2) \\ &\leq (m/2)(2/m + 1/m^2) \\ &= 1 + 1/(2m) \\ &< 1 + 2/m \\ &\leq 2|S|/m. \end{aligned}$$

For $i \leq m$, $v_1^{(i)}(S) \leq 1 < 2|S|/m$, so the maximum value is $v_1(S) = 2|S|/m$. □

Now, consider an instance of PFS_1 with a single player whose valuation is constructed as above for some T and $k = m/2$. For any allocation $S \neq T$, $|S| = m/2$, so the social welfare is $2|S|/m = 1$. If T is allocated, the social welfare is $1 + 2/m$. Thus, in order to maximize the social welfare, T must be allocated.

We now show that finding T requires exponentially many value queries in the worst case. In particular, we show that this requires $\binom{m}{m/2} - 1$ queries. Suppose a query is made with a set $S, |S| \neq m/2$. In this case, $v_1(S)$ only depends on $|S|$. So nothing is learned about T by making this query. Now, suppose $|S| = m/2$. Then $v_1(S) = 1$ regardless of T if $S \neq T$ and $1 + m/2$ if $S = T$. So if $S \neq T$, this query only rules out S.

Suppose we make queries $\mathcal{S} = \{S_1, \ldots, S_\ell\}$ where $S_i \neq T$ for all I, and let $\mathcal{T} = \{S : |S| = m/2, S \notin \mathcal{S}\}$. For every $S \in \mathcal{T}$, $S = T$ is consistent with the queries so far. So while $|\mathcal{T}| > 1$, we don't know what T is, so for some T, the next query is not T. Thus, in the worst case, we make queries $S \neq T$ until $|\mathcal{T}| = 1$. As $|\mathcal{T}|$ starts at $\binom{m}{m/2}$ and each query reduces it by at most 1, it requires $\binom{m}{m/2} - 1$ queries in the worst case before $|\mathcal{T}| = 1$. Thus, it requires exponentially many value queries to maximize the social welfare. □

A result like Lemma 2.6 is interesting, but ultimately any communication complexity result depends on an inability to communicate player valuations. We feel that this type of result is unsatisfying for succinctly represented valuations, and therefore avoid the use of communication complexity. To demonstrate the ineffectiveness of this approach for the problems we study, we show that value queries are sufficient to produce a succinct representation for all of the submodular valuation classes in Figure 1.1 other than multi-unit-demand. As a polynomial number of demand queries are sufficient to simulate a value query [5], this also demonstrates that demand queries can be used to produce a succinct valuation.

Lemma 2.7. *There exists a polynomial-time algorithm which finds a succinct representation for a capped-additive valuation using only value queries.*

Proof. A capped-additive valuation can be represented by the per-item values v_i^j for each j and the value cap c. $v(\{j\}) = v_i^j$, so this value query is sufficient to find v_i^j and $v([m]) = \min(c, \sum_j v_i^j)$, so this query either finds c or shows that c is large enough that the actual value of c is irrelevant. Thus, setting $c = v([m])$ yields a succinct representation for v_i. This only requires $m + 1$ queries, which is polynomial in m. □

Corollary 2.8. *There exists a polynomial-time algorithm which finds a succinct representation for an additive valuation using only value queries.*

Proof. Additive valuations are a special case of budget-additive valuations, so this is a special case of Lemma 2.7. □

Lemma 2.9. *There exists a polynomial-time algorithm which finds a succinct representation for a unit-demand valuation using only value queries.*

Proof. As in Lemma 2.7, we can find the per-item values v_i^j by querying the values $v_i(\{j\})$. This fully describes the unit-demand valuation and takes only $O(m)$ time. □

Lemma 2.10. *Let A be a randomized algorithm which makes demand queries about a valuation function v, and outputs a possible representation r of v. For any constants $\epsilon, \delta > 0$, if A outputs a correct representation r of v with probability at least ϵ, then for sufficiently large m, it makes more than $2^{m/4}$ queries with probability at least $1 - \delta$.*

Proof. In order to show this, we will build an exponentially large set of valuation functions such that any given query will have the same result on all but at most one of the functions.

We create a set of valuation functions over $m = 2\ell + 1$ items. For each set $T \subseteq [2\ell]$ such that $|T| = \ell$, we define a function v_T such that for any $S \subseteq [2\ell]$,

$$v_T(S) = \begin{cases} 2|S|, & |S| \leq \ell \\ 2\ell + 1, & \text{otherwise} \end{cases}$$

$$v_T(S \cup \{2\ell + 1\}) = \begin{cases} 2|S| + 3, & |S| < \ell \\ 2\ell + 1, & S = T \\ 2\ell + 2, & \text{otherwise} \end{cases} .$$

Let S^* be the set returned by a demand query with prices $p_1, \ldots, p_m$. If $p_1, \ldots, p_m$ are such that S^* is a demand set for each $v_S, S \in \binom{[2\ell]}{\ell}$, nothing is learned from this query. So we will analyze this assuming that there exists some S, T such that S^* is a demand set for v_T but not v_S.

Claim 2.10.1. *Let $p_1, \ldots, p_m$ be a demand query such that for some S, T, there exists some S^* that is a demand set for v_S but not for v_T. Then one of the following is true:*

1. *For some $R \in \{S, T\}$, $R \cup \{2\ell + 1\}$ is the only demand set for all value functions v_L, $L \neq R$ and $R \cup \{2\ell + 1\}$ is not a demand set for v_R*
2. *$S \cup \{2\ell + 1\}$ and $T \cup \{2\ell + 1\}$ are the demand sets for all value functions v_L, $L \neq S, T$ and $S \cup \{2\ell + 1\}$ is the only demand set for v_T and $T \cup \{2\ell + 1\}$ is the only demand set for v_S.*

Proof. We will examine two cases of S^*.

The first case we examine is that $S^* = T \cup \{2\ell+1\}$. Let S' be a demand set for v_S. Assume by way of contradiction that that $S^* \neq S'$.

$$\begin{aligned}
v_T(S') - \sum_{j \in S'} p_j &\geq v_S(S') - \sum_{j \in S'} p_j \quad \text{because } S' \neq T \cup \{2\ell+1\} \\
&\geq v_S(S^*) - \sum_{j \in S^*} p_j \quad \text{because } S' \text{ is a demand set for } v_S \\
&= 2\ell + 2 - \sum_{j \in S^*} p_j \\
&> 2\ell + 1 - \sum_{j \in S^*} p_j \\
&= v_T(S^*) - \sum_{j \in S^*} p_j,
\end{aligned}$$

which contradicts that S^* is a demand set for v_T. So if $S^* = T \cup \{2\ell+1\}$ is a demand set for v_T, it is also a demand set for any v_S. Therefore, $S^* \neq T \cup \{2\ell+1\}$.

The second case we consider is that $S^* \neq T \cup \{2\ell+1\}$. In this case, we will now show that $T \cup \{2\ell+1\}$ is the only demand set for v_S for any $S \neq T$. Suppose by way of contradiction that v_S has some demand set $S' \neq T \cup \{2\ell+1\}$. Note that $S' \neq S \cup \{2\ell+1\}$, as if $S' = S \cup \{2\ell+1\}$,

$$\begin{aligned}
v_T(S') - \sum_{j \in S'} p_j &> v_S(S') - \sum_{j \in S'} p_j \\
&\geq v_S(S^*) - \sum_{j \in S^*} p_j \quad \text{because } S' \text{ is a demand set for } v_S \\
&= v_T(S^*) - \sum_{j \in S^*} p_j \quad \text{because } S^* \neq T \cup \{2\ell+1\}, S'.
\end{aligned}$$

So

$$\begin{aligned}
v_T(S') - \sum_{j \in S'} p_j &\geq v_S(S') - \sum_{j \in S'} p_j \quad \text{because } v_T(S') < v_S(S') \Rightarrow S' = T \cup \{2\ell+1\} \\
&> v_S(S^*) - \sum_{j \in S^*} p_j \quad \text{because } S' \text{ is a demand set of } v_S \text{ but } S^* \text{ is not} \\
&= v_T(S^*) - \sum_{j \in S^*} p_j \quad \text{because } S' \neq S \cup \{2\ell+1\}, T \cup \{2\ell+1\},
\end{aligned}$$

which contradicts that S^* is a demand set of v_T. So the only demand set for v_S under prices $p_1, \ldots, p_m$ is $T \cup \{2\ell+1\}$.

Note that if every $S \neq T$ does not have S^* as a demand set, the above proof shows that every v_S has $T \cup \{2\ell+1\}$ as a demand set, satisfying case 1. So now we assume that there exists some $U \neq T$ where v_U has S^* as a demand set. Let $T' = T \cup \{2\ell+1\}$. Suppose by way of contradiction that v_U has a demand set U' which is neither $T \cup \{2\ell+1\}$ nor $S \cup \{2\ell+1\}$.

$$\begin{aligned}
v_U(U') - \sum_{j \in U'} p_j &= v_S(U') - \sum_{j \in U'} p_j \\
&< v_S(T') - \sum_{j \in T'} p_j \quad \text{because } T' \text{ is the only demand set for } v_S \\
&= v_U(T') - \sum_{j \in T'},
\end{aligned}$$

which contradicts that U' is a demand set for v_U. So the only possible demand sets for v_U are $S \cup \{2\ell+1\}$ and $T \cup \{2\ell+1\}$. Note that if these are both demand sets for v_U, then they are the both demand sets for any demand function v_W other than v_S and v_T as all these functions have the same values on these sets. This is case 2 in the statement of the claim.

If only one of these sets is a demand set for v_U, then as $S^* \neq T \cup \{2\ell+1\}$ is a demand set for v_U, S^* must equal $S \cup \{2\ell+1\}$ and be the only demand set for v_U. Similarly, this will be the only demand set for any v_W other than v_S, satisfying case 1 in the statement of the claim. $\square$

So looking at the two cases in Claim 2.10.1, any query that can differentiate between two functions will either have the same result for all valuations other than v_T, or it will return one of $T \cup \{2\ell+1\}, S \cup \{2\ell+1\}$ indicating that the valuation is not v_S or not v_T. Note that if we make two queries of the first type, one for v_S and one for v_T, that we learn everything that the second type of query could have shown, plus possibly confirmation that the valuation function is v_T. Thus, any query matching the second case in Claim 2.10.1 can be simulated by two queries matching the first case in Claim 2.10.1. Thus, we can assume that all queries match the first case with only a factor of 2 increase in the number of queries. So every query will result in $T \cup \{2\ell+1\}$ for all valuation functions except v_T, and is therefore an indicator for whether the valuation function is v_T. We denote such a query by Q_T.

Now, suppose that we choose a valuation function v_T by choosing T uniformly at random. We will show inductively that if we only make γ different queries Q_S, the probability that one of the queries is equal to Q_T is $\gamma / \binom{2\ell}{\ell}$. For $\gamma = 0$, this is trivially true. If this is true for γ, then by symmetry, all sets S such that Q_S has not been asked are equally likely. So the probability of any of these is $1 / \left(\binom{2\ell}{\ell} - \gamma \right)$. Thus, the probability that query $\gamma+1$ is the first query equal to Q_T is

$$\left(1 - \frac{\gamma}{\binom{2\ell}{\ell}}\right) \frac{1}{\binom{2\ell}{\ell} - \gamma} = \frac{\binom{2\ell}{\ell} - \gamma}{\binom{2\ell}{\ell}} \cdot \frac{1}{\binom{2\ell}{\ell} - \gamma} = \frac{1}{\binom{2\ell}{\ell}}.$$

So the probability that some query of the first $\gamma+1$ is Q_T is the probability that one of the first γ is Q_T, plus the probability that query $\gamma+1$ is the first one equal to Q_T, or $\gamma/\binom{2\ell}{\ell} + 1/\binom{2\ell}{\ell} = (\gamma+1)/\binom{2\ell}{\ell}$.

Thus, the probability that Q_T is queried within $2 \cdot 2^{m/4}$ queries is $2 \cdot 2^{m/4} / \binom{2\ell}{\ell}$. For any $\delta > 0$,

$2 \cdot 2^{m/4} \in o\left(\binom{2\ell}{\ell}\right)$, so $2 \cdot 2^{m/4}/\binom{2\ell}{\ell}$ is less than δ for sufficiently large m.

Note that if none of the $2 \cdot 2^{m/4}$ queries is Q_T, then the remaining $\binom{2\ell}{\ell} - 2^{m/4}$ sets of size ℓ are T with equal probability by symmetry. So the probability of returning the correct one after these queries is $1/\left(\binom{2\ell}{\ell} - 2 \cdot 2^{m/4}\right)$, which is less than ϵ for sufficiently large ℓ, as $2 \cdot 2^{m/4} \in o\left(\binom{2\ell}{\ell}\right)$. As Q_T is not queried within $2 \cdot 2^{m/4}$ queries with probability at least $1 - \delta$, this shows that with probability $1 - \delta$, A must perform more than $2 \cdot 2^{m/4}$ queries Q_S. As $2 \cdot 2^{m/4}$ queries Q_S are sufficient to simulate any $2^{m/4}$ queries, this shows that for sufficiently large m, A must perform more than $2^{m/4}$ queries with probability at least $1 - \delta$.

Now, we need only to see that v_T is a multi-unit-demand function. We will define $\ell + 1$ unit-demand valuation functions. The first ℓ will be identical and defined by

$$v_T^{(i)}(S) = \begin{cases} 0, & S = \emptyset \\ 3, & 2\ell + 1 \in S \\ 2, & \text{otherwise} \end{cases}$$

which is unit-demand, as it gives value 2 to items 1 through 2ℓ and value 3 to item $2\ell + 1$. The last unit-demand function is the one that depends on T,

$$v_T^{(\ell+1)} = \begin{cases} 1, & S \cap ([2\ell] \setminus T) \neq \emptyset \\ 0, & \text{otherwise} \end{cases}$$

which is unit-demand because it gives value 1 to everything in $[2\ell]\setminus T$ and 0 to everything else.

Now we consider valuations over sets $S \subseteq [2\ell]$. If $|S| \leq \ell$, then each of the items in S can be matched with one of the first ℓ unit-demand valuations, for a value of 2 each. As the other valuation gives each of these value at most 1, this is the maximum possible per-item value. So the total value is $2|S|$.

If $|S| > \ell$, then there is at least one item in S from $[2\ell \setminus T]$. Match this item with $v_T^{(i)}$ for value 1, and match ℓ of the other items with the first ℓ for value 2ℓ. This gives a total value $2\ell + 1$, and each unit-demand valuation yields maximum value for items in S. Thus, the value of this set is $2\ell + 1$. So our construction is equal to $v_T(S)$ above.

Now we consider valuations over sets $S \cup \{2\ell + 1\}$ for $S \subseteq [2\ell]$. If $|S| < \ell$, then one of the first ℓ unit-demand functions can be matched with item $2\ell + 1$ for value 3, and all other items can be matched with others of the first ℓ items for value $2|S|$, for a total value of $2|S| + 3$. No higher value is possible, as no unit-demand function values $2\ell + 1$ more than 3, and no unit-demand function values anything in S more than 2.

If $S = T$, then nothing can be matched with $v_T^{(\ell+1)}$ for any value, as $v_T^{(\ell+1)}$ gives value 0 to items in T as well as to $\ell + 1$. So we have 2 possibilities. First, we don't match item $2\ell + 1$. In this case, we can match all items in S to the first ℓ unit-demand valuations for value 2ℓ. Otherwise, we match item $2\ell + 1$ to one of the first ℓ unit-demand valuations, and $\ell - 1$ items from S to the rest for a total value $3 + 2(\ell - 1) = 2\ell + 1$. So the value in this case is $2\ell + 1$.

Finally, we are left with the case $|S| \geq \ell, S \neq T$. We are again left with 2 possibilities. If item $2\ell + 1$ is not matched, we have value at most $2\ell + 1$, 1 from $v_T^{(\ell+1)}$ and 2 from every other $v_T^{(i)}$. Otherwise, since $|S| \geq \ell$ and $S \neq T$, there is some item from $[2\ell] \setminus T$ in S. Match this item to $v_T^{(\ell+1)}$ for value 1. Match item $2\ell + 1$ to one of the first ℓ unit-demand valuations, and $\ell - 1$ of the other items to the rest. This gives value $1 + 3 + 2(\ell - 1) = 2\ell + 2$. All unit-demand valuations not matched with item $2\ell + 1$ have maximum value given that they are not matched with this item. So no higher value can be found in this case. Thus, the value for $S \cup \{2\ell + 1\}$ is $2\ell + 2$ in this case. So our construction is equal to $v_T(S \cup \{2\ell + 1\})$ above, completing the proof. □

Lemma 2.11. *There exists a polynomial-time algorithm which finds a succinct representation for a weighted coverage valuation using only value queries if the universe U over which the weighted coverage valuation is defined has polynomial size.*

Proof. The valuation v_i consists of subsets $S_1, \ldots, S_m$ of U, together with weights w_u. Let $T_u = \{j : u \in S_j\}$ for each $u \in U$. If two elements u, u' of U have identical sets $T_u, T_{u'}$, then the valuation does not change if we replace them with a single element u'' with set $T_{u''} = T_u$ and $w_{u''} = w_u + w_{u'}$. So we will construct a representation in which each element $u \in U$ has a unique set T_u. If we can find all the sets T_u together with their corresponding weights w_u, we can construct the succinct representation by simply creating a universe U with an element u for each T_u we find and creating sets $S_1, \ldots, S_m$ such that u is in each set $S_j, j \in T_u$.

We find all the sets T_u through an iterative process which finds all sets $T_u \cap [1], \ldots, T_u \cap [m]$. At each step, we keep track of the weights of the sets we've built so far, $\sum_{T_u \cap [i] = T} w_u$. We begin with the empty set and $\sum_{T_u \cap \emptyset = \emptyset} w_u = v_i([m])$.

Given all sets $T_u \cap [j]$, we find the sets $T_u \cap [j + 1]$ and their corresponding sum of weights as follows. For a given set $T \subseteq [j]$, let w be the total weight of sets T_u such that $T_u \cap [j] = T$. We want to figure out the total weights of sets such that $T_u \cap [j + 1] = T$ and the weight of sets such that $T_u \cap [j + 1] = T \cup \{j + 1\}$. If we make two queries to compute $v_i(\{j + 1\} \cup [j] \setminus T) - v_i([j] \setminus T)$, the resulting difference is the total weight of sets T_u which contain $j + 1$ but do not contain anything from $[j + 1] \setminus T$. So this is the total weight of subsets of $T \cup \{j + 1\}$ which contain $j + 1$.

In order to find the weight of $T_u \cup \{j + 1\}$, we need to subtract the weight of sets which contain $j + 1$ and are proper subsets of T. If we process each T in order of size, then each of these has already been computed as the weight of some $T_{u'} \cap [j + 1]$ which is a proper subset of $T \cap [j + 1]$ containing $j + 1$. So we can subtract these weights to find the weight for $T_u \cap [j + 1] = T \cup \{j + 1\}$. Subtracting this from w, we get the total weight of sets T_u such that $T_u \cap [j + 1] = T$.

After iterating through all T, we have computed the total weight of each of the at most $|U|$ sets $T_u \cap [j+1]$ with nonzero weight. We need not keep track of sets with 0 weight, as these aren't part of the valuation. So after m iterations, we can construct a succinct representation. Each iteration requires that we make at most 2 queries and $O(|U|)$ subtractions per set T_u. As U is polynomial in size, each iteration thus takes $O(|U|^2) \in poly(m)$ time, for a total runtime of $m \cdot poly(m) \in poly(m)$. □

Corollary 2.12. *There exists a polynomial-time algorithm which finds a succinct representation for a coverage valuation using only value queries.*

Proof. A coverage valuation is a special case of weighted coverage where all weights are 1, so we need only find a weighted coverage representation as in Lemma 2.11, then create w_u copies of each item u. □

Corollary 2.13. *There exists a polynomial-time algorithm which finds a succinct representation for a scaled coverage valuation using only value queries.*

Proof. As scaled coverage is a special case of weighted coverage where all weights are equal, we can begin by finding a weighted coverage valuation as shown in Lemma 2.11. Unlike in Corollary 2.12, we can't simply pull the answer out of the weights. If all weights are equal, this is possible, but there may be multiple copies of some items, resulting in different weights in the weighted coverage representation. To find an appropriate scale factor, pick an item $u \in U$. Assuming there are ℓ copies of u, the scale factor is $\alpha = w_u/\ell$. This is consistent if each $w_{u'}/\alpha$ is an integer for each $u' \neq u$. So we need only check values of ℓ starting at 1 and increasing until we find a consistent α. If $|U|$ is polynomial in a representation of v_i, then $\ell \leq |U|$, so finding α only requires a polynomial number of iterations, after which we can create w_u/α copies of each u to arrive at a succinct representation. □

We build a model in Chapter 5 in which mechanisms make use of queries in addition to succinct representations.

Chapter 3

Combinatorial Public Projects

The combinatorial public projects problem was first introduced in [36], where it was shown that efficient truthful mechanisms cannot achieve an approximation ratio better than $\sqrt{m}$ for general submodular valuations unless NP has polynomial circuits. This matches a $\sqrt{m}$ truthful approximation algorithm shown in [41] which only requires the ability to compute value queries, which can be trivially computed in polynomial time directly from the definition of all valuation classes we consider (except for multi-unit-demand, for which polynomial-time computation of value queries is demonstrated in Lemma 1.3). The hardness result was shown in two parts. First, it was demonstrated that all truthful mechanisms for the problem are affine maximizers (see Definition 1.14), then it was shown that affine maximizers cannot achieve a constant approximation ratio unless NP has polynomial circuits.

As combinatorial public projects are a relatively new problem class, we seek to get an understanding of the purely computational issues as well as the results when truthful computation is taken into account. In this way, we hope to gain a better understanding of the problem by looking into which features lead to worst-case computational hardness and which ones lead to hardness to approximate truthfully.

As we will show, public projects retain their computational complexity even for very simple valuation classes. Some of these classes are so simple that truthful approximations which are not VCG-based exist and can even achieve better approximation ratios than the bounds we show for maximal-in-range mechanisms. We will demonstrate some of these truthful mechanisms for a few interesting cases.

Our VCG-based hardness results make use of the VC-dimension.

Definition 3.1 (Shattered). *Let S be a subset of the power set 2^U for some universe U. A set $T \subseteq U$ is* shattered *by S if $\{s \cap T : s \in S\} = 2^T$.*

Definition 3.2 (VC-dimension). *Let S be a subset of the power set 2^U for some universe U. The VC-dimension of S is the size of the largest set $T \subseteq U$ such that T is shattered by S. We say that*

T exhibits *the VC dimension of* S.

In order to demonstrate a large VC dimension, we use the Sauer-Shelah lemma.

Lemma 3.1 (Sauer-Shelah Lemma)**.** *Let S be a subset of 2^T where $|T| = \ell$ and $|S| > \sum_{i=0}^{k-1} \binom{\ell}{i}$. The VC dimension of S is at least k.*

As the binomial coefficients in this lemma are not particularly useful for our purposes, we make use of the following corollary.

Corollary 3.2. *Let S be a subset of 2^T with $|T| = \ell$. For any constants $\alpha > 0, \delta > \alpha$, and $\epsilon > 0$, the following holds for all sufficiently large ℓ: if $|S| > (1+\epsilon)^{\epsilon \ell^{\delta}}$ then S has VC dimension at least ℓ^{α}.*

Proof. Note that for sufficiently large ℓ, $\ell^{\alpha} < \ell/2$, so $\binom{\ell}{\ell^{\alpha}} \geq \binom{\ell}{i}$ for $i < \ell^{\alpha}$.

$$\begin{aligned}
\sum_{i=0}^{\ell^{\alpha}-1} \binom{\ell}{i} &\leq \sum_{i=0}^{\ell^{\alpha}-1} \binom{\ell}{\ell^{\alpha}} \\
&\leq \ell^{\alpha} \left(\frac{e\ell}{\ell^{\alpha}}\right)^{\ell^{\alpha}} \\
&= \ell^{\alpha} \left(e\ell^{1-\alpha}\right)^{\ell^{\alpha}} \\
&= (1+\epsilon)^{\alpha \log_{1+\epsilon} \ell + \ell^{\alpha} \log_{1+\epsilon}(e\ell^{1-\alpha})} \\
&= (1+\epsilon)^{\ell^{\alpha}((1-\alpha)\log_{1+\epsilon} \ell + \log_{1+\epsilon} e + o(1))} \\
&= (1+\epsilon)^{\ell^{\alpha}(\ell^{o(1)})} \\
&= (1+\epsilon)^{\ell^{\alpha+o(1)}}
\end{aligned}$$

which is less than $|S| = (1+\epsilon)^{\epsilon \ell^{\delta}}$ for sufficiently large ℓ, since $\delta > \alpha$. Thus, for sufficiently large ℓ, $|S| > \sum_{i=0}^{\ell^{\alpha}-1} \binom{\ell}{i}$, so by Lemma 3.1, S has VC dimension at least ℓ^{α}. □

The general approach we will use is to first show that the range of any maximal-in-range algorithm which achieves an approximation ratio better than $\sqrt{m}$ must have a VC-dimension of at least m^{α} for some constant α. Thus, there are m^{α} items which are allocated in every possible way, with the rest of the items coming from outside of this set to reach an allocation of size k. As the Sauer-Shelah lemma is non-constructive, we use this to produce a non-uniform reduction which embeds an NP-hard problem into the set of items exhibiting the VC dimension of the range.

The embedding must be done carefully, so that an optimal solution involves some subset of the items exhibiting the VC dimension, together with enough arbitrary items from outside the set to reach an allocation of size k, as $k > m^{\alpha}$ and we have no guarantee of how the range behaves outside of the set exhibiting its VC dimension. After this, we see that if the algorithm runs in polynomial time, we have created a non-uniform family of polynomial circuits for an NP-hard problem. So the algorithm cannot run in polynomial time unless NP has polynomial circuits.

3.1 Unit-Demand Valuations

Unit-demand players are those for whom each item is given a value, and the value of a set S is the maximum value of any item in S. For a formal definition, see Definition 1.17. 2-{0,1}-unit-demand players are unit-demand players who have value 0 or 1 for each item, and value 1 for at most 2 items (see Definition 1.18).

Theorem 3.3. *No polynomial-time algorithm for PU has an approximation ratio of* $e/(e-1)-\epsilon$ *for any constant* $\epsilon > 0$ *unless* $P = NP$*.*

Proof. We will show an approximation preserving reduction from MAX-t-COVER. The problem of MAX-t-COVER takes as input a collection of subsets $\mathcal{F}$ of a set A and an integer t. The goal is to find t sets in $\mathcal{F}$ which have a union of maximum cardinality. It was shown in [21] that MAX-t-COVER cannot be approximated in polynomial time to within $e/(e-1)-\epsilon$ for any constant $\epsilon > 0$ unless P = NP.

Consider a MAX-t-COVER instance over set A with $\mathcal{F} = \{S_1, \ldots, S_m\}$ and number of sets to be chosen t. We create a PU instance with $m = |\mathcal{F}|$ items and $n = |A|$ players. Let $A = \{a_1, \ldots, a_n\}$. Player i's valuation function $v_i(S) = \max_{S_j \in S} v_i^j$ is defined by

$$v_i^j = \begin{cases} 1, & a_i \in S_j \\ 0, & \text{otherwise} \end{cases} .$$

So the value for player i is 1 if a_i is contained in some set $S_j \in S$ and 0 otherwise. As there is one player for each item, the social welfare is the number of items in the union of sets in S. By setting the number of resources allowed to be chosen to $k = t$, the maximum social welfare is equal to the cardinality of the maximum t cover. So if we can approximate the social welfare to within any factor α, we get an α-approximation of MAX-t-COVER as well. So by [21], no polynomial time algorithm can approximate PU with a factor of $e/(e-1)-\epsilon$ unless P = NP. □

The above hardness of approximation is tight, as a greedy approximation achieves a ratio of $e/(e-1)$ [32]. Note that as unit-demand is a special case of multi-unit-demand and of capped-additive, Theorem 3.3 implies the following corollary.

Corollary 3.4. *No polynomial-time algorithm for PMU or for PC has an approximation ratio of* $e/(e-1)-\epsilon$ *for any constant* $\epsilon > 0$ *unless* $P = NP$*.*

Note that the above proof required n players. With only a constant number of players, a polynomial-time exact algorithm is possible.

Theorem 3.5. *For any constant n, PU_n can be solved in polynomial time.*

Proof. In the case that $n \leq k$, it is possible to simply choose an item j maximizing v_i^j for each player i, then choose more items arbitrarily to reach k items. This achieves the maximum value for each player, and thus maximizes social welfare. This clearly only requires polynomial time.

If $n > k$, then there are $\binom{m}{k} \leq m^k < m^n$ possible allocations. As n is a constant, m^n is polynomial in n. So we can simply enumerate all possible solutions in polynomial time, computing the social welfare of each to find the maximum. $\square$

We now consider the limits of truthful mechanisms with unit-demand and 2-{0,1}-unit-demand valuations. We begin by showing limits on maximal-in-range mechanisms. To help with this, we will first show a lemma based on work in [36].

Lemma 3.6. *Let V be a valuation class such that for any set T, it is possible to create an instance of PV (PV_n) where the social welfare of a set S is equal to $|S \cap T|$. Then any algorithm for PV (PV_n) which approximates the social welfare to within $m^{1/2-\epsilon}$ must have a range with VC-dimension at least m^α.*

Proof. Let A be an algorithm for PV (PV$_n$) with range R_A. Let $k = m^{1/2+\epsilon}$. We construct a set $T \subseteq [m]$ by including each $i \in m$ with probability $m^{-1/2+\epsilon}$. Consider an instance where the social welfare of a set S is $|T \cap S|$. We will show that any $S \in R_A$ is exponentially unlikely to approximate the maximum welfare to within $m^{1/2-\epsilon}$, necessitating that $|R_A|$ is exponentially large to have a probability of 1 of approximating T to within $m^{1/2-\epsilon}$.

Applying a Chernoff bound, we find that for any $0 < \delta < 1/2$, the probability that $|T| < (1-\delta)m^{1/2+\epsilon}$ (and therefore, that the maximum social welfare is less than $(1-\delta)m^{1/2+\epsilon}$) is at most $e^{-\delta^2 m^{1/2+\epsilon}/2}$. Applying another Chernoff bound, the probability that $|T \cap S| \geq (1+\delta)m^\epsilon$ is at most $e^{-\delta^2 m^\epsilon/4}$. However, $\frac{(1-\delta)m^{1/2+\epsilon}}{(1+\delta)m^\epsilon} = \frac{1-\delta}{1+\delta}m^{1/2} \geq m^{1/2}/3$, which is greater than $m^{1/2-\epsilon}$ for sufficiently large m. So the first Chernoff bound tells us that the probability that some element $S \in R_A$ has $|S \cap T| > (1+\delta)m^\epsilon$ must be at least $1 - (e^{-\delta^2 m^{1/2+\epsilon}/2})$ in order to guarantee an approximation ratio of at most $m^{1/2-\epsilon}$. By the union bound, we thus have

$$\begin{aligned} |R_A| &\geq \frac{1 - e^{-\delta^2 m^{1/2+\epsilon}/2}}{e^{-\delta^2 m^\epsilon/4}} \\ &= e^{\delta^2 m^\epsilon/4} - e^{-(\delta^2 m^{1/2+\epsilon}/2 - \delta^2 m^\epsilon/4)} \\ &> e^{\delta^2/4m^\epsilon} - 1 \\ &\in e^{\Omega(m^\epsilon)}. \end{aligned}$$

Now, we simply apply Corollary 3.2 to see that for some constant α, this implies that R_A has a VC dimension of at least m^α. □

Lemma 3.7. *Any algorithm which approximates P2U with a ratio of $m^{1/2-\epsilon}$ must have VC dimension m^α for some constant $\alpha > 0$.*

Proof. Given a set T, we can construct an instance of P2U such that the social welfare of a set S is $|S \cap T|$ as follows. Set $n = |T|$ and label the items in T as $T = \{t_1, \dots, t_n\}$. Each player i has a valuation function

$$v_i(S) = \begin{cases} 1, & t_1 \in S \\ 0, & \text{otherwise} \end{cases}$$

which is 2-{0,1}-unit-demand because it sets v_i^j to either 0 or 1 and only has $v_i^j = 1$ for the one value $j = i$. Adding these values over S to find the social welfare, we get value 1 for each t_i in S, so the social welfare is the number of t_i in S, or $|S \cap T|$.

Now we can apply Lemma 3.6 to show that any algorithm which achieves a $m^{1/2-\epsilon}$ ratio has VC dimension at least m^α. □

Theorem 3.8. *No polynomial-time maximal-in-range mechanism can approximate P2U within $m^{1/2-\epsilon}$ for any constant $\epsilon > 0$ unless $NP \subseteq P/poly$.*

Proof. Our proof uses a general structure similar to [36], where the use of VC dimension to bound the approximation ratios of maximal-in-range mechanisms is introduced. Lemma 3.7 shows that any mechanism which approximates P2U within $m^{1/2-\epsilon}$ has VC dimension m^α for some α. We can perform a non-uniform reduction using the subset of items which realizes the VC dimension.

Re-order the items such that the m^α items corresponding to the set which realizes the VC-dimension are the first m^α items. As Corollary 3.2 is non-constructive, this step is not uniform. We show a reduction from vertex cover with m^α vertices. Let k' be the parameter such that the vertex cover instance is positive if there is a vertex cover of size at most k'. Note that $k' \leq m^\alpha \leq k$. Label the vertices $V_1, \dots, V_{m^\alpha}$ and the edges $E_1, \dots, E_\ell$.

The first $2|E|$ players correspond 2 to each edge, and have value 1 if the corresponding edge is covered. So for $i = 1, \dots, \ell$ $v_i(S) = \max_{j \in S} v_i^j$ where

$$v_i^j = \begin{cases} 1, & V_j \in E_i \\ 0, & \text{otherwise} \end{cases}$$

which is 2-{0,1}-unit-demand because each edge contains exactly 2 vertices. For $i = \ell + 1, \dots, 2\ell$, let $v_i = v_{i-\ell}$, so there are 2 players with each of the above valuations. Finally, we have $m - m^\alpha$ more players $2\ell + 1, 2\ell + m - m^\alpha$, one corresponding to each item outside of $[m^\alpha]$. Player $2\ell + i$ has valuation

$$v_{2\ell+i}(S) = \begin{cases} 1, & m^\alpha + i \in S \\ 0, & \text{otherwise} \end{cases}$$

which is 2-{0,1}-unit-demand because only one item has value 1 and the rest have value 0.

If an allocation S maximizes the social welfare, the vertices corresponding to items chosen from $[m^\alpha]$ form a vertex cover. To see this, suppose that there is an S which maximizes the social welfare and does not correspond to a vertex cover. As $k \geq m^\alpha$ and not all items in $[m^\alpha]$ are chosen, some item j in S is larger than m^α. As we don't have a vertex cover, some edge $E_i = (V_{j'}, V_{j''})$ is not covered, so j' is not in S. Consider the social welfare of the set $\{j'\} \cup S \backslash j$ which replaces j with j' in S. The welfare decreases by 1 as player $2\ell + j - m^\alpha$ loses value 1, but this is more than offset by an increase of 1 each for players i and $i + \ell$. So the total change in social welfare is an increase of 1, contradicting that S maximized the social welfare.

So the maximum social welfare is obtained when the items from $S \cap [m^\alpha]$ correspond to a vertex cover. The social welfare from these items is 2ℓ, as each of the first 2ℓ players has value 1 for such an allocation. The remaining items add value 1 each to the welfare, as each of the other items is only valued by 1 player. So if we have a vertex cover of size C, the social welfare is $2\ell + k - C$. So given the maximum social welfare, we can determine the size of the minimum vertex cover.

The maximal-in-range algorithm will find the maximum social welfare, as $[m^\alpha]$ realizes its VC dimension. So for every subset of $[m^\alpha]$, there is an element in the range which contains all items in that subset, together with enough items from $[m] \backslash [m^\alpha]$ to reach k items. Thus, the range includes a set which corresponds to a minimum vertex cover and therefore maximizes the social welfare. As vertex cover is NP-hard and this was a non-uniform reduction, the algorithm cannot run in polynomial time unless $NP \subset P/poly$. □

Theorem 3.8 shows that no maximal-in-range algorithm can achieve an approximation ratio better than $\sqrt{m}$ for P2U. In contrast, a simple non-adaptive greedy algorithm achieves an approximation ratio of 2 without even requiring payments. This establishes a large gap between what is achievable via maximal-in-range and general truthful mechanisms.

Theorem 3.9. *There exists a computationally efficient algorithm for $P\ell U$ that achieves an approximation ratio of ℓ and is truthful without payments.*

Proof. We use a simple algorithm.

1. For each item j let $s_j = |\{i : v_i(\{j\}) = 1\}|$.

2. Sort the m items in decreasing order by the value of s_j, breaking ties arbitrarily

3. Output the set S consisting of the k first resources in the above ordering.

First, we show that the algorithm runs in polynomial time. Note that $0 \leq s_j \leq n$ for all j, so a bucket sort can be used to perform the sorting in $O(m+n)$ time. Selecting the k smallest elements after sorting only requires $O(k)$ time. So this is a linear time algorithm.

We will now show that this has an approximation ratio of at most ℓ. Let S be the allocation chosen by the above algorithm. Every player has a value of either 0 or 1 for S. If a player has a value of 1, we call that player satisfied. For each resource j, let s_j be the number of players satisfied by $\{j\}$. For any set T, $\sum_{j \in T} s_j$ is an upper bound on the social welfare of T. As S contains the k elements which individually satisfy the largest number of players, S maximizes $\sum_{j \in S} s_j$ for sets of size k. So $\sum_{j \in S} s_j$ is an upper bound on the maximum social welfare. Furthermore, as players give value to at most ℓ items each, each player is satisfied by at most ℓ items in S, so the social welfare of S is at least $1/\ell \sum_{j \in S} s_j$, or at least $1/\ell$ times the maximum social welfare. This shows that the algorithm has an approximation ratio of at most ℓ.

Now, we will show that this algorithm has an approximation ratio of at least ℓ, to demonstrate that it is exactly ℓ in the worst case. Consider the PℓU instance with ℓ players, $2\ell - 1$ items and $k = \ell$. Let player 1 have value 1 for items 1 through ℓ and player $i+1$ have value 1 only for item $\ell + i$. If we allocate items 1 and $\ell + 1$ through 2ℓ, we achieve the maximum possible welfare of ℓ, as each player has value 1. However, each item satisfies exactly one of the players. So if we sort the items, they may remain in order $1, \ldots, 2\ell$. Thus, this algorithm will choose items 1 through ℓ, and only player 1 will have nonzero value, resulting in a welfare of 1. So this algorithm is a factor of ℓ from the maximum social welfare in this case.

Finally, we need only show that this algorithm is truthful without payments. Using this as a mechanism for PℓU, each player submits an ℓ-{0,1}-unit-demand function. For the sake of analysis, we will interpret this as votes for each of the at most ℓ items valued by the player. The algorithm as described then chooses the k items with the highest vote tallies. We will consider 2 possible types of lies. First, suppose a player votes for some items which he does not value. This can only change the outcome if it moves one or more of these items into the top k, displacing other items which would have been in the top k without this lie. At best, this change is not helpful, and at worst, all of the items which the player values will be displaced, resulting in a loss of value. So there is no incentive for this type of lie.

Now, suppose that a player lies by not voting for an item he values. The only way this can change the outcome is if it lowers the vote total enough that the item is no longer in the top k. As the player's value is maximized by having this item in the top k, there is no incentive to change the outcome in this way. As there is no incentive for either kind of lie, the player will submit his valuation function truthfully. $\square$

3.2 Multi-Unit-Demand Valuations

Multi-unit-demand valuations (termed "OXS" in [30, 33]) are a generalization of unit-demand valuations in which a player has several unit-demand functions and can assign items to each one (see Definition 1.19).

Unit-demand valuations are a special case of multi-unit-demand valuations, and so our negative results in Sec. 3.1 for P2U extend to PMU. We will therefore focus on PMU$_n$ for constant n. Lemma 1.3 shows how value queries can be computed in polynomial time via bipartite matching. In Theorem 3.13, we show how a similar idea can be used to compute exact solutions to PMU$_2$ (and therefore also PMU$_1$) in polynomial time. First, we show hardness results for PMU$_n$, $n \geq 3$.

Theorem 3.10. *PMU$_3$ is NP-hard.*

Proof. We reduce from 3-Dimensional Matching (3DM). Given a 3DM instance $M \subseteq [q] \times [q] \times [q]$, the goal is to determine whether there exists a set $M' \subset M$ of size q such that no two members of M' share a coordinate. Our reduction is as follows. There are $m = |M|$ items. Label the items $M = \{M_1, \ldots, M_m\}$. The number of items to be chosen is $k = q$. If there is a set of size q such that no two members share a coordinate, then there are q different values for each coordinate in the set. We will simply create a player for each coordinate that has a valuation equal to the number of distinct values seen in that coordinate, so that the social welfare is maximized with a value of $3q$ if no two items coincide on any coordinate.

The ith player values set S by the number of different values for the ith coordinate in triples corresponding to items in S. This valuation is multi-unit-demand because it can be built out of the q unit-demand valuations that value 1 to any item corresponding to a triple with a j in the ith coordinate and 0 to all other items. The maximum possible value is q (1 from each unit-demand valuation), and this is only achievable if every possible value of the ith coordinate appears in S.

If there exists a 3-dimensional matching, we can assign the items corresponding to the triples in the matching to arrive at a social welfare of $3q$. If there is no 3-dimensional matching, then any q of the triples will not cover all values of all coordinates, so for any set of $k = q$ items, some player will have value less than q, for a total value less than $3q$. Thus, the maximum social welfare is $3q$ iff the 3DM instance is positive. □

We now build on the proof of Theorem 3.10 to show that PMU$_3$ cannot be approximated better than $\sqrt{m}$ by a VCG-based mechanism.

Theorem 3.11. *No computationally efficient maximal-in-range mechanism can approximate PMU to $m^{1/2-\epsilon}$ for any constant $\epsilon > 0$ unless $NP \subset P/poly$.*

Proof. The proof here is essentially similar in style to that of Theorem 3.8, in that the proof of NP-hardness can be modified to work within the set which exhibits the VC dimension. As 2-{0,1}-

unit-demand is a special case of multi-unit-demand, Lemma 3.7 shows that any algorithm with a ratio of at most $m^{1/2-\epsilon}$ has a range with VC dimension at least m^{α}.

Re-order the items so that the first m^{α} correspond to the VC dimension. We reduce from 3DM as in the proof of Theorem 3.10. Perform this reduction with 3DM instance with m^{α} triples. The players have value for the items in $[m^{\alpha}]$ according to the reduction in the proof of Theorem 3.10.

We now add unit-demand valuations to player 1's valuation function. For each item $j > m^{\alpha}$, we add a unit-demand function for player 1 which values j at $1/(k+1)$ and all other items at 0. So the social welfare of an allocation S becomes the number of distinct indices covered by the triples corresponding to items in $S \cap [m^{\alpha}]$, plus $1/(k+1)$ times the number of items larger than m^{α}. Thus, a social welfare of $3q + (k-q)/(k+1)$ is achievable iff there is a 3-dimensional matching.

Running the algorithm will result in the set which maximizes the social welfare, as its range includes all possible subsets of of $[m^{\alpha}]$ and the social welfare depends only on how many items outside of $[m^{\alpha}]$ are chosen, not which ones are chosen. Thus, the algorithm cannot run in polynomial time unless $NP \subset P/poly$. □

While Theorem 3.10 leaves open the possibility of a PTAS for any constant number of players, we can rule out this possibility by presenting a hardness of approximation result for 10 (or more) players.

Theorem 3.12. *There exists a constant $\epsilon > 0$ such that it is NP-hard to approximate the PMU_{10} to a ratio of $1+\epsilon$.*

Proof. We will reduce from MAX-3SAT-5. This is a special case of MAX-3SAT in which each variable occurs in exactly 5 clauses. Consider an instance of MAX-3SAT-5 consisting of ℓ clauses, $c_1, \ldots, c_\ell$. Because each clause has 3 variables and each variable is contained in 5 clauses, there are $3\ell/5$ variables $v_1, \ldots, v_{3\ell/5}$. We will start by reducing to an instance with n unit-demand players, then demonstrate that these players can be compressed into 10 multi-unit-demand players without changing the social welfare of any set.

There are $6\ell/5$ items, 2 corresponding to each variable. For each variable v_j, we will have two items labeled j and $\bar{j}$. Choosing j corresponds to setting v_j to true, while choosing $\bar{j}$ corresponds to setting v_j to false. We allow $k = 3\ell/5$ items to be chosen, so one value can be chosen for each variable.

There are two classes of players. The first class has ℓ players, one corresponding to each clause. The player corresponding to clause c_i has value 1 for each item j such that v_j is in c_i and 1 for each item $\bar{j}$ such that $\neg v_j$ is in c_i. Thus, these players have value 1 if their clause is satisfied and 0 otherwise.

The second class of players has $3\ell/5$ players, one for each variable. The player corresponding to v_j has value 5 for items j and $\bar{j}$ and 0 for all other items. If there is some item j for which both

j and $\overline{j}$ are chosen, then by the pigeonhole principle, there is some j' for which neither j' nor $\overline{j'}$ is chosen. This leads to a loss of 5 to the social welfare from these players compared to replacing one of $j, \overline{j}$ with one of $j', \overline{j'}$. As the social welfare from players of the first class for both j and $\overline{j}$ is at most 5, choosing j and j' rather than j and $\overline{j}$ cannot decrease the social welfare. Thus, we will assume without loss of generality that any allocation corresponds to a proper assignment to the variables of the MAX-3SAT-5 instance.

As each allocation corresponds to a proper assignment, the social welfare from the first class of players is the number of satisfied clauses and the social welfare from the second class of players is exactly 3ℓ. So the total social welfare is 3ℓ plus the number of satisfied clauses. Thus, a social welfare of $3\ell + x$ corresponds to an assignment satisfying x clauses.

It was shown in [21] that there exists a positive constant δ such that it is NP-hard to distinguish between the case that all clauses are satisfiable in a MAX-3SAT-5 instance and that only a $1 - \delta$ fraction are. Let ϵ be a positive constant such that $1/(1 + \epsilon) > 1 - \delta/4$. If the instance of MAX-3-SAT that we are reducing from is one in which all ℓ clauses are satisfiable, then in the produced PU instance, the corresponding assignment has a social welfare of 4ℓ. So if it is possible to approximate the social welfare to a factor of $1 + \epsilon$, we will find a social welfare of at least

$$\begin{aligned} \frac{1}{1+\epsilon} 4\ell &> (1 - \delta/4)4\ell \\ &= 4\ell - \delta\ell \\ &= 3\ell + (1 - \delta)\ell. \end{aligned}$$

As mentioned above, a social welfare of $3\ell + x$ corresponds to an assignment satisfying x clauses, so this demonstrates the existence of an assignment satisfying more than a $1 - \delta$ fraction of the clauses, allowing us to distinguish between the case that all clauses are satisfiable and the case that at most a $1 - \delta$ fraction are satisfied. As this is NP-hard, we have shown that approximating the social welfare for these unit-demand players to within $1 - \epsilon$ is NP-hard as well.

Finally, we show how these $8\ell/5$ unit-demand players can be combined into 10 multi-unit-demand players without changing the social welfare. We do so by combining groups of unit-demand players that don't value any of the same items into a single multi-unit-demand player. Then the multi-unit-demand value to that player of any set is the sum of the values of each of the individual unit-demand players, as no two unit-demand valuations will count the same item twice for any player. Note that we can assume without loss of generality that there is no i for which the clause players never value i (or similarly, $\overline{i}$), as we could simply remove all players valuing i or $\overline{i}$, then perform a $1 - \epsilon$ approximation to find the best $k - 1$ items for the remaining players, and then add either item i or $\overline{i}$ to the resulting allocation to get an improved approximation. Thus, each item i or $\overline{i}$ is only valued by at most 4 clause players.

Start with 10 multi-unit-demand players with value 0 for all sets of items. We add the unit-demand valuations to their valuations greedily, beginning with the valuations of the clause players. For each unit-demand clause player, simply add its valuation to any multi-unit-demand player which does not yet value any of the 3 items it values. Since each item i or $\bar{i}$ is valued by at most 3 other clause players and there are three items valued by any clause player, there are at most $3 \cdot 3 = 9$ multi-unit-demand clause players with values for these 3 items. Thus, one of the 10 multi-unit-demand players can accommodate the value of this unit-demand player.

Now, we add the second class of players, each of which values items i and $\bar{i}$ at 1 for some i. For each i, there are 5 clause players which value either i or $\bar{i}$. So the player from the second class corresponding to i can be added to the valuations of one of the 5 multi-unit-demand players that do not yet value items i or $\bar{i}$ from the clause valuations.

Thus, we can compress these valuations into 10 multi-unit-demand players while preserving the social welfare of every assignment. As the social welfare is preserved, it remains NP-hard to approximate the social welfare to within a factor of $1 + \epsilon$. □

We now show that PMU$_2$ can be optimally solved in a computationally efficient manner via minimum cost flow.

Theorem 3.13. *There exists a polynomial-time truthful mechanism which solves PMU$_2$ exactly.*

Proof. As we are solving the problem exactly, a truthful mechanism is implied by the VCG payments. So we demonstrate the theorem with a polynomial time algorithm.

Our algorithm uses minimum cost flow, for which an optimal integral solution (one in which flow along each edge is integral) can be found in polynomial time [37]. Minimum cost flow is similar to network flow, except that each edge has a cost and the goal is to find a flow with f units of flow and minimum total edge cost. We now formally describe the algorithm.

Input: an instance of PMU$_2$ where each player i has a multi-unit-demand valuation v_i such that each v_i is composed of w_i unit-demand valuations $v_i^{(1)}, \ldots, v_i^{(w_i)}$ (see Definition 1.19).

1. **Step I: Add "dummy" unit-demand valuations** (that equal 0 for all subsets of resources) if necessary to ensure that $w_1 = w_2 = w \geq k$. This simplifies our description and analysis.

2. **Step II: Create a minimum-cost flow network (see Fig. 3.1)**. In addition to the source and target nodes s and t, the network contains node $p_{i,r}$ corresponding to player i's rth unit-demand valuation $v_i^{(r)}$, and two nodes $q_{1,j}$ and $q_{2,j}$ for each resource $j \in [m]$. The edge set contains an edge from s to each node $p_{1,j}$, and an edge from each node $p_{2,j}$ to t. In addition, for each $j \in [m]$, create an edge from $q_{1,j}$ to $q_{2,j}$. Set the cost of each of these edges to be 1.

   Let $v_{\max}$ be a positive real value that is strictly higher than both players' values for any single resource (say, $v_{\max} = \max_{i \in [2], j \in [m]} v_i(\{j\}) + 1$). Create, for each $j \in [m], r \in [w]$, an edge

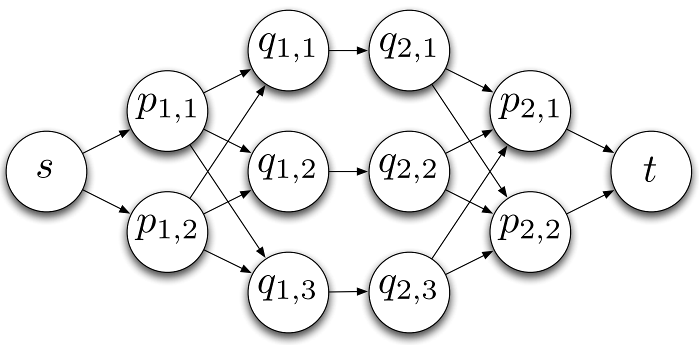

Figure 3.1: example of minimum-cost flow network construction for $w = 2$ and $m = 3$.

from $p_{1,r}$ to $q_{1,j}$ of cost $v_{\max} - v_1^{(r)}(\{j\})$ and an edge from $q_{2,j}$ to $p_{2,r}$ of cost $v_{\max} - v_2^{(r)}(\{j\})$. Observe that all costs are positive.

Set the capacities of all edges to be 1.

3. **Step III: Compute a minimum-cost flow** f with flow value k and integer flow along each edge (i.e., the flow along each edge is in $\{0, 1\}$).

4. **Step IV: Set S to be the subset of $[m]$ such that $j \in S$ iff the flow in f along the edge from $q_{1,j}$ to $q_{2,j}$ is positive**. Observe that the k units of flow in f emanating from s must traverse exactly k edges of the form $(q_{1,j}, q_{2,j})$, and hence $|S| = k$.

5. **Step V: Output the allocation S.**

Clearly, the mechanism is computationally efficient (recall that the computation of minimum-cost flow with integer values is tractable [37]). So we need only show that the algorithm maximizes the social welfare.

Observe that the k units of flow in f emanating from s, and the k units of flow going into t use edges that have a total cost of $2k$, and that the k units of flow along the edges from $q_{1,j}$ nodes to $q_{2,j}$ nodes traverse edges that have a total cost of k. Hence, the total cost of these edges is $3k$ regardless of how the flow f is achieved.

Consider $j \in S$ (computed in Step IV of the mechanism). Observe that because there is 1 unit of flow traversing the edge $(q_{1,j}, q_{2,j})$, there must be exactly one incoming edge leading to node $q_{1,j}$, and exactly one outgoing edge leaving node $q_{1,j}$, on which the flow in f is 1. Consider an edge $(q_{1,j}, q_{2,j})$ and let $(p_{1,r}, q_{1,j})$ and $(q_{2,j}, p_{2,r'})$ be the other edges incident on $q_{1,j}$ and $q_{2,j}$ through which the flow in f equals 1. Observe that the total cost of these two edges is $2v_{\max} - v_1^{(r)}(\{j\}) - v_2^{(r')}(\{j\})$. We define $c : S \to \mathbb{Z}^+$ to be the function that maps each $j \in S$ to the total cost of the incoming and outgoing edges to $q_{1,j}$ and $q_{2,j}$ (not including the edge between them).

Now, for some pair of partitions of S $P_1 = (P_1^1, \ldots, P_1^w)$ and $P_2 = (P_2^1, \ldots, P_2^w)$,

$$\begin{aligned}\sum_{j\in S} c(j) &= 2kv_{\max} - \sum_{r=1}^{w} v_1^{(r)}(P_1^r) - \sum_{r=1}^{w} v_2^{(r)}(P_2^r) \\ &\geq 2kv_{\max} - \max_{P=(P^1,\ldots P^w)} \sum_{r=1}^{w} v_1^{(r)}(P^r) - \max_{P=(P^1,\ldots,P^w)} \sum_{r=1}^{w} v_2^{(r)}(P^r) \\ &= 2kv_{\max} - v_1(S) - v_2(S),\end{aligned}$$

where the maxima in the above equations are taken over w-partitions of S.

Therefore, the total cost of flow f (including the edges leaving s, the edges entering t and the edges leading from the $q_{1,j}$'s to the $q_{2,j}$'s) is at least $2kv_{\max} + 3k$ minus the social welfare of the set S. Choosing the set maximizing the social welfare and the incoming and outgoing flows that correspond to the unit-demand valuations that maximize each v_i guarantees a total cost of exactly $2kv_{\max} + 3k$ minus the maximum social welfare. Hence, the computation of the k-flow of minimum cost determines the value of the social-welfare maximizing outcome, and the set S produced achieves this maximum. $\square$

Through use of VCG payments, the above algorithm becomes a truthful mechanism for PMU$_2$. We saw in Corollary 2.4 that the above algorithm can also be used to produce a universally truthful $n/2$ approximation for PMU. As PMU$_n$ is a special case of PMU, this mechanism also provides an $n/2$ approximation for PMU$_n$. For $n = 3$, this is a fairly good approximation ratio.

Corollary 3.14. *There is a polynomial-time universally truthful mechanism for PMU$_3$ which has an approximation ratio of at most* $3/2$ *in expectation.*

3.3 Capped-Additive Valuations

Intuitively, a capped-additive valuation is a valuation function that is additive (the value for each bundle of resources is the additive sum of the per-resource values) but value stops being added after the value cap is reached (see Definition 1.20).

2-{0,1}-unit-demand valuations are a subclass of capped-additive valuations (where the value cap is $c = 1$), and so our negative results in Sec. 3.1 for P2U extend to PC. So we focus on PC$_n$ for constant n. Observe that finding the optimal solution for PC$_1$ is in P, as we need only find the k most valuable items. We show that PC$_2$ is NP-hard.

Theorem 3.15. *PC$_2$ is NP-hard.*

Proof. We reduce from Subset Sum, where we are given a set of positive integers $w_1, \ldots, w_\ell$ and a target t, and the goal is to find a subset of $w_1, \ldots, w_\ell$ that sums to t. Given an instance of Subset Sum, we construct an instance to our problem with $m = 2\ell$ items, $k = \ell$, and 2 players with valuations $v_1(S) = \min(\sum_{j\in S} 2v_i^j, 2t)$ and $v_2(S) = \min(\sum_{j\in S} v_2^j, C)$, where $C = k \cdot \max_j w_j$ and v_i^j are defined by

$$v_1^j = \begin{cases} 2w_j, & j \le \ell \\ 0, & \text{otherwise} \end{cases}$$

$$v_2^j = \begin{cases} C/k - w_i, & j \le \ell \\ C/k, & \text{otherwise} \end{cases} .$$

Observe that if there exists a subset S s.t. $\sum_{i\in S} a_i = t$, by choosing the set of resources $S' = S \cup \{\ell+1, \dots, 2\ell - |S|\}$ we have $v_1(S') + v_2(S') = C + t$.

Conversely, consider an allocation S of size k with social welfare of at least $C + t$. Consider the subset of items in S with index at most ℓ. If the corresponding values w_i sum to more than t, then player 2 would have total value less than $C - t$, while player 1 would have value of only $2t$, for a total value of less than $C + t$. If the corresponding values sum to less than t, then the social welfare would be C plus the sum of the values w_i, which is less than $C + t$. So there must be a subset of $w_1, \dots, w_\ell$ which sums to t. $\square$

However, using dynamic programming we obtain an FPTAS for any constant number of players.

Theorem 3.16. *There exists an FPTAS for PC_n for any constant n.*

Proof. We will use a dynamic programming procedure. First, let c be the maximum effective value cap. That is,

$$c = \max_i \max_{S\subseteq[m],|S|=k} v_i(S).$$

For any i and any set S of size at most k, $v_i(S) \le c$. c can be found by taking S to be the k items of highest value for player i, then computing $v_i(S)$ and comparing these for all i. Note that the optimal social welfare is at least c.

We divide the interval $[0, c]$ into nm/ϵ segments, each of length $\epsilon c/(nm)$, and denote $p(x) = \lfloor x \cdot mn/(\epsilon c) \rfloor$. We will maintain an n-dimensional table A with $(nm/\epsilon)^n$ entries (this is polynomial, as n is constant), where in each entry $A_{i_1,\dots,i_n}$ we will store a subset $S \subseteq [m]$ of minimum cardinality for which $p(v_1(S)) = i_1, \dots, p(v_n(S)) = i_n$, if such a subset exists. For every subset $S \subseteq [m]$ contained in A, $A(S)$ shall denote the cell in A containing S, namely cell $A_{p(v_1(S)),\dots,p(v_n(S))}$.

We fill the table using the following procedure. We initialize the table with the empty set in all entries. At stage j, for each subset S contained in A at the start of stage j such that $|S| < k$, let $T = A(S \cup \{j\})$. If $|S \cup \{j\}| \le |T|$ or $T = \emptyset$, we set $A(S \cup \{j\}) = S \cup \{j\}$. After the mth stage, we are done filling the table in, so we iterate over all entries in the table and choose the subset of size k with highest social welfare. The procedure runs in $O((m+1) \cdot (mn/\epsilon)^n)$ steps, which is polynomial in m and $1/\epsilon$ as required.

Let O denote the optimal solution and let $O_j = \{i \in O : i \leq j\}$. By induction on the stage of the algorithm, we can show that after stage ℓ there is a subset S_ℓ in A such that $|S_\ell| \leq |O_\ell|$ and for every player i we have that $v_i(O_\ell) - v_i(S_\ell) \leq \ell\epsilon c/(mn)$. For $\ell = 0$ the claim is trivial, as both sides of the equation are 0.

For a $\ell \leq m$, first consider the case that $\ell \notin O_\ell$, if $S_{\ell-1}$ is still in A, then $S_{\ell-1}$ still satisfies the inductive hypothesis. Otherwise, $A(S_{\ell-1})$ contains some new set S_ℓ, such that $|S_\ell| < |S_{\ell-1}|$. Since $\ell \notin O_\ell$ $O_\ell = O_{\ell-1}$, so $|O_\ell| = |O_{\ell-1}| \geq |S_{\ell-1}| > |S_\ell|$. Furthermore, because each cell only contains a range of values of length $\epsilon c/(nm)$ for each player, each player's value can only have decreased by at most $\epsilon c/(nm)$. So

$$\begin{aligned}
v_i(O_\ell) - v_i(S_\ell) &= v_i(O_{\ell-1}) - v_i(S_\ell) \\
&\leq v_i(O_{\ell-1}) - (v_i(S_{\ell-1}) - \epsilon c/(mn)) \\
&= (v_i(O_{\ell-1}) - v_i(S_{\ell-1})) + \epsilon c/(mn) \\
&\leq (\ell-1)\epsilon c/(mn) + \epsilon c/(mn) \\
&= \ell\epsilon c/(mn).
\end{aligned}$$

Otherwise, $\ell \in O_\ell$. Consider $S_{\ell-1} \cup \ell$. Let S_ℓ be the allocation contained in cell $A(S_{\ell-1} \cup \{\ell\}|$. Note that since $|S_{\ell-1}| \leq |O_{\ell-1}| < |O_\ell| \leq k$, $S_{\ell-1} \cup \{\ell\}$ is a valid candidate for cell $A(S_{\ell-1} \cup \ell)$. So $|S_\ell| \leq |S_{\ell-1}|+1 \leq |O_{\ell-1}|+1 = |O_\ell|$. Now consider $v_i(O_\ell)-v_i(S_\ell)$. Note that since S_ℓ is in table cell $A(S_{\ell-1}\cup\{\ell\})$, $v_i(S_\ell) \geq v_i(S_{\ell-1}\cup\{\ell\})-\epsilon c/(mn)$. If $v_i(S_{\ell-1}\cup\{\ell\}) = c_i$, then $v_i(S_\ell) \geq c_i-\epsilon c/(mn)$, so $v_i(O_\ell)-v_i(S_\ell) \leq c_i-(c_i-\epsilon c/(mn)) = \epsilon c/(mn)$, which is at most $\ell\epsilon c/(mn)$ for $\ell \geq 1$. So the inductive hypothesis holds. Otherwise, $v_i(S_{\ell-1} \cup \{\ell\}) = v_i(S_{\ell-1}) + v_i(\{\ell\})$ and $v_i(O_\ell) \leq v_i(O_{\ell-1}) + v_i(\{\ell\})$. So

$$\begin{aligned}
v_i(O_\ell) - v_i(S_\ell) &\leq v_i(O_\ell) - (v_i(S_{\ell-1} \cup \{\ell\}) - \epsilon c/(mn)) \\
&\leq v_i(O_{\ell-1}) + v_i(\{\ell\}) - (v_i(S_{\ell-1}) + v_i(\{\ell\}) - \epsilon c/(mn)) \\
&= v_i(O_{\ell-1}) - v_i(S_{\ell-1}) + \epsilon c/(mn) \\
&\leq (\ell-1)\epsilon c/(mn) + \epsilon c/(mn) \\
&= \ell\epsilon c/(mn).
\end{aligned}$$

So we have proven that for all ℓ, $v_i(O_\ell) - v_i(S_\ell) \leq \ell\epsilon c/(mn)$ for some S_ℓ contained in table A. In particular, $v_i(O_m) - v_i(S_m) \leq \epsilon c/n$. Summing over i, we find that $\sum_i v_i(O_m) - \sum_i v_i(S_m) \leq \epsilon c$. As $\sum_i v_i(O_m) \geq c$, $\epsilon c \leq \epsilon \sum_i v_i(O_m)$, so $\sum_i v_i(O_m) - \sum_i v_i(S_m) \leq \epsilon \sum_i v_i(O_m)$, or $\sum_i v_i(S_m) \geq (1-\epsilon)\sum_i v_i(O_m)$. This completes the proof, as $\sum_i v_i(O_m)$ is the optimal social welfare and the social welfare of the allocation returned by our algorithm is at least $\sum_i v_i(S_\ell)$. $\square$

We now demonstrate that although there exists an FPTAS for PC_n, no maximal-in-range algorithm can approximate even PC_2 to any constant factor.

Theorem 3.17. *No computationally efficient maximal-in-range algorithm can approximate PC_2 to within $m^{\frac{1}{2}-\epsilon}$ for any constant $\epsilon > 0$ unless $NP \subset P/poly$.*

Proof. As in the proofs in previous sections, we begin by showing that for any T, we can construct an instance for any T such that the social welfare of a set S is $|S \cap T|$. We simply set $v_1^j = 1$ for $j \in T$ and $v_1^j = 0$ for $j \notin T$, $c_1 = |T|$, and $v_2(S) = 0$ for all S. Now, by Lemma 3.6, we know that for some constant α, the range has VC dimension at least m^α. Re-order the items such that $[m^\alpha]$ realizes the VC dimension. We now show modify our previous proof such that an exact solution can be found by examining all subsets of $[m^\alpha]$.

Consider the NP-hardness reduction used in Theorem 3.15. The number of resources which are valued by player 1 at 0 and player 2 at C/k doesn't matter as long as it's larger than ℓ and at least k, because this is equivalent to adding 0 values to a subset sum instance. So we can embed this reduction from a set of size m^α into the values of items $[m^\alpha]$, and give items $m^\alpha + 1, \ldots, m$ value 0 for player 1 and C/k for player 2. The reduction remains valid, as we have essentially added a polynomial number of zeroes to the subset sum instance.

All possible allocations are simply some subset of $[m^\alpha]$, together with enough items larger than m^α to reach k. As particular choice of items larger than m^α does not matter, the algorithm contains all relevant allocations in its range. Thus, using the same reduction with this modification, we see that a maximal-in-range algorithm for PC_2 with an approximation ratio of $m^{1/2-\epsilon}$ can be used to solve subset sum instances of size m^α, and therefore does not run in polynomial time unless $NP \subset P/poly$. □

3.4 Coverage Valuations

Intuitively, in a coverage valuation each item corresponds to a set of elements in some universe U, and the value of each set of items $S \subseteq [m]$ equals the number of elements of U covered by the union of the corresponding sets. For a formal definition, see Definition 1.21. We also consider scaled and weighted versions of this problem (see Definitions 1.22 and 1.23).

[36] shows that no computationally-efficient and truthful mechanism for combinatorial public projects with 2 players with submodular valuation functions can obtain an approximation ratio better than $\sqrt{m}$. One might suspect that the hardness of truthful computation in [36] stems from the conflict of interests between the two players. When there is only one player, the interests of the mechanism designer and the single player are aligned because the social welfare is the player's value. Surprisingly, our next result demonstrates that algorithmic mechanism design can be non-trivial

even in single-player environments. This seemingly paradoxical result raises intriguing questions. The questions raised by Theorem 3.18 and Corollary 3.19 were the inspiration behind the work in Chapter 5, so these results are particularly important.

Theorem 3.18. *No computationally efficient maximal-in-range mechanism for $PCOV_1$ achieves an approximation ratio of $m^{1/2-\epsilon}$ for any constant $\epsilon > 0$ unless NP $\subseteq$ P/poly.*

Proof. We begin by showing that for any T, we can construct an instance with social welfare $|T \cap S|$ for each set S. This is rather simple, as we need only set $V_1^j = \{j\}$ for $j \in T$ and $V_1^j = \emptyset$ for $j \notin T$. So $v_i(S) = |\cup_{j \in S} V_1^j| = |\{j \in S : S \in T\}| = |S \cap T|$. Thus, by Lemma 3.6, any maximal-in-range algorithm for PCOV_1 which achieves an approximation ratio of $m^{1/2-\epsilon}$ must have a VC dimension of at least m^α for some constant α.

We now show a reduction from the NP-hard t-COVER [21] with m^α sets. In t-COVER, the input is m^α subsets $T_1, \ldots, T_{m^\alpha}$ of a universe $E = \{E_1, \ldots, E_\ell\}$, and an integer t, and the objective is to determine whether there are t sets that cover E. We now construct the valuation function of player 1. We create a universe $U = [2\ell + m]$. To define the coverage valuation v_1 we need to define the sets $S_1, \ldots, S_m \subseteq U$ (see Definition 1.21). Re-order the resources such that the m^α resources corresponding to this VC-dimension are the set $[m^\alpha]$. For each $j \in m^\alpha$, let

$$S_j = \{i : E_i \in T_j\} \cup \{i + \ell : E_i \in T_j\}$$

so S_j contains two items corresponding to each element of T_j. For each $j \in \{m^\alpha + 1, \ldots, m\}$ let $S_j = \{2\ell + j\}$.

Consider the social welfare of an allocation. If the allocation contains r items from $[m^\alpha]$ and $k - r$ items from $m^\alpha + 1, \ldots, m$, then the social welfare is $k - r$ plus twice the size of the number of items covered by sets T_j such that j is in the allocation. Furthermore, if not all items from E are covered, the welfare can be improved by at least 1 by removing an item from $m^\alpha + 1, \ldots, m$ and replacing it with an item from $[m^\alpha]$ which covers some missing item, as it will cover both j and $\ell + j$. So any set maximizing the social welfare corresponds to a cover of E.

If the minimal number of sets needed to cover E in t-COVER is r, then the optimal social welfare in our PCOV_1 instance is $2|E| + k - r$. As the optimal outcome only depends on finding the right subset of items from $[m^\alpha]$ to choose and $[m^\alpha]$ exhibits the VC dimension of the range of the maximal-in-range algorithm, the algorithm will exactly maximize the social welfare. So subtracting the social welfare from $2|E| + k$, we get r, the size of the smallest possible cover of E. Thus, such an algorithm cannot run in polynomial time unless NP $\subseteq$ P/poly. □

Corollary 3.19. *No truthful polynomial-time algorithm for $PSCOV_1$ can achieve an approximation ratio of $m^{1/2-\epsilon}$ for any constant $\epsilon > 0$ unless NP $\subseteq$ P/poly.*

Proof. We begin by presenting a simple characterization of truthful mechanisms for combinatorial public projects with a single player. Our characterization shows that every truthful mechanism is an affine maximizer (see Definition 1.14).

Lemma 3.20. *Any truthful mechanism for a combinatorial public project with a single player must be an affine maximizer.*

Proof. Let R be the range of a truthful mechanism for a single-player public project. In truthful mechanisms, the payment of a player is independent of his own valuation function, and can only depend on the outcome and on the valuations of the other players. Otherwise, the player would be incentivized to lie in some situations to receive the same set with a smaller payment. As there is only one player, we can associate each outcome $r \in R$ with the payment p_r that player 1 must pay if r is allocated. As the mechanism is truthful, for any valuation function v, the mechanism must output some allocation $S \in R$ which maximizes $v(S) - p_S$ over R. This matches the definition of an affine maximizer (with $C_S = -p_S$), completing the proof. □

Now, simply note that if a mechanism for PSCOV$_1$ is an affine maximizer, it returns a set maximizing $v_1(S) + C_S$. Let $C_{\max} = \max_S |C_S| + 1$ If we take an instance of PCOV$_1$ and set the scale factor to $\alpha = 2m{\cdot}C_{\max}$, the affine maximizer returns a set S maximizing $(2m{\cdot}C_{\max})|\bigcup_{j\in S} V_1^j| + \sum_j C_j$, but $2m{\cdot}C_{\max} > |\sum_{j\in S} p_j - \sum_{j\in S'} p_j|$ for any $S, S' \subseteq 2^{[m]}$, so a set S' will be chosen over S by the affine maximizer if it contains even one more element. Thus, the affine maximizer is a maximal-in-range algorithm for the PCOV$_1$ instance, and Corollary 3.19 follows from Theorem 3.18. □

3.5 Fractionally-Subadditive Valuations

Intuitively, a valuation is fractionally-subadditive (termed "XOS" in [30, 33]) if it is the maximum of a collection of additive valuations (see Definition 1.24).

Although multi-unit-demand players are a special case of fractionally subadditive players in general, using our representation for fractionally subadditive valuation functions to represent multi-unit-demand valuations leads to an exponential increase in description size. So while 3 multi-unit demand players result in an NP-hard problem, combinatorial public projects and combinatorial auctions with any constant number of fractionally-subadditive players can be solved in polynomial time.

Theorem 3.21. *PFS$_n$ and AFS$_n$ can be solved in polynomial time for any constant n.*

Proof. For each player i, $v_i(S) = \max_j v_i^{(j)}(S)$, where the $v_i^{(j)}$ are additive functions. So for each S, there is some collection of values $j_1, \ldots, j_n$ such that $v_i(S) = v_i^{j_i}(S)$. Let S^* be the optimal allocation and let $j_1^*, \ldots, j_n^*$ be such that $v_i(S^*) = v_i^{(j_i^*)}(S^*)$. Because S^* maximizes the social welfare, S^* also maximizes the social welfare of the additive public project where player i has valuation $v_i^{(j_i^*)}$. This public project can be solved in $O(mn) = O(m)$ time by Theorem 1.2.

So in order to find S^*, we need only find $j_1^*, \ldots, j_n^*$. Fortunately, there are only linearly many additive functions $j_i^{(\ell)}$ for each i, as the number of functions $j_i^{(\ell)}$ cannot exceed the size of the input in our representation. Let L be the maximum number of additive functions in any player's valuation. The number of possible choices for $j_1, \ldots, j_n$ is $O(L^n)$, which is polynomially large for constant n. So we can try all possible values of $j_1, \ldots, j_n$ and solve each public project in $O(m)$ time, for a total polynomial runtime of $O(mL^n)$. As one of the choices will be equal to $j_1^*, \ldots, j_n^*$, one of these iterations will find S^*. So by taking the set which maximizes social welfare, we can find S^* in polynomial time, which solves PFS$_n$.

We can use the same trick to compute optimal allocations for combinatorial auctions. Let $S_1, \ldots, S_n$ be an optimal allocation for an instance of AFS$_n$. Then for some $j_1^*, \ldots, j_n^*$, $S_1, \ldots, S_n$ is also an optimal solution for the auction with valuation functions $v_1^{(j_1^*)}, \ldots, v_n^{(j_n^*)}$. Theorem 1.2 shows that we can solve this auction, so we again need only enumerate over all possible values for $j_1^*, \ldots, j_n^*$ and solve each auction in order to find an optimal solution. By the same reasoning as above, there are only $O(L^n)$ possible choices of $j_1^*, \ldots, j_n^*$, for a total runtime of $O(mL^n)$. $\square$

Much like with unit-demand, fractionally-subadditive functions become hard to approximate truthfully when the number of players becomes unlimited. In fact, this is a corollary of Theorem 3.8 because unit-demand is a special case of fractionally-subadditive functions in which each additive function gives nonzero value to at most 1 item. This maintains succinct representation, as at most m additive functions are required. Thus, we get Corollary 3.22.

Corollary 3.22. *No polynomial-time maximal-in-range mechanism can approximate PFS within $m^{\frac{1}{2}-\epsilon}$ for any constant $\epsilon > 0$ unless $NP \subseteq P/poly$.*

We now show a result which does not depend on the assumption of truthfulness using a reduction from LABEL-COVER$_{\max}$ to PFS which preserves an approximation gap. First, we define LABEL-COVER$_{\max}$ and discuss the complexity of its approximation. A LABEL-COVER$_{\max}$ instance consists of a regular bipartite graph $G = (V_1, V_2, E)$, a set of n labels $[n]$, and for each edge $e \in E$ a partial function $\Pi_e : [n] \to [n]$. Consider applying a label from $[n]$ to each of the vertices, such that a vertex v has label l_v. We say that the edge $e = \{x, y\}$ for $x \in V_1, y \in V_2$ is satisfied if $\Pi_e(l_x) = l_y$. The goal of LABEL-COVER$_{\max}$ is to find an assignment of labels to the nodes in V_1 and V_2 such that each node has exactly one label and as many edges as possible are satisfied. It was shown in [1] that LABEL-COVER$_{\max}$ is quasi-NP-hard to approximate.

Theorem 3.23 ([1])**.** *For any sufficiently small constant $\gamma > 0$, it is quasi-NP-hard to distinguish between the following two cases in* LABEL-COVER$_{\max}$: *(1) YES case: all edges are covered, and*

(2) NO case: at most a $2^{-\log^{1-\gamma} n}$ *fraction of the edges are covered, where* n *is the size of the* LABEL-COVER$_{\max}$ *instance.*

We make use of Theorem 3.23 to show a similar hardness result for PFS.

Theorem 3.24. *For any sufficiently small constant* $\gamma > 0$*, it is quasi-NP-hard to obtain an approximation ratio of* $2^{\log^{1-\gamma}(\min(n,m))}$ *for PFS.*

Proof. We prove this using a gap-preserving reduction from LABEL-COVER$_{\max}$: We are given as input an instance of LABEL-COVER$_{\max}$ consisting of a graph $G = (V_1, V_2, E)$, a set of labels $[n]$, and a set of partial functions Π_e for each $e \in E$. We create a PFS instance with $|V_1|$ players, one corresponding to each node in V_1. For convenience, we assume $V_1 = [|V_1|]$, so we can refer to the node corresponding to player i as node i. There are $|V_2| \cdot n$ items. We associate each item with some pair $(w, \ell) \in V_2 \times [n]$. Let $j_{(w,\ell)}$ refer to the item associated with (w, ℓ). We now define the fractionally-subadditive valuation v_i of each player i. For every label $l \in [n]$, we define the additive valuation function $v_i^{(l)}$ by

$$v_i^{(l)}(\{j_{(w,\ell)}\}) = \begin{cases} 1, & \{i, w\} \in E \text{ and } \Pi_{\{i,w\}}(l) = \ell \\ 0, & \text{otherwise} \end{cases}.$$

So $v_i^{(l)}(S)$ is equal to the number of edges incident on i which can covered if we choose label l for vertex i and choose label ℓ for vertex w for each $j_{(w,\ell)} \in S$. Note that if both $j_{w,\ell}$ and $j_{w,\ell'}$ are in S, this will count $\{i, w\}$ as covered if $\Pi_{\{i,w\}}(l)$ is equal to either ℓ or ℓ'.

The fractionally-subadditive valuation of player i is defined by

$$v_i(S) = \max_{l \in N} \{a_{i,l}(S)\}.$$

So player i gets the value for the best possible choice of a single label for vertex i given the label choices for V_2 implied by S. We set the number of items chosen to be $k = |V_2|$.

If the LABEL-COVER$_{\max}$ instance is a YES case, we can find a set of resources with social welfare $|E|$. Simply take any labeling that covers every edge and for every $w \in V_2$, choose the resource $j_{(w,\ell)}$, where $w \in V_2$ is labeled by ℓ in the labeling. Call this set S. $v_i(S)$ equals the degree of node i, so the social welfare given these resources is $|E|$.

We now show that if the LABEL-COVER$_{\max}$ instance is a NO case, then the maximum social welfare is bounded by $2^{-\frac{\log^{1-\gamma}\min(n,m)}{6}}|E|$ for sufficiently large n. Note that if n' is the size of the

LABEL-COVER$_{\max}$ instance, our construction guarantees $\min(n,m) \le n'$. So our bound is at least

$$\begin{aligned} 2^{-\frac{\log^{1-\gamma}(n'^2)}{6}}|E| &= 2^{-2^{1-\gamma}\frac{\log^{1-\gamma} n'}{6}} \\ &> 4 \cdot 2^{-2\frac{\log^{1-\gamma} n'}{6}} \quad \text{for sufficiently large } n' \\ &= 4 \cdot 2^{-\frac{\log^{1-\gamma} n'}{3}}. \end{aligned}$$

In order to simplify our expressions in the rest of the proof, let $\alpha = 2^{-\frac{\log^{1-\gamma} n}{6}}$. Using the above bound, we see

$$\alpha \ge 4 \cdot 2^{-\frac{\log^{1-\gamma} n'}{3}}. \tag{3.1}$$

Suppose by way of contradiction that we reduced from a NO case, but the maximum social welfare is at least $\alpha|E|$.

Let S be a set of resources with a social welfare of at least $\alpha|E|$. Recall also that each player i's fractionally-subadditive valuation v_i is defined as the maximum over a set of additive valuations $v_i^{(l)}$. Define l_i such that $v_i(S) = v_i^{(l_i)}(S)$. We will define a labeling in which each $i \in V_1$ is labeled with l_i.

For any fixed w, $v_i^{(l_i)}$ assigns a value of 1 to at most one of the resources (w,ℓ) for $\ell \in [n]$. Moreover, $v_i^{(l_i)}$ can only assign value to a resource (w,ℓ) if $\{i,j\} \in E$. We say that an edge between vertex $i \in V_1$ and vertex $w \in V_2$ is satisfied by the set S if $(w, \Pi_{\{w,j\}}(l_i)) \in S$. Observe that the total social welfare value of S equals the number of edges satisfied by S.

Let d be the number of edges incident on a vertex in V_2. As G is a regular bipartite graph, $d = \frac{|E|}{|V_2|}$. Let V_2' denote all vertices $w \in V_2$ in which the number of edges incident on w satisfied by S is at least $\alpha d/2$. A counting argument shows that $|V_2'| \ge \frac{\alpha}{2}|V_2|$: If $|V_2'|$ were less than $\frac{\alpha}{2}|V_2|$, the number of satisfied edges incident upon vertices in V_2' is at most $|V_2'|d < \frac{\alpha}{2}|E|$, and the number of satisfied edges incident upon vertices outside of V_2 would be less than $|V_2|\frac{\alpha}{2}d = \frac{\alpha}{2}|E|$. So summing these, we would see that the number of satisfied edges is less than $\alpha|E|$, a contradiction. So $|V_2'| \ge \frac{\alpha}{2}|V_2|$.

If S contains β different items $j_{(w,\ell)}$ for a fixed w, we say that w is labeled β times by S. Since there are $k = |V_2|$ items chosen, at most $\frac{\alpha}{4}|V_2|$ of the nodes $j \in V_2$ are labeled more than $\frac{4}{\alpha}$ times by S. So letting V_2'' be the subset of V_2' which is labeled at most $\frac{4}{\alpha}$ times by S,

$$|V_2''| \ge |V_2'| - \frac{\alpha}{4}|V_2| = \frac{\alpha}{2}|V_2| - \frac{\alpha}{4}|V_2| = \frac{\alpha}{4}|V_2|.$$

Since S labels each $w \in V_2''$ at most $\frac{4}{\alpha}$ times, and S satisfies at least $\frac{\alpha}{2}d$ edges incident upon each vertex in V_2'' (because $V_2'' \subseteq V_2'$), we can find a single label ℓ, $j_{(w,\ell)} \in S$ that satisfies at least $\frac{\alpha}{2}d/\frac{4}{\alpha} = \frac{\alpha^2}{8}d$ of the edges incident upon w. So if we label each $w \in V_2''$ according to S_j, and label each $i \in V_1$ by the l_i, we have a labeling that satisfies at least

$$|V_2''|\frac{\alpha^2}{8}d \geq \frac{\alpha}{4}|V_2|d \cdot \frac{\alpha^2}{8} = \frac{\alpha}{4}|E| \cdot \frac{\alpha^2}{8} = \frac{\alpha^3}{32}|E|$$

edges, regardless of how the vertices in $V_2 - V_2''$ are labeled. This contradicts that we had a NO case, as we can see by (3.1) that

$$\frac{\alpha^3}{32}|E| > \frac{\left(4 \cdot 2^{-\frac{\log^{1-\gamma} n'}{3}}\right)^3}{32} = 2^{-\log^{1-\gamma} n'}|E|.$$

Thus, we see that the maximum social welfare of our PFS instance is at least $|E|$ if we reduced from a YES case, and at most $\alpha|E|$ if we reduced from the NO case. Therefore, it is quasi-NP-hard to achieve an approximation ratio of $2^{\frac{\log^{1-\gamma}\min(n,m)}{6}}$.

Finally, we need only show how to remove the $1/6$ factor in the exponent. For sufficiently large n, m, and $\gamma' < \gamma$, $2^{\frac{\log^{1-\gamma'}(\min(n,m))}{6}} < 2^{\log^{1-\gamma}(\min(n,m))}$, and we have shown that it is quasi-NP-hard to achieve an approximation ratio of $2^{\frac{\log^{1-\gamma'}(\min(n,m))}{6}}$, which is a better ratio than $2^{\log^{1-\gamma}(\min(n,m))}$, proving the theorem. $\square$

Chapter 4

Combinatorial Auctions

In this chapter, we analyze auctions with capped-additive valuations. A capped-additive valuation v_i consists of a value v_i^j for each item and a value cap c_i. Unlike the combinatorial public projects studied in Chaper 3, allocations are not a single set, but rather a partition of the items among the players. We begin by demonstrating a maximal-in-range approximation algorithm for subadditive auctions which was originally shown in [15].

Given valuation functions v_i for each player i, form a bipartite graph $G = (V_1, V_2, E)$ with $V_1 = \{P_1, \ldots, P_n\}$, $V_2 = \{I_1, \ldots, I_m\}$ and $E = \{\{P_i, I_j\} : i \in [n], j \in [m]\}$. Define a weight function on the edges $w(\{P_i, I_j\}) = v_i(\{j\})$. Find a maximum weighted matching in G with edge weights w. Call the value of this matching $V_{matching}$.

Now, consider $v_i([m])$, the value to player i of getting all the items. Let $V_{all} = \max_i v_i([m])$, and let i^* be the player that maximizes $v_i([m])$.

If $V_{matching} \geq V_{all}$, allocate item j to player i for each edge $\{P_i, I_j\}$ in the maximum matching. Otherwise, allocate every item to player i^*.

Theorem 4.1 ([15])**.** *The above algorithm is maximal-in-range and achieves a* $\min(n, 2m^{1/2})$ *approximation of the social welfare under subadditive valuations.*

Proof. The range of this mechanism is all allocations in which each player gets at most one item, together with the allocations in which one player gets all items. $V_{matching}$ is the maximum value possible in which each player gets at most one item, while V_{all} is the maximum value in which some player gets all items. As the allocation gets the maximum of these, it is the maximum over the entire range.

For n players, the maximum social welfare is at most $n \cdot V_{all}$. So this algorithm is at most an n approximation. We now proceed to show that this algorithm is also at most a $2\sqrt{m}$ approximation.

Consider an allocation $A = S_1, \ldots, S_n$ which maximizes the social welfare. There are at most $\sqrt{m}$ players who get $\sqrt{m}$ or more of the items each. Call this set of players P_{high}, and call the others P_{low}.

V_{all} is at least as great as the maximum value received by any player in P_{high}. Thus, V_{all} is at least $1/|P_{high}| \geq 1/\sqrt{m}$ times the social welfare from players in P_{high}. So if the players in P_{high} account for at least half the social welfare, the maximum social welfare is at most $2\sqrt{m}$ times V_{all}.

Otherwise, the players in P_{low} account for at least half the social welfare. Consider the allocation in which every player in P_{low} receives the item in S_i maximizing $v_i(\{j\})$. Since the valuations are subadditive and each player in P_{low} receives at most $\sqrt{m}$ items, the total value from players in P_{low}, each player gets at least a $\sqrt{m}$ fraction of his value from the optimal allocation. Since $V_{matching}$ is the maximum value of any allocation in which each player gets at most one item, $V_{matching}$ is at least as good as this allocation. So the social welfare from P_{low} in A is at most $\sqrt{m}V_{matching}$. Thus, since P_{low} gets at least half the social welfare, the social welfare of the optimal allocation is at most $2\sqrt{m}$ times $V_{matching}$.

As V_{all} is always an n approximation and one of $V_{all}, V_{matching}$ is a $2\sqrt{m}$ approximation of the social welfare, assigning items to achieve the max of these two welfares yields a $\min(n, 2\sqrt{m})$ approximation. $\square$

4.1 Hardness for MIR Mechanisms

In this section, we prove a hardness result for maximal-in-range algorithms for AC and AC$_n$ via Theorem 4.2:

Theorem 4.2. *No polynomial-time maximal-in-range algorithm for AC with $n = n(m) \leq m^{\eta}$ for positive constant $\eta < 1/2$ can approximate the social welfare with a ratio of $n(1-\epsilon)$ for positive constant ϵ unless $NP \subseteq P/poly$.*

4.1.1 Our Proof

The proof of Theorem 4.2 is more difficult than the proofs of maximal-in-range approximation hardness in Chapter 3. This is because allocations in a combinatorial public project are simply subsets of the items, which allow for easy use of the Sauer-Shelah lemma through Corollary 3.2. We turn the partitions in the range into something more like subsets in order to apply the Sauer-Shelah lemma. First, Lemma 4.5 shows how we can restrict to a subset of the items for which a large subset of the range allocates them all. In the 2-player case, this means that each item is either allocated to player 1 or player 2, so the allocations can be represented as the subset of items allocated to player 1, and we can apply the Sauer-Shelah lemma. Lemma 4.6 shows how if there are more than 2 players, all but one can be grouped together so we can again represent the range by just the set of items allocated to one player.

Finally, Lemma 4.7 shows how the large VC dimension shown by applying the Sauer-Shelah lemma allows an NP-hard problem to be solved by any maximal-in-range mechanism with approximation ratio of $n(1-\epsilon)$, completing the proof of Theorem 4.2.

Theorem 4.2 gives us the following corollary, which shows that it is not possible to find a polynomial-time maximal-in-range mechanism that achieves an approximation much better than the $\min(n, 2m^{1/2})$ in Theorem 4.1.

Corollary 4.3. *For any positive constant ϵ and any relation $n = n(m) \leq poly(m)$, no polynomial-time maximal-in-range auction mechanism can approximate the social welfare for AC with n players and m items to a ratio of $\min((1-\epsilon)n, m^{1/2-\epsilon})$ unless $NP \subseteq P/poly$.*

Proof. Suppose by way of contradiction that there exists a polynomial-time maximal-in-range algorithm A which achieves a $\min(n(1-\epsilon), m^{1/2-\epsilon})$ approximation ratio. Let $\eta = 1/2 - \epsilon/2$ and $n'(m) = \min(n(m), m^{\eta})$. We construct a polynomial-time maximal-in-range mechanism A' for $n' = n'(m) \leq m^{\eta}$ players from A by simply running A with the first n' players, and all other players given $v_i = 0$. This produces an allocation of items to n players, from which we produce an allocation to n' players by not allocating any items which are given to players larger than n'.

Now we will see that A' produces an approximation ratio of at most $n'(1-\epsilon)$. For $n \leq m^{\eta}$, $n' = n$, and A has an approximation ratio of $\min((1-\epsilon)n, m^{1/2-\epsilon}) \leq (1-\epsilon)n = (1-\epsilon)n'$. For $n \geq m^{\eta}$, A has an approximation ratio of

$$\begin{aligned}
\min((1-\epsilon)n, m^{1/2-\epsilon}) &\leq m^{1/2-\epsilon} \\
&= m^{1/2-\epsilon/2} \cdot m^{-\epsilon/2} \\
&= m^{\eta} \cdot m^{-\epsilon/2} \\
&= n' \cdot m^{-\epsilon/2} \\
&< n'(1-\epsilon) \quad \text{for } m > (1-\epsilon)^{-2/\epsilon}.
\end{aligned}$$

So A achieves a ratio of $n'(1-\epsilon)$ when $n \geq m^{\eta}$ for $m > (1-\epsilon)^{-2/\epsilon}$. For $m \leq (1-\epsilon)^{-2/\epsilon}$, m has a constant bound, so A' can find an exact solution by trying all ${n'}^{m} \in poly(n')$ possible allocations and choosing the one with maximum social welfare.

Thus, A' gets an approximation ratio of $n'(1-\epsilon)$ for a function $n' = n'(m) \leq m^{\eta}$, contradicting Theorem 4.2. □

Note that because Corollary 4.3 applies to any relation $n = n(m)$ between the number of players and players, it shows that unless NP has polynomial circuits, the best polynomial-time maximal-in-range approximation for AC is $\min(n, O(\sqrt{m}))$, as well as that the best such approximation for AC_n for constant n is n.

We begin the proof of Theorem 4.2 by examining the structure of the range.

4.1.2 The Counting Argument

Let A be a maximal-in-range algorithm with range $R \subseteq ([n] \cup \{\star\})^m$, where for $x \in R$, $x_i = j$ means that item i is given to player j, while $x_i = \star$ indicates that no player is given item i. For $S \subseteq [m]$, we define R_S to be the subset of the range where all of the items in S are assigned to players,

$$R_S = \{x \in R \ : \ x_i \neq \star \forall i \in S\}.$$

When considering R_S we wish to focus on the players that the items in S are assigned to, so we define T_S to be the projection of R_S to the indices in S. So $T_S \subseteq [n]^{|S|}$.

In order to show that A can solve an NP-hard problem, we will need find a subset of the items which are allocated in enough ways to exactly solve an NP-hard problem. Finding a large T_S so that we can ignore the issue of unassigned items is the first step toward this goal. We begin with a helpful lemma.

Lemma 4.4. *For any positive constant ϵ and any m, n for which the binomial coefficients below are positive,*

$$\frac{\binom{m}{\epsilon m/n}}{\binom{((1+2\epsilon)/n)m}{\epsilon m/n}} < \left(\frac{n}{1+\epsilon}\right)^{\epsilon m/n}.$$

Proof. First, note that

$$\begin{aligned}
\frac{\binom{m}{\epsilon m/n}}{\binom{((1+2\epsilon)/n)m}{\epsilon m/n}} &= \frac{\prod_{i=0}^{\epsilon m/n-1} \frac{m-i}{\epsilon m/n-i}}{\prod_{i=0}^{\epsilon m/n-1} \frac{((1+2\epsilon)/n)m-i}{\epsilon m/n-i}} \\
&= \prod_{i=0}^{\epsilon m/n-1} \frac{\frac{m-i}{\epsilon m/n-i}}{\frac{((1+2\epsilon)/n)m-i}{\epsilon m/n-i}} \\
&= \prod_{i=0}^{\epsilon m/n-1} \frac{m-i}{((1+2\epsilon)/n)m-i}.
\end{aligned}$$

Now, we bound the individual terms of the above product:

$$\begin{aligned}
\frac{m-i}{((1+2\epsilon)/n)m-i} &\leq \frac{m}{(1+2\epsilon)m/n-i} \\
&< \frac{m}{(1+2\epsilon)m/n-\epsilon m/n} \\
&= \frac{n}{(1+2\epsilon)-\epsilon} \\
&= \frac{n}{1+\epsilon}.
\end{aligned}$$

Multiplying the $\epsilon m/n$ terms together, we have

$$\begin{aligned}\frac{\binom{m}{\alpha m}}{\binom{((1+2\epsilon)/n)m}{\epsilon m/n}} &= \prod_{i=0}^{\epsilon m/n-1} \frac{m-i}{((1+2\epsilon)/n)m-i} \\ &< \prod_{i=0}^{\epsilon m/n-1} \frac{n}{1+\epsilon} \\ &= \left(\frac{n}{1+\epsilon}\right)^{\epsilon m/n}\end{aligned}$$

which proves the lemma. □

Using the above lemma, we can now show that there must be some large T_S.

Lemma 4.5. *Let A be a maximal-in-range algorithm for AC which approximates the social welfare with a ratio of $n/(1+2\epsilon)$, for positive constant ϵ. There exists a set $S \subseteq [m]$ with $|S| = \epsilon m/n$ where T_S has size $|T_S| \geq (1+\epsilon)^{\epsilon m/n}$.*

Proof. To begin, we associate with each $x \in [n]^m$ a set of capped-additive valuation functions $v_{x,i}(S) = \min\left(\sum_{i\in S} v^j_{x,i}, c_i\right)$ where

$$\begin{aligned} v^j_{x,i} &= \begin{cases} 1 & x_j = i \\ 0 & \text{otherwise} \end{cases} \\ c_i &= m. \end{aligned}$$

Let x be an arbitrary element of $[n]^m$. Because A approximates the social welfare with a ratio of $n/(1+2\epsilon)$ and the maximum social welfare of the instance with players whose valuation functions are $v_{x,1}, \ldots, v_{x,n}$ is m, there must be an allocation r in the range R of A such that $r_i = x_i$ for at least $((1+2\epsilon)/n)m$ different indices i. Let S_x be the set of these indices,

$$S_x = \{i \ : \ r_i = x_i\}.$$

There are at least $\binom{|S_x|}{\epsilon m/n} \geq \binom{((1+2\epsilon)/n)m}{\epsilon m/n}$ subsets $S' \subseteq S_x$ of size $\epsilon m/n$. For each such set S', $T_{S'}$ contains the projection of x to S'. If T_S contains the projection of x to S, we say that x is *covered* by T_S. If $t \in T_S$ is the projection of x to S, we say that t *covers* x. Note that each x is covered by at least $\binom{((1+2\epsilon)/n)m}{\epsilon m/n}$ sets T_S of size $|S| = \epsilon m/n$.

For a subset $S \subseteq [m]$, define $C(S)$ to be the number of vectors $x \in [n]^m$ which are covered by T_S. Since each $x \in [n]^m$ is covered by at least $\binom{((1+2\epsilon)/n)m}{\epsilon m/n}$ sets T_S with $|S| = \epsilon m/n$,

$$\sum_{S\subseteq[m],|S|=\epsilon m/n} C(S) \geq n^m \binom{((1+2\epsilon)/n)m}{\epsilon m/n}. \tag{4.1}$$

Suppose by way of contradiction that for every subset $S \subseteq [m]$ of size $\epsilon m/n$, $|T_S| < (1+\epsilon)^{\epsilon m/n}$. Using this assumption, we will find a bound on $\sum_{S \subseteq [m], |S| = \epsilon m/n} C(S)$ which contradicts (4.1). Consider a subset $S \subset [m]$ such that $|S| = \epsilon m/n$. Each $t \in T_S$ covers $n^{m - \epsilon m/n}$ elements of $[n]^m$. So $C(S) < (1+\epsilon)^{\epsilon m/n} n^{m-\epsilon m/n}$, which gives the bound

$$\sum_{S \subseteq [m], |S| = \epsilon m/n} C(S) < \binom{m}{\epsilon m/n} (1+\epsilon)^{\epsilon m/n} n^{m-\epsilon m/n}. \tag{4.2}$$

Combining bounds (4.1) and (4.2), we have

$$\binom{m}{\epsilon m/n} (1+\epsilon)^{\epsilon m/n} n^{m-\epsilon m/n} > n^m \binom{((1+2\epsilon)/n)m}{\epsilon m/n},$$

which we simplify to

$$\frac{\binom{m}{\epsilon m/n}}{\binom{((1+2\epsilon)/n)m}{\epsilon m/n}} (1+\epsilon)^{\epsilon m/n} > n^{\epsilon m/n}. \tag{4.3}$$

By Lemma 4.4, we get

$$\frac{\binom{m}{\epsilon m/n}}{\binom{((1+2\epsilon)/n)m}{\epsilon m/n}} (1+\epsilon)^{\epsilon m/n} < \left(\frac{n}{1+\epsilon}\right)^{\epsilon m/n} (1+\epsilon)^{\epsilon m/n} = n^{\epsilon m/n},$$

which contradicts (4.3), proving that there exists some $S \subseteq [m]$ with $|S| = \epsilon m/n$ such that $|T_S| \geq (1+\epsilon)^{\epsilon m/n}$. $\square$

4.1.3 Using the VC Dimension

In Chapter 3, showing a $2^{\Omega(m)}$ sized subset of the range was enough to show large VC-dimension and move on to the reduction. One more step is required with three or more players because for $n > 2$ there exist sets of size $(n-1)^m \geq 2^m$ such that no item is ever allocated to player n. So an exponentially large subset does not imply that some polynomial-sized subset of the items is allocated in every way. To get around this difficulty, we map T_S injectively from $[n]^{\epsilon m/n}$ into $2^{[\epsilon m]}$ by combining all but one of the players into a meta-player. We then apply the Sauer-Shelah lemma via Corollary 3.2 to show that there is a polynomial-sized subset of the items which are allocated in every possible way in the meta-player auction. We finish by showing how finding the exact solution in the restricted meta-player auction allows for a solution to the NP-hard subset sum problem.

Definition 4.1 (ϕ_i, Φ_i)**.** *Let $U = \{u_i^j : i \in [n], j \in [m]\}$. Let $\phi_i(r) = \{u_i^j : r_j = i\}$. So $\phi_i(r)$ is the set of u_i^j such that item j is allocated to player i in the allocation r. We define $\Phi_i(T) = \{\phi_i(t) : t \in T\}$.*

The next lemma is the main lemma in this section; it applies to the large set T_S shown to exist in Section 4.1.2.

Lemma 4.6. *Let A be a maximal-in-range algorithm for AC with n players and m items, with $n = n(m) \leq m^\eta$ for positive constant $\eta < 1/2$. For all sufficiently large m, if there exists a subset $S \subseteq [m]$ with $|S| = \epsilon m/n$ such that $|T_S| \geq (1+\epsilon)^{\epsilon m/n}$, then there exists a player i^* such that $\Phi_{i^*}(T_S)$ has VC dimension at least $\sqrt{\epsilon} \cdot m^{1/2-\eta}$.*

Proof. Let $U = \{u_i^j : i \in [n], j \in [m]\}$. We define $f : [n]^{\epsilon m/n} \to 2^U$ by $f(x) = \{u_i^j : x_j = i\}$. So u_i^j is in $f(x)$ if the allocation defined by x gives item j to player i. Note that this differs from ϕ_i in that i is not specified. So $f(x) = \bigcup_i \phi_i(x)$. Similar to our definition of Φ_i, we define $F(T) = \{f(t) : t \in T\}$.

Note that for $x \neq x'$, there must be some j such that $x_j \neq x'_j$. Thus, $f(x)$ contains $u_{x_j}^j$, and $f(x')$ does not. So f is injective, and therefore

$$|F(T_S)| = |T_S| \geq (1+\epsilon)^{\epsilon m/n}.$$

Note that $(1+\epsilon)^{\epsilon m/n} \geq (1+\epsilon)^{\epsilon m^{1-\eta}} \geq (1+\epsilon)^{\epsilon(\epsilon m)^{1-\eta}}$. We can apply Corollary 3.2 (with $\alpha = 1/2$, $\delta = 1 - \eta > \alpha$, and $\ell = \epsilon m$) to conclude that for sufficiently large m (and therefore sufficiently large $\ell = \epsilon m$), $F(T_S)$ has VC dimension at least $(\epsilon m)^{1/2}$.

Let Q be a set of size at least $(\epsilon m)^{1/2}$ which exhibits the VC dimension of $F(T_S)$. Let $U_I = \{u_i^j : j \in [m]\}$. Partition Q into sets $Q_i = Q \cap U_i$. There are n parts in the partition, so $\sum_i |Q_i| = (\epsilon m)^{1/2}$, which implies that there is some $i^* \in [n]$ for which $|Q_{i^*}| \geq (\epsilon m)^{1/2}/n$.

Because Q is shattered (see Definition 3.1) by $F(T_S)$ so is Q_{i^*}. As Q_{i^*} is shattered by $F(T_S)$, it is also shattered by $\Phi_{i^*}(T_S)$, as, if there exists a $t \in F(T_S)$ such that $t \cap Q_{i^*} = V$, there is a corresponding $t' = \{u_i^j \in t : i = i^*\} \in \Phi_{i^*}(T_S)$ such that $t' \cap Q_{i^*} = V$ (as the only items in either intersection are of the form $u_{i^*}^j$, and t and t' agree on these items). So $\phi_{i^*}(T_S)$ has VC dimension at least $|Q_{i^*}| \geq (\epsilon m)^{1/2}/n \geq \sqrt{\epsilon} \cdot m^{1/2-\eta}$. □

For a more intuitive understanding of Lemma 4.6, consider viewing all players other than i^* as a single meta-player. Lemma 4.6 states that there is a polynomially large set of items which are fully allocated in every possible way under this 2-player view.

4.1.4 Embedding Subset Sum

We now show that if $\phi_{i^*}(T_S)$ has VC dimension at least m^α for constant $\alpha > 0$, we can embed a subset sum instance into the auction in such a way that it is solved by A. We use a reduction similar to one used in [30] to show that exactly maximizing the social welfare of these auctions is NP-hard.

Lemma 4.7. *Let A be a polynomial-time maximal-in-range auction for auctions with n players and m items. Suppose there exists a constant $\alpha > 0$ such that for all sufficiently large m, there exists a player i^* and a subset $S \subseteq [m]$ such that $\phi_{i^*}(T_S)$ has VC dimension at least m^α. Then $NP \subseteq P/poly$.*

Proof. Let T' be a set which exhibits the VC dimension of $\phi_{i^*}(T_S)$. Let $L = \{j : u_{i^*}^j \in T'\}$. The subset of items in L are allocated in every possible way by A, if we consider only whether an item is given to player i^* or to some other player. For ease of exposition we re-order the items so that L is the set of the first m^α items. Let $a_1, \ldots, a_{m^\alpha}$ be a subset sum instance with target sum K. For all players $i \neq i^*$, we set

$$\begin{aligned} v_{ij} &= \begin{cases} a_j, & j \leq m^\alpha \\ 0, & j > m^\alpha \end{cases} \\ c_i &= \sum_j a_j \end{aligned}$$

and for player i^*, we set

$$\begin{aligned} v_{i^*}^j &= \begin{cases} 2a_j, & j \leq m^\alpha \\ 0, & j > m^\alpha \end{cases} \\ c_{i^*} &= 2K. \end{aligned}$$

If there is a subset V of $\{a_1, \ldots, a_{m^\alpha}\}$ with sum K, there is an allocation in A's range with social welfare of $\sum_j a_j + K$. This can be any assignment where player i^* gets the items in V, the other items from the first m^α are allocated arbitrarily to other players, and items greater than m^α are allocated or not arbitrarily. A's range must contain such an assignment because the first m^α items are allocated in every possible way in A's range. So A will output an assignment with social welfare at least $\sum_j a_j + K$ if there is a subset summing to K.

If there is no subset of $\{a_1, \ldots, a_{m^\alpha}\}$ summing to K, A will assign player i^* a subset $M \subseteq [m]$ such that $\sum_{j \in M} a_j \neq K$. If $\sum_{j \in M} a_j < K$, the social welfare is at most

$$\begin{aligned} \sum_{j \notin M} a_j + \sum_{j \in M} 2a_j &= \left(\sum_{j \notin M} a_j + \sum_{j \in M} a_j \right) + \sum_{j \in M} a_j \\ &= \sum_j a_j + \sum_{j \in M} a_j \\ &< \sum_j a_j + K. \end{aligned}$$

If $\sum_{j\in M} a_j > K$, player i^* gets value $2K$. So the total value is at most

$$\begin{aligned}
\sum_{j\notin M} a_j + 2K &= \sum_{j\notin M} a_j + \left(\sum_{j\in M} a_j - \sum_{j\in M} a_j\right) + 2K \\
&= \left(\sum_{j\notin M} a_j + \sum_{j\in M} a_j\right) - \sum_{j\in M} a_j + 2K \\
&= \sum_{j} a_j - \sum_{j\in M} a_j + 2K \\
&< \sum_{j} a_j - K + 2K \\
&= \sum_{j} a_j + K.
\end{aligned}$$

So unless there is a subset V with sum K, every assignment has social welfare less than $\sum_j a_j + K$. Taking L as advice, we can therefore solve a subset sum instance with m^α values in polynomial time. Therefore, subset sum is in P/poly, so NP $\subseteq$ P/poly. $\square$

4.1.5 Proof of the Main Result

We can now prove Theorem 4.2. Suppose that there exists a a polynomial-time maximal-in-range mechanism A which achieves an approximation ratio of $n(1-\epsilon)$ for AC with m items and $n = n(m) \leq m^\eta$ players for a positive constant $\eta < 1/2$. Let ϵ' be the positive constant such that $n/(1+2\epsilon') = n(1-\epsilon)$. By Lemma 4.5, for each m there exists a subset $S \subseteq [m]$ of size $\epsilon' m/n$ such that $|T_S| \geq (1+\epsilon')^{\epsilon' m/n}$. By Lemma 4.6, this implies that for sufficiently large m, there exists an i^* such that $\phi_{i^*}(T_S)$ has VC dimension at least $\sqrt{\epsilon'} \cdot m^{1/2-\eta}$. Since $\eta < 1/2$, we have $\sqrt{\epsilon'} \cdot m^{1/2-\eta} \geq m^\gamma$ for some fixed positive constant γ. By Lemma 4.7, we thus have that $NP \subseteq P/poly$. So a polynomial time maximal-in-range algorithm for AC with m items and $n = n(m) \leq m^\eta$ players implies that NP has polynomial circuits.

4.2 Super-Polynomially Many Players

In this section, we observe that our results can be extended to handle the case where n is not bounded by a polynomial in m, at the expense of a stronger complexity assumption. For n larger than m, our technique shows a limit of $m^{1/2-\epsilon}$ on the approximation ratio of any mechanism which runs in time polynomial in m. However, an efficient mechanism need only run in time polynomial in the input size, which is at least as large as n, in order to represent all n valuation functions. So the runtime need only be polynomial in n, which is greater than *poly*(m) if n is not bounded by a polynomial in m.

To see how larger values of n affect the necessary complexity assumptions, consider the case where n is sub-exponential in m. Applying the same reduction leads to a circuit family of size $poly(n, m)$ (or sub-exponential in m), which solves subset sum instances of size m^γ for constant $\gamma > 0$, and this implies that NP has sub-exponential size circuits. So for sub-exponential $n = n(m)$, there is no polynomial-time algorithm for AC with m items and $n = n(m)$ players unless NP has sub-exponential circuits.

If n is sufficiently large as a function of m, it can even become possible to solve AC in polynomial time unconditionally.

Theorem 4.8. *There exists a maximal-in-range mechanism $\mathcal{M}$ for auctions with n players and m items, which maximizes the social welfare and runs in polynomial time when $B_m \in O(poly(n))$, where B_m is the mth Bell number, the number of partitions of $[m]$ into any number of disjoint subsets with union $[m]$.*

Proof. If $B_m \in O(poly(n))$, it is possible to enumerate all of the partitions of $[m]$ into any number of disjoint subsets in polynomial time. For each such partition into sets $S_1, \ldots, S_k$, we need only find the maximum social welfare obtainable by giving each set S_j in the partition to some player i_j. If we find this maximum for every partition, we can take the maximum such value as the maximum social welfare.

To find the maximum welfare for a partition, create a bipartite graph $G = (V_1, V2, E)$ with $V_1 = \{S_1, \ldots, S_k\}$, $V_2 = [n]$ and $E = V_1 \times V_2$. Define a weight function $w(\{S_j, i\}) = v_i(S_j)$. A maximum weighted bipartite matching results in the maximum sum of $\sum_j v_{i_j}(S_j)$, so this is the maximum social welfare obtainable with this partition.

The amount of time required to compute the maximum welfare for each partition is polynomial in the size of the graph, which is polynomial in n and $k \leq m$. The number of partitions is polynomial in n, so the total runtime is polynomial in n and m. $\square$

Chapter 5

Reducing Asymmetry in Truthfulness

In Chapter 3, we saw the strange result of Corollary 3.19 that PSCOV_1 cannot be truthfully approximated better than $\sqrt{m}$ in polynomial time unless NP $\subseteq$ P/poly. This is extremely surprising, as truthfulness should intuitively be easy to achieve when incentives are aligned. In the case of PSCOV_1, both the player and the mechanism aim to maximize the social welfare if all payments are zero. Because incentives are aligned, one should expect that the restriction to truthfulness would not prevent use of the worst-case optimal $e/(e-1)$ polynomial-time approximation algorithm.

The hardness to approximate PSCOV_1 truthfully may seem to be just a strange quirk that can easily be ignored, as single player public projects lack players with competing interests, and therefore are not even really games. It is not a problem if techniques from algorithmic game theory fail when applied to situations which are not games. However, this issue cannot be ignored if we wish to study PSCOV_n for $n \geq 2$ or even PSCOV. These cannot be approximated efficiently and truthfully because they contain PSCOV_1 as a special case. In order to ignore the PSCOV_1 hardness result, we must somehow first eliminate the PSCOV_1 as the source of complexity for these problems.

Our counter-intuitive hardness result for PSCOV_1 is a result of asymmetry in the definition of "efficient truthful mechanism." In order to be efficient, a mechanism is limited to polynomial computation. In order to be truthful, there must not exist a lie by which a player could benefit. It does not matter whether it is possible to find this lie in polynomial time. For example, if we solve PSCOV_1 using a greedy $e/(e-1)$ approximation and implement this as a mechanism, the player must find a solution which approximates the social welfare better than $e/(e-1)$ in order to lie effectively. It is NP-hard to approximate PSCOV_1 better than $e/(e-1)$, but the existence of such a lie still demonstrates that this mechanism is not truthful.

5.1 Different Approaches

The most obvious approach to solve this problem is to redefine truthfulness in a manner that takes player computation into account. For example, a mechanism is truthful if no polynomial-time algorithm can compute a lie. This approach immediately fails, as a polynomial-time algorithm could simply have a lie built in for just one special case in which a beneficial lie is possible. If we are using the greedy algorithm for PSCOV_1, the player's algorithm need only have the optimal solution for one case that the greedy algorithm fails to solve exactly.

If truthfulness is instead defined such that a mechanism is truthful if no polynomial-time algorithm can compute a lie in the worst case, we still run into problems. Building lies for special cases into a polynomial-time algorithm no longer demonstrates that an algorithm is not truthful, but truthfulness loses much of its meaning. Now, a mechanism may be hard to construct a lie for in the worst case, but easy to construct a lie for in the average case. We do not wish to classify such a mechanism as truthful.

Based on these observations, one might consider average case complexity as the proper tool with which to redefine truthfulness. This approach can be complicated by players who have algorithms specifically tailored to their particular valuation function. It also depends heavily on the distribution assumed. For these reasons, average case complexity is not the approach we take.

5.1.1 Communication Complexity

The hardness for truthful mechanisms for PSCOV_1 goes away if the mechanism is allowed to ask the player which allocation it would prefer. Noting this, one might suggest that a communication complexity approach would be the proper avenue for a solution to this conundrum. If we simply look at number of bits of communication necessary to find a solution in the multi-party communication complexity model, then Corollary 2.13 shows that only a polynomial number of bits are necessary in order to communicate the succinct representation of any PSCOV_n or PSCOV instance. So under a communication complexity model, we cannot hope to show hardness for PSCOV_n and PSCOV. Corollary 2.13 also shows that the same is true even if communication is limited to value queries.

5.1.2 The Nisan-Ronen Approach

In [34], a "Second Chance" mechanism was suggested to address the asymmetry between truthfulness and efficiency. The mechanism can use any allocation algorithm A. The players submit their valuation functions to the mechanism, and A is used to compute an allocation. Next, each player i submits a function f_i which maps the n valuation functions submitted by the players to n new valuation functions for which the player believes A will compute an allocation with better social welfare.

Each f_i is applied to the submitted valuations to arrive at a new collection of valuations. A is then applied to each of these to arrive at n additional candidate allocations. Of the $n+1$ allocations computed by the mechanism, the one with maximum social welfare is chosen. VCG-style payments are used to ensure that each player is incentivized to maximize the social welfare. So any player who wishes to lie should prefer to put that lie into the function f_i.

Unfortunately, the need to compute each player's function f_i requires the mechanism to limit the computation time of f_i to some polynomial n^c. So a player capable of more computation than this n^c limit would still benefit more by lying than by submitting a good f_i, so this approach still suffers from a similar asymmetry problem.

5.1.3 Our Approach

We propose a new general class of mechanisms which take advantage of player computation. Instead of trying to get a handle on player computation and limiting it to the resources available to the mechanism, our mechanism makes use of the same resources that players are using to lie. The mechanism supplements its own computation by making queries to the players. For example, in order to lie a player may need to be able to compute which allocation they prefer given a set of per-item prices which they would be charged. We call such a query in a public project a *k-demand query.* We will also define an analogous query for combinatorial auctions, called a *demand query.*

In the public project with one coverage valuation player described above, making a k-demand query with item prices all equal to 0 results in a solution to the problem. By assuming that the player has the computational capabilities necessary to lie, we are able to come up with a mechanism where lying is no longer beneficial. The use of VCG payments incentives the player to maximize social welfare, so as long as the mechanism maximizes social welfare there is no benefit to lying. A further advantage of our approach is that it can be used to model various assumptions of player power by choosing appropriate types of queries.

Proving that a mechanism like the one above finds an exact solution does not require any more new concepts or techniques. But not every pairing of an oracle and a mechanism design problem will result in an exact algorithm. Some classes of queries will not be powerful enough to allow for better mechanisms. The second main contribution of this dissertation is a framework for analyzing the computational complexity of these allocation problems paired with queries, as well as several substantive results showing hardness within this framework, thus strengthening the results from previous chapters.

5.1.4 Oracles and Reductions

We model the ability of players to answer queries as instance oracles.

Definition 5.1 (Instance Oracle). *Consider an n-player game A, with instances described by $(a_1, \ldots, a_n)$, where a_i is the private information held by player i. An* instance oracle *O is a black box function such that for some f, on input x it returns the n results $f(x, a_1), \ldots, f(x, a_n)$.*

For example, an oracle for k-demand queries would take prices $p_1, \ldots, p_n$ as the input x and for each i return $f(x, a_i)$, the set of k items which maximizes player i's utility with prices x. Note that this differs from a traditional oracle in that part of the input is fixed to the player valuations. So even if the k-demand query is NP-hard in general, an instance oracle for demand queries would not necessarily allow the mechanism to solve arbitrary NP-hard problems. For example, if in a particular instance, all players happen to have valuation functions $a_i(S) = 0$ for all sets S, a k-demand query instance oracle would simply return the items with the k lowest prices. This can easily be computed in polynomial time and therefore adds no power over polynomial computation.

We denote a problem A with associated oracle O by A^O. We can now define complexity classes in terms of problems with oracles. The class IONP is the class of all problems A^O such that $A \in$ NP. O can be any oracle. In order to show reductions between these problems, we need to modify the idea of a polynomial-time reduction to take the oracles into account. We do so as follows.

We say that A^O reduces to B^Q (or $A^O \leq_{IO} B^Q$) if there is a polynomial time reduction r from A to B such that for any a, $a \in A \Leftrightarrow r(a) \in B$ and O can be used on a to answer queries to Q on $r(a)$. So a reduction pairs a standard reduction (optionally making use of O) from A to B together with a way to simulate Q using O. This means that even showing $A^O \leq_{IO} A^Q$ can require a nontrivial reduction in order to be able to simulate Q on the result of the reduction, as seen in Section 5.6.1.

A problem $A^O \in$ IONP is IONP-complete if $A^O \in$ IONP and for all $B^Q \in$ IONP, $B^Q \leq_{IO} A^O$. Note that if A is NP-complete, A^O is IONP-complete for any $O \in$ FP. Most of the proofs of IONP-completeness in this dissertation either show that $O \in$ FP or reduce from an IONP-hard problem A^O where O is a trivial oracle (one for which the output does not depend on the input).

Instance oracle reductions fulfill the standard properties expected of reductions, as can be seen by the following lemmas.

Lemma 5.1. *If $A^O \leq_{IO} B^Q$ and B^Q has a polynomial-time solution, then A can be solved in polynomial time using queries to O.*

Proof. Let a be an instance of A. As $A^O \leq_{IO} B^Q$, we can create an instance b of B which has the same solution of A and for which queries to Q on b can be simulated in polynomial time using queries to O on A. Thus, we can solve b in polynomial time using the algorithm for B^Q, with only a polynomial slowdown for query computation. As a and b have the same solutions, this solves a in polynomial time. □

Lemma 5.2 (Contrapositive of 5.1). *If $A^O \leq_{IO} B^Q$ and no algorithm can solve A^O in polynomial time, then no algorithm can solve B^Q in polynomial time.*

Proof. Suppose by way of contradiction that no algorithm can solve A^O in polynomial time, but some algorithm exists which can solve B^Q in polynomial time. By Lemma 5.1, the existence of a polynomial-time algorithm for B^Q implies one for A^O, contradiction that no algorithm can solve A^O in polynomial time. □

Lemma 5.3. *If $A^O \leq_{IO} B^Q$ and $B^Q \leq_{IO} C^R$, then $A^O \leq_{IO} C^R$.*

Proof. Let R_1 and R_2 be the reductions from A to B and from B to C, respectively. We can simply compose R_1 and R_2 to find a reduction from A to C. Now, we need only see how queries to R on C can be simulated in polynomial time using queries to O on A. Consider some query x to O. This can be simulated in polynomial time using a polynomial number of queries $y_1, \ldots, y_\ell$ to Q on the intermediate instance of B. Furthermore, each query $y_1, \ldots, y_\ell$ can be simulated in polynomial time using queries to O on A. Thus, we can simulate queries to R on instances produced by our reduction in polynomial time using O. □

5.1.5 Definitions

We study two instance oracles. The first is the demand oracle alluded to in Section 5.1.4.

Definition 5.2 (Demand oracle (dem))**.** *A demand oracle takes as input a list of prices $p_1, \ldots, p_m$ and for each player i, returns a set S maximizing $v_i(S) - \sum_{j \in S} p_j$. This definition is the same for both auctions and public projects.*

We also define a similar oracle which is more useful in public projects.

Definition 5.3 (k-demand oracle (kdem))**.** *A k-demand oracle takes as input a list of prices $p_1, \ldots, p_m$ and for each player i, returns a set $S, |S| = k$ maximizing $v_i(S) - \sum_{j \in S} p_j$. This oracle is only defined for public projects, as auctions have no parameter k.*

5.1.6 Our Results

Our results are comprised of several reductions, pictured in Figure 5.1. Reductions between problems shown are indicated by black-headed arrows. All other reductions are from existing NP-hard problems paired with arbitrary oracles (as the oracles are not used in the reduction). The IONP-hard public projects also cannot be approximated by a maximal-in-range algorithm to a factor better than $\sqrt{m}$ using the associated oracles unless NP has polynomial circuits.

Figure 5.2 shows the hardness results shown in this chapter for maximal-in-range polynomial time mechanisms. All problems listed cannot be approximated better than $\sqrt{m}$ unless NP has polynomial circuits.

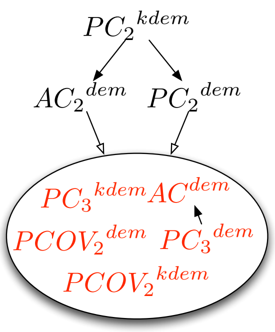

Figure 5.1: A summary of the results shown in this chapter, excluding those from Section 5.2. Arrows with a dark head indicate a reduction from the problem at the start of the arrow to the one at the end. Arrows with a clear head indicate a reduction which is implied by IONP-hardness. The circled problems are IONP-hard.

Valuation class	hard problems
Unit-demand	P2U^{dem} P2U^{kdem}
Multi-unit-demand	PMU$_3{}^{dem}$ PMU$_3{}^{kdem}$
Capped-additive	PC$_3{}^{dem}$ PC$_3{}^{kdem}$
Coverage	PCOV$_2{}^{dem}$ PCOV$_2{}^{kdem}$
Fractionally-subadditive	PFS^{dem} PFS^{kdem}

Figure 5.2: The hardness results for maximal-in-range mechanisms for public projects. The column labeled "hard problems" refers to problems which are both NP-hard and cannot be approximated better than $\sqrt{m}$ by a polynomial-time maximal-in-range mechanism unless NP has polynomial circuits. We do not list problems which contain others in the table as special cases. So we do not list PU^{dem} because P2U^{dem} is already in the table.

We also use techniques from Chapter 3 to extend our IONP-hardness results to also show that none of the IONP-hard public projects can be approximated better than $\sqrt{m}$ unless NP has polynomial circuits. We begin with a few results which easily follow from results in earlier chapters.

5.2 Simple Results

We show a few simple results here in the instance oracle model. First, we demonstrate that proof techniques from previous chapters do not always work in the instance oracle model by showing how instances of PC$_2$ and AC$_2$ produced in reductions in earlier chapters become easy with access to a *kdem* or *dem* oracle. After this, we will show how many of the hardness results from previous chapters immediately become IONP-hardness results with *dem* and *kdem* oracles, as these queries can be computed in polynomial time.

5.2.1 Special Cases of Games with 2 Capped-Additive Valuations

In the reductions showing that PC$_2$ and AC$_2$ are NP-hard, the instances produced have the property that Player 2 has value cap $c_2 = \sum_{j\in[m]} v_2^j$, so that this player actually has an additive valuation function. We show that instance oracles can be used to solve this special case, necessitating a new proof if IONP-hardness is desired.

Theorem 5.4. *In the special case where Player 2 has an additive valuation function, both $AC_2{}^{dem}$ and $PC_2{}^{kdem}$ can be solved exactly in polynomial time.*

Proof. In the case of AC$_2{}^{dem}$, consider the demand query to Player 1 with prices $p_j = v_2^j$. This will return a set S maximizing

$$\begin{aligned}
v_1(S) - \sum_{j\in S} p_j &= v_1(S) - \sum_{j\in S} v_2^j \\
&= v_1(S) - \sum_{j\in S} v_2^j - \left(\sum_{j\in[m]} v_2^j - \sum_{j\in[m]} v_2^j\right) \\
&= v_1(S) - \sum_{j\in[m]} v_2^j - \left(\sum_{j\in S} v_2^j - \sum_{j\in[m]} v_2^j\right) \\
&= v_1(S) - \sum_{j\in[m]} v_2^j - \left(\sum_{j\in S} v_2^j - \left(\sum_{j\in S} v_2^j + \sum_{j\in S^C} v_2^j\right)\right) \\
&= v_1(S) - \sum_{j\in[m]} v_2^j - \left(- \sum_{j\in S^C} v_2^j\right) \\
&= v_1(S) + \sum_{j\in S^C} v_2^j - \sum_{j\in[m]} v_2^j \\
&= v_1(S) + v_2(S^C) - \sum_{j\in[m]} v_2^j.
\end{aligned}$$

As $\sum_{j\in[m]} v_2^j$ does not depend on S, the set S therefore maximizes $v_1(S) + v_2(S^C)$. So (S, S^C) is an optimal allocation for this instance. Thus, an optimal allocation can be found with only one demand query.

In the case of PC$_2{}^{kdem}$, consider the demand query to Player 1 with prices $p_j = -v_2^j$. This will return a set S of size k maximizing

$$\begin{aligned}
v_1(S) - \sum_{j\in S} p_j &= v_1(S) - \sum_{j\in S} -v_2^j \\
&= v_1(S) + v_2(S)
\end{aligned}$$

so S is an optimal allocation, and can be found with only one query, completing the proof. $\square$

5.2.2 Hard Problems with Easy Oracles

In this section, we show how demand and k-demand queries can be answered for many of the valuation classes we study, leading to hardness results which are directly implied by earlier hardness results without oracles. Note that all results, both NP-hardness results and those for maximal-in-range algorithms, are preserved in the presence of oracles which can be computed in polynomial time.

Lemma 5.5. *For any V such that*

- *both PV_2 and AV_2 can be solved optimally in polynomial time*
- *additive functions are a special case of V, and can be constructed with a succinct representation in V in time polynomial in m*
- *given a valuation function v, it is possible to construct a valuation function $v'(S) = v(S) + \sum_{j\in S} w_j$ in polynomial time for any non-negative values $w_1, \ldots, w_m$*

it is possible to compute demand and k-demand queries for V in polynomial time.

Proof. Given a valuation function v, we create an instance of AV$_2$ in order to answer a demand query as follows. Let $w_j = \max(0, -p_j)$. So w_j is $-p_j$ when p_j is negative, and 0 otherwise. Player 1 has valuation function $v_1(S) = v(S) + \sum_{j\in S} w_j$ and player 2 has additive valuation function $v_2(S) = \sum_{j\in S} v_2^j$ where $v_2^j = \max(p_i, 0)$. So v_2^j is p_i when p is positive and 0 otherwise. Both of these functions can be constructed in polynomial time by the assumptions about V in the statement of the lemma.

Given v_1, v_2, let S, S^C be an allocation maximizing the social welfare, so player 1 gets items S and player 2 gets items S^C. If the allocation maximizing the social welfare does not allocate all items, allocate any left-over items arbitrarily so that the allocation is of this form. The social welfare of a set S is equal to

$$\begin{aligned}
v_1(S) + v_2(S^C) &= v(S) + \sum_{j\in S} w_j + \sum_{j\in S^C} v_2^j \\
&= v(S) - \sum_{j\in S, p_j<0} p_j + \sum_{j\in S^C} v_2^j \\
&= v(S) - \sum_{j\in S, p_j<0} p_j + \sum_{j\in[m]} v_2^j - \sum_{j\in S} v_2^j \\
&= v(S) - \sum_{j\in S, p_j<0} p_j - \sum_{j\in S} v_2^j + \sum_{j\in[m]} v_2^j \\
&= v(S) - \sum_{j\in S, p_j<0} p_j - \sum_{j\in S, p_j\geq 0} p_j + \sum_{j\in[m]} v_2^j \\
&= v(S) - \sum_{j\in S} p_j + \sum_{j\in[m]} v_2^j.
\end{aligned}$$

As $\sum_{j\in[m]} v_2^j$ does not depend on S or S^C, the optimal allocation maximizes $v(S) - \sum_{j\in S} p_j$, and is therefore an answer to the demand query with prices $p_1, \ldots, p_m$. Furthermore, as AV_2 can be solved optimally in polynomial time, we can find this query result in polynomial time.

In order to answer a k-demand query, we construct an instance of PV_2 as follows. Let $p_{\max} = \max_j p_j$. Player 1 has valuation function $v_1 = v$ and player 2 has additive valuation function $v_2(S) = \sum_{j\in S} v_2^j$ where $v_2^j = p_{\max} - p_j$. Note that $p_{\max} - p_j \geq 0$. Both of these can be constructed in polynomial time by the assumptions in the statement of this lemma. The social welfare of a set S of size k is equal to

$$\begin{aligned} v_1(S) + v_2(S) &= v(S) + \sum_{j\in S} p_{\max} - p_j \\ &= v(S) - \sum_{j\in S} p_j + \sum_{j\in S} p_{\max} \\ &= v(S) - \sum_{j\in S} p_j + k \cdot p_{\max}. \end{aligned}$$

As $k \cdot p_{\max}$ does not depend on S, the optimal allocation maximizes $v(S) - \sum_{j\in S} p_j$, thus giving an answer to a k-demand query with prices $p_1, \ldots, p_m$. Furthermore, as PV_2 can be solved optimally in polynomial time, we can find this query result in polynomial time. □

Now, we need only show that the valuation classes *multi-unit-demand* and *fractionally-subadditive* both meet the criteria outlined in Lemma 5.5.

Lemma 5.6. *Both demand and k-demand queries can be computed in polynomial time for valuation functions from the classes multi-unit-demand and fractionally-subadditive.*

Proof. First, we will show that this is the case for multi-unit-demand. Theorem 3.13 shows that PMU_2 can be solved in polynomial time and Corollary 1.4 shows that AMU_2 can be solved in polynomial time. So we need only to see how to construct additive valuation functions and how to construct $v'(S) = v(S) + \sum_{j\in S} w_j$ in polynomial time.

In order to construct an additive valuation function $v_i(S) = \sum_{j\in S} v_i^j$, we simply construct the m unit-demand functions

$$v_i^{(j)}(S) = \begin{cases} v_i^j, & j \in S \\ 0, & \text{otherwise} \end{cases}$$

and the multi-unit-demand function $v_i(S) = \max_{S_1,\ldots,S_m} \sum_j v_i^{(j)}(S_j)$ is the additive function we desire.

In order to construct a valuation function $v'(S) = v(S) + \sum_{j\in S} w_j$, we take two steps. First, we add w_j to the value of each item j in each unit-demand function. That way, if j is the item which gets value from a particular unit-demand function, the total value increases by w_j. Furthermore, we add m additional unit-demand functions $v^{extra}(S) = \max_{j\in S} w_j$. Thus, any items $j \in S$ which don't add to the value originally can be matched to these extra valuation functions for an extra w_j value. So both matched and unmatched items j add w_j to the total, giving the set S value of $\sum_{j\in S} w_j$, plus the value of the original matching corresponding to this. So this is maximized by the new value $v'(S) = v(S) + \sum_{j\in S} w_j$.

Now, we show this for fractionally-subadditive functions. Theorem 3.21 shows that both AFS$_2$ and PFS$_2$ can be solved optimally in polynomial time. As a fractionally-subadditive valuation is simply the maximum over additive valuations, an additive valuation can be constructed in polynomial time by simply making a function which is the maximum over a single additive valuation. Finally, we can create a function $v'(S) = v(S) + \sum_{j\in S} w_j$ by simply adding w_j to the value of j in each additive function that v is composed of. Thus,

$$\begin{aligned} v'(S) &= \max_{\ell}(v^{(\ell)}(S) + \sum_{j\in S} w_j) \\ &= (\max_{\ell} v^{(\ell)}(S)) + \sum_{j\in S} w_j \\ &= v(S) + \sum_{j\in S} w_j \end{aligned}$$

which completes the proof. □

As unit-demand valuations are a special case of multi-unit-demand, this also implies the following corollary.

Corollary 5.7. *It is possible to compute demand and k-demand queries for unit-demand valuation functions in polynomial time.*

Lemma 5.6 and Corollary 5.7, together with previous hardness results, give us the following hardness results.

Theorem 5.8. *PU^{kdem}, PU^{dem}, $PMU_3{}^{kdem}$, $PMU_3{}^{dem}$, PFS^{kdem}, and PFS^{dem} are all IONP-hard and furthermore cannot be approximated by a polynomial-time maximal-in-range mechanism with a ratio of $m^{1/2-\epsilon}$ unless $NP \subseteq P/poly$.*

5.3 Coverage Valuations

One of the more surprising results from Chapter 3 is that no truthful mechanism can achieve an approximation ratio for PSCOV$_1$ better than $\sqrt{m}$ unless $NP \subset P/poly$. Clearly, PSCOV$_1{}^{kdem}$ and even PWCOV$_1{}^{kdem}$ have polynomial time solutions, as a single oracle call can determine the social welfare maximizing set. We show that PCOV$_2{}^{kdem}$ and PCOV$_2{}^{dem}$ are IONP-hard.

Theorem 5.9. *$PCOV_2{}^{dem}$ and $PCOV_2{}^{kdem}$ are both IONP-hard.*

Proof. We reduce from the NP-hard problem of vertex cover on a 3-regular graph $G = (V, E)$ [24]. By Vizing's theorem [43], there exists an edge coloring of this graph using 4 colors such that no two adjacent edges share the same color. Furthermore, there is a standard constructive proof of Vizing's theorem in [6] which demonstrates how such a coloring can be found in polynomial time. We begin by coloring the edges using colors C_1, C_2, C_3, C_4. Let E_1 be the set of edges colored C_1 and C_2 and E_2 be the set of edges colored C_3 and C_4. An example illustrating the effects of partitioning the edges into sets E_1 and E_2 appears in Figure 5.3.

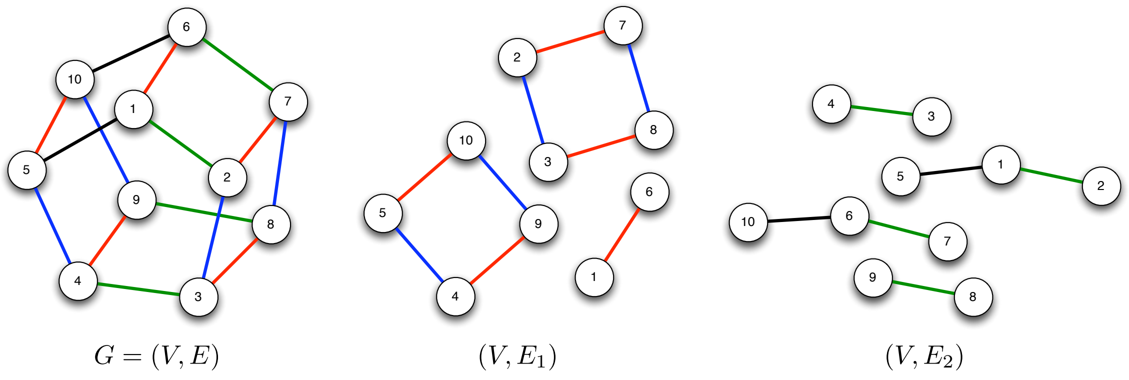

Figure 5.3: A 3-regular graph G, together with graphs representing the edge sets E_1 and E_2.

There are $m = |V|$ items. Let $V = \{v_1, \ldots, v_m\}$. Each item j corresponds to the vertex v_j. Player 1 has a valuation $v_1(S) = |\bigcup_{j \in S} V_1^j|$ where

$$V_1^j = \{e \in E_1 : v_j \in e\}$$

is the set of edges colored C_1 or C_2 incident on v_j. So the value of a set S to player 1 is the number of edges colored C_1 or C_2 incident upon the vertices corresponding to the values in S. Player 2's value is defined similarly, with

$$V_2^j = \{e \in E_2 : v_j \in e\}.$$

The social welfare for allocating a set S is the number of edges incident upon vertices corresponding to values in S. So there is a social welfare of at least $|E|$ iff there is a vertex cover of size at most k. This completes the reduction.

Now that we have demonstrated a reduction, we need only show that the *dem* and *kdem* queries can be computed in polynomial time to complete the proof of IONP-hardness. A query consists of a list of item prices $p_1, \ldots, p_m$. For each player, we must find a set maximizing the difference between the number of appropriately colored edges covered and the sum of the item prices. We achieve this in two steps. We will focus on player 1, as the solution for player 2 can be found in the same manner by symmetry.

First, we examine each connected component in $G_1 = (V, E_1)$. As illustrated by Figure 5.3, the fact that G_1 can be edge colored with 2 colors implies that each of these components is a path or a cycle. Label these components $P_1, \ldots, P_c$. For each component P_i, we compute the at most n values P_i^j, the maximum value obtainable for player 1 choosing at most j items corresponding to vertices in P_i. These values can be computed by a simple dynamic programming approach as follows.

For a path P_i with ℓ vertices, order the vertices from one end to the other. Label these vertices $\rho_1, \ldots, \rho_\ell$. For each α from 0 to ℓ and i from 0 to ℓ, we compute two values. We compute α_i^j, the most value obtainable using j vertices from $\rho_1, \ldots, \rho_{i-1}$. We also compute β_i^j, the most value obtainable using j vertices from $\rho_1, \ldots, \rho_i$ which necessarily includes ρ_i. So α_i^j is the most value not using ρ_i, and β_i^j is the most value using it. Note that $\alpha_i^j = \max(\alpha_{i-1}^j, \beta_{i-1}^j)$ and $\beta_i^j = \max(\alpha_{i-1}^{j-1}+2, \beta_{i-1}^{j-1}+1)$ for $i < c$ and $\beta_c^j = \max(\alpha_{c-1}^{j-1}+1, \beta_{c-1}^{j-1})$. The last two formulas come from observing that if we add vertex ρ_i to a cover, which uses vertices from $\rho_1, \ldots, \rho_{i-1}$, it may cover edges $\{\rho_{i-1}, \rho_i\}$ and $\{\rho_i, \rho_{i+1}\}$. The first edge is already covered if ρ_{i-1} is in the cover, and the second edge only exists for $i < c$. Note that the first edge only exists for $i > 1$, so we set base cases $\alpha_1^j = 0$, $\beta_1^j = 1$ for any path containing at least 1 edge. As i and j are both bounded by n, these values can all be computed in $O(|P_i|^2)$ time via a dynamic programming table. We then have $P_i^j = \max(\alpha_c^j, \beta_c^j)$.

For a cycle with $c+1$ nodes, we simply choose a node in the path, and include it in the cover. This covers two edges. Remove these two edges and use the above algorithm to compute the best covers for each of the remaining paths. Set P_i^j to be the maximum of $\alpha_c^{j-1}+2, \beta_c^{j-1}+2$. Repeat this for every possible choice of node to remove at the beginning, and take the maximum value of P_i^j found by this method (and set $P_i^0 = 0$). This computes the correct value because if the node removed is in the actual best cover using j items, it covers the two edges incident on it, and the other $j-1$ items form a best cover of the remaining path. The path algorithm is run $|P_i|$ times, so this requires $O(|P_i|^3)$ time.

Next, we combine these partial results for each connected component using more dynamic programming. Let U_i^j be the maximum utility using at most i items from components $P_1, \ldots, P_i$. Note that $U_i^j = \max_\ell U_{i-1}^{j-\ell} + P_i^\ell$, which is the maximum value using $j-\ell$ items from $P_1, \ldots, P_{i-1}$ and ℓ items from P_i. As both i and j are limited by n, these can be computed by dynamic programming in time $O(n^2)$.

The total runtime for this is at most $\sum_i O(|P_i|^3) + O(n^2) = O(n^3)$. After these computations, the answer to a *kdem* query is simply U_c^k, and the answer to a *dem* query is $\max_i U_c^i$. We can also return the corresponding sets of items by backtracking through the dynamic programming table or keeping a copy of the set which achieves the value in each cell of the table. Either approach only requires polynomial overhead at each step.

Thus, we have shown an NP-hardness reduction for PCOV$_2$ in which both *kdem* and *dem* queries can be answered in polynomial time. Thus, both ${\text{PCOV}_2}^{kdem}$ and ${\text{PCOV}_2}^{dem}$ are IONP-hard. $\square$

We can now apply techniques from Chapter 3 to strengthen the above proof into a proof that no MIR algorithm can approximate ${PCOV_2}^{kdem}$ or ${PCOV_2}^{dem}$ to a factor better than $\sqrt{m}$.

Theorem 5.10. *No polynomial time maximal-in-range algorithm for ${PCOV_2}^{kdem}$ or ${PCOV_2}^{dem}$ has an approximation ratio of $m^{1/2-\epsilon}$ for any constant ϵ unless $NP \subset P/poly$.*

Proof. In the proof of Theorem 3.18, we showed how Lemma 3.6 implies that that any maximal-in-range mechanism for $PCOV_1$ (a special case of $PCOV_2$) must have a range with VC dimension at least m^α for some constant $\alpha > 0$. That is, for every subset S of these m^α items, there is an allocation S' in the range that contains S, together with $k-|S|$ items from the other $m-m^\alpha$. We use this fact to embed the previous reduction in the m^α items in such a way that the maximal-in-range algorithm solves it exactly.

Assume that a polynomial-time maximal-in-range algorithm A for ${PCOV_2}^{kdem}$ or ${PCOV_2}^{dem}$ achieves an approximation ratio of $m^{1/2-\epsilon}$. Order the items such that the first m^α are the ones that exhibit the VC dimension of A's range. This ordering corresponds to the polynomial advice in P/poly. Perform a reduction from a vertex cover instance with m^α vertices to these m^α items as in the proof of Theorem 5.9, but for each edge, create $k+1$ corresponding items in each set, rather than just 1. This has the effect of multiplying the social welfare by $k+1$. For the other $m-m^\alpha$ items, add $m-m^\alpha$ new elements $n_{m^\alpha+1}, \ldots, n_m$ to U_1, and set $V_i^j = \{n_j\}$ for $j > m^\alpha$.

Now, A will find a set of k items with social welfare $(k+1)|E| + k - k'$ iff there is a set of k' vertices which covers the original graph. Clearly, if A finds a set with welfare $(k+1)|E| + k - k'$, then this must correspond to a covering of the graph in order to get the $(k+1)|E|$ term, as the total value from items greater than m^α is at most k. The rest of the welfare must therefore come from having at least $k-k'$ items chosen from items greater than m^α. So the covering must have had at most k' items.

Furthermore, if there exists such a covering, it will be found by A, as every possible subset of the m^α items is in As range and A is maximal-in-range. The rest will be filled in by the other $m-m^\alpha$ items, but the way in which these are chosen doesn't matter by construction. So A will solve PCOV_2, demonstrating that NP $\subseteq$ P/poly.

Now, we need only see that we can still simulate the oracle queries in order to show that the oracles do not add any extra power beyond that of polynomial circuits. We already know how to simulate queries to determine the best coverage choosing j of the m^α items corresponding to the original reduction. We can also choose the best j of the $m-m^\alpha$ items by simply choosing the j with lowest price, as the added value of j of these items depends only on j and not on which particular items are chosen. So we can find the best j items overall by checking all i and finding the best i from the first m^α, and the best $j-i$ from the rest. Thus, we can still compute the *dem* and *kdem* queries in polynomial time by either taking the maximum utility for k items, or the maximum utility for j

items over all j. $\square$

5.4 Three-Player Public Projects

In this section, we examine ${PC_3}^{dem}$ and ${PC_3}^{kdem}$, and show that they are both IONP-hard and cannot be approximated to a factor better than $\sqrt{m}$ by an efficient maximal-in-range mechanism unless NP has polynomial circuits. We begin by proving the following theorem.

Theorem 5.11. *${PC_3}^{kdem}$ and ${PC_3}^{dem}$ are IONP-hard.*

Proof. We reduce from 3-dimensional matching (3DM). A 3DM instance consists of a set of triples $T \subseteq [k] \times [k] \times [k]$, and the decision problem is does there exist a set $M \subseteq T$, $|M| = k$ such that for each $i \in [3], j \in [k]$, there is an element of M such that its ith coordinate is j?

Starting from such an instance of 3DM, we create the following instance of PC_3. Let $\tau = 2^{\lceil \log_2 |T| \rceil + 1}$ (so $\tau > |T|$). For each item $(\alpha_1, \alpha_2, \alpha_3) \in T$, create an item i such that

- Player 1 values i at $\tau^{3k+1} + \tau^{2k+\alpha_1} - \tau^{k+\alpha_2}$
- Player 2 values i at $\tau^{3k+1} + \tau^{k+\alpha_2} + \tau^{\alpha_3}$
- Player 3 values i at $\tau^{3k+1} - \tau^{2k+\alpha_1} - \tau^{\alpha_3}$

and

- Player 1 has budget $k \cdot \tau^{3k+1} + \sum_{j\in[k]} \tau^{2k+j} - \tau^{k+j}$
- Player 2 has budget $k \cdot \tau^{3k+1} + \sum_{j\in[k]} \tau^{k+j} + \tau^{j}$
- Player 3 has budget $k \cdot \tau^{3k+1} + \sum_{j\in[k]} -\tau^{2k+j} - \tau^{j}$.

We now show that there exists a set $M \subseteq T$ of size k covering each coordinate iff there is a set of k items that gives each player value equal to its budget.

Lemma 5.12. *The above reduction is a valid reduction from 3-dimensional matching to PC_3.*

Proof. Assume that there exists a set of k items S such that all budgets are met or exceeded. If player 1's budget is exceeded, then either

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{2k+\alpha_1} > \sum_{j\in[k]} \tau^{2k+j} \tag{5.1}$$

or

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{k+\alpha_2} < \sum_{j\in[k]} \tau^{k+j}. \tag{5.2}$$

If (5.1), this player 3 would be under budget, a contradiction. Otherwise, (5.2) means that player 2 is under budget, also a contradiction. So we have

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{2k+\alpha_1} = \sum_{j\in[k]} \tau^{2k+j} \tag{5.3}$$

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{k+\alpha_2} = \sum_{j\in[k]} \tau^{k+j}. \tag{5.4}$$

If $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{\alpha_3} > \sum_{j\in[k]} \tau^j$, then player 3 is under budget, and if $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{\alpha_3} < \sum_{j\in[k]} \tau^j$, then player 2 is under budget. So

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{\alpha_3} = \sum_{j\in[k]} \tau^j. \tag{5.5}$$

Together, (5.3), (5.4), and (5.5) imply that for every j, there is some $\alpha_1 = j$, some $\alpha_2 = j$, and some $\alpha_3 = j$, implying a 3-dimensional matching. Now, we note that if a 3-dimensional matching exists, then choosing the corresponding items will result in all 3 budgets being met. So a 3-dimensional matching exists iff there is some set of items for which the social welfare is equal to the sum of the players' budgets. $\square$

In order to complete the proof of Theorem 5.11, we now need show how demand and k-demand queries for instances produced by the reduction can be computed in polynomial time. We show that for any $\ell \in [m]$, the utility maximizing set of size ℓ for any player can be computed in polynomial time. Using this, it is trivial to compute demand and k-demand queries by taking the maximum utility over all ℓ or just taking the utility for $\ell = k$, respectively.

Lemma 5.13. *For any set of item prices $p_1, \ldots, p_m$ and any ℓ, it is possible to compute the utility maximizing set of size ℓ for any player resulting from the reduction in this section in polynomial time.*

Proof. Each player has a value function of the form

$$v(S) = \sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{3k+1} \pm \tau^{(3-i)k+\alpha_i} \pm \tau^{(3-j)k+\alpha_j}$$

for some $1 \leq i < j \leq 3$. For any $\ell < k$, the player's budget cannot be reached, so the utility maximizing set consists of the ℓ items with maximum individual utility. For $\ell > k$, the budget will be exceeded by any set of items, so the ℓ items of smallest price are chosen. We now need only to examine the $\ell = k$ case.

In this case, we can ignore the τ^{3k+1} terms, as they will always sum to $k \cdot \tau^{3k+1}$. Now, for each value α_i from k to 1, we consider 3 cases. Either zero, one, or more than one items with ith coordinate α_i are chosen.

If zero, then we know that if the sign of $\tau^{(3-i)k+\alpha_i}$ is positive, the budget will be exceeded, and if negative, the budget will not be reached. In either case, it is easy to choose the items with smaller α_i values by either maximizing individual utility or minimizing prices.

If more than one item, we also know that the budget will either be exceeded or not met based on the sign of $\tau^{(3-i)k+\alpha_i}$, so we choose the 2 of lowest price or highest utility, then choose items to fill out the rest of the k with smaller or equal α_i values based on lowest price or highest individual utility.

Finally, if we choose exactly 1 such item, we move on to $\alpha_i - 1$. We reserve the choice of item corresponding to α_i until later. If we reach a point where the budget will definitely be reached or not, we can choose the item of smallest price or highest utility accordingly.

After we have iterated through all k values of α_i^*, we move on to values α_j^* of α_j. Again we have 3 possibilities, zero, one, or more than one items. With one item, we still move on. With zero items or more than one item, we again know whether the budget will be exceeded, but it is more difficult to determine how to maximize utility. We use weighted bipartite matching for this purpose.

We use a graph $G = (V_1, V_2, E)$. $V_1 = \{v_1, \ldots, v_k\}$ corresponds to the values of α_i which must each be covered once and

$$V_2 = \{w^1_{a_j^*+1}, \ldots, w^1_k\} \cup \bigcup_{\ell \in [k]} \{w^\ell_1, \ldots, w^\ell_{\alpha_j^*-1}\} \cup \bigcup_{\ell \in T} w^\ell_j,$$

where T is equal to either $\emptyset$ or $[k]$, depending on whether we want zero or more than one item with this value.

For each triple $(\alpha_1, \alpha_2, \alpha_3)$ in the 3-dimensional matching instance, we add an edge $\{v_{\alpha_i}, w^\ell_{\alpha_j}\}$ for each $w^\ell_{\alpha_j}$ in V_2. We give these edges weight either equal to the negative of the price of the corresponding item (if we're choosing more than one item for α_j^* and will therefore exceed the budget) or equal to the utility (if we're choosing 0 items for α_j^* and will therefore not reach the budget). After this weighting, shift the weights by adding the same positive value to all of them to ensure that they are all positive.

Let $W_{\max}$ be the maximum edge weight. To ensure that at least 2 items corresponding to α_j^* are chosen, add $W_{\max} \cdot k$ to the weights of edges incident on $w^1_{\alpha_j}$ and $w^2_{\alpha_j}$. To ensure that one item is chosen corresponding to each $w^1_k, \ldots, w_{\alpha_j^*+1}$, add $W_{\max} \cdot k$ to the weights of edges incident on these as well. As $W_{\max} \cdot k$ is larger than the weight of any matching consisting of edges not given this extra factor, any maximum matching must match edges incident on all vertices from $w^1_{\alpha_j^*+1}, \ldots, w^1_k$, as well as $w^1_{\alpha_j^*}$ and $w^2_{\alpha_j^*}$ if they exist.

Compute a maximum weighted k matching. This will result in a set of edges corresponding to an allocation that covers each α_i exactly once, each $\alpha_j > \alpha_j^*$ exactly once, and α_j^* either 0 or more than one time as appropriate while maximizing utility.

The last step is to notice that after all α_j are covered exactly once, the budget is exactly reached. So the above procedure can be used to find the maximum allocation with $\alpha_j^* = 0$.

Now that we have shown how to compute the utility maximizing set, we need only show that the computation only requires polynomial time. At each step going through α_i or α_j, we consider covering α_i or α_j 0, 1, or more than 1 time. We have shown how to compute the solution for 0 or more than 1 in polynomial time. For exactly 1, we simply move on to the next step of the algorithm. So we have $2k$ steps, each requiring polynomial time, which comes out to a polynomial runtime overall.

After completing all steps, the set which maximizes the utility is the set of maximum utility computed at any step in the algorithm. Thus, the utility maximizing set will be found by the end of the algorithm. $\square$

This completes the proof of Theorem 5.11 $\square$

As with Theorem 5.10, $PCOV_2$, the proof of Theorem 5.11 can be modified to show that ${PC_3}^{kdem}$ and ${PC_3}^{dem}$ cannot be approximated well by maximal-in-range mechanisms.

Theorem 5.14. *No polynomial time maximal-in-range algorithm for ${PC_3}^{kdem}$ or ${PC_3}^{dem}$ has an approximation ratio of $m^{1/2-\epsilon}$ for any constant ϵ unless $NP \subset P/poly$.*

Proof. In Theorem 3.17, we showed how Lemma 3.6 can be used to show that any algorithm for PC$_2$ (a special case of PC$_3$) which achieves an approximation of at least $m^{1/2-\epsilon}$ must have a range with VC dimension at least m^α for some constant α. As in the proof of Theorem 5.10, we embed the reduction into these m^α items, and design valuations for the other $m - m^\alpha$ items such that exactly $k - k'$ of them will be chosen in an optimal allocation and can be considered separately from the m^α items from the original reduction when answering *dem* and *kdem* queries.

The $m - m^\alpha$ items not corresponding to those in the initial proof have a very simple valuation. Player 1 values them at τ^{4k}, and players 2 and 3 value them at 0. Player 1's budget is increased by $(k - k')\tau^{4k}$.

If more than $k - k'$ of these extra items are chosen, then fewer than k' of the original items are chosen, and therefore players 2 and 3 do not reach their budgets. If fewer than $k' - k$ of these extra items are chosen, then player 1's budget cannot be reached. So in the optimal allocation, k' of the original m^α items are chosen, and the other $k - k'$ are chosen arbitrarily from the rest. So this algorithm will find the optimal solution, which will in turn solve the original instance.

Now, we need to see that the ability to compute demand sets of size ℓ for any ℓ is preserved. This is easy, as we need only determine the best way to choose i items from the first m^α and $\ell - i$ from the rest. This is trivial for players 2 and 3, as we simply choose i items as in Lemma 5.13, then choose the $\ell - i$ items of lowest price from the rest (as these do not affect the value).

The contribution from the $\ell - i$ items chosen from the other $m - m^\alpha$ to player 1's value is the same regardless of choice, so the $\ell - i$ of these that are chosen must be those of lowest price. If $\ell - i < k - k'$, then player 1's budget can not be exceeded, so simply choose the i of greatest utility. If $\ell - i > k - k'$, then player 1's budget is exceeded by these items alone, so the other i chosen must be those of lowest price.

Otherwise, $\ell - i = k - k'$. This leaves the remaining budget and item prices for the m^α items the same as in the original reduction. Therefore, the proof of Lemma 5.13 demonstrates how to compute the solution to this part of the query. By taking the result of maximum utility over all i, we can thus compute the query response in polynomial time. $\square$

5.5 Auctions with Many Players are Hard

In this section, we show that $\text{PC}^{dem} \leq_{IO} \text{AC}^{dem}$, proving that capped-additive auctions with demand queries are IONP-hard, as we have already shown that ${\text{PC}_3}^{dem}$ is IONP-hard, and ${\text{PC}_3}^{dem}$ is a special case of PC^{dem}.

Theorem 5.15. *$PC^{dem} \leq_{IO} AC^{dem}$*

Proof. We prove Theorem 5.15 via the following reduction.

Reduction 5.1. *We begin with an instance of PC^{dem} in which player i has value v_i^j for item j and value cap c_i, and k items are to be chosen. We produce an instance of AC^{dem} with $mn + k$ items and $n + m$ players.*

The first mn items, we label with pairs (i, j) for $1 \leq i \leq n, 1 \leq j \leq m$. The final k items, we label $\alpha_1, \dots, \alpha_k$. For $i = 1, \dots, n$,

- *Player i has value v_i^j for items $(i, 1), \dots, (i, m)$ and value 0 for all other items.*
- *Player i has budget c_i.*

Let $C = \sum_i c_i + 1$. For $j = 1, \dots, m$,

- *Player $n + j$ has value C for items $(1, j), \dots, (n, j)$, value nC for items $\alpha_1, \dots, \alpha_k$, and value 0 for all other items.*
- *Player $n + j$ has budget nC.*

The idea behind Reduction 5.1 is that each item from the public project is turned into n new items and that for each original item, either all of the first n players get a copy, or one of the last m players receives all of the copies. Any of the last m players that does not receive all copies of his item instead receives one of the items $\alpha_1, \dots, \alpha_k$. These free up k items to be given to each of the first n players while allowing the last m players to all reach their budgets.

Lemma 5.16. *Let $S_1, \ldots, S_{m+n}$ be an allocation to an auction produced by Reduction 5.1. If $S_1, \ldots, S_{m+n}$ has social welfare at least mnC, then for all but k values j from 1 to m, every item $(i,j), i = 1, \ldots, n$ is allocated to player $n+j$.*

Proof. Suppose by way of contradiction that there exists an allocation with social welfare at least mnC, yet there are $k+1$ players $n+j$ who don't receive all of the items (i,j). At most k of these players receive some item α_i, and thus have value nC. Thus, one of these players has value at most $(n-1)C$, as it receives at most $n-1$ items with value each C, and has 0 value for all other received items. So the social welfare from players $n+j, j = 1, \ldots, m$ is at most $mnC - C$. The social welfare from players $i = 1, \ldots, n$ is strictly less than C by definition, for a total social welfare strictly less than $mnC - C + C = mnC$, a contradiction. □

Lemma 5.17. *A public project has social welfare V iff the auction produced by Reduction 5.1 has social welfare $V + mnC$*

Proof. If the public project has social welfare V, let S be the allocation achieving that welfare. The following allocation has social welfare $V + mnC$ in the auction. For every $i = 1, \ldots, n$, give player i the items $\{(i,j) : j \in S\}$. For every $j \in [m] \backslash S$, give player $n+j$ the items $\{(i,j) : i \in [n]\}$. For the remaining k players $n+j, j \in S$, give them each one of the items $\alpha_i, i = 1, \ldots, k$.

The players $i = 1, \ldots, n$ have values equal to that from the public project, so they contribute V to the social welfare. The players $n+j, j = 1, \ldots, m$ each have value nC, for a total contribution of mnC. Thus, the social welfare of this allocation is $V + mnC$.

Now, suppose that the auction has social welfare $V + mnC$ for some $V > 0$. By Lemma 5.16, for all but k values j from 1 to m, every item $(i,j), i = 1, \ldots, n$ is allocated to player $n+j$. Let S be the set of k values j from 1 through m such that not every (i,j) is allocated to player $n+j$. As players $n+j, j = 1, \ldots, m$ have total value at most mnC, players 1 through n must have total value at least V. Furthermore, player i has value at most $\min(\sum_{j \in S} v_i^j, c_i)$, as the only items that it gets value from that are not given to players $n+j$ are $(i,j), j \in S$. Thus, $\sum_i \min(\sum_{j \in S} v_i^j, c_i) \geq V$, so allocation S has social welfare at least V in the original public project. □

Lemma 5.18. *The demand oracle in a public project can be used to compute answers to demand queries in the auction resulting from Reduction 5.1.*

Proof. To simulate a query to player i, simply query player i in the public project with price for item j taken from the price (i,j), as these are the only items the player values. Add all other items with negative price to arrive at the query result.

To simulate a query to player $n+j$, there are only a few relevant choices for value. Either the α_i of minimum price is chosen, or the ℓ items (i,j) of lowest price for some ℓ. In addition to these, add all items of negative price to arrive at the proper query result. Clearly, one of these choices maximizes utility and they can all be computed in polynomial time. □

As we have shown a valid reduction and the ability to simulate oracle queries, Lemmas 5.16, 5.17, and 5.18 together prove Theorem 5.15. □

5.6 The Relative Power of k-Demand and Demand Queries

In the previous sections, we demonstrated IONP-hardness. Here, we show how IONP reductions can be used to better understand the relationships between problems whose hardness we have not exactly determined. The three problems we examine are ${PC_2}^{kdem}$, ${PC_2}^{dem}$, and ${AC_2}^{dem}$. For these three problems, we do not have proofs of IONP-hardness or polynomial-time algorithms using the oracles. We demonstrate that ${PC_2}^{kdem}$ can be reduced to the other two, and is therefore a potentially easier problem.

5.6.1 Public Projects Self-Reduction

In this section we show a self-reduction between PC_2 and itself that allows for *dem* queries to be simulated by *kdem* queries, showing that the *dem* oracle is no more powerful than the *kdem* oracle.

Theorem 5.19. ${PC_2}^{kdem} \leq_{IO} {PC_2}^{dem}$.

Proof. We prove Theorem 5.19 via the following reduction.

Reduction 5.2. *We start with an instance of the combinatorial public projects problem with* 2 *capped-additive players. Player i has value v_i^j for item j and value cap c_i. Let $D = 2m \max(c_1, c_2)$. The reduction produces an instance with 2 capped-additive players, where*

- *Player i has value $w_i^j = v_i^j + D$ for item j*
- *Player i has value cap $d_i = c_i + kD$.*

We assume without loss of generality that for all i, j, $v_j^j \leq c_i$.

Lemma 5.20. *A public project has social welfare V iff the public project produced by Reduction 5.2 has social welfare at least $V + 2kD$.*

Proof. Consider player i's value for a set S of size k. In the original public project, the value is

$$\min\left(\sum_{j \in S} v_i^j, c_i\right).$$

In the instance produced by the reduction, player i's value is

$$\begin{aligned}\min\left(\sum_{i\in S} w_j^{(i)}, c_i\right) &= \min\left(\sum_{i\in S} v_i^j + D, c_i + kD\right)\\ &= \min\left(\sum_{i\in S} v_i^j, c_i\right) + kD.\end{aligned}$$

So summing over both players, if there is a social welfare of V for set S in the original auction, there is a social welfare of $V + 2kD$ for S in the auction produced by Reduction 5.2. □

Lemma 5.21. *The kdem oracle for a public project can be used to answer dem queries for the public project resulting from Reduction 5.2 in polynomial time.*

Proof. Consider the result of a *dem* query for player i with prices $p_1, \ldots, p_m$. There are 3 possible cases:

Case 1: It returns a set S, $|S| > k$. In this case, the value of the set to player i is at least $\min((k+1)D, c_i + kD)$. As $D > c_i$, $(k+1)D > c_i + kD$, so this minimum is $d_i = c_i + kD$ regardless of the choice of S. Thus, the query will simply return a set of size at least $k+1$ of minimum price. This is either the $k+1$ items of lowest price, or all items of negative price if there are more than $k+1$ of these.

Case 2: It returns a set S, $|S| < k$. In this case, the value of the set to player i is at most $\min((k-1)(D+c_i), c_i+kD)$. As $(k-1)c_i < D$, this is less than $\min((k-1)D+D, kD+c_i) = kD \leq d_i$. So the budget is not reached, and the utility of S is the sum of the individual utilities of the items in S. Thus, S consists of the at most $k-1$ items of maximum positive utility.

Case 3: It returns a set S, $|S| = k$. In this case, player i's utility is

$$\min\left(\sum_{j\in S} v_j^{(i)} + D, b_i + kD\right) - \sum_{j\in S} p_j = \min\left(\sum_{j\in S} v_j^{(i)}, b_i\right) + kD - \sum_{j\in S} p_j.$$

To find a set of size k maximizing this utility, we simply need a set of size k maximizing

$$\min\left(\sum_{j\in S} v_i^j, c_i\right) - \sum_{j\in S} p_j,$$

which is exactly what the original *kdem* oracle returns for player i given prices $p_1, \ldots, p_m$.

So by computing the results for each of these three cases, we can choose the one with maximum utility and return it as the result of the *dem* query. The solutions to all of these cases can be computed in polynomial time as shown above, so *dem* query results can be computed in polynomial time. □

As we have a valid reduction, and the *kdem* oracle can simulate the *dem* oracle on instances produced by the reduction, we have proven Theorem 5.19. □

5.6.2 Reducing Public Projects to Auctions

In this section, we reduce from PC_2 to AC_2 in such a way that the *kdem* oracle can again be used to simulate the *dem* oracle.

Theorem 5.22. $PC_2{}^{kdem} \leq_{IO} AC_2{}^{dem}$.

Reduction 5.3. *We begin with a public project in which the goal is to choose k items, and player i has valuation function $v_i(S) = \min(\sum_{j\in S} v_i^j, c_i)$. Assume without loss of generality that $c_i \geq v_i^j$ for all i, j.*

Let $W = \sum_i v_2^i$ and $C = \max(c_1, c_2, W) + 1$. We create an instance of AC_2 with the players 1,2 defined by $w_i(S) = \min(\sum_{j\in S} w_i^j, d_j)$ where:

- *Player 1 has value $w_1^j = v_1^j + mC$ for item j.*
- *Player 1 has value cap $d_1 = c_1 + mkC$.*
- *Player 2 has value $w_2^j = mC - v_2^j$ for item j.*
- *Player 2 has value cap $d_2 = c_2 + m(m-k)C - W$.*

We claim that the original public project has social welfare at least V iff the auction produced has social welfare at least $V + m^2C - W$.

Lemma 5.23. *In an optimal allocation for the auction produced by Reduction 5.3, player 1 gets k items and player 2 gets $m - k$ items.*

Proof. Suppose by way of contradiction that player 1 gets fewer than k items in an optimal allocation. Let S be the set of items player 1 gets. Player 2 has value at most $d_2 = c_2 + m(m-k)C - W$.

Player 1 has value at most

$$\sum_{j\in S} v_j + mC \leq (k-1)C + (k-1)mC,$$

so the social welfare is at most

$$\begin{aligned} d_2 + (k-1)C + (k-1)mC &= c_2 + m(m-k)C - W + (k-1)C + (k-1)mC \\ &= c_2 + m(m-1)C + (k-1)C - W \\ &\leq c_2 + m(m-1)C + (m-1)C - W \\ &= c_2 + (m+1)(m-1)C - W \\ &= c_2 + (m^2-1)C - W \\ &< m^2C - W \quad \text{because } C > c_2. \end{aligned}$$

However, any allocation for which player 1 gets k items and player 2 gets $m-k$ items has social welfare of at least $k \cdot mC + (m-k)mC - \sum_j v_2^j = m^2C - W$. So any allocation in which player 1 gets k items and player 2 gets $m-k$ items has higher social welfare than one in which player 1 gets fewer than k items.

Now, suppose by way of contradiction that player 1 receives more than k items in an optimal allocation. Then player 1 has value at most $d_1 = c_1 + mkC$. Player 2 receives a set S of at most $m-(k+1)$ items, so player 2 has value at most

$$\begin{aligned} \sum_{j\in S} w_2^j &= \sum_{j\in S} mC - v_2^j \\ &\leq \sum_{i\in S} mC \\ &\leq (m-k-1)mC \end{aligned}$$

so the social welfare is at most

$$\begin{aligned} d_1 + (m-k-1)mC &= c_1 + mkC + (m-k-1)mC \\ &= m^2C + c_1 - mC \\ &< m^cC + C - mC \quad \text{because } c_1 < C \\ &= m^2C - (m-1)C. \end{aligned}$$

For $m > 1$, $(m-1)C > W$, so

$$m^2C - (m-1)C < m^2C - W.$$

This is again less than the lower bound of $m^2C - W$ for giving k items to player 1 and $m-k$ items to player 2, completing the proof that player 1 gets k items and player 2 gets $m-k$ in any allocation which maximizes the social welfare. $\square$

Lemma 5.24. *A public project has social welfare at least V iff the auction produced using Reduction 5.3 has social welfare at least $V + m^2C - W$.*

Proof. Suppose the original public project has social welfare V. Let S be a set of k items achieving this social welfare, so that $V = v_1(S) + v_2(S)$. Give S to player 1, and $S^C = [m]\backslash S$ to player 2. In the auction allocation described above, player 1 will have value

$$\begin{aligned}
w_1(S) &= \min\left(\sum_{j\in S} w_1^j, d_1\right) \\
&= \min\left(\sum_{j\in S}(v_i^j + mB), c_1 + mkB\right) \\
&= \min\left(mkB + \sum_{j\in S} v_i^j, c_1 + mkB\right) \\
&= \min\left(\sum_{j\in S} v_1^j, c_1\right) + mkB \\
&= v_1(S) + mkB
\end{aligned}$$

for S and player 2 will have value

$$\begin{aligned}
w_2(S^C) &= \min\left(\sum_{j\in S^C} w_2^j, d_2\right) \\
&= \min\left(\sum_{j\in S^C} mC - v_2^j, c_2 + m(m-k)C - W\right) \\
&= \min\left(m(m-k)C + \sum_{j\in S^C} -v_2^j, c_2 + m(m-k)C - W\right) \\
&= \min\left(\sum_{j\in S^C} -v_2^j, c_2 - W\right) + m(m-k)C \\
&= \min\left(\left(\sum_j v_2^j - W\right) + \sum_{j\in S^C} -v_2^j, c_2 - W\right) + m(m-k)C \\
&= \min\left(\left(\sum_j v_2^j + \sum_{j\in S^C} -v_2^j\right) - W, c_2 - W\right) + m(m-k)C \\
&= \min\left(\sum_j v_2^j - \sum_{j\in S^C} v_2^j, c_2\right) - W + m(m-k)C \\
&= \min\left(\sum_{j\in S} v_2^j, c_2\right) - W + m(m-k)C \\
&= v_2(S) - W + m(m-k)C
\end{aligned}$$

for S^C. So the social welfare of (S, S^C) is

$$\begin{aligned}
w_1(S) + w_2(S^C) &= v_1(S) + mkC + v_2(S) - W + m(m-k)C \\
&= v_1(S) + v_2(S) + m(m-k+k)C - W \\
&= V + m^2C - W.
\end{aligned}$$

So if there is an allocation S with social welfare V in the original public project, there is an allocation in the auction produced by the reduction with social welfare $V + m^2C - W$.

Now, suppose that there is an allocation in the auction produced by the reduction with social welfare $V + m^2C - W$. By Lemma 5.23, we know that player 1 gets exactly k items and player 2 gets $m-k$ items. Let S be the set allocated to player 1 and $S^C = [m]\backslash S$ be the set allocated to player 2. We can assume that all items are allocated, as this will only increase the welfare. As shown above, player 1 has value $w_1(S) = v_1(S) + mkC$ and player 2 has value $w_2(S^C) = v_2(S) + m(m-k)C - W$, for a total social welfare of $w_1(S) + w_2(S^C) = v_1(S) + v_2(S) + m^2C - W$. So if the auction resulting from the reduction has social welfare $V + m^2C - W$, then the set S of items allocated to player

1 satisfies $v_1(S) + v_2(S) = V$. Therefore, the original public project has social welfare at least V, completing the proof. $\square$

Now we need only see that the *kdem* oracle on the original public project can be used to answer oracle queries for *dem* in the auction resulting from the reduction in polynomial time.

Lemma 5.25. *The kdem oracle an instance of* ${PC_2}^{kdem}$ *can be used to answer dem queries in the* ${PC_2}^{dem}$ *instance resulting from Reduction 5.3 in polynomial time.*

Proof. Consider the result S of the *dem* query. We will examine 6 cases, 3 for each player.

Case 1a: *dem* returns a set S for player 1, $|S| > k$. In this case, the value to player 1 of any set of size $k+1$ or more is at least $(k+1)mC > d_1$, so any set of size $k+1$ of more will give the same value. So to maximize the utility, S must be the $k+1$ items of lowest price. If there are more than $k+1$ items of negative price, S is all items of negative price.

Case 1b: *dem* returns a set S for player 1, $|S| < k$. In this case, the value to player 1 of any set of size $k-1$ or less is at most $(k-1)(C+mC) = kmC - mC + kC - C < kmC < d_1$. So the budget is not reached. Thus, query result is simply the at most $k-1$ items of highest non-negative utility.

Case 1c: *dem* returns a set S for player 1, $|S| = k$. In this case, we have already seen that player 1's utility is $\min\left(\sum_{j\in S} v_1^j, c_1\right) + mkC$, so the query result is a set of size k maximizing $\min\left(\sum_{j\in S} v_1^j, c_1\right) - \sum_{j\in S} p_j$, which is exactly what the *kdem* oracle gives us as the result for player 1 with prices $p_1, \ldots, p_k$.

Case 2a: *dem* returns a set S for player 2, $|S| > m-k$. In this case,

$$\begin{aligned}
\sum_{j\in S} w_2^j &= \sum_{j\in S}(mC - v_2^j) \\
&\geq (m-k+1)mC - \sum_{j\in S} v_2^j \\
&\geq (m-k+1)mC - W \\
&> c_2 + (m-k)mC - W \quad \text{because } mC > c_2 \\
&= d_2,
\end{aligned}$$

so player 2 has value $w_2(S) = \min(\sum_{j\in S} w_2^j, d_2) = d_2$ regardless of the choice of S. So as in case 1a, S is the set of size at least $m-k+1$ with lowest total price.

Case 2b: *dem* returns a set S for player 2, $|S| < m-k$. In this case, the value to player 2 of any set of size $m-k-1$ or less is at most

$$\begin{aligned}\sum_{j\in S} w_2^j &= (mC - v_2^j)\\ &\leq \sum_{j\in S} mC\\ &\leq (m-k-1)mC\\ &= m(m-k)C - mC\\ &< m(m-k)C - W\\ &\leq c_2 + m(m-k)C - W\\ &= d_2,\end{aligned}$$

so the budget is not reached. Thus, as in case 1b, S is simply the at most $m-k-1$ items of highest non-negative utility.

Case 2c: *dem* returns a set S for player 2, $|S| = m-k$. We have already seen that player 2's utility for S is $\min\left(\sum_{j\in S^C} v_2^j, c_2\right) + m(m-k)C - W$. So we need only find a set S of size k maximizing $\min\left(\sum_{j\in S^C} v_2^j, c_2\right) - \sum_{j\in S} p_i$, as $m(m-k)C - W$ does not depend on S. If we make a query to *kdem* with prices $-p_1, \ldots, -p_m$ we get a set T maximizing

$$\min\left(\sum_{j\in T} v_2^j, c_2\right) - \sum_{j\in T} -p_j = \min\left(\sum_{j\in T} v_2^j, c_2\right) - \sum_{j\in T^C} p_j + \sum_{j\in[m]} p_j,$$

which in turn maximizes $\min\left(\sum_{j\in T} v_2^j, c_2\right) - \sum_{j\in T^C} p_j$, as $\sum_{i\in[m]} p_i$ does not depend on the choice of T. Setting $S = T^C$, we have found a set maximizing $\min\left(\sum_{j\in S^C} v_2^j, c_2\right) - \sum_{i\in S} p_i$.

So in order to solve a *dem* query, we simply need to compute candidates for each of the 3 cases for each player, then choose the result of highest utility. Each candidate only requires polynomial time to compute given access to the *kdem* oracle for $\mathrm{PC_2}^{kdem}$, so the entire procedure runs in polynomial time. □

Lemma 5.24 shows that we have a valid reduction, and Lemma 5.25 shows that we can simulate the oracle after the reduction, completing the proof of Theorem 5.22.

Chapter 6

Conclusions

We have explored the landscape of VCG-based mechanisms for combinatorial public projects and combinatorial auctions with subadditive valuation functions. We saw that for public projects, hardness to approximate with a maximal-in-range mechanism better than a $\sqrt{m}$ ratio can follow fairly easily from worst-case hardness. The $\sqrt{m}$ can follow so easily from worst-case hardness that some 1-player games become harder to approximate truthfully than under pure computation.

We also saw that the same ideas can be used to show hardness of approximation for VCG-based mechanisms for combinatorial auctions. These results do not follow quite as easily, however, as auctions are not as amenable to the study of truthfulness as public projects. Determining the complexity of VCG-based mechanisms for auctions with fractionally-subadditive and coverage valuations remains an open problem.

Most importantly, we developed a framework for the study of mechanism design problems which allows player computation to be taken into account. Within this framework, we were able to replicate most of the maximal-in-range hardness results for combinatorial public projects, as well as many of the NP-hardness results for both auctions and public projects. This allowed us to demonstrate that these results are very robust and not just artifacts of the definitions used. How this framework can be applied to other problems in the auction and public project domains, as well as other domains entirely remain interesting open problems.

Bibliography

[1] Sanjeev Arora and Carsten Lund. Hardness of approximations. In *Approximation algorithms for NP-hard problems*, pages 399–446. PWS Publishing Co., 1997.

[2] Sushil Bikhchandani, Shurojit Chatterji, Ron Lavi, Ahuva Mualem, Noam Nisan, and Arunava Sen. Weak monotonicity characterizes deterministic dominant-strategy implementation. *Econometrica*, 74(4):1109–1132, 2006.

[3] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[4] Liad Blumrosen and Noam Nisan. On the computational power of iterative auctions. In John Riedl, Michael J. Kearns, and Michael K. Reiter, editors, *ACM Conference on Electronic Commerce*, pages 29–43. ACM, 2005.

[5] Liad Blumrosen and Noam Nisan. Combinatorial auctions. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 267–299. Cambridge University Press, 2007.

[6] Béla Bollobás. Edge coloring. In S. Axler, F.W. Gehring, and K.A. Ribet, editors, *Modern Graph Theory*, pages 152–154. Springer, 1998.

[7] Dave Buchfuhrer, Shaddin Dughmi, Hu Fu, Robert Kleinberg, Elchanan Mossel, Christos H. Papadimitriou, Michael Schapira, Yaron Singer, and Christopher Umans. Inapproximability for VCG-based combinatorial auctions. *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 518–536, 2010.

[8] Dave Buchfuhrer, Michael Schapira, and Yaron Singer. Computation and incentives in combinatorial public projects. *Proceedings of the 11th ACM Conference on Electronic Commerce (EC)*, pages 33–42, 2010.

[9] Shuchi Chawla, Jason D. Hartline, and Robert Kleinberg. Algorithmic pricing via virtual valuations. *Proceedings of the 8th ACM Conference on Electronic Commerce (EC)*, pages 243–251, 2007.

[10] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 2:17–33, 1971.

[11] Peerapong Dhangwatnotai, Tim Roughgarden, and Qiqi Yan. Revenue maximization with a single sample. *Proceedings 11th ACM Conference on Electronic Commerce (EC)*, pages 129–138, 2010.

[12] Shahar Dobzinski. An impossibility result for truthful combinatorial auctions with submodular valuations. *Proceedings of the 43rd annual ACM Symposium on Theory of Computing (STOC)*, 2011. (To appear).

[13] Shahar Dobzinski and Shaddin Dughmi. On the power of randomization in algorithmic mechanism design. *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 505–514, 2009.

[14] Shahar Dobzinski and Noam Nisan. Limitations of VCG-based mechanisms. *Proceedings of the 39th annual ACM symposium on Theory of Computing (STOC)*, pages 338–344, 2007.

[15] Shahar Dobzinski, Noam Nisan, and Michael Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. In *STOC*, 2005.

[16] Shahar Dobzinski, Noam Nisan, and Michael Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. *Mathematics of Operations Research*, 35(1):1–13, 2010.

[17] Shahar Dobzinski and Michael Schapira. An improved approximation algorithm for combinatorial auctions with submodular bidders. *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1064–1073, 2006.

[18] Shahar Dobzinski and Mukund Sundararajan. On characterizations of truthful mechanisms for combinatorial auctions and scheduling. *Proceedings of the 9th ACM conference on Electronic Commerce (EC)*, pages 38–47, 2008.

[19] Shaddin Dughmi and Tim Roughgarden. Black-box randomized reductions in algorithmic mechanism design. *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 775–784, 2010.

[20] Shaddin Dughmi, Tim Roughgarden, and Qiqi Yan. From convex optimization to randomized mechanisms: Toward optimal combinatorial auctions. *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, 2011. (To appear).

[21] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[22] Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing*, 39(1):122–142, 2009.

[23] Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 667–676, 2006.

[24] M. R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. *Proceedings of the sixth annual ACM symposium on Theory of computing (STOC)*, pages 47–63, 1974.

[25] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[26] Jason D. Hartline and Tim Roughgarden. Optimal mechanism design and money burning. *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 75–84, 2008.

[27] Jason D. Hartline and Tim Roughgarden. Simple versus optimal mechanisms. *Proceedings of the 10th ACM Conference on Electronic Commerce (EC)*, pages 225–234, 2009.

[28] Subhash Khot, Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Inapproximability results for combinatorial auctions with submodular utility functions. *Algorithmica*, 52(1):3–18, 2008.

[29] Ron Lavi, Ahuva Mu'alem, and Noam Nisan. Towards a characterization of truthful combinatorial auctions. *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, page 574, 2003.

[30] Benny Lehmann, Daniel J. Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.

[31] R. B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

[32] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions ii. *Math. Programming Study 8*, pages 73–87, 1978.

[33] Noam Nisan. Bidding and allocation in combinatorial auctions. In *ACM Conference on Electronic Commerce*, pages 1–12, 2000.

[34] Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. *Journal of Artificial Intelligence Research (JAIR)*, 29:19–47, 2007.

[35] Noam Nisan and Ilya Segal. The communication requirements of efficient allocations and supporting prices. *Journal of Economic Theory (JET)*, 129(1):192–224, 2006.

[36] Christos Papadimitriou, Michael Schapira, and Yaron Singer. On the hardness of being truthful. *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008.

[37] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization; Algorithms and Complexity.* Dover Publications, 1998.

[38] Kevin Roberts. The characterization of implementable choice rules. In Jean-Jacques Laffont, editor, *Aggregation and Revelation of Preferences. Papers presented at the 1st European Summer Workshop of the Econometric Society*, pages 321–349. North-Holland, 1979.

[39] Amir Ronen. On approximating optimal auctions. *Proceedings of the 3rd ACM Conference on Electronic Commerce (EC)*, pages 11–17, 2001.

[40] Amir Ronen and Amin Saberi. On the hardness of optimal auctions. *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS)*, pages 396–405, 2002.

[41] Michael Schapira and Yaron Singer. Inapproximability of combinatorial public projects. In Christos H. Papadimitriou and Shuzhong Zhang, editors, *WINE*, volume 5385 of *Lecture Notes in Computer Science*, pages 351–361. Springer, 2008.

[42] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[43] V.G. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz*, 3:25–30, 1964. In Russian.

[44] Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–74, 2008.