Single-Value Combinatorial Auctions and Algorithmic Implementation in Undominated Strategies *

Moshe Babaioff† Ron Lavi‡ Elan Pavlov§

Abstract

In this paper we are interested in general techniques for designing mechanisms that approximate the social welfare in the presence of selfish rational behavior. We demonstrate our results in the setting of Combinatorial Auctions (CA). Our first result is a general deterministic technique to decouple the algorithmic allocation problem from the strategic aspects, by a procedure that converts any *algorithm* to a dominant-strategy ascending *mechanism*. This technique works for any single value domain, in which each agent has the same value for each desired outcome, and this value is the only private information. In particular, for "single-value CAs", where each player desires any one of several different bundles but has the same value for each of them, our technique converts any approximation algorithm to a dominant strategy mechanism that almost preserves the original approximation ratio. Our second result provides the first computationally efficient deterministic mechanism for the case of single-value multi-minded bidders (with private value and private desired bundles). The mechanism achieves an approximation to the social welfare which is close to the best possible in polynomial time (unless P=NP). This mechanism is an *algorithmic implementation in undominated strategies*, a notion that we define and justify, and is of independent interest.

1 Introduction

Algorithmic Mechanism Design [29] studies the design of computationally efficient algorithms under the assumption that the input is distributed among a set of rational selfish players. One successful approach to this problem is of designing *truthful* mechanisms, in which a player always maximizes his utility by simply revealing his true input. The classic VCG [10, 18, 33] scheme from the economic literature that converts algorithms to truthful mechanisms is computationally infeasible for most interesting domains, including Combinatorial Auctions. Thus the important task of designing computationally-efficient mechanisms for combinatorial auctions with selfish agents should be addressed by other means.

In this context, this paper makes several contributions. We first describe a deterministic technique for converting arbitrary algorithms into strategic mechanisms. We then describe how this technique may be applied in the setting of single-value multi-minded bidders, for which no non-trivial deterministic mechanism was previously known. This mechanism is an *algorithmic implementation in undominated strategies*, a new notion that we define and discuss. We believe that this notion is of independent interest.

1.1 General methods for creating dominant-strategy mechanisms

We first consider the issue of general methods for creating dominant-strategy mechanisms. In principle, we would like to decouple the algorithmic problem (solving or approximating the optimum) from the strategic problem (eliciting the privately known input). Such a decoupling would allow us to utilize the full power of

*An extended abstract of this work has appeared in SODA'06.

†moshe@microsoft.com. Microsoft Research - Silicon Valley. 1065 La Avenida, Mountain View, CA 94043

‡ronlavi@ie.technion.ac.il. Faculty of Industrial Engineering and Management, The Technion – Israel Institute of Technology

§elan@MIT.EDU. Media Lab, Massachusetts Institute of Technology

approximation algorithms without reference to the strategic properties. Several papers suggest methods to convert algorithms to truthful mechanisms for several different problem domains (we discuss these papers in details at Section 1.4 below). A common drawback of all these methods is the use of randomization. Randomized mechanisms suffer from several drawbacks, as compared to deterministic mechanisms, e.g. the approximation guarantees are only in expectation, and it is **not** possible (as with approximation algorithms) to repeat the computation and take the best outcome, in order to approach the expected guarantee, as this will damage the truthfulness properties.[1] Our first contribution is a *deterministic* technique that converts any given *algorithm* into a dominant-strategy ascending *mechanism* for any one-parameter Combinatorial Auction (CA) domain. For example, for combinatorial auctions with "known" single-minded bidders [26], where each player desires one specific *publicly known* subset of items for a private value of $\bar{v}_i$, our method deterministically converts any c-approximation of the social welfare to a dominant-strategy mechanism with $O(log(\bar{v}_{max}) \cdot c)$-approximation, where $\bar{v}_{max} = \max_i \bar{v}_i$.[2] If players are multi-minded, with publicly known desired bundles and the same private value for all desired bundles, our technique yields a dominant-strategy mechanism with $O(log^2(\bar{v}_{max}) \cdot c)$-approximation.

The conversion method takes an approximation algorithm that receives as input a vector of assumed players' values (bids), and then utilizes an ascending procedure as a wrapper to iteratively output a set of provisional winners. Every provisional loser is then required to either double his bid (reported value) or to permanently retire. This is repeated until all losers decide to retire. Each winner pays his final bid. The analysis shows that the number of iterations is surprisingly low (only $O(log(\bar{v}_{max}))$), and this by itself is enough to bound the approximation loss, relative to the original approximation of the algorithm.

While this method cannot improve the existing results for single-minded combinatorial auctions, as these already obtain the best possible $\sqrt{m}$-approximation [25][3], it is fruitful for special cases in which some further structure on the bundles is assumed, allowing to break the $\sqrt{m}$ lower bound (see e.g. [1]). It is additionally fruitful if one wishes to use the many well-known heuristics that work well in practice but are not truthful to begin with. Since our method converts the given algorithm to an ascending auction, i.e. players compete by placing bids, and winners pay their last offer, it has a more realistic structure than standard direct revelation mechanisms, in which each player simply reveals his value. In particular, it has the advantage that winners do not reveal their true values. It has been argued many times (see e.g. [31]) that such indirect mechanisms should be preferred over the more common "direct revelation" mechanisms.

1.2 Single-value Combinatorial Auctions

The next case we consider is when players are multi-minded, and their desired bundles are *private information*. Specifically, we look at the class of valuations in which a player desires several different bundles, all for the same value. We call these players "single-value" players. Here, both the player's value and his collection of desired bundles are assumed to be private information. The best deterministic truthful approximation guarantee for general combinatorial auctions is $O(\frac{m}{\sqrt{\log m}})$[19], and no better bound is known even for single-value players. This is in sharp contrast to the best deterministic approximation that is achievable *without* strategic considerations, which is $\sqrt{m}$.[4]

The model of single-value players does not fall into the family of one-parameter domains defined by Archer and Tardos [2] since the desired bundles are not public information, and hence value monotonicity by itself is no longer sufficient for dominant strategy implementation. In [4] we discuss general single-value domains and the sufficient conditions for dominant strategy implementations in such domains. We emphasize that the model of single-value players is *significantly* richer than the model of single-minded players, as a single-value

[1] In addition, randomized methods sometimes assume that players are neutral to risk; a detailed discussion can be found in [14].

[2] This assumes that $v_i \geq 1$ for all i; a normalization that is equivalent to the assumption that values are bounded away from 0 by some known constant. We also note that our technique does not assume that $\bar{v}_{max}$ is known a-priori.

[3] For any $\epsilon > 0$, an approximation of $m^{1/2-\epsilon}$ can not be computed in polynomial time, unless P=NP [25, 34], where m is the number of goods.

[4] There exist several other special cases of combinatorial auctions for which deterministic mechanisms achieve good approximation guarantees, e.g. in [6]. We discuss these papers in details at Section 1.4 below.

player may be multi-minded, and may desire even an exponential number of bundles that are not contained one in the other. Even if the players' values were fixed and known, single-value valuations might require an exponential representation in the number of goods [27]. Nisan and Segal [27] show that, for single-value domains, even with no computational constraints, exponential communication in the number of goods m is required in order to get approximation of $m^{1/2-\epsilon}$, for any fixed $\epsilon > 0$. Thus, the best approximation bound we can hope for is $\sqrt{m}$, and this is indeed achievable if one ignores the strategic issue.

A natural single-value multi-minded CA domain is the Edge Disjoint Paths (EDP) problem: given a graph, (graph edges may be thought of as the items for sale), each player i obtains a value $\bar{v}_i$ from receiving any path from an unknown source node s_i to an unknown target node t_i. The algorithm is required to allocate edge disjoint paths to maximize the sum of values of players that receive a desired path. In this problem, players are naturally single valued (a player obtains the same value from any source-target path), but are **not** single minded. Even for this special case, no deterministic mechanism with approximation guarantees that approach $\sqrt{m}$ is currently known.

Our second contribution is a deterministic mechanism for "unknown" single-value players, with $O(log^2(\bar{v}_{max}) \cdot \sqrt{m})$ approximation. In particular, this applies to the EDP problem.

The mechanism for "unknown" single-value players is composed of the above wrapper technique, on top of a specific algorithm. As the collection of desired bundles is assumed to be private, we maintain an "active bundle" for each player. At the beginning of the auction each active bundle contains all items, and at the end of the auction, each winner receives his last active bundle. In addition to increasing their bids, as before, losing players may also shrink their active bundle, by this focusing on one of their desired bundles.

This additional building block of active bundles forces a transition from *dominant* strategies to *undominated* strategies: the mechanism admits more than one reasonable strategic choice for a rational player (and not just one as implied by dominant strategies), but the approximation is guaranteed for *any* rational choices that the players make. Intuitively, if a player shrinks his active bundle, he loses the ability to win some of his desired bundles, as any subset he will eventually win must be contained in his current active bundle, and this cannot expand. Therefore, a losing player faces the trade-off of increasing his price, or shrinking his active bundle. A mechanism with dominant strategies implies that the player has a specific optimal choice, no matter what the other players choose. In our mechanism there is no such "clear cut" decision for a player. The crucial point in the analysis shows that such a unique optimal choice is also not necessary: to guarantee the approximation, all we need is that a *retiring* player bids up to his true value *and* reveals one of his true desired subsets, the exact "path" is not important!

1.3 Algorithmic implementation in undominated strategies

This leads us to a new solution concept for strategic mechanisms. Requiring that players have dominant strategies limits the family of allocation algorithms that can be used [2, 4, 21]. Clearly, this requirement is not the real essence, but rather a tool to obtain the underlying goal: reaching approximately optimal outcomes in the presence of selfish behavior. Here, we obtain this goal although we allow the mechanisms to leave in several "reasonable" strategies for the players to choose from. The basic assumption still remains that a player prefers not to choose a *dominated* strategy, since this means that there exists another strategy that *always* performs at least as well. We say that a mechanism M is an *algorithmic implementation of a c-approximation (in undominated strategies)* if there exists a set of strategies, D, such that (1) M obtains a c-approximation for any combination of strategies from D, in polynomial time, and (2) for any strategy that does not belong to D, there exists a strategy in D that dominates it (D stands for Dominating strategies). Furthermore, we require that this "improvement step" can be computed in polynomial time.

Thus, while we leave some uncertainty on the game-theoretic side, we compensate by strengthening the algorithmic analysis, showing that the mechanism performs well for *any combination of strategies from the set D*. This moves some burden from the game theoretic constraints on mechanism design to the algorithmic part in mechanism design.

We argue that this concept captures all the truly important ingredients of the dominant strategies notion: First, the only assumption is that a player is willing to replace any chosen strategy with a strategy that dominates it. Indeed, this guarantees at least the same utility (even in the worst-case), and can be done in

polynomial time. In addition, as in dominant strategies, our notion does not require any form of coordination among the players (unlike Nash equilibrium), or that players have any assumptions on the rationality behavior of the others (unlike "iterative deletion of dominated strategies"). Our mechanisms also ensure that the resulting utility is non-negative, thus players have no risk in participating. Algorithmic implementation differs from truthfulness in one main aspect: it cannot predict what strategies from the set D the players will choose to play, as there is no single strategy that dominates all the rest. In fact, it is not straight-froward for a player to choose a strategy from D, and it may well be that a player regrets his choice in retrospect, realizing that another strategy from D would have performed better. However we are guaranteed that players will choose *some* strategies from D, and this is enough to ensure the approximation, which is what the mechanism designer cares about. In a companion paper [5], we present an additional technique for algorithmic implementation, completely different than our iterative wrapper method. This further demonstrates its potential usefulness.

As we explain in Section 5, the notion of algorithmic implementation generalizes the notion of implementation in undominated strategies. This is a well-known game-theoretic concept, but very few positive results have been achieved by using it (one such positive example is [20]). To the best of our knowledge, we are the first to adapt it to the context of algorithmic mechanism design and to demonstrate its usefulness.

1.4 Related work

One of the first papers to consider the problem of constructing computationally-efficient and truthful combinatorial auctions with a good approximation guarantee is the paper of Lehmann, O'Callaghan, and Shoham [25]. They consider the case of unknown single-minded bidders, and provide a deterministic truthful $O(\sqrt{m})$-approximation mechanism. Two randomized mechanisms that obtain the best possible approximation ratio for the general problem are known: Lavi and Swamy [23] give a randomized mechanism that is truthful in expectation, and provides an $O(\sqrt{m})$-approximation. Their mechanism gives tight bounds for several additional problem domains, including the domain of multi-unit combinatorial auctions, and some special cases of the EDP problem that admit approximation guarantees better than $\sqrt{m}$. The mechanism of [23] obtains the approximation guarantee only in expectation, hence for some coin tosses the outcome can be far from optimal. Their solution also assumes that players are risk-neutral. Dobzinski, Nisan, and Schapira [14] and Dobzinski [11] improve these two problematic points. They give a randomized mechanism that provides an $O(\sqrt{m})$-approximation with high probability, and is universally-truthful (truthful for any realization of the coin tosses).

Despite many efforts to complete this line of papers and to provide a deterministic truthful mechanism with a tight approximation guarantee, the problem is still open. The best known approximation of a truthful deterministic mechanism is $O(\frac{m}{\sqrt{\log m}})$ [19], while the best possible approximation without truthfulness is $\sqrt{m}$. Bartal, Gonen, and Nisan [6] were the first to construct a truthful deterministic mechanism with a significantly improved approximation guarantee for a multi-dimensional combinatorial auction domain, assuming that each item is supplied in several copies, while each player desires at most a small fraction of the total number of copies. For example, if there are $B \geq 3$ copies of each item and each player desires at most one copy from each item then their mechanism is a deterministic and truthful $O(B \cdot m^{\frac{1}{B-2}})$ approximation. When the number of copies is logarithmic in the number of items then the approximation ratio becomes logarithmic in the number of items (i.e. if $B = O(\log m)$ then $O(B \cdot m^{\frac{1}{B-2}}) = O(\log m)$). Dobzinski, Nisan, and Schapira [13] study another special case, were the valuation functions of the players are assumed to be sub-additive, and provide a truthful deterministic mechanism with $O(\sqrt{m})$ approximation. Briest, Krysta, and Vöcking [9] discuss several single-dimensional domains, including the EDP problem under the assumption that the source and target nodes of each player are public knowledge.

To tackle the problem from its other end, several papers have tried to pin-point the impossibilities of deterministic mechanism design. Lavi, Mu'alem, and Nisan [21] show that a certain family of algorithms that are "strongly monotone" cannot yield a good deterministic and truthful approximation to the general combinatorial auctions problems, and Dobzinski and Sundararajan [15] give slightly different conditions that yield the same impossibility. Dobzinski and Nisan [12] show that VCG-based mechanisms cannot yield

an approximation factor better than $\Omega(\min(n, m^{1/6}))$ when valuations are sub-additive. Given all this, it is clear that despite the many efforts of the community, the question of constructing deterministic combinatorial auctions is far from being solved.

The model of combinatorial auctions also presents pure algorithmic questions that are interesting and challenging, and the interested reader is referred to the book chapter [8] for further details. In addition to the theoretical algorithms that were constructed for combinatorial auctions, many heuristics that work extremely well in practice have also been designed, see for example the papers [16, 17, 28, 32], and the references therein. The book [30] contains many more details on this subject.

As motivated above, an independent line of study is the question of constructing methods to convert algorithms to mechanisms. This will yield the decoupling of algorithmic constructions from strategic considerations, and will enable to convert existing algorithms to new mechanisms, in a black-box fashion. Several papers have designed such conversion methods, for various problem domains. Awerbuch, Azar, and Meyerson [3] give such a method for a family of online settings with one-parameter agents. Lavi and Swamy [23] base their results for combinatorial auctions on such a conversion method, that operates whenever the underlying algorithm bounds the integrality gap of the LP relaxation of the problem. Lavi and Swamy [24] give a different conversion method, for the problem domain of job-scheduling, assuming processing times that can be either "low" or "high". All these conversion techniques are randomized, and in this paper we give the first deterministic conversion method.

1.5 Organization of the paper

The rest of the paper is organized as follows. We formally describe the model, and give basic definitions, in Section 2. Section 3 describes the wrapper technique for known single-minded players. In Section 4 we move to the general single-value players case. Section 5 describes the new notion of algorithmic implementation in undominated strategies, and in Section 6 we fully analyze the wrapper technique according to this notion. In Appendix A we show how to use the general wrapper in order to convert algorithms to mechanisms for known multi-minded players, complementing our results for known single-minded players, and in Appendix B we generalize our results to the case of Multi-Unit Combinatorial Auctions.

2 Model and Definitions

2.1 Combinatorial Auctions and Single-Value Players

We consider mechanisms for the extensively studied domain of Combinatorial Auctions (the textbook [30] provides a detailed survey). In a Combinatorial Auction (CA), we need to allocate a set Ω of m items to n players, where an "allocation" is a tuple of disjoint subsets $s_1, ..., s_n$ of items (i.e. player i gets the items in s_i). Note that some of the items may be left unallocated. Players have values for *subsets* of items, and $\bar{v}_i(s)$ denotes player i's value for the subset of items s. These value functions are assumed to be monotone (this is often called the "free disposal" property), i.e. $\bar{v}_i(s) \subseteq \bar{v}_i(t)$ for every $s \subseteq t$, and normalized ($\bar{v}_i(\emptyset) = 0$). We denote by V_i the set of all valid valuation functions of player i. The goal is to (approximately) maximize the sum of true values of players for the subsets they receive (the "social welfare"), i.e. to find an allocation $s_1, ..., s_n$ that maximizes $\sum_i \bar{v}_i(s_i)$.

We mainly consider several special cases of CAs, as follows. In the case of "known" single-minded players [26], each player desires one specific publicly known subset of items, for a private value of $\bar{v}_i$. For the main result of the paper, we suggest to look at a more general class of valuations, in which a player desires several different bundles, all for the same value:

Definition 1 (Single-Value players)**.** *Player i is a* single-value *(multi-minded) player if there exists a real value $\bar{v}_i > 0$ such that for any bundle s, $\bar{v}_i(s) \in \{0, \bar{v}_i\}$. Both the player's value and his collection of desired bundles are assumed to be private information, known only to the player himself.*

In other words, the player desires any bundle that belongs to a *collection* of bundles $\bar{S}_i$, each for a value of $\bar{v}_i$. Both the player's value and his collection of desired bundles are assumed to be private information,

known only to the player[5]. We additionally assume that $\bar{v}_i \geq 1$. We denote by $\bar{v}_{max}$ the maximal value of any player (this need not be known a-priori). Since $\bar{S}_i$ may be of size exponential in m, we need to assume that an oracle access is provided. The specific query types that we use are detailed below, in the relevant sections.

A natural single-value multi-minded CA domain is the Edge Disjoint Paths (EDP) problem: given a graph, (graph edges may be thought of as the items for sale), each player i obtains a value $\bar{v}_i$ from receiving any path from a source node s_i to a target node t_i. The algorithm is required to allocate edge disjoint paths to maximize the sum of values of players that receive a desired path. In this model, players are naturally single valued (a player obtains the same value from any source-target path), but are **not** single minded, and in general the source-target pair of a player is **not** publicly known.

2.2 Strategic issues and mechanism design

The literature on Algorithmic Mechanism Design [29] studies algorithmic design when players behave strategically: the "input" is held by the players, and they can "lie" in order to increase their utility. To cope with this, the literature suggests to construct a *mechanism*, as follows. A mechanism defines a set of actions $\mathcal{A}_i$ for every player i. Let $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$. Each player is required to choose an action, and, based on the actions of the players, the mechanism chooses an outcome, and a payment for every player. The outcome is determined by the function $\mathcal{O} : \mathcal{A} \rightarrow \{\text{all allocations}\}$, and the payment for every player i is determined by the function $\mathcal{P}_i : \mathcal{A} \rightarrow \Re$. Thus, a mechanism M can be written as a tuple $M = (\mathcal{A}, \mathcal{O}, \mathcal{P})$. Each player chooses an action based on her privately known valuation function. Formally, a player chooses a strategy $st_i : V_i \rightarrow \mathcal{A}_i$ that determines her action as a function of her valuation. The basic assumption is that player i will choose a strategy in order to maximize her *utility*. The utility of a player is assumed to be the player's value that is being obtained from the chosen outcome, minus the payment of the player. Therefore a player aims to choose a strategy $st_i(\cdot)$ that will maximize the term

$$\bar{v}_i(O(st_i(\bar{v}_i), a_{-i})) - \mathcal{P}_i(st_i(\bar{v}_i), a_{-i})$$

for every valuation function $\bar{v}_i(\cdot)$, and for every tuple of actions of the other players, a_{-i}.

The basic goal of the mechanism designer is to design the mechanism such that the players' strategies could be predicted. Based on that, the outcome of the mechanism can be analyzed. The "usual" notion that is being used in the algorithmic mechanism design literature is *dominant strategies*:

Definition 2 (The domination relation)**.** *A strategy st_i dominates another strategy st'_i if for every $\bar{v}_i \in V_i$, and for any possible action of the others $a_{-i} \in \mathcal{A}_{-i}$,*

$$\bar{v}_i(O(st_i(\bar{v}_i), a_{-i})) - \mathcal{P}_i(st_i(\bar{v}_i), a_{-i}) \geq \bar{v}_i(O(st'_i(\bar{v}_i), a_{-i})) - \mathcal{P}_i(st'_i(\bar{v}_i), a_{-i}).$$

In other words, st_i dominates st'_i if the player's utility from st_i is not less than her utility from strategy st'_i, no matter what are the actions of the other players, and what is her valuation (i.e. in the *worst case*).

Definition 3 (A dominant strategy)**.** *A strategy st_i is dominant if it dominates any other strategy st'_i.*

Although we construct some dominant-strategy mechanisms in this paper, the main focus of the paper is a new, novel solution concept that we introduce in Section 5.

3 The Iterative Ascending Wrapper for Known single-minded Players

We start by describing our iterative wrapper technique for domains in which the only private information is the players' values, with a focus on known single-minded CAs. For this domain, our method converts *any*

[5] We also consider the case of "known single-value players", in which the only private information is the value.

given c-approximation to a dominant strategy mechanism with $O(\log(\bar{v}_{max}) \cdot c)$ approximation, showing how to incorporate value monotonicity [6], which is equivalent to truthfulness [26], into any given algorithm.

A formal description of our Iterative Wrapper mechanism for the case of "known single-minded players" is given as Mechanism 1. Informally, suppose that ALG is an algorithmic procedure such that, when given as input a set of player values, outputs a set of winners that is a c-approximation to the social welfare. We assume w.l.o.g[7] that ALG outputs a Pareto efficient outcome (Pareto, in short), i.e. that there does not exist a loser that can be added to the set of winners. The Iterative Wrapper Mechanism is a simple wrapper to ALG: a vector of player values, initialized to $v^0 = \vec{1}$ (line 1), is iteratively handed in as input to ALG, who, in return, outputs a set of winners W_{j+1} (line 3), where j is the iteration number. If the new allocation W_{j+1} has a value lower than the previous allocation W_j, we keep the previous allocation (lines 4-6, using the notation $v(W, v) = \sum_{i \in W} v_i$). Every loser is then required to either double his value or to permanently retire (this is denoted by setting $v_i^j = 0$) (lines 7-9). A retired player will not be able to win any bundle. This is iterated until all non-retired players are declared winners by ALG. These are the winners of the mechanism. Each winner gets his desired bundle and pays his last bid, v_i^J, where J denotes the total number of iterations.

Mechanism 1 The Iterative Wrapper

1: Initialize $j = 0$, $W_0 = \emptyset$, $L_0 = \emptyset$ and $\forall i,\ v_i^0 = 1$.
2: **while** $W_j \cup L_j \neq N$ **do**
3: $\quad W_{j+1} = ALG(v^j)$
4: $\quad$ **if** $v(W_{j+1}, v^j) < v(W_j, v^j)$ **then**
5: $\quad\quad W_{j+1} = W_j$
6: $\quad$ **end if**
7: $\quad$ **for** $i \notin W_{j+1}$ **do**
8: $\quad\quad i$ chooses $v_i^{j+1} = 2v_i^j$ or $v_i^j = 0$ (retire)
9: $\quad$ **end for**
10: $\quad L_{j+1} = \{i \in N | v_i^j = 0\}$, j=j+1
11: **end while**
12: J=j

Output: Each winner $i \in W_J$ receives his bundle and pays his final reported value v_i^J, others receive nothing and pay 0.

Proposition 1. *It is a dominant strategy of any player to increase his reported value when asked, as long as the increased value is lower than his true value $\bar{v}_i$.*

Proof. If player i bids up to some value larger than $\bar{v}_i$ his utility will be non-positive in any case, which can be improved by retiring at the last value not larger than $\bar{v}_i$. If i retires at a value smaller than $\bar{v}_i/2$, then the mechanism ensures that he will not receive any item, hence his utility will be zero, while if i continues to bid and retires at the largest value smaller than $\bar{v}_i$, his utility is non negative. □

This implies that if player i retires at iteration j then $\bar{v}_i/2 \leq v_i^j \leq \bar{v}_i$.

In order to analyze the approximation bounds of this mechanism, the crucial point is that the number of iterations is low, independent of the number of players:

Lemma 1. *If all players are single minded and play their dominant strategy then the number of iterations is at most $2 \cdot \log \bar{v}_{max} + 1$.*

[6] For our domains, an algorithm is value monotone if a winner continues to win when he plays the dominant strategy with respect to a higher value, fixing the strategies of the others.

[7] Any set of winners that is not Pareto efficient can be expanded to a Pareto efficient set in polynomial-time, as follows: go over all losing players, and for each player determine if her bundle intersects some winner's bundle. If not, make the player a winner. This can be done in polynomial-time since players are single-minded, hence this greedy approach works. For non-single-minded players such a Pareto improvement might not be polynomial-time computable.

Proof. Suppose by contradiction that at iteration $2\log\bar{v}_{max}+1$ there exists a loser, i_1, who is willing to double his (reported) value. This implies that there exists another winner, i_2, such that his desired bundle intersects the desired bundle of i_1 (Pareto). Hence in every previous iteration at least one of them was a loser, and doubled his value. Since each player doubles his value at most $\log(\bar{v}_{max})$ times before retiring (in the dominant strategy), the only possibility is that both i_1 and i_2 have value $\bar{v}_{max}$ at iteration $2\log\bar{v}_{max}+1$. But this contradicts the assumption that i_1 is willing to increase his value in this iteration. □

This also implies that if *ALG* has polynomial running time then the Iterative Wrapper has polynomial running time. For the case of single-minded players, not much work is left to show that the mechanism almost preserves the original approximation ratio. We first define some more notation.

Notation: If W is a set of players that can be simultaneously satisfied[8], we define $v(W,v)=\sum_{i\in W}v_i$. For a general set of players X, we define $OPT(X,v)$ to be the maximal $v(W,v)$ over all sets $W\subseteq X$ of players that can be simultaneously satisfied.

Lemma 2. *Suppose that ALG is a c-approximation algorithm, and that the Iterative Wrapper performs J iterations. Then it obtains an* $(2\cdot J\cdot c+1)$*-approximation for known single-minded bidders.*

Proof. For any $1\le j\le J$, let R_j be the set of players that retired at iteration j. Note that $\cup_{j=1}^{J}R_j$ is exactly the set of all losing players, and for any j, $OPT(R_j,v^j)\le c\cdot v(W_j,v^j)\le c\cdot v(W_J,\bar{v})$, where the last inequality holds by step 5 (improvement of the allocation), and $v_i^j\le\bar{v}_i$ for all i when playing dominant strategies. For any player $i\in R_j$ we have that $v_i^j\ge\bar{v}_i/2$, thus $\frac{1}{2}OPT(R_j,\bar{v})\le OPT(R_j,v^j)$, and therefore $OPT(R_j,\bar{v})\le 2\cdot c\cdot v(W_J,\bar{v})$. We can now bound the optimal value of the entire set of players $N=W_J\cup(\cup_{j=1}^{J}R_j)$:

$$OPT(N,\bar{v})=OPT(W_J\cup(\cup_{j=1}^{J}R_j),\bar{v})\le OPT(W_J,\bar{v})+\sum_{j=1}^{J}OPT(R_j,\bar{v})\le v(W_J,\bar{v})+J\cdot 2\cdot c\cdot v(W_J,\bar{v})$$

Thus, $OPT(N,\bar{v})\le(J\cdot 2\cdot c+1)\cdot v(W_J,\bar{v})$, which is exactly what we need to prove. □

These two lemmas give us:

Theorem 1. *Given any polynomial time c-approximation algorithm for single-minded CAs, the Iterative Wrapper obtains an* $O(\log\bar{v}_{max}\cdot c)$*-approximation in dominant strategies for known single-minded players, in polynomial time.*

Proof. By Lemma 1 we have $J\le 2\cdot\log\bar{v}_{max}+1$, and from Lemma 2 we have that the approximation is $(2\cdot J\cdot c+1)$. Combining these two terms, we get that the approximation is $O(\log\bar{v}_{max}\cdot c)$. □

This finishes the analysis of the Iterative Wrapper, for the case of known single-minded players.

In addition to being the framework for our undominated strategies implementation that we describe in the sequel, our construction is the first deterministic technique to convert any algorithm to a dominant strategy mechanism. One way to convert an algorithm for known single-minded players to a mechanism is to define the action space of a player so that each player is required to simply declare a value. The outcome of the mechanism is then the algorithm's output when the input is the declared values. In this case, one would like to construct payment functions so that, in the resulting mechanism, declaring the *true* value will be a dominant strategy. It is known that such payment functions can be constructed if and only if the algorithm is value-monotonic[9] [26]. Note that the Iterative Wrapper is indeed value-monotonic (in the sense of Footnote 6), even if the original algorithm was not value-monotonic to begin with. Most useful approximation techniques (e.g. LP rounding, or taking the maximum over several algorithms) are not value-monotonic, and our method converts such techniques to dominant strategy mechanisms (with some additional

[8] A player is satisfied by some allocation if he receives one of his desired bundles in that allocation.

[9] An algorithm is value-monotonic if, whenever a winner increases his value, he remains a winner.

approximation loss). The method can also be used to create dominant-strategy mechanisms from algorithms based on heuristics (algorithms which perform well on many instances, but with no worse case guarantee). Many such heuristics were previously studied for CAs (e.g. [28, 32, 17]) but no strategic construction of those was previously known. In particular, it enables us to pick the best result over a polynomial number of heuristics and approximation algorithms (by a MAX operator), enjoying the benefits of all methods (with a small approximation loss), while having dominant strategies.

We make few additional technical remarks that shed more light on the technical analysis. We first note that, in order to optimize the constants, one needs to multiply a loser's value by e instead of by 2. For ease of notation we use the constant 2. We also point out that, in principle, if $\log(\bar{v}_{max}) \cdot c > \min(m, n)$ then the iterative wrapper can output a result which is far from optimal by a ratio even larger than $\min(m, n)$. However, this extreme case can be avoided by adding the following modification to the algorithmic procedure ALG: if there exists a single player with value larger than the value of the computed allocation, the resulting allocation should be any Pareto allocation that contains this player. Such a modification keeps the approximation and ensures that the sum of values of the winners is larger than the value of any single player. In turn, this implies that the approximation ratio will not exceed $\min(m, n)$. [10]

Our analysis of the Iterative Wrapper for known single-minded players is tight, up to a constant. In Appendix C we prove that for any constant $c \geq 1$ and large enough $\bar{v}_{max}$ there exists a c-approximation algorithm such that the Iterative Wrapper only gets $\Omega(c \cdot \log(\bar{v}_{max}))$-approximation.

For multi-minded players, the arguments that were used to show the approximation fail. In particular, the number of iterations of the basic Wrapper mechanism may become large, dependent on the number of players:

Proposition 2. *There exists a scenario with n double-minded players for which the basic Wrapper performs exactly n iterations.*

Proof. Assume that there are $n-1$ goods. Player $i \in \{1, ..., n-1\}$ is double minded with bundles $\{i\}$ and $\{(i+1) \ mod \ (n-1)\}$. Player n wants the bundle $\{1\}$. All players have values equal to 2. Suppose that the algorithmic procedure we use is the optimal algorithm. The important point to notice is that any $n-1$ players can be satisfied jointly, while the n players cannot be satisfied together. Therefore the course of the wrapper will be as follows: at each iteration, all the players with a current value of 2 will be winners, and exactly one of the players with a current value of 1 will be the only loser. That player will then double his value. After n iterations, when all players have doubled their values, one of them will be the loser, at which point he will retire and the auction will end. Hence we have exactly n iterations. □

We note, however, that we do not know whether the mechanism achieves a bad approximation for multi-minded players – only that our arguments for the single-minded case do not follow through for the multi-minded case. For multi-minded players we therefore continue in a slightly different trail, that will also enable the shift to the "unknown" case.

4 An Approximation Mechanism for Single-Value Players

The General Iterative Wrapper mechanism for single-value players (formally defined below as Mechanism 2) is an adaptation of the Iterative Wrapper (Mechanism 1 of Section 3), for the case where each player desires many different bundles, and his set of desired bundles is private information. Given a procedure that outputs an approximately optimal allocation, we use the iterative wrapper in order to turn it to an algorithmic implementation in undominated strategies. Since players are now multi-minded, and we do not know their desired bundles, we introduce two differences: First, a player has an "active bundle", s_i^j, which contains all possible items that a player may receive at the end of the auction. It is initialized to the entire set of items, Ω, and is gradually shrunken by the given procedure. Second, the given procedure now interacts with the players in order to decide how to shrink the active bundle, in each step j, from s_i^j to s_i^{j+1}.

[10] The optimal allocation can choose at most $\min(m, n)$ winners. Therefore its value is at most $\min(m, n)$ times the largest value. Since the sum of values of the winners in the auction is at least the largest value, the ratio will be at most $\min(m, n)$.

Mechanism 2 The General Iterative Wrapper

initialize $j = 0$, $W_j = \emptyset$, $L_j = \emptyset$. initialize $v_i^0 = 1$ and $s_i^0 = \Omega \ \forall i$
while $W_j \cup L_j \neq N$ **do**
  $(W_{j+1}, s^{j+1}) \leftarrow 1 - CA - SUB(v^j, s^j, W_j)$. \\ 1-CA-SUB is defined below as Procedure 3
  for $i \notin W_{j+1}$ **do**
    i chooses $v_i^{j+1} = 2v_i^j$ or $v_i^j = 0$ (retire)
  end for
  $L_{j+1} = \{i \in N | v_i^j = 0\}$, j=j+1
end while
J=j

Output: Each winner $i \in W_J$ receives his final active bundle s_i^J and pays his final bid v_i^J. All other players get nothing and pay 0.

Procedure 3 formally defines the 1-CA-SUB procedure, it is followed by an informal description. We first present some notation.

Notation: For a set of players X and an allocation $s^j = (s_1^j, ..., s_n^j)$, we denote by $Free(X, s^j)$ the set of goods that are not allocated to any player in X, i.e. the items that are not in $\cup_{i \in X} s_i^j$.

Procedure 3 The 1-CA-SUB procedure:

Input: vectors of values v^j and of bundles s^j (with one element for each player), and a set of winners W_j such that the allocation (W_j, s^j) is valid.

initialize $MAX_{j+1} \leftarrow \text{argmax}_{i \in N}\{ v_i^j \}$, $GREEDY_{j+1} \leftarrow \emptyset$, and $\forall i \in N,\ s_i^{j+1} \leftarrow s_i^j$.
for all player i with $v_i^j > 0$ in descending order of the values v_i^j **do**
  // **Shrinking the active set:**
  if $i \notin W_j$ **then**
    allow i to update s_i^{j+1} to any bundle s that satisfies $s \subseteq Free(GREEDY_{j+1}, s^{j+1}) \cap s_i^j$ and $|s| \leq \sqrt{m}$.
  end if
  // **Updating the current winners and ensuring a Pareto allocation:**
  if $|s_i^{j+1}| \leq \sqrt{m}$ **then**
    add i to any of the allocations $W \in \{W_j, MAX_{j+1}, GREEDY_{j+1}\}$ for which $s_i^{j+1} \subseteq Free(W, s^{j+1})$.
  end if
end for

Output: The vector of bundles s^{j+1} and an allocation $W \in \{W_j, MAX_{j+1}, GREEDY_{j+1}\}$ with maximal value of $\sum_{i \in W} v_i^j$.

This procedure is essentially a modification of the 1-CA algorithm of Mu'alem and Nisan [26]. However, while the original 1-CA is truthful only for known single-minded players, we construct a procedure that is suitable for unkonown multi-minded players by using our wrapping technique and the new notion of algorithmic implementation.

At iteration $j + 1$, 1-CA-SUB receives the current players' values and bundles, as well as the previous allocation W_j (Input), and constructs two allocations, MAX_{j+1} and $GREEDY_{j+1}$. The allocation $GREEDY_{j+1}$ holds only players with active sets of size at most $\sqrt{m}$. It is constructed by going over all such players from the highest value to the lowest value, inserting a player to the allocation if and only if her chosen bundle does not intersect some other bundle of a player that already belongs to $GREEDY_{j+1}$. The allocation MAX_{j+1} holds the player with the maximal current value (line 1), plus perhaps some other players, to make it Pareto efficient (we give more details below). The construction of the two allocations is done by iterating over all non-retired players in descending order of values (line 2), and performing two steps for each player, as follows.

In the first step (lines 3-6), a player that was previously a loser is given an option to "shrink" her bundle. If a player chooses to shrink her bundle, the new bundle must be: (1) contained in the old bundle, (2) of size at most $\sqrt{m}$, and (3) disjoint from the bundles of players that were already added to $GREEDY_{j+1}$. This ensures that a player that actually shrinks her bundle will be added to $GREEDY_{j+1}$. (the actual addition is done in line 9).

In the second step (lines 7-10), the player is added to any of the allocations $W_j, MAX_{j+1}, GREEDY_{j+1}$, for which no existing player's bundle intersects the new player's bundle. While approximation considerations require the player to be added only to $GREEDY_{j+1}$, the player may also be added to W_j and/or to MAX_{j+1}, to ensure that all allocations $W_j, MAX_{j+1}, GREEDY_{j+1}$ are Pareto efficient with respect to the new active bundles of the players. As a result, a losing player cannot be added to the output set of winners, since there exists a winner with a new active set that intersects that losing player's new active set.

Once all players have been considered, the procedure outputs the allocation with maximal value out of the three allocations $W_j, MAX_{j+1}, GREEDY_{j+1}$ as the new set of winners W_{j+1}. It also outputs the updated active sets. At the end of a call to 1-CA-SUB, we have:

- The new allocation W_{j+1} has value at least as large as:
  1. The player with the maximal value (since we compare value with MAX_{j+1}).
  2. The greedy construction that chooses winners by going over all players in decreasing order of values, announcing each player to be a winner if the previous winners' bundles do not intersect her bundle (since we compare value with $GREEDY_{j+1}$).
  3. The value of the old allocation (since we compare value with W_j).
- The new allocation is Pareto efficient with respect to the new active sets of the players, i.e. no losing player can be added to the set of winners.

Note that the procedure relies on the assumption that a player can answer the following query in polynomial time: Given a set of items, return a desired bundle contained in this set, of size at most $\sqrt{m}$, if such a bundle exists. The query in line 5 presents the set $Free(GREEDY_{j+1}, s^{j+1}) \cap s_i^j$ to the player as the input set to this query. For example, in the Edge-Disjoint-Paths problem, this query can be answered in polynomial time since it is equivalent to finding a shortest path of length at most $\sqrt{m}$ in a given graph.

We give the full analysis in Section 6 below, but since the transition to the new solution concept of algorithmic implementation is a bit subtle, we first give some intuition. Consider the case of EDP, being solved with the Iterative Wrapper and the 1-CA-SUB. Each edge is an item, and the entire set of edges is to be allocated to the different source-target players. As we go over players, e.g. in the first iteration, each player is asked to specify a desired subset of edges. Suppose that the player has two edge disjoint paths of length at most $\sqrt{m}$ in the original graph, but some edges in the first path are not available when the player is reached, while the second path is available. At this point, the player may shrink his active subset, removing the currently occupied edges, or she may choose not to shrink the active set. By the first alternative, she is included in $GREEDY_{j+1}$, and possibly avoids the need to double her price. However this also limits her choices in future iterations, as she will no longer be able to win the other desired path. The second option will cause her to double her winning payment. Losing players face such a decision in every iteration. A mechanism with dominant strategies implies that the player has a specific optimal choice, no matter what the other players choose. In our case, we do not rule out either of these choices. Instead, we show that the approximation ratio will be maintained *in either case*.

In particular, we will show in Section 6 that all we need in order to guarantee the approximation is that in the iteration at which the player actually retires, she discloses one of her true paths, if such a path is still available. More concretely, in the actual mechanism, a player will be forced to retire at some iteration if the player is not included in the set of winners, and cannot (subsequently) double her value, as it will be higher than her true value. A player can recognize that she faces such a situation already in line 5 of 1-CA-SUB: If the player does not belong to W_j nor to MAX_{j+1}, and her current active set intersects previous winners in all three allocations $W_j, MAX_{j+1}, GREEDY_{j+1}$, then the player will have to retire if she does not shrink her active set. We term this situation a "loser-if-silent":

Definition 4 (loser-if-silent)**.** *Player i is a "loser-if-silent" at iteration j if, when she is asked to shrink her bundle at line 5 of the 1-CA-SUB procedure, all the following hold:*

1. *(Retires if losing) $v_i^j \geq \bar{v}_i(s_i^j)/2$.*
2. *(Not included previously) $i \notin W_j$ and $i \notin MAX_{j+1}$.*
3. *(Loses at the end) $s_i^j \cap (\cup_{i \in W_j} s_i) \neq \emptyset$, $s_i^j \cap (\cup_{i \in MAX_{j+1}} s_i) \neq \emptyset$, and $s_i^j \cap (\cup_{i \in GREEDY_{j+1}} s_i) \neq \emptyset$.*

In words, i is a loser-if-silent if she will lose unless she will shrink her active set when she is approached at line 5, no matter what the others will declare. Our approximation analysis depends on the fact that a loser-if-silent player will indeed shrink her active set, if she can. Intuitively and informally, the player will indeed do this – "what does she have to lose"? If she will not shrink the set, she is *guaranteed* to lose, and if she shrinks the set she might win. Phrasing this informall explanation using a formal game-theoretic reasoning involves the statement that choosing a strategy in which a loser-if-silent will not shrink her set is a *dominated* strategy. Note that there are *many* strategies in which a loser-if-silent shrinks her set, and so we do not highlight one particular dominating strategy. We just rule out some dominated strategies, and argue that all remaining strategies lead to a good approximation. The next section discusses more formally the game-theoretic definitions, and the subsequent section will continue with the formal analysis.

5 Algorithmic Implementation in Undominated Strategies

Requiring that players have dominant strategies limits the family of allocation algorithms that can be used [2, 4, 21]. As stated in the introduction it is clear that this requirement is not the real essence, but rather a tool to obtain the underlying goal: reaching approximately optimal outcomes in the presence of selfish behavior. For the case of the general wrapper with the 1-CA-SUB, we obtain this goal although we leave in several "reasonable" strategies for the players to choose from. The basic assumption is that a player prefers not to choose a *dominated* strategy, since this means that there exists another strategy that *always* performs at least as well. Thus, *any* rational player will move from a given strategy to another strategy that dominates it. Our notion of algorithmic implementation captures this intuition, even if players are computationally bounded:

Definition 5. *A mechanism M is an* algorithmic implementation of a c-approximation *(in undominated strategies) if there exists a set of strategies, D, with the following properties:*

1. *M obtains a c-approximation for any combination of strategies from D, in polynomial time.*
2. *For any strategy that does not belong to D, there exists a strategy in D that dominates it. Furthermore, we require that this "improvement step" can be computed in polynomial time.*

It seems trivial to assume that players will not play any strategy outside of D, since there exists a strategy in D that dominates it, and that can be obtained using a "short" (polynomial-time) calculation. When players play *any* tuple of strategies from D, the outcome is a c-approximation. Note that the definition does not simply require the ability to compute *some* undominated strategy in polynomial time. This will not suffice, as it might be rational to play a different, dominated strategy, that is not dominated by the calculated strategy. Instead, the definition requires that given *any* strategy not in D, we can compute a strategy in D that dominates the given strategy, in polynomial time. This stronger requirement does ensure that players will restrict themeselves to playing only strategies that belong to D.

Thus, algorithmic implementation shares all the important properties of truthfulness in dominant strategies:

- Both concepts do not require any form of coordination among the players (unlike e.g. Nash equilibrium)
- Both concepts do not require that players have any assumptions on the rationality behavior of the others (unlike e.g. "iterative deletion of dominated strategies").

- The only assumption underlying both concepts is that players do not play a strategy that is clearly dominated.

Our specific mechanism also ensures "individual rationality": the resulting utility is non-negative, thus players have no risk in participating.

However, two differences do exist, and should be explicitly mentioned: (1) in our case, a player might regret his chosen strategy, realizing in retrospect that another strategy from D would have performed better (it is an implementation concept, not an equilibrium concept), and (2) while in dominant strategies it is a straight-forward task for a player to find a strategy to play, here finding the strategy to play is not straight-forward, as there are many strategies in D to choose from. Both these differences are not important for the mechanism designer. The designer should not care which tuple of strategies in D is chosen, since the approximation is obtained for *any such tuple*.

Two further remarks about the general notion are important:

1. Algorithmic implementation generalizes the well-known game-theoretic concept of "implementation in undominated strategies", since all undominated strategies must belong to D.[11] Algorithmic implementation adds the aspect of computationally bounded players, emphasizing that, for such players, the transition from a dominated strategy to the strategy that dominates it must be computed efficiently.

2. Algorithmic implementation only requires the mechanism to choose a close to optimal outcome for every tuple of *pure* strategies from D. However even if the players randomize over pure strategies (i.e. forming a mixed strategy so as to maximize their expected utility), the approximation guarantee still holds, since the support of any rational choice of a mixed strategy should contain pure strategies only from D.

6 Analysis

With the new notion of algorithmic implementation, we can formally state our main result:

Theorem 2. *Suppose all players are single valued, where the set of desired bundles of each player is private information to him. Then the Iterative Wrapper with the 1-CA-SUB procedure is an algorithmic implementation in undominated strategies of an $O(\log^2(\bar{v}_{max}) \cdot \sqrt{m})$-approximation.*

We prove this in three parts. We first define the set D in Section 6.1. We then analyze the approximation that these strategies produce. We give general conditions on the procedure used inside the wrapper, that lead to an approximation, in Section 6.2. Finally, in Section 6.3 we show that the 1-CA-SUB indeed satisfies these conditions.

6.1 Analysis of Strategies

The first step of the proof defines the set D of strategies:

Definition 6 (The set D of dominating strategies)**.** *Let D be all strategies that satisfy the following, in every iteration j:*

1. *$\bar{v}_i(s_i^j) \geq v_i^j$, and, if i retires at iteration j, then $v_i^j \geq \bar{v}_i(s_i^j)/2$.*

2. *If player i is a loser-if-silent (Def. 4), then i will declare some desired bundle s_i^{j+1} that satisfies the conditions of line 5 of Procedure 3 if such a bundle exists.*[12]

[11] More accurately, if an undominated strategy st does not belong to D then there exists a different strategy $st' \in D$ that always yield the *same* utility as st, so both strategies are completely equivalent for the player.

[12] That is, if i is a loser-if-silent and there exists a desired bundle $s \subseteq Free(GREEDY_{j+1}, s^{j+1}) \cap s_i^j$ such that $|s| \leq \sqrt{m}$, then i will choose to shrink her active subset to such an s.

We note that any strategy in D is ex-post individually rational – a player cannot obtain a negative utility, no matter what are the actions of the other players, if she chooses a strategy in D. This follows from the first requirement of D.

To show that our mechanism is an algorithmic implementation with respect to the set D, we have to show two things: (1) that any tuple of strategies from D results in a $O(\log^2(\bar{v}_{max}) \cdot \sqrt{m})$-approximation, and (2) that there exists a polynomial-time algorithm that receives any strategy $st \notin D$ and finds a strategy $st' \in D$ that dominates st. We start by showing the latter requirement:

Lemma 3. *There exists a polynomial time algorithm to find a strategy $st' \in D$ that dominates a given strategy st.*[13]

Proof. Consider the following algorithm that checks, at each played decision point of the strategy st, if the two conditions of Def. 6 are satisfied.

If the player is a "loser-if-silent" and does not shrink her bundle, check if there exists a desired subset that can be declared in line 5 (this can be done in polynomial time by our assumption on the query capabilities). If such a desired bundle exists, we change the strategy to a strategy that ensures non-negative utility (while the original strategy will yield zero utility). Instead, the player shrinks the active set to some arbitrary desired set that does satisfy the requirement. In the changed strategy, the player never increases her value in later iterations. Since a player receives her last active bundle, this will change the utility from zero to a non-negative utility, hence this creates a strategy that dominates the former.

When a player shrinks her active subset, check if the new set has value at least v_i^j. If not, this strategy always results with non-positive utility, and we change it to exactly the same strategy that was used in the previous paragraph.

When st decides to double the value v_i^j, check if the new value is lower than the true value of the currently active bundle. If not, change st so that it will instead decide to retire. When st decides to retire and not to double the value, check if twice the value is higher than the true value of the currently active bundle. If not, change st so that it will instead decide to double the value (after that, keep increasing the value up to the true value of the active set, and when "loser-if-silent" follow the procedure above). Since a winner pays her last reported value, and can only gain by not retiring at a lower value (has no risk if she stays active), this again creates a strategy that dominates the former one. □

This implies the second requirement of the algorithmic implementation definition. The next two sections handle the first requirement: showing that if all players play strategies in D then the approximation holds.

6.2 Analysis of the Iterative Wrapper

We next describe sufficient conditions such that, when plugging any sub-procedure that satisfies them inside the iterative wrapper, the result will be a c-approximation. In the subsequent section we show that the 1-CA-SUB procedure indeed satisfies the conditions that we give here.

Notation: Given a set of players, W, and a vector of player subsets, $s = (s_1, ..., s_n)$, we say that (W, s) is a *valid allocation* if $s_i \cap s_{i'} = \emptyset$ for any $i, i' \in W$ such that $i \neq i'$.

Recall that the sub-procedure receives as input (1) the players' current values v^j, (2) the players' active subsets s^j, and (3) a subset of the players, W_j, such that (W_j, s^j) is a valid allocation. The output of the procedure is a set of winning players, W_{j+1}, and a set of active sets, s^{j+1}, such that (W_{j+1}, s^{j+1}) is a valid allocation. We list six properties that we require from this procedure, whenever players play strategies in D. The first three properties are parallel to the properties discussed in Section 3:

- **(Pareto)** For any $i \notin W_{j+1}$, $s_i^{j+1} \cap (\cup_{l \in W_{j+1}} s_l^{j+1}) \neq \emptyset$.

[13] Note that the algorithm does not take a strategy and *explicitly* outputs another strategy that dominates the first, as this is not poly-time – it needs to handle exponentially many decision points. Rather, at each decision point the player actually faces, the algorithm checks to see if the strategy is taking a dominated action, and if it does, replaces the action and changes the strategy to fit the new action, in a rather straight-forward way.

- **(Improvement)** $\sum_{i \in W_{j+1}} v_i^j \geq \sum_{i \in W_j} v_i^j$.
- **(Value bounds)** $\bar{v}_i(s_i^{j+1}) \geq v_i^j$, and, if i retires at j then also $\bar{v}_i(s_i^{j+1}) \leq 2 \cdot v_i^j$.

As in Section 3, the Pareto property requires that no loser can be added to the set of winners, the improvement property requires the value of the tentative winning allocation to always increase between iterations, and the value-bounds property implies that a player retires if and only if the declared value is approximately equal to the true value of the current active set.

Two additional properties are introduced since the sub-procedure handles the issue of shrinking the active subsets, in contrast to the setting of Section 3 were the desired subsets were assumed to be known.

- **(Shrinking sets)** For every player i, $s_i^{j+1} \subseteq s_i^j$.
- **(First time shrink)** Let FIRST-SHRINK$_j = \{i : |s_i^j| = m \ \& \ |s_i^{j+1}| < m\}$. Then for any $i_1, i_2 \in$ FIRST-SHRINK$_j$ such that $i_1 \neq i_2$ it holds that $s_{i_1}^{j+1} \cap s_{i_2}^{j+1} = \emptyset$.

The shrinking-sets property simply requires the updated active set of each player to be contained in the previous active set, so that active sets "shrink" as the wrapper advances. The first-time-shrink property examines FIRST-SHRINK$_j$, which is the set of players that shrink their active set *for the first time* at iteration j, and requires the allocation (FIRST-SHRINK$_j, s^{j+1}$) to be valid.

The last property concerns the approximation guarantee that the sub-procedure provides. The definition here is a bit subtle, since the sub-procedure does not "see the real valuations". There are two differences between the actual valuation of a player and the valuation that the 1-CA-SUB procedure observes: (1) Inside the procedure the player is restricted to bid only on desired bundles that are contained in s_i^j, and not on all his true desired bundles, and (2) inside the procedure a player's value is fixed to be v_i^j, which in general is different than the player's true value.

To capture these two differences we define the following two concepts: First, we define a *restricted* valuation $\bar{v}_i|_{s_i^j}(\cdot)$ to be:

$$\bar{v}_i|_{s_i^j}(s) = \bar{v}_i(s \cap s_i^j)$$

for every $s \in \Omega$. In words, a player with a value function $\bar{v}_i|_{s_i^j}(\cdot)$ values only bundles that are contained in s_i^j, and each such bundle is valued by its true value. Note that $\bar{v}_i|_\Omega(\cdot) = \bar{v}(\cdot)$. Second, we define a *rounded and restricted* valuation $\bar{v}_i^j|_{s_i^j}(\cdot)$ to be:

$$\bar{v}_i^j|_{s_i^j}(s) = \begin{cases} v_i^j & \bar{v}_i(s \cap s_i^j) = \bar{v}_i \\ 0 & \text{otherwise} \end{cases}$$

for every $s \in \Omega$. In words, the rounded and restricted value $\bar{v}_i^j|_{s_i^j}(s)$ of a bundle s is v_i^j if the restricted value of s is positive, otherwise the rounded and restricted value of s is zero.

After this preparation, we can now define the last property that we need. Let RRV_j be the set of all the restricted and rounded valuations of players that retired at iteration j, i.e.

$$RRV_j = \{ \ \bar{v}_i^j|_{s_i^j}(\cdot) \mid \text{for all players } i \text{ that retired at iteration } j \ \} \tag{1}$$

The last property that we require captures the fact that the result of the sub-procedure is only an approximation with respect to rounded and restricted valuations:

- **(c-local-approximation)** At any iteration j, $OPT(RRV_j) \leq c \cdot \sum_{i \in W_{j+1}} v_i^j$.

In other words, the property of a c-local-approximation requires that the winners at the j'th iteration (players in the set W_{j+1}) have total value at least $1/c$ of the optimal allocation for the *rounded and restricted* valuations of players that retire at iteration j.

We have listed six properties, and we now prove that any sub-procedure that satisfies these properties can be used in the wrapper, to obtain a $O(\log^2(\bar{v}_{max}) \cdot c)$-approximation. The main difficulty is to translate the local approximation to a global approximation. Specifically, we are given that that $OPT(RRV_j) \leq c \cdot \sum_{i \in W_{j+1}} v_i^j$ for any iteration j, and we wish to prove that $OPT(\text{true valuations}) = OPT(\bar{v}_1, ..., \bar{v}_n) \leq O(c \cdot J^2) \cdot \sum_{i \in W_J} v_i^J$. We will then show that $J = O(\bar{v}_{max})$, which will yield the desired result. Let

$$RV = \{\ \bar{v}_1|_{s_1^J}(\cdot), ..., \bar{v}_n|_{s_n^J}(\cdot)\ \},$$

i.e. RV contains all the final restricted valuations. We will show that $OPT(\bar{v}_1, ..., \bar{v}_n) \leq O(c \cdot J^2) \cdot \sum_{i \in W_J} v_i^J$ in two parts. First, we show $OPT(\bar{v}_1, ..., \bar{v}_n) \leq (J+1) \cdot OPT(RV)$. Second, we show $OPT(RV) \leq 2 \cdot J \cdot c \cdot \sum_{i \in W_J} v_i^J$. The two inequalities imply the global approximation.

Claim 1. $OPT(\bar{v}_1, ..., \bar{v}_n) \leq (J+1) \cdot OPT(RV)$.

Proof. For each iteration $0 \leq j \leq J-1$, let $P_j = \{\ \bar{v}_i(\cdot)\ :\ |s_i^j| = m,\ \text{and}\ |s_i^{j+1}| < m\ \}$, i.e. P_j contains all players who first shrank their bundles at iteration j, with their true value function. By the first-time-shrink property, all players in P_j have disjoint bundles since they all shrank there bundles for the first time in iteration j. In other words, $s_{i_1}^{j+1} \cap s_{i_2}^{j+1} = \emptyset$ for every $i_1, i_2 \in P_j$ such that $i_1 \neq i_2$. Since active sets only shrink, we also have $s_{i_1}^J \cap s_{i_2}^J = \emptyset$. Every player i in P_j corresponds to a player in RV, and the above implies that all these players have disjoint bundles in RV. Therefore $OPT(RV) \geq \sum_{i \in P_j} \bar{v}_i(s_i^J)$ for every $0 \leq j \leq J-1$.

In the other direction, we trivially have $OPT(P_j) \leq \sum_{i \in P_j} \bar{v}_i(\Omega) = \sum_{i \in P_j} \bar{v}_i(s_i^J)$, where the right equality follows since the value bounds property imply $\bar{v}_i(s_i^J) > 0$ and since players are single valued.

Putting these two inequalities together we get, for any j, $OPT(P_j) \leq OPT(RV)$. Let $P = \cup_j P_j$, and $\bar{P} = \{\ \bar{v}_i(\cdot)\ :\ |s_i^J| = m\ \}$, i.e. $\bar{P}$ contains all players that did not shrink their bundle at all. Thus $\{\bar{v}_1, ..., \bar{v}_n\} = P \cup \bar{P}$. For any player i in $\bar{P}$ we have $\bar{v}_i(\cdot)|_{s_i^J} = \bar{v}_i(\cdot)$ since $s_i^J = \Omega$. Thus $\bar{P} \subseteq RV$. We get:

$$OPT(\bar{v}_1, ..., \bar{v}_n) \leq OPT(\cup_{j=0}^{J-1} P_j) + OPT(\bar{P}) \leq \sum_{j=0}^{J-1} OPT(P_j) + OPT(RV) \leq J \cdot OPT(RV) + OPT(RV)$$

and the claim follows. □

Claim 2. $OPT(RV) \leq 2 \cdot J \cdot c \cdot \sum_{i \in W_J} \bar{v}_i(s_i^J)$.

Proof. Let RV_j contain all players from RV that retired at iteration j, for $0 \leq j \leq J-1$. Consider $OPT(RV_j)$ versus $OPT(RRV_j)$. Suppose that the optimal allocation for RV_j is $s_1, ..., s_n$. For any player i,

$$\bar{v}_i|_{s_i^J}(s_i) \leq \bar{v}_i(s_i \cap s_i^J) \leq \bar{v}_i(s_i^J) \leq 2 \cdot v_i^j$$

where the last inequality follows from the value bounds property, since player i retires at iteration j. Thus

$$OPT(RV_j) = \sum_i \bar{v}_i|_{s_i^J}(s_i) = 2 \cdot \sum_{i:\ \bar{v}_i|_{s_i^J}(s_i) > 0} v_i^j = 2 \cdot \sum_{i:\ \bar{v}_i|_{s_i^J}(s_i) > 0} \bar{v}_i^j|_{s_i^J}(s_i) \leq 2 \cdot OPT(RRV_j).$$

where the last equality follows since, if $\bar{v}_i|_{s_i^J}(s_i) > 0$ then $\bar{v}_i(s_i \cap s_i^J) > 0$, and therefore $\bar{v}_i^j|_{s_i^J}(s_i) = v_i^j$.

By the c-local-approximation and improvement properties, we have $OPT(RRV_j) \leq c \cdot \sum_{i \in W_{j+1}} v_i^j \leq c \cdot \sum_{i \in W_J} v_i^J$. For every $i \in W_J$, $\bar{v}_i(s_i^J) \geq v_i^J$ by value-bounds. We conclude that

$$OPT(RV_j) \leq 2 \cdot c \cdot \sum_{i \in W_J} \bar{v}_i(s_i^J).$$

Let W contain all players in RV that did not retire. By definition these players are the winners, hence $OPT(W) = \sum_{i \in W_J} \bar{v}_i(s_i^J)$. We get

$$OPT(RV) = OPT(W \cup (\cup_j RV_j)) \leq OPT(W) + \sum_j OPT(RV_j) \leq (2 \cdot J \cdot c + 1) \sum_{i \in W_J} \bar{v}_i(s_i^J)$$

and the claim follows. □

Claim 3. *If the sub-procedure satisfies the properties: pareto, shrinking set, and value bounds then the number of iterations of the General Iterative Wrapper of definition 2 is at most* $2 \log \bar{v}_{max} + 1$.

Proof. Consider iteration $j = 2 \cdot \log(\bar{v}_{max}) + 1$, and some $i_1 \notin W_{j+1} \cup L_j$ that (by contradiction) doubles his value. By the pareto property, there exists $i_2 \in W_{j+1}$ such that $s_{i_1}^{j+1} \cap s_{i_2}^{j+1} \neq \emptyset$. By the shrinking-sets property, in every iteration $j' < j$ their winning bundles intersect, hence at j' at least one of them was not a winner, and doubled his value. But then at least one of the players doubled his value more that $\log(\bar{v}_{max})$ times, hence his tentative value is more than $\bar{v}_{max}$, a contradiction to the property of value-bounds. □

By all the above claims we conclude the following corollary:

Corollary 1. *Suppose that players are single valued and that the six properties: Pareto, improvement, value bounds, shrinking sets, first time shrink, and c-local-approximation, are satisfied. Then the General Iterative Wrapper obtains an* $O(\log^2(\bar{v}_{max}) \cdot c)$*-approximation.*

6.3 Analysis of the 1-CA-SUB

It now only remains to show that the 1-CA-SUB satisfies the above six properties whenever players play any tuple of strategies from D. It is straight-forward to verify the first five properties: Line 9 verifies that all three allocations $W_j, MAX_{j+1}, GREEDY_{j+1}$ are pareto with respect to the active bundles of the players, hence the pareto requirement is satisfied. The improvement requirement is taken care of by the output step, where it is explicitly verified that the new allocation has value not less than W_j, the previous allocation. The value-bounds requirement is immediate from the first requirement of the definition of D. Line 5 explicitly verifies the shrinking-sets requirement by requiring the new active set to be contained in the old one. The first-time-shrink requirement holds since any two players that shrink their subsets at iteration j must both belong to the $GREEDY_{j+1}$ allocation, hence their active subsets are disjoint. The only non-trivial property is the local approximation:

Claim 4. *The 1-CA-SUB procedure is an* $O(\sqrt{m})$*-local-approximation when players play any tuple of strategies from set D of Def. 6.*

Proof. Fix an iteration j, and recall that we need to show that $OPT(RRV_j) \leq O(\sqrt{m}) \cdot \sum_{i \in W_{j+1}} v_i^j$, where RRV_j is the set of rounded and restricted valuations of all players that retire at iteration j, as defined in Eq. (1). For any $W \in \{MAX_{j+1}, GREEDY_{j+1}, W_j\}$, $\sum_{i \in W} v_i^j \leq \sum_{i \in W_{j+1}} v_i^j$ by the improvement property.

Let X_j contains all valuations in RRV_j that correspond to players that do not belong to $MAX_{j+1} \cup GREEDY_{j+1} \cup W_j$, i.e.

$$X_j = \{\ \bar{v}_i^j|_{s_i^j}(\cdot) \mid \forall i \text{ such that } i \notin MAX_{j+1} \cup GREEDY_{j+1} \cup W_j \text{ and } i \text{ retires at iteration } j\ \}.$$

Claim 5 below shows that $OPT(X_j) \leq 2 \cdot \sqrt{m} \cdot \sum_{i \in W_{j+1}} v_i^j$. We conclude that

$$OPT(RRV_j) \leq \sum_{i \in W_j \cup MAX_{j+1} \cup GREEDY_{j+1}} v_i^j + OPT(X_j) \leq (3 + 2 \cdot \sqrt{m}) \sum_{i \in W_{j+1}} v_i^j,$$

and the claim follows. □

Claim 5. $OPT(X_j) \leq 2 \cdot \sqrt{m} \cdot \sum_{i \in W_{j+1}} v_i^j$.

Proof. Let $s_1^*, ..., s_n^*$ be an optimal allocation for X_j. Assume w.l.o.g. that for every player i it holds that s_i^* does not contain any smaller set of the same value with respect to $\bar{v}_i^j|_{s_i^j}(\cdot)$ (minimal to containment). As the valuations in X_j are the rounded and restricted valuations, $s_i^* \subseteq s_i^j$ for all players i, and every player that receives a non-empty bundle has positive value for it.

Let $X_M = \{i : |s_i^*| \geq \sqrt{m}\}$ contain all players that receive a desired bundle of size at least $\sqrt{m}$ in $OPT(X_j)$. We present a mapping $f : X_M \to MAX_{j+1}$ such that f is $\sqrt{m}$ to 1, and for any $i \in X_M$, $v_i^j \leq v_{f(i)}^j$: simply map i to the player with highest value in MAX_{j+1}. There can be at most $\sqrt{m}$ players in X_M since each player receives a bundle of size at least $\sqrt{m}$ and these bundles do not intersect. Thus the first property of f holds. The second property holds since the player with maximal value in MAX_{j+1} has the highest value among all players (according to v^j). The mapping f shows that

$$v(X_M, v^j) \leq \sqrt{m} \cdot v(MAX_{j+1}, v^j)$$

Next, let $X_G = \{ i : 0 < |s_i^*| \leq \sqrt{m}\}$. We show a mapping $f : X_G \to GREEDY_{j+1}$ such that f is $\sqrt{m}$ to 1, and for any $i \in X_G$, $v_i^j \leq v_{f(i)}^j$. We claim that if player i is in X_G then he is loser-if-silent (Def. 4): since player i retired at iteration j it follows that $v_i^j \geq \bar{v}_i(s_i^j)/2$, hence the retires-if-losing condition holds. By the definition of X_j it also holds that $i \notin W_j$ and $i \notin MAX_{j+1}$, hence the not-included-previously condition holds. The definition of 1-CA-SUB (Line 9) implies that $s_i^j \cap (\cup_{i \in W_j} s_i) \neq \emptyset$, $s_i^j \cap (\cup_{i \in MAX_{j+1}} s_i) \neq \emptyset$, and $s_i^j \cap (\cup_{i \in GREEDY_{j+1}} s_i) \neq \emptyset$, otherwise we would had $i \notin MAX_{j+1} \cup GREEDY_{j+1} \cup W_j$, and i would not have been included in X_j. Therefore the loses-at-the-end condition at the end holds. Therefore indeed player i is loser-if-silent at iteration j.

Now, player i has a desired bundle $s_i^* \subseteq s_i^j$ with size at most $\sqrt{m}$, but he was not included in $GREEDY_{j+1}$. Since player i plays some strategy in D, this implies that player i could not have shrunken his active bundle to s_i^*. This means that there exists a player $i' \in GREEDY_{j+1}$ such that $s_{i'}^j \cap s_i^* \neq \emptyset$ and $v_i^j \leq v_{i'}^j$. We map i to i'. It remains to show that f is $\sqrt{m}$ to 1: for any i_1, i_2 with $i_1 \neq i_2$ that were mapped to i' we have that $s_{i_1}^* \cap s_{i_2}^* = \emptyset$ since both sets belong to the optimal allocation. Since the size of $s_{i'}^j$ is at most $\sqrt{m}$ it follows that at most $\sqrt{m}$ players can be mapped to i'. The mapping f shows that $v(X_G, v^j) \leq \sqrt{m} \cdot v(GREEDY_{j+1}, v^j)$. We conclude:

$$OPT(X_j) = v(X_M, v^j) + v(X_G, v^j) \leq \sqrt{m} \cdot v(MAX_{j+1}, v^j) + \sqrt{m} \cdot v(GREEDY_{j+1}, v^j) \leq 2\sqrt{m} \cdot v(W_{j+1}, v^j),$$

and the claim follows. □

7 Conclusions

In this paper we have explored the advantages of a simple wrapper technique that converts algorithms to mechanisms. This technique enables us to provide two main contributions: a deterministic method to convert *algorithms* to *mechanisms* for CAs with "known" single-value players, and a deterministic mechanism for "unknown" single-value players. This is the first deterministic mechanism for a single-value settings. An additional advantage of the wrapper technique is that it creates *ascending auctions*, which is a much more realistic auction format than direct-revelation auctions.

While our main use of the wrapper technique is to design a mechanism for "unknown" single-value players, the technique can also be extended in other directions, and in the appendix we give two such examples:

- In Appendix A we generalize the wrapper technique to the case of "known" single-value players. Similarly to the "known" single-minded case of Section 3, we show in the appendix how the wrapper technique can be used to convert c-approximation algorithms for "known" single-value players to ascending auctions that obtain $O(log^2(\bar{v}_{max}) \cdot c)$ approximation in dominant strategies. Since a single-value player may require exponential number of bundles, we have to require an oracle access to the players, see the formal definitions and the theorem statement in the appendix.

- In Appendix B we generalize the wrapper technique to the case of *multi-unit* combinatorial auctions with "known" single-minded players: there are B copies of each item, each player desires a single bundle, and each desired bundle contains at most one copy of each item. Similarly to the "known" single-minded case of Section 3, we show in the appendix how the wrapper technique can be used to convert any c-approximation algorithm for the multi-unit combinatorial auctions case to an ascending auction that obtains $O(B \cdot log(\bar{v}_{max}) \cdot c)$ approximation in dominant strategies.

This paper also contributes a significant conceptual shift from the "usual" goal of constructing dominant-strategy mechanisms. We have defined the powerful concept of "algorithmic implementation in undominated strategies", and demonstrated its usefulness. An additional demonstration of its powers is given in [5]. In this context we should also mention [22], which advocates the similar (though significantly weaker) concept of "Set-Nash".

The above results and discussion raise several open questions. We wish to highlight two such questions. First, the limitations of dominant-strategy mechanisms for CAs are still unclear:

Open question 1: *Does there exist a deterministic polynomial-time dominant-strategy implementation for single value CAs with $O(\sqrt{m})$ approximation?*

Second, the power of algorithmic implementations versus dominant-strategy implementations is unclear:

Open question 2: *Does there exist a problem domain in which an algorithmic implementation achieves a better approximation ratio than any dominant strategy implementation?*

Acknowledgments

We wish to thank the two anonymous referees and the area editor for many helpful and thoughtful comments. We are grateful to Noam Nisan for his invaluable help. We also thank Liad Blumrosen, Edith Elkind, Jason Hartline, Daniel Lehmann, and Chaitanya Swamy, for their many comments.

References

[1] K. Akcoglu, J. Aspens, B. DasGupta, and M. Kao. An opportunity-cost algorithm for combinatorial auctions. In E. J. Kontoghiorghes, B. Rustem, and S. Siokos, editors, *Applied Optimization: Computational Methods in Decision-Making, Economics, and Finance*. Kluwer Academic, 2002.

[2] Aaron Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *FOCS*, pages 482–491, 2001.

[3] B. Awerbuch, Y. Azar, and A. Meyerson. Reducing truth-telling online mechanisms to online optimization. In *STOC*, pages 503–510, 2003.

[4] M. Babaioff, R. Lavi, and E. Pavlov. Mechanism design for single-value domains. In *AAAI*, pages 241–247, 2005.

[5] M. Babaioff, R. Lavi, and E. Pavlov. Impersonation-based mechanisms. In *AAAI*, 2006.

[6] Y. Bartal, R. Gonen, and N. Nisan. Incentive compatible multi-unit combinatorial auctions. In *TARK*, pages 72–87, 2003.

[7] L. Blumrosen and N. Nisan. On the computational power of iterative auctions. In *ACM-EC*, pages 29–43, 2005.

[8] L. Blumrosen and N. Nisan. Combinatorial auctions. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University press, 2007.

[9] P. Briest, P. Krysta, and B. Vocking. Approximation techniques for utilitarian mechanism design. In *STOC*, pages 39–48, 2005.

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

[11] S. Dobzinski. Two randomized mechanisms for combinatorial auctions. In *RANDOM-APPROX*, 2007.

[12] S. Dobzinski and N. Nisan. Limitations of vcg-based mechanisms. In *STOC*, 2007.

[13] S. Dobzinski, N. Nisan, and M. Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. In *STOC*, pages 610–618, 2005.

[14] S. Dobzinski, N. Nisan, and M. Schapira. Truthful randomized mechanisms for combinatorial auctions. In *STOC*, 2006.

[15] S. Dobzinski and M. Sundararajan. On characterizations of truthful mechanisms for combinatorial auctions and scheduling. In *The Proc. of the 9th ACM Conference on Electronic Commerce*, 2008.

[16] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of IJCAI'99*, Stockholm, Sweden, July 1999. Morgan Kaufmann.

[17] R. Gonen and D. J. Lehmann. Optimal solutions for multi-unit combinatorial auctions: branch and bound heuristics. In *ACM Conference on Electronic Commerce*, pages 13–20, 2000.

[18] T. Groves. Incentives in teams. *Econometrica*, pages 617–631, 1973.

[19] R. Holzman, N. Kfir-Dahav, D. Monderer, and M. Tennenholtz. Bundling equilibrium in combinatorial auctions. *Games and Economic Behavior*, 47:104–123, 2004.

[20] M. O. Jackson. Implementation in undominated strategies: A look at bounded mechanisms. *Review of Economic Studies*, 1992.

[21] R. Lavi, A. Mu'alem, and N. Nisan. Towards a characterization of truthful combinatorial auctions. In *FOCS*, pages 574–583, 2003.

[22] R. Lavi and N. Nisan. Online ascending auctions for gradually expiring items. In *SODA*, pages 1146–1155, 2005.

[23] R. Lavi and C. Swamy. Truthful and near optimal mechanism design via linear programming. In *FOCS*, pages 595–604, 2005.

[24] R. Lavi and C. Swamy. Truthful mechanism design for multidimensional scheduling. In *The Proc. of the 8th ACM Conference on Electronic Commerce (ACM-EC)*, 2007.

[25] D. Lehmann, L. O'Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):1–26, 2002.

[26] Ahuva Mu'alem and Noam Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. In *AAAI/IAAI*, pages 379–384, 2002.

[27] N. Nisan and I. Segal. The communication requirements of efficient allocations and supporting prices. *Journal of Economic Theory*, 2006.

[28] N. Nisan and E. Zurel. An efficient approximate allocation algorithm for combinatorial auctions. In *ACM Conference on Electronic Commerce*, pages 125–136, 2001.

[29] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

[30] Y. Shoham P. Cramton and R. Steinberg. *Combinatorial Auctions*. The MIT press, 2005.

[31] D. Parkes. Iterative combinatorial auctions. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*. The MIT press, 2005.

[32] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Cabob: A fast optimal algorithm for winner determination in combinatorial auctions. *Management Science*, 51(3):374–390, 2005.

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

[34] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *ACM Symposium on Theory of Computing (STOC)*, pages 681–690, 2006.

A The Iterative Ascending Wrapper for Known Single-Value Players

In Section 3 we have seen that the Iterative Wrapper can turn *algorithms* for known single-minded players into dominant-strategy *mechanisms*, with only a small loss in the approximation. In this section we show that the Wrapper technique can be extended to the more general case of known single-value players. The basic idea is to turn a single-value player into a single-minded player, by "fixing" the first bundle he provisionally receive, as his only desired bundle. By this treatment of the agents as single-minded agents we keep the number of iteration to $O(log(\bar{v}_{max}))$, and use the six properties of Section 6.2 to show the approximation. In other words, we show that ignoring all the other subsets incurs an additional approximation loss of factor of at most $O(\log(\bar{v}_{max}))$.

Since the collection $\bar{S}_i$ of the desired bundles of player i may have exponential size, we need to assume that the given algorithm is equipped, for each player i, with an oracle O_i that answers queries about $\bar{S}_i$. A space of valid bundle-collections $\mathcal{S}$ is a family of collections of sets of items, each collection $\bar{S}_i$ includes all sets that i desires ($\bar{S}_i$ satisfies monotonicity). For example, for the Edge-Disjoint-Paths domain, each $\bar{S}_i$ is a collection of paths, that includes all paths that will satisfy a player (each path in $\bar{S}_i$ is a desired set of items), and $\mathcal{S}$ includes all such valid collections of paths. Fixing $S_i \in \mathcal{S}$, we are given an oracle that answers the queries that the algorithm-designer determines:

Definition 7. *A "bundles oracle" for a space of valid bundle-collections $\mathcal{S}$, and an algorithm-dependent query space Q, is a function $O : \mathcal{S} \times Q \to 2^{\Omega}$. I.e. for any fixed type $S_i \in \mathcal{S}$ and for any query $q \in Q$, the oracle outputs a set of items.*

Note that the assumption of "known" players implies that the oracle cannot lie, i.e. it "knows" that $\bar{S}_i$ is the player's true type, and answers queries accordingly. The player's only secret parameter is his value $\bar{v}_i$. Thus, this definition provides a decoupling of the player's type, differentiating between his value and his desired bundles. This is different from previously considered classes of oracles (e.g. value oracles and demand oracles [7]), in which the underlying "knowledge-base" of the oracle did not model any such difference. For single-value players it seems natural to make this decoupling.

Figure 1 describes a sub-procedure for the case of known single-value players, to be placed inside of the wrapper of Section 4, replacing 1-CA-SUB. In a similar manner to the single-minded players case, this procedure converts any given c-approximation ALG to a dominant-strategy mechanism that obtains an $O(log^2(\bar{v}_{max}) \cdot c)$ approximation. We use the analysis of Section 6 to argue about the approximation, by showing that this sub-procedure satisfies the six required properties.

The basic idea of the sub-procedure is similar to the case of single-minded players. The only important modification is a "freezing" of first-time winners: every player, from the moment of his first win and later on, is regarded as a single-minded player with a desired bundle that is the bundle he first received (this is step 1 in Figure 1). For this modification to perform well we need to be careful and make sure that indeed we can *replace* a player by a single-minded player. We need the space $\mathcal{S}$ to include all such single-minded

A Sub-Procedure for Known Multi-Minded Players:

Input:
A vector of values v^j, a vector of bundles s^j (with one element for each player), and a vector of oracles $\{ORACLE_i\}_i$. An allocation W_j which is legal w.r.t. s^j.

Allocation:

1: For any player i with $s_i^j \neq \Omega$ **change** $ORACLE_i$ to be the oracle that corresponds to a single-minded player that desires the bundle s_i^j.

2: $(W_{j+1}, outSet^j) = ALG(v^j, \{ORACLE_i\}_i)$

3: if $v(W_{j+1}, v^j) > v(W_j, v^j)$ then set $s_i^{j+1} = outSet_i$ for any $i \in W_{j+1}$, else set $W_{j+1} = W_j$.

4: For any $i \in N \setminus W_{j+1}$ with $v_i^j > 0$ (in an arbitrary order), [Pareto stage].
if $s_i^{j+1} \cap (\cup_{l \in W_{j+1}} s_l^{j+1}) = \emptyset$ then $W_{j+1} = W_{j+1} \cup \{i\}$.

Output:
The vector of bundles s^{j+1} and the allocation W_{j+1}.

Figure 1: A "sub-procedure" for known multi-minded players, to be placed inside the general wrapper described in Mechanism 2, Section 4, replacing 1-CA-SUB. This sub-procedure requires the Wrapper to pass on the oracles from one iteration to the other (a trivial addition).

players, since otherwise it might be the case that a c-approximation algorithm for a space $\mathcal{S}$ will not yield a c-approximation (or will not even be well-defined) if we freeze a bundle of a winner (since we lose some structure on the bids).

Definition 8. *A type space $\mathcal{S}$ "includes the single-minded players extension" if for any $S_i \in \mathcal{S}$ and any $s_i \in S_i$, $\{s_i\} \in \mathcal{S}$ (where $\{s_i\}$ is a collection of bundles that contains one element which corresponds to a single-minded player with bundle s_i).*

For example, for the EDP domain, if there exists some $S \in \mathcal{S}$ that contains *all* paths from a node s to a node t then for any path $\mathcal{P} \in S$ there must also exist $S' \in \mathcal{S}$ that contains *only* the path $\mathcal{P}$. This enables us to convert the single-value player, that desires any one of the paths from s to t, so a single-minded player that desires only the specific path $\mathcal{P}$.

When we freeze the bundle of a player, we essentially ignore his original oracle, and answer the queries that the algorithm presents by ourself. Since we would like our mechanism to run in polynomial time, we need:

Definition 9. *An oracle O is "polynomially computable for single-minded players" if for every $S_i \in \mathcal{S}$ with $|S_i| = 1$ (i.e. $S_i = \{s_i\}$ for some $s_i \in 2^\Omega$) and for every $q \in Q$, $O(q, S_i)$ can be computed in polynomial time in the number of items.*

Note that we do not assume anything on the complexity of O for multi-minded bidders (this is an oracle access). With these definitions, we have:

Theorem 3. *Assume that ALG is a c-approximation algorithm for the type space $\mathcal{S}$ that includes the single-minded players extension. Then the General Iterative Wrapper with the sub-procedure of Figure 1 implements an $O(log^2(\bar{v}_{max}) \cdot c)$ approximation in dominant strategies.*

Furthermore, if ALG has polynomial running time and uses an oracle access which is polynomially computable for single-minded players then the created mechanism has polynomial running time.

Proof. In a similar manner to the known single minded case, the dominant strategy of a player is to double his value (whenever he is a loser) if and only if the doubled value will not be higher than his true value. We show that when the player plays this dominant strategy the six properties of Section 6.2 are satisfied, which proves the claim. Line 4 verifies Pareto. Line 3 verifies Improvement. The value-bounds requirement is exactly the dominant strategy. The shrinking-sets property follows from the method of freezing the desired set of

a player, since if the set s_i^j does not contain all items, it never changes. The first-time-shrink requirement holds since players that shrink their subsets (Line 2) must be declared provisional winners by ALG. The c-local-approximation property holds since ALG is a c-approximation with respect to the oracles that it receives and with respect to the vector of values v^j, and these exactly represent the valuations in the set RRV_j. □

One may consider plugging randomized algorithms to this procedure. Since all the above holds for any fixed realization of the random string that *ALG* uses, our construction follows through for randomized algorithms as well, and produces a dominant-strategy mechanism for every realization of the random string (a.k.a "universal truthfulness").

B Multi-Unit Combinatorial Auctions

All the above assumes that we have one copy of each item. In fact, it is possible to generalize the above results to the case of Multi-Unit Combinatorial Auctions, where we have B copies of each item, and a player desires at most one copy of each item. For the sake of simplicity, we conduct the discussion for "known" single-minded players, but it is not hard to verify that this generalization holds for the known single-value case as well.

Recall that the crucial point that enabled the Iterative Wrapper of Section 3 to obtain a good approximation was that the number of iterations was low. More accurately, Lemma 2 shows that, if *ALG* is a c-approximation, and the Iterative Wrapper performs J iterations, then it obtains an $(2 \cdot J \cdot c + 1)$-approximation.

We next give a generalization to the condition that enables the Iterative Wrapper to achieve a small number of iterations, and discuss its implications for Multi-Unit Combinatorial Auctions.

We say that a set of players $T \subseteq N$ *get along* if there exists an allocation in which all players in T receive one of their desired subsets.

Definition 10 (A d-wise domain)**.** *A combinatorial auction domain is "d-wise" if for any set of players T that get along, with $|T| \geq d-1$, and any $i \notin T$: $T \cup \{i\}$ get along if and only if $X \cup \{i\}$ get along for any $X \subseteq T$ such that $|X| = d-1$.*

Lemma 4. *If a given combinatorial auctions domain is d-wise then the Iterative Wrapper of Figure 1 performs at most $d \cdot \log \bar{v}_{max} + 1$ iterations.*

Proof. Consider iteration number $d \cdot \log \bar{v}_{max} + 1$. Suppose by contradiction that there exists a loser, i, who is willing to double his value. Since the algorithm is Pareto this implies that i does not get along with the set of winners. Therefore there exists a set of players X of size at most $d-1$ such that $X \cup \{i\}$ do not get along. Thus, in any previous iteration, at least one player out of $X \cup \{i\}$ doubled his value. Since we had all together $d \cdot \log \bar{v}_{max}$ such value doublings it follows that all players in $X \cup \{i\}$ have value $\bar{v}_{max}$, contradicting the fact that i is now willing to increase his value. □

Corollary 2. *Given any c-approximation algorithm for a d-wise combinatorial auction domain, the Iterative Wrapper obtains an $O(d \cdot \log(\bar{v}_{max}) \cdot c)$-approximation in dominant strategies for known single-minded players, in polynomial time.*

It is not hard to verify that the single-minded CA domain is a 2-wise domain [14]. This can be generalized to multi-unit combinatorial auctions with single-minded players. If there are B copies of each item, we get a $(B+1)$-wise domain. By the following proposition we conclude that given any c-approximation algorithm for a multi-unit combinatorial auction with B copies of each good, the Iterative Wrapper obtains an $O(B \cdot \log(\bar{v}_{max}) \cdot c)$-approximation in dominant strategies for known single-minded players, in polynomial time.

[14] Proposition 2 actually shows that for double-minded players, the CA domain is not a d-wise domain for any $d < n$.

Name	Val	g_1	g_2	g_3	g_4
r_1	2	—			
r_2	4	—	—		
r_3	8	—	—	—	
r_4	16	—	—	—	—
d_1	8	—			
d_2	8		—		
d_3	8			—	
d_4	8				—

Iteration	r_1	r_2	r_3	r_4	d_1	d_2	d_3	d_4
1	1	1	1	1	1	1	1	1
2	2	2	2	2	1	1	1	1
3	2	4	4	4	2	1	1	1
4		4	8	8	4	2	1	1
5			8	16	8	4	2	1
6				16		8	4	2
7				16			8	4
8				16				8
9				16				

Figure 2: Example for the proof of Prop. 4. Left table describes the players' valuations; the right table describes the sequence of iterations of the wrapper on this instance. The entries are the bids of the players in the corresponding iteration, and the winners' bids are shaded.

Proposition 3. *Combinatorial auctions with single-minded players and $d-1$ copies of each item, such that any player desires at most one copy of every item, is a d-wise domain.*

Proof. Fix a set of players T with $|T| \geq d-1$ that get along, and any $i^* \notin T$. We have to show that $T \cup \{i^*\}$ get along if and only if $X \cup \{i^*\}$ get along for any $X \subseteq T$ such that $|X| = d-1$. If $T \cup \{i^*\}$ get along then clearly for any subset X of T, $X \cup \{i^*\}$ get along. Now suppose $T \cup \{i^*\}$ do not get along. Let $s_1, ..., s_{|T|}$ be the desired subsets of the players of T. Since T get along then for any item, g, $D(g) = |\{i \in T \ : \ g \in s_i\}| \leq d-1$. Since $T \cup \{i^*\}$ do not get along this implies that there exists an item $g^* \in s_{i^*}$ (where s_{i^*} is the desired subset of i^*) such that $D(g^*) = d-1$ ($d-1$ agents in T desire the item g^* that i^* also desires). Thus, if we take X to be all players $i \in T$ such that $g^* \in s_i$ we have a set X of size $d-1$ such that the agents $X \cup \{i^*\}$ do not get along. □

C On the Tightness of the Analysis of Section 3

In this section we show that our analysis of the Iterative Wrapper for known single-minded players is tight, up to a constant. To help clarifying the proof we first shows this when the wrapper uses an algorithm that always outputs the optimal allocation Following that we prove that for any constant $c \geq 1$ and large enough $\bar{v}_{max}$ there exists a c-approximation algorithm such that the Iterative Wrapper only gets $\Omega(c \cdot \log(\bar{v}_{max}))$-approximation.

Proposition 4. *Suppose that the wrapper uses an algorithm that outputs the optimal allocation. Then, for any value of $\bar{v}_{max}$, the approximation factor of the Iterative Wrapper is $\Omega(\log(\bar{v}_{max}))$.*

Before the formal proof, let us give a short intuition. The proof constructs a scenario in which the optimal welfare is $\log(\bar{v}_{max}) \cdot \bar{v}_{max}/2$ while the Iterative Wrapper outputs an allocation with welfare of $\bar{v}_{max}$, which is a factor of $\log(\bar{v}_{max})/2$ smaller, proving the claim. The example in Fig. 2 illustrates the proof when $\bar{v}_{max} = 16$. There are four items, g_1, g_2, g_3, g_4, and there are 8 players. The left table in the figure shows the players, their values, and their desired bundles. The first set of players are "row players": each player r_i ($i = 1, ..., 4$) desires a bundle that is composed of items $g_1, ..., g_i$. The second set of players are "dot players": each player d_i ($i = 1, ..., 4$) desires a bundle that is composed of a single-item, g_i. In the optimal allocation the winners are the four dot players, with a total welfare of 32. The Iterative Wrapper, however, will end up choosing player r_4 as the sole winner, and the table to the right shows the sequence of iterations that leads to this result. Each row in the right table contains the current values declared by the players, an empty cell denotes the fact that the player already retired, and a cell with a grey background denotes the fact that the

player is a tentative winner in that iteration. Since the winner is r_4 the ratio between the optimal outcome and the outcome of the iterative wrapper is exactly $\log(\bar{v}_{max})/2 = 2$, proving the claim for this case. The general proof has the same structure:

Proof. By losing a factor of 2 we can assume that $\bar{v}_{max}$ is a power of 2. We run the Iterative Wrapper using the optimal allocation rule (c=1). We build an instance in which the optimal allocation has value of $\log(\bar{v}_{max}) \cdot \bar{v}_{max}/2$ while the Iterative Wrapper outputs an allocation with welfare of $\bar{v}_{max}$, proving the claim.

The instance has $m = \log(\bar{v}_{max})$ goods and $n = 2 \cdot \log(\bar{v}_{max})$ bidders. Define the set of items to be $\{g_1, g_2, g_3, ..., g_{\log \bar{v}_{max}}\}$. Let the set of players be composed of two different subsets: (1) Dot players $\{d_1, ..., d_{\log \bar{v}_{max}}\}$, where each player d_i desires the bundle $\{g_i\}$ for value $\bar{v}_{max}/2$, and (2) Row players $\{r_1, ..., r_{\log \bar{v}_{max}}\}$, where r_i wants the bundle $\{g_1, ..., g_i\}$ for value 2^i.

Note that in any feasible allocation, at most one row player can win, and if player r_i wins, this excludes players $d_1, ..., d_i$ from winning. Thus, in the optimal solution, players $d_1, ..., d_{\log \bar{v}_{max}}$ win, with a total welfare of $\log(\bar{v}_{max}) \cdot \bar{v}_{max}/2$. We next show that when running the Iterative Wrapper, the sole winner is $r_{\log \bar{v}_{max}}$, and as his value is $2^{\log(\bar{v}_{max})} = \bar{v}_{max}$. This implies the claim.

Assume that ties are broken such that players of type d_i are preferred. The Iterative Wrapper runs in a similar manner to the example of Fig. 2: The first allocation has $d_1, ..., d_{\log \bar{v}_{max}}$ as the provisional winners. In the i'th iteration, for $1 < i \leq \log(\bar{v}_{max}) + 1$, the winners are $r_{i-1}, d_i, ..., d_{\log \bar{v}_{max}}$, player r_{i-2} retires, and the rest double their value. Note that at every such iteration, the set of winners is the optimal one according to their current values. Therefore at iteration $\log(\bar{v}_{max})+1$ the sole winner is $r_{\log \bar{v}_{max}}$ and $r_1, ..., r_{\log(\bar{v}_{max})-1}$ have already retired. The current value of $r_{\log \bar{v}_{max}}$ is $\bar{v}_{max}$. The current values of $d_1, ..., d_{\log \bar{v}_{max}}$ are $\bar{v}_{max}/2, \bar{v}_{max}/4, \bar{v}_{max}/8, ..., 1$, respectively, and the total value of these players is only $\bar{v}_{max} - 1$ (thus indeed $r_{\log \bar{v}_{max}}$ wins).

At this point, player d_1 retires, and players $d_2, ..., d_{\log \bar{v}_{max}}$ double their values. Thus, at iteration $\log(\bar{v}_{max}) + 2$ the values of the remaining dot players are $\bar{v}_{max}/2, \bar{v}_{max}/4, \bar{v}_{max}/8, ..., 2$, i.e. the sum of their values is smaller than the sum of the values of the dot players in the previous iteration. Hence player $r_{\log \bar{v}_{max}}$ still wins. This continues similarly, the sum of values of the dot players continues to decrease, and player $r_{\log \bar{v}_{max}}$ remains a winner until the end. Since his value is $\bar{v}_{max}$ and the optimal social welfare is $\log(\bar{v}_{max}) \cdot \bar{v}_{max}/2$, the claim follows. □

As a parallel to Prop. 4, we next show that the analysis is tight even when a non-optimal algorithm is used inside the wrapper.

Proposition 5. *For any $c \geq 1$ and any $\bar{v}_{max}$ that is large enough, there exists a c-approximation algorithm and an instance for which the Iterative Wrapper mechanism outputs an allocation with welfare that is only $O(\frac{1}{c \cdot \log \bar{v}_{max}})$ fraction of the optimum welfare.*

Proof. If $c \leq 2$ the claim follows from Proposition 4 so we proceed assuming that $c > 2$. By losing a constant factor we can assume that c and $\bar{v}_{max}$ are both a power of 2.

Let $x = \bar{v}_{max}/(c/2)$. Define the set of items to be $\{g_1, g_2, g_3, ..., g_{\log x}\}$. Let the set of players be composed of two different subsets: (1) Dot players $\{d_1, ..., d_{\log x}\}$, where each dot player d_i desires the bundle $\{g_i\}$ for value $\bar{v}_{max}$, (2) Row players $\{r_1, ..., r_{\log x}\}$, where r_i wants the bundle $\{g_1, ..., g_i\}$ for value 2^i.

In the optimal solution, the dot players $d_1, ..., d_{\log x}$ win, with a total welfare of $\bar{v}_{max} \cdot \log x$. We next show that there exists a c-approximation algorithm such that when running the Iterative Wrapper with that algorithm it outputs $r_{\log x}$ as the sole winner. As his value is $x = \bar{v}_{max}/(c/2)$ this implies that we get only $\frac{\bar{v}_{max}/(c/2)}{\bar{v}_{max} \cdot \log(\bar{v}_{max}/(c/2))} = \frac{2}{c \cdot (1 + \log \bar{v}_{max} - \log c)}$ fraction of the optimal welfare. If $\bar{v}_{max} = c^2$ then $\log \bar{v}_{max} = 2 \log c$. Thus the mechanism gets a fraction of $\frac{2}{c \cdot (1 + (\log \bar{v}_{max})/2)}$ and the claim follows.

The c-approximation algorithm breaks ties in favor of the dot players (players of type d_i). The Iterative Wrapper runs as follows: The first set of (provisional) winners are all players of type d_i, that is $W_1 = \{d_1, ..., d_{\log x}\}$. In the i'th iteration, for $1 < i \leq \log(x) + 1$, the winners are $W_i = \{r_{i-1}, d_i, ..., d_{\log x}\}$, player r_{i-2} retires, and the rest double their value. Note that at every such iteration, the set of winners is the optimal one according to their current values.

At iteration $\log(x)+1$, $W_{\log(x)+1} = \{r_{\log x}\}$, that is, the sole winner is $r_{\log x}$. This keeps the approximation: at this phase, $r_1, ..., r_{\log(x)-1}$ have already retired. The current value of $r_{\log x}$ is x. The current values of $d_1, ..., d_{\log x}$ are $x/2, x/4, ..., 1$, respectively, so even if they all win together the total welfare is only $x - 1$ (thus $r_{\log x}$ wins as he has a bid of x).

Players $d_1, ..., d_{\log(x)}$ will keep increasing their values until their values are $(c/2)\cdot x, (c/2)\cdot(x/2), ..., (c/2)\cdot 1$ (note that these are the values $\bar{v}_{max}, \bar{v}_{max}/2, \bar{v}_{max}/4, ..., (c/2)$), this takes $\log(c/2)+1 = \log(c)$ iterations. The approximation algorithm will output player $r_{\log x}$ as the winner in all these iterations (that is, $W_i = \{r_{\log x}\}$ for $\log(x) + 1 \leq i \leq \log(x) + 1 + \log(c)$). This is still within the approximation bound, as the optimal allocation in all these phases is at most $2 \cdot (c/2) \cdot \bar{v}_{max} = c \cdot \bar{v}_{max}$.

Let $k = \log(x) + \log(c) + 1$. As at this point $r_{\log x}$ wins and d_1 reached his value $\bar{v}_{max}$, he is not willing to increase his bid anymore and retires. So at iteration $k + i$ for $1 \leq i \leq \log(x) - 1$, the losers that do not retire are $d_{i+1}, ..., d_{\log x}$ with bids $(c/2) \cdot x, (c/2) \cdot (x/2), ..., (c/2) \cdot 2^{i+1}$, respectively. The winner is always $r_{\log x}$ (his value is within a factor of c of the optimal allocation) and after each round the agent that reaches his value retires and the other double their bids.

At iteration $\log(x) + \log(c) + \log(x) + 1$,[15] $d_{\log(x)}$ retires so now all players besides player $r_{\log x}$ have retired, and so $r_{\log x}$ is the sole winner. Since his value is $x = \bar{v}_{max}/(c/2)$ and the optimal social welfare is $\bar{v}_{max} \cdot \log(x)$, the claim follows. □

[15]Note that the number of iterations is less than $2 \cdot \log \bar{v}_{max} + 1$, as $x = \bar{v}_{max}/(c/2)$.