



# On Black-Box Transformations in Downward-Closed Environments

Warut Suksompong<sup>1,2</sup>

Published online: 23 November 2018  
© The Author(s) 2018

## Abstract

Black-box transformations have been extensively studied in algorithmic mechanism design as a generic tool for converting algorithms into truthful mechanisms without degrading the approximation guarantees. While such transformations have been designed for a variety of settings, Chawla et al. showed that no fully general black-box transformation exists for single-parameter environments. In this paper, we investigate the potentials and limits of black-box transformations in the prior-free (i.e., non-Bayesian) setting in *downward-closed* single-parameter environments, a large and important class of environments in mechanism design. On the positive side, we show that such a transformation can preserve a constant fraction of the welfare at every input if the private valuations of the agents take on a constant number of values that are far apart, while on the negative side, we show that this task is not possible for general private valuations.

**Keywords** Black-box transformation · Downward-closed · Mechanism design · Social welfare

## 1 Introduction

Mechanism design is a science of rule-making. Its goal is to design rules so that individual strategic behavior of the agents leads to desirable global outcomes. Algorithmic mechanism design, one of the initial and most well-studied branches of algorithmic game theory, studies the tradeoff between optimizing the global outcome, respecting the incentive constraints for individual agents, and maintaining the computational tractability of the mechanism [13]. A major line of work in algorithmic

---

An abstract appeared in Proceedings of the 10th International Symposium on Algorithmic Game Theory

✉ Warut Suksompong  
[warut@cs.stanford.edu](mailto:warut@cs.stanford.edu)

<sup>1</sup> Department of Computer Science, Stanford University, Stanford, CA USA

<sup>2</sup> Department of Computer Science, University of Oxford, Oxford OX1 3QD, UK

mechanism design involves taking a setting where the optimization problem is computationally intractable, and designing computationally tractable mechanisms that yield a good global outcome and such that the agents have a truth-telling incentive. Ideally, the mechanisms would match the best-known approximation guarantees for computationally tractable optimization algorithms in that setting. In other words, we want to obtain truthfulness from agents in as many settings as possible without having to pay for more computation.

In the past two decades, this goal of algorithmic mechanism design has been met in a wide range of prior-free as well as Bayesian settings. For instance, Briest et al. [3] showed how to transform pseudopolynomial algorithms for several problems, including knapsack, constrained shortest path, and scheduling, into monotone fully polynomial time approximation schemes (FPTAS), which lead to efficient and truthful auctions for these problems. Lavi and Swamy [11] constructed a general reduction technique via linear programming that applies to a wide range of problems. The widespread success of designing computationally tractable mechanisms with optimal approximation guarantees has raised the question of whether there exists a generic method for transforming any computationally tractable algorithm into a computationally tractable mechanism without degrading the approximation guarantee. Such a method would not be allowed access to the description of the algorithm but instead would only be able to query the algorithm at specific inputs, and is therefore known as a “black-box transformation”.

An important work that demonstrates a limit of the powers of black-box transformations was done by Chawla et al. [5], who showed among other things that no fully general black-box transformation exists for single-parameter environments in the prior-free setting. In particular, for any transformation, there exists an algorithm (along with a feasibility set) such that the transformation degrades the approximation ratio of the algorithm by at least a polynomial factor. The result holds even when the private valuations can take on only two values; Chawla et al. provided a construction with two private valuations  $l < h$  satisfying  $h/l = n^{7/10}$ , where  $n$  is the number of agents. Pass and Seth [14] extended this result by allowing the transformation access to the feasibility set while assuming the existence of cryptographic one-way functions.

Even though no fully general black-box transformation exists for single-parameter environments, it is still conceivable that there are transformations that work for certain large subclasses of such environments. One important subclass, which is the main subject of our paper, is that of *downward-closed* environments, i.e., environments in which any subset of a feasible allocation is also feasible. The construction used by Chawla et al. [5], later built upon by Pass and Seth [14], relies heavily on the non-downward-closedness of the feasibility set. The construction only includes three feasible allocations, and it is crucial that the transformation cannot arbitrarily “round down” the allocations as it would be able to if the feasibility set were downward-closed. Since downward-closed environments occur in a wide variety of settings in mechanism design, including knapsack auctions and combinatorial auctions, we find the question that we study to be a natural and important one. We consider such settings and assume, crucially, that the black-box transformation is aware that the feasible set is downward-closed. As a result, when the transformation makes a query to the algorithm, it can potentially learn many more feasible allocations than merely

the one it obtains. In this paper, we investigate the potentials and limits of black-box transformations when they are endowed with this extra power.

## 1.1 Our results

Throughout the paper, we consider the prior-free (i.e., non-Bayesian) setting. In Section 3, we show the limits of black-box transformations in downward-closed environments. We prove that such transformations cannot preserve the full welfare at every input, even when the private valuations can take on only two arbitrary values (Theorem 1). Preserving a constant fraction of the welfare pointwise is impossible if the ratio between the two values  $l < h$  is sublinear, i.e.,  $h/l \in O(n^\alpha)$  for  $\alpha \in [0, 1)$ , where  $n$  is the number of agents (Theorems 2 and 3), while preserving the approximation ratio is also impossible if the values are within a constant factor of each other and the transformation is restricted to querying inputs of Hamming distance  $o(n)$  away from its input (Theorem 4).

In Section 4, we show the powers of black-box transformations in downward-closed environments. We prove that when the private valuations can take on only a constant number of values, each pair of values separated by a ratio of  $\Omega(n)$ , it becomes possible for a transformation to preserve a constant fraction of the welfare pointwise, and therefore the approximation ratio as well (Theorem 5). The same is also true if the private valuations are all within a constant factor of each other (Theorem 8). Combined with the negative results, this gives us a complete picture of transformations that preserve the welfare pointwise for any number of input values. Not only are these results interesting in their own right, but they also demonstrate the borders of the negative results that we can hope to prove.

**Table 1** Summary of our results for the case where the private valuations take on two values  $l < h$

	100% pointwise	Constant fraction pointwise	100% approx ratio	Constant fraction approx ratio
$h/l = n^{7/10}$ ; $\mathcal{F}$ cannot be queried, not downward-closed	No [5]	No [5]	No [5]	No [5]
$h/l \in \Omega(n)$ ; $\mathcal{F}$ can be queried, downward-closed	No (Theorem 1)	Yes (Theorem 5)	?	Yes (Theorem 5)
$h/l \in \Theta(1)$ ; $\mathcal{F}$ can be queried, downward-closed	No (Theorem 1)	No (Theorem 2)	No if restricted to Hamming distance $f(n) \in o(n)$ , $\mathcal{F}$ cannot be queried (Theorem 4)	Yes (Theorem 8)
$h/l \in O(n^\alpha)$ for $\alpha \in (0, 1)$ ; $\mathcal{F}$ can be queried, downward-closed	No (Theorem 1)	No (Theorem 3)	?	?

The results can be generalized to any constant-size range of private valuations

The results are summarized in Table 1 for the case where the private valuations can take on two values, but they can be generalized to any constant-size range of private valuations as well.

## 1.2 Related work

Besides the works already mentioned, black-box transformations have been obtained in a variety of other prior-free and Bayesian settings. In the prior-free setting, Goel et al. [7] presented a reduction for symmetric single-parameter problems with a logarithmic loss in approximation, and later Huang et al. [10] improved the reduction to obtain arbitrarily small loss. Dughmi and Roughgarden [6] designed a reduction for the class of multi-parameter problems that admit an FPTAS and can be encoded as a packing problem, while Babaioff et al. [1] considered reductions for single-valued combinatorial auction problems. Reductions that preserve the approximation guarantees have also been obtained in the single-parameter Bayesian setting by Hartline and Lucier [9], and their work was later extended to multi-parameter settings by Bei and Huang [2], Cai et al. [4], and Hartline et al. [8].

## 2 Preliminaries

We will be concerned with single-parameter environments. Such an environment consists of some number  $n$  of agents. Each agent  $i$  has a private valuation  $v_i \in \mathbb{R}$ , its value “per unit of stuff” that it gets. In addition, there is a feasibility set  $\mathcal{F}$ , which specifies the allocations that can be made to the agents. Each element of  $\mathcal{F}$  is a vector  $(x_i)_{i=1}^n$ , where  $x_i \in \mathbb{R}$  denotes the “amount of stuff” given to agent  $i$ . For instance, in single-item auctions,  $\mathcal{F}$  consists of the vectors with  $x_i \in \{0, 1\}$  and  $\sum_{i=1}^n x_i = 1$ . A more general and well-studied type of auctions is called knapsack auctions, in which each agent is endowed with a public size  $w_i$  along with its private valuation  $v_i$ , and the seller has some public capacity  $W$ . The feasibility set of a knapsack auction consists of the vectors with  $x_i \in \{0, 1\}$  and  $\sum_{i=1}^n w_i x_i \leq W$ . In this paper, we assume that the feasibility set  $\mathcal{F}$  is *downward-closed*, which means that if we take a feasible allocation and decrease the amount of stuff given to one of the agents in the allocation, then the resulting allocation is also feasible. Downward-closedness is an assumption that holds in many natural settings, including the auctions above. We assume in addition that each  $x_i$  is either 0 or 1; this is also the case for both of the aforementioned auctions.

**Algorithms** An *algorithm* (or *allocation rule*)  $\mathcal{A}$  is a function that takes as input a valuation vector  $\mathbf{v} = (v_i)_{i=1}^n$  and outputs an allocation  $\mathbf{x} = (x_i)_{i=1}^n$ . We will consider the social welfare objective—the *welfare* of  $\mathcal{A}$  at  $\mathbf{v}$  is given by  $\mathbf{v} \cdot \mathbf{x} = v_1 x_1 + \dots + v_n x_n$ , where  $\mathbf{x} \in \mathcal{F}$  is the allocation that  $\mathcal{A}$  returns at  $\mathbf{v}$ . We denote by  $OPT_{\mathcal{F}}(\mathbf{v})$  the maximum welfare at valuation vector  $\mathbf{v}$  over all allocations in  $\mathcal{F}$ . The (*worst-case*) *approximation ratio* of  $\mathcal{A}$  is given by  $\text{approx}_{\mathcal{F}}(\mathcal{A}) = \min_{\mathbf{v}} \frac{\mathcal{A}(\mathbf{v})}{OPT_{\mathcal{F}}(\mathbf{v})}$ , where we slightly abuse notation and use  $\mathcal{A}(\mathbf{v})$  to denote the allocation returned by  $\mathcal{A}$  at  $\mathbf{v}$  as

well as the welfare of that allocation at  $\mathbf{v}$ . Note that by definition,  $\text{approx}_{\mathcal{F}}(\mathcal{A}) \leq 1$  for all  $\mathcal{F}$  and  $\mathcal{A}$ .

**Transformations** A *transformation*  $\mathcal{T}$  is an algorithm that has black-box access to some other algorithm  $\mathcal{A}$ , i.e., it can make queries to  $\mathcal{A}$ . In each query,  $\mathcal{T}$  specifies a valuation vector  $\mathbf{v}$  and obtains the allocation that  $\mathcal{A}$  returns at  $\mathbf{v}$ . We write  $\mathcal{T}(\mathcal{A})$  for a transformation  $\mathcal{T}$  with access to the algorithm  $\mathcal{A}$ . Importantly, we assume that  $\mathcal{T}$  has the knowledge that the feasibility set  $\mathcal{F}$  is downward-closed. For the strongest possible negative results, we assume whenever possible that (i)  $\mathcal{T}$  can make a polynomial number of queries to ask whether a particular allocation belongs to  $\mathcal{F}$ ; (ii)  $\mathcal{T}$  is adaptive, i.e., it can adjust its next query based on the responses it received for previous queries; and (iii)  $\mathcal{T}$  is randomized. For the strongest positive results, our transformation  $\mathcal{T}$  does not make queries about  $\mathcal{F}$  and is also not adaptive. We will be clear about our assumptions on  $\mathcal{T}$  for each result.

**Mechanisms** A *mechanism* is a procedure that consists of eliciting declared private valuations  $(b_i)_{i=1}^n$  from the agents, and then applying an allocation rule and a payment rule on the elicited valuations. The allocation rule determines the allocation  $(x_i)_{i=1}^n$  and the payment rule determines the prices  $(p_i)_{i=1}^n$  to charge the agents. We are interested in transformations that, when coupled with any algorithm, lead to *truthful* mechanisms, meaning that it is always in the best interest for each agent  $i$  to declare the true valuation  $v_i$  to the mechanism, no matter what the other agents do. A seminal result by Myerson [12] states that an allocation rule can be supplemented with a payment rule to yield a truthful mechanism exactly when the allocation rule is *monotone*. Monotonicity of an allocation rule means that if an agent increases its declared valuation while the declared valuations of the remaining agents stay fixed, then the agent is allocated at least as much stuff as before by the allocation rule. Therefore, the transformations that yield truthful mechanisms are exactly the ones that constitute a monotone allocation rule for any algorithm.

**Properties of transformations** We call a transformation  $\mathcal{T}$  *monotone* if  $\mathcal{T}(\mathcal{A})$  is a monotone allocation rule for any algorithm  $\mathcal{A}$ . Furthermore,  $\mathcal{T}$  is called *welfare-preserving* if  $\mathcal{T}(\mathcal{A})$  preserves the welfare of  $\mathcal{A}$  at every input for any algorithm  $\mathcal{A}$ , and *constant-fraction welfare-preserving* if  $\mathcal{T}(\mathcal{A})$  preserves a constant fraction of the welfare of  $\mathcal{A}$  at every input for any algorithm  $\mathcal{A}$ . Similarly,  $\mathcal{T}$  is *approximation-ratio-preserving* if  $\mathcal{T}(\mathcal{A})$  preserves the approximation ratio of  $\mathcal{A}$  for any algorithm  $\mathcal{A}$ , and *constant-fraction approximation-ratio-preserving* if  $\mathcal{T}(\mathcal{A})$  preserves a constant fraction of the approximation ratio of  $\mathcal{A}$  for any algorithm  $\mathcal{A}$ . Note that a (constant-fraction) welfare-preserving transformation is also (constant-fraction) approximation-ratio-preserving.

### 3 Negative Results

In this section, we consider the limits of black-box transformation in downward-closed environments. First, we show that no monotone black-box transformation

preserves, up to a constant factor, the welfare of any original algorithm  $A$  pointwise. We then show that if a monotone black-box transformation preserves the approximation ratio of any given input algorithm  $A$ , then on some input  $\mathbf{v}$  it must query  $A$  on an input that has Hamming distance  $\Omega(n)$  from  $\mathbf{v}$ .

### 3.1 Welfare-preserving transformations

We begin by considering the highest possible benchmark for the transformation: preserving the full welfare of any algorithm at every input. Our first theorem shows that this benchmark is impossible to fulfill even when the private valuations can take on only two arbitrary values.

**Theorem 1** *Let  $l < h$  be arbitrary values (possibly depending on  $n$ ), and assume that the private valuation of each agent is either  $l$  or  $h$ . There does not exist a polynomial-time, monotone, welfare-preserving transformation, even when the transformation is allowed to be randomized and adaptive and make a polynomial number of queries to  $\mathcal{F}$ .*

Before we go into the formal proof, we give a high-level intuition. We will consider a class of algorithms from which one algorithm  $\mathcal{A}$  is selected randomly. For each algorithm  $\mathcal{A}$ , our feasibility set contains two maximal allocations  $C$  and  $D$  along with some low-value allocations that we insert to prevent the transformation from learning new information when it makes queries on allocations with low welfare. The allocation  $C$  is only returned at a “special input”  $B_1$ , and the allocation itself as well as the special input depends on the algorithm  $\mathcal{A}$  we choose from the class. At any other input, the allocation  $D$  is returned. Since  $C$  yields higher welfare than  $D$  at  $B_1$ , the transformation must return  $C$  at  $B_1$  in order to preserve the full welfare. By considering a chain of inputs starting from  $B_1$ , each consecutive pair of which differ in one position, and using the monotonicity of the transformation, we can show that at an input that is “far away” from  $B_1$ , the transformation still needs to know the allocation  $C$  in order to preserve the full welfare. However, because of the randomization, the probability the transformation can discover either the allocation  $C$  or the special input  $B_1$  when it is given the faraway input is exponentially low, meaning that the transformation cannot achieve its goal.

*Proof* Assume first that the transformation  $\mathcal{T}$  is deterministic. Suppose that the input is of length  $n = 4m$ . The algorithm  $\mathcal{A}$  will be chosen randomly. To begin, we define the preliminary algorithm  $\mathcal{A}'$  as follows.

- At input  $B_1 = \overbrace{hh \dots h}^{2m} \overbrace{hh \dots h}^m \overbrace{ll \dots l}^m$ ,  $\mathcal{A}'$  returns output  $C = \underbrace{10110 \dots 1}_{m+1 \text{ are } 1\text{'s}} \underbrace{00 \dots 0}_m \underbrace{11 \dots 1}_m$ , where the  $m + 1$  1's in the first  $2m$  positions are uniformly randomized. Note that the randomization is in the step of choosing the algorithm  $\mathcal{A}'$ , but the resulting algorithm  $\mathcal{A}'$  itself is a deterministic algorithm. We call this input the *special input*;

- At any other input,  $\mathcal{A}'$  returns  $D = \overbrace{00 \dots 0}^{2m} \overbrace{11 \dots 1}^m \overbrace{11 \dots 1}^m$ .

Besides the two allocations  $C$  and  $D$  and their subsets, we also insert all allocations with  $m/2 - 1$  1's in the first  $2m$  positions and all 1's in the last  $m$  positions along with subsets of these allocations into  $\mathcal{F}$ .

In the real algorithm  $\mathcal{A}$ , we permute uniformly at random the last  $3m/2$  positions of the inputs as well as the corresponding allocations. Again, this permutation is only for choosing the algorithm  $\mathcal{A}$ , which is a deterministic algorithm.

Consider any algorithm  $\mathcal{A}$  that we might choose, and assume without loss of generality that in the special input of this algorithm, the  $l$ 's are in the last  $m$  positions. To preserve the welfare at input  $B_1$ ,  $\mathcal{T}$  must return  $C = \mathcal{A}(B_1)$  itself, since returning any strict subset of  $C$  or returning  $D$  (or any subset of  $D$ ) would yield a lower welfare. Indeed, the welfare of  $C$  at  $B_1$  is  $(m+1)h + ml$ , while the welfare of  $D$  at this input is only  $mh + ml$ .

Next, consider the input  $B_2 = \overbrace{hh \dots h}^{2m} \overbrace{hh \dots hl}^m \overbrace{ll \dots l}^m$ , with the only change from  $B_1$  being in the rightmost position of the middle block. By monotonicity,  $\mathcal{T}$  must return 0 in that position. In order to preserve the welfare at  $B_2$ ,  $\mathcal{T}$  must return a subset of  $C$ , since otherwise it would have to return a strict subset of  $D$ , which would yield a lower welfare.

Now, consider inputs  $B_3 = \overbrace{hh \dots h}^{2m} \overbrace{hh \dots hll}^m \overbrace{ll \dots l}^m$ ,  $B_4 = \overbrace{hh \dots h}^{2m} \overbrace{hh \dots hlll}^m \overbrace{ll \dots l}^m$ , and so on with one extra  $l$  in each input, up to  $B_{m/2+1} = \overbrace{hh \dots h}^{2m} \overbrace{hh \dots hll \dots l}^{m/2} \overbrace{ll \dots l}^{m/2}$ . By a similar argument,  $\mathcal{T}$  must return a subset of  $C$  at all of these inputs. In particular,  $\mathcal{T}$  must return a subset of  $C$  at  $B_{m/2+1}$ .

In order to preserve the welfare at  $B_{m/2+1}$ ,  $\mathcal{T}$  must return at least  $m/2$  1's in the first  $2m$  positions. If  $\mathcal{T}$  tries to find such an allocation by querying  $\mathcal{F}$ , then since the positions of the  $m+1$  1's in the first  $2m$  positions are chosen randomly, the probability of success for each query is at most  $\frac{\binom{m+1}{m/2}}{\binom{2m}{m/2}} < \frac{1}{\text{poly}(n)}$ . Note also that because of the “fake” allocations that we insert into  $\mathcal{F}$ , any query to  $\mathcal{F}$  that contains at most  $m/2 - 1$  1's in the first  $2m$  positions does not give  $\mathcal{T}$  any information. Hence  $\mathcal{T}$  will succeed within a polynomial number of queries with less than constant probability.

Alternatively,  $\mathcal{T}$  might try to find the special input  $B_1$  by querying  $\mathcal{A}$ . In order to find the special input  $B_1$ ,  $\mathcal{T}$  must correctly choose  $m/2$  out of the  $3m/2$  positions to change to  $h$ . However, recall that when we define the real algorithm  $\mathcal{A}$  based on the preliminary algorithm  $\mathcal{A}'$ , we randomly permute the last  $3m/2$  positions of the inputs and their corresponding allocations. The probability of success of  $\mathcal{T}$  for each query is therefore at most  $\frac{1}{\binom{3m/2}{m/2}} < \frac{1}{\text{poly}(n)}$ . Moreover, each unsuccessful query to  $\mathcal{A}$  gives  $\mathcal{T}$  no new information. Hence  $\mathcal{T}$  will again succeed within a polynomial number of queries with less than constant probability. Combined with the previous paragraph, this means that the probability that  $\mathcal{T}$  succeeds within a polynomial number of queries is less than constant if  $\mathcal{T}$  is deterministic.

Finally, assume that  $\mathcal{T}$  is allowed to be randomized. As before,  $\mathcal{T}$  still has to guess correctly either the special input or an allocation that returns at least  $m/2$  1's in the first  $2m$  positions, and any unsuccessful query gives  $\mathcal{T}$  no new information. Since we choose the positions of the  $m + 1$  1's in the first  $2m$  positions uniformly at random in the preliminary algorithm  $\mathcal{A}'$ , the probability that each guess of  $\mathcal{T}$  (no matter whether deterministic or randomized) is successful is still at most  $\frac{\binom{m+1}{m/2}}{\binom{2m}{m/2}} < \frac{1}{\text{poly}(n)}$ . Likewise, since we permute the last  $3m/2$  positions of the inputs and their corresponding allocations uniformly at random when we define the real algorithm  $\mathcal{A}$  based on the preliminary algorithm  $\mathcal{A}'$ , the probability that each guess of  $\mathcal{T}$  (no matter whether deterministic or randomized) is successful is still at most  $\frac{1}{\binom{3m/2}{m/2}} < \frac{1}{\text{poly}(n)}$ . We therefore conclude that the probability of success of  $\mathcal{T}$  in guessing the special input or an allocation that returns at least  $m/2$  1's in the first  $2m$  positions cannot increase even if it randomizes its choices.  $\square$

### 3.2 Constant-fraction welfare-preserving transformations

Even though Theorem 1 shows that it is impossible for a transformation to preserve the full welfare pointwise, it would still be interesting if the transformation can preserve a *constant fraction* of the welfare pointwise. However, as we show in this subsection, it turns out that this weaker requirement is also impossible to satisfy. Our next two theorems show that preserving a constant fraction pointwise is impossible when the ratio  $h/l$  is sublinear, i.e.,  $h/l \in O(n^\alpha)$  for some  $\alpha \in [0, 1)$ . We first consider the case where  $h/l$  is constant (Theorem 2), and later generalize to  $h/l \in O(n^\alpha)$  for some  $\alpha \in [0, 1)$  (Theorem 3). Together with Theorem 5, which exhibits an example of a constant-fraction welfare-preserving transformation when  $h/l \in \Omega(n)$ , we have a complete picture of constant-fraction welfare-preserving transformations when there are two input values.

**Theorem 2** *Let  $l < h$  be such that  $h/l$  is constant, and assume that the private valuation of each agent is either  $l$  or  $h$ . There does not exist a polynomial-time, monotone, constant-fraction welfare-preserving transformation, even when the transformation is allowed to be randomized and make a polynomial number of queries to  $\mathcal{F}$ .*

The proof of Theorem 2 uses similar ideas to that of Theorem 1 but contains differences in the execution.

*Proof* Assume first that the transformation  $\mathcal{T}$  is deterministic. Suppose that the input is of length  $n = m^6 + m^3$ . Note that the sets  $\text{poly}(m)$  and  $\text{poly}(n)$  are identical.

Let  $X$  denote the set of inputs with  $m^5 l$ 's in the first  $m^6$  positions followed by  $m^3 h$ 's, and let  $Y$  denote the set of inputs with  $m^5 - ml$ 's in the first  $m^6$  positions, followed by  $m^3 h$ 's. We have  $|X| = \binom{m^6}{m^5}$  and  $|Y| = \binom{m^6}{m^5 - m}$ . Since  $|X| > |Y| \cdot \text{poly}(n)$ , there exists an input in  $X$  that is not in the (polynomially long) query list of  $\mathcal{T}$  for any input in  $Y$ . Assume without loss of generality that  $B_1$ , defined below, is one such input.



Consider the algorithm  $\mathcal{A}$  as follows:

- At input  $B_1 = \overbrace{hh \dots h}^{m^6-m^5} \overbrace{ll \dots l}^{m^5} \overbrace{hh \dots h}^{m^3}$ ,  $\mathcal{A}$  returns  $C = \overbrace{00 \dots 0}^{m^6-m^5} \overbrace{10110 \dots 1}^{m^4 \text{ are } 1\text{'s}} \overbrace{00 \dots 0}^{m^3}$ . For now, we assume an arbitrary set of positions for the  $m^4$  1's in the  $m^5$  positions of the middle block; we will choose a particular set later. We call  $B_1$  the *special input*, and the corresponding allocation  $C$  the *special allocation*;
- At any other input,  $\mathcal{A}$  returns  $D = \overbrace{00 \dots 0}^{m^6-m^5} \overbrace{00 \dots 0}^{m^5} \overbrace{11 \dots 1}^{m^3}$ .

For large enough  $m$ , to preserve a constant fraction of the welfare at input  $B_1$ ,  $\mathcal{T}$  cannot return a subset of  $D$ . Hence  $\mathcal{T}$  must return a subset of  $C = \mathcal{A}(B_1)$ .

Consider the input  $B_2 = \overbrace{hh \dots h}^{m^6-m^5} \overbrace{hll \dots l}^{m^5} \overbrace{hh \dots h}^{m^3}$ , with the only change from  $B_1$  being in the leftmost position of the middle block. (Here we choose the leftmost position because this position of  $C$  contains a 1 in the particular choice of  $C$  above; otherwise we choose any position of  $C$  that contains a 1.) By monotonicity,  $\mathcal{T}$  must return a 1 in the middle block for  $B_2$ , so it cannot return a subset of  $D$ . Moreover, for large enough  $m$ , to preserve a constant fraction of the welfare at  $B_2$ ,  $\mathcal{T}$  must return at least  $m^2$  1's in the middle block. In particular, there is still a 1 corresponding to an  $l$  in the middle block.

Similarly, we can define inputs  $B_3, B_4, \dots, B_{m+1}$  so that  $B_i$  has  $i - 1$   $h$ 's in the middle block and there is still a 1 corresponding to an  $l$  in the middle block. For each of these inputs,  $\mathcal{T}$  must return at least  $m^2$  1's in the middle block. Note also that  $B_{m+1} \in Y$ .

Now, the special allocation  $C$  is at  $B_1$ , and by our assumption above,  $\mathcal{T}$  does not discover  $C$  by querying  $\mathcal{A}$  at  $B_1$  when it is presented with  $B_{m+1} \in Y$ . The only other possibility for  $\mathcal{T}$  to discover  $C$  is to query  $\mathcal{F}$ . There are  $\binom{m^5}{m}$  inputs of  $Y$  whose first  $m^6 - m^5$  positions are all  $h$ 's, and these are the only inputs at which  $\mathcal{T}$  can benefit from a “successful” query to  $\mathcal{F}$ . When  $\mathcal{T}$  makes a query at each of these inputs, it must pick an allocation with at least  $m^2$  1's in the  $m^5$  positions. (Similarly to the proof of Theorem 1, we insert all allocations with at most  $m^2 - 1$  1's in the  $m^5$  positions into  $\mathcal{F}$  to ensure that  $\mathcal{T}$  learns no new information by querying such allocations.) From the perspective of us preventing the transformation  $\mathcal{T}$  from achieving its goal, this rules out at most  $\binom{m^5-m^2}{m^4-m^2}$  allocations. The total number of possible allocations  $C$  that we can choose is  $\binom{m^5}{m^4}$ . Since  $\binom{m^5}{m} \cdot \binom{m^5-m^2}{m^4-m^2} \cdot \text{poly}(n) < \binom{m^5}{m^4}$ , this means that for some choice of 1's in  $m^4$  out of the  $m^5$  positions in the middle block,  $\mathcal{T}$  does not succeed in finding an allocation with at least  $m^2$  1's. By making this choice of  $C$ , we ensure that  $\mathcal{T}$  cannot succeed within a polynomial number of queries.

Finally, assume that  $\mathcal{T}$  is allowed to be randomized. We choose an input  $B_1 \in X$  for which the total probability of querying  $X$  given any input in  $Y$  is exponentially small; such an input must exist since  $|X| > |Y| \cdot \text{poly}(n)$ . The choice of the allocation  $C$  is made in the same way as before.  $\square$

Using a similar construction, we can generalize the impossibility result to the case where  $h/l \in O(n^\alpha)$  for any  $\alpha \in [0, 1)$ .

**Theorem 3** *Let  $l < h$  be such that  $h/l \in O(n^\alpha)$  for some constant  $\alpha \in [0, 1)$ , and assume that the private valuation of each agent is either  $l$  or  $h$ . There does not exist a polynomial-time, monotone, constant-fraction welfare-preserving transformation, even when the transformation is allowed to be randomized and make a polynomial number of queries to  $\mathcal{F}$ .*

*Proof* We extend the example for the case where  $h/l$  is constant (Theorem 2). Let  $b > c > d > e \geq 3$  be integer constants that we will choose later. Suppose that the three blocks have length  $m^b, m^c$ , and  $m^e$ , respectively, and that there are  $m^d$  1's in the middle block. We construct inputs  $B_1, B_2, \dots, B_{m+1}$  as before. In order for the same argument to go through, we need three conditions:

1. For the transformation to necessarily return a subset of  $C$  at the special input  $B_1$ , it is sufficient to have  $m^d = \omega(m^e(m^b + m^c + m^e)^\alpha)$ . This is because the welfare from the allocation  $C$  at  $B_1$  is  $m^d \cdot l$ , while the welfare from the allocation  $D$  at  $B_1$  is  $m^e \cdot h \leq k \cdot m^e(m^b + m^c + m^e)^\alpha \cdot l$  for some constant  $k$ . To satisfy the asymptotic relation, one may choose  $b > d > e$  so that  $d > e + b\alpha$ . Since  $\alpha < 1$ , such a choice is possible.
2. To guarantee the existence of an input in  $X$  that is not in the (polynomially long) query list of  $\mathcal{T}$  for any input in  $Y$ , we need  $\binom{m^b+m^c}{m^c-m} \cdot \text{poly}(n) < \binom{m^b+m^c}{m^c}$ . This is because  $|X| = \binom{m^b+m^c}{m^c}$  and  $|Y| = \binom{m^b+m^c}{m^c-m}$ . The inequality is equivalent to

$$\frac{(m^b + m)(m^b + m - 1) \dots (m^b + 1)}{m^c(m^c - 1) \dots (m^c - m + 1)} > \text{poly}(n).$$

For each  $i = 1, 2, \dots, m$ , the ratio between the  $i$ th term of the numerator and the  $i$ th term of the denominator is at least  $m^{b-c}/2 > m^{(b-c)/2}$  for large enough  $m$ . Hence the inequality holds for any  $b > c$ .

3. To ensure that there is a valid choice of  $C$ , we need  $\binom{m^c}{m} \cdot \binom{m^c-m^{e-1}}{m^d-m^{e-1}} \cdot \text{poly}(n) < \binom{m^c}{m^d}$ . The reasoning follows that in the penultimate paragraph of the proof of Theorem 3, which corresponds to the special case  $(c, d, e) = (5, 4, 3)$ . The inequality is equivalent to

$$\frac{m!}{(m^d)(m^d - 1) \dots (m^d - m + 1)} \cdot \frac{(m^c - m) \dots (m^c - m^{e-1} + 1)}{(m^d - m) \dots (m^d - m^{e-1} + 1)} > \text{poly}(n).$$

The first fraction on the left-hand side is at least  $1/m^{dm}$ . For each  $i = 1, \dots, m^{e-1} - m$ , the ratio between the  $i$ th term of the numerator and the  $i$ th term of the denominator of the second fraction is at least  $m^{c-d}/2 > m^{(c-d)/2}$  for large enough  $m$ . Hence the second fraction is at least  $m^{(c-d)(m^{e-1}-m)/2}$ . The product of the two fractions is therefore at least  $(m^m)^{[(c-d)(m^{e-2}-1)/2]-d} > \text{poly}(n)$ . Hence the inequality holds for any  $c > d > e$ .

Therefore we can choose  $b, c, d, e$  so that all three conditions hold.  $\square$

Note that the examples so far cannot be used to show the non-existence of a monotone (constant-fraction) approximation-ratio-preserving transformation. Indeed, consider the transformation that simply returns the canonical allocation  $D$ . The points at which this transformation fails to preserve the welfare of the algorithm are points at which the algorithm is optimal, and elsewhere the algorithm is far from optimal, implying that the approximation ratio is preserved.

### 3.3 Approximation-ratio-preserving transformations

In this subsection, we consider a weaker benchmark than preserving full welfare pointwise: preserving the approximation ratio. We show that this benchmark is still impossible to satisfy if we restrict the transformation  $\mathcal{T}$  to querying inputs at Hamming distance less than some function  $f(n) \in o(n)$  from its input, and disallow  $\mathcal{T}$  from querying  $\mathcal{F}$ . Note that our transformations for two or more input values that are far apart (Theorems 5, 6, and 9) only query inputs at constant Hamming distance from the given input. The following result therefore implies that such transformations or minor modifications thereof cannot be approximation-ratio-preserving.

**Theorem 4** *Let  $l < h$  be such that  $h/l$  is constant, and assume that the private valuation of each agent is either  $l$  or  $h$ . Let  $f(n) \in o(n)$ . There does not exist a polynomial-time, monotone, approximation-ratio-preserving transformation  $\mathcal{T}$ . The transformation  $\mathcal{T}$  is allowed to be randomized and adaptive, but it cannot make queries to  $\mathcal{F}$  and can only make queries to  $\mathcal{A}$  on inputs that are of Hamming distance less than  $f(n)$  from the original input.*

*Proof* Suppose that the input is of length  $n = 2m$ , and consider the algorithm  $\mathcal{A}$  as follows:

- At any input with at most  $m + f(n)h$ 's,  $\mathcal{A}$  returns  $\overbrace{00 \dots 0}^m \overbrace{11 \dots 1}^m$ ;
- At any other input,  $\mathcal{A}$  returns  $\overbrace{11 \dots 1}^m \overbrace{00 \dots 0}^m$ .

One can check that  $\text{approx}_{\mathcal{F}}(\mathcal{A}) = l/h$ . Let  $B_1$  denote the input  $\overbrace{hh \dots h}^m \overbrace{ll \dots l}^m$ .

We have  $\mathcal{A}(B_1) = \overbrace{00 \dots 0}^m \overbrace{11 \dots 1}^m$ . At input  $B_1$ , the transformation  $\mathcal{T}$  cannot discover the other (undominated) allocation because of the Hamming distance restriction. Hence it must return a subset of  $\mathcal{A}(B_1)$ . Moreover, since the approximation ratio of  $\mathcal{A}$  is worst at input  $B_1$ ,  $\mathcal{T}$  must return exactly  $\mathcal{A}(B_1)$ .

Consider the input  $B_2 = \overbrace{hh \dots h}^m \overbrace{hll \dots l}^m$ , with the only change from  $B_1$  being in the leftmost position of the second half. By monotonicity,  $\mathcal{T}$  must return a subset of  $\mathcal{A}(B_1)$  at  $B_2$ . Moreover, for large enough  $n$ , to preserve the approximation ratio,  $\mathcal{T}$  must return at least one 1 on  $l$  in the second half.

Similarly, we can define inputs  $B_3, B_4, \dots, B_{m+2f(n)+1}$  so that  $B_i$  has  $i - 1$   $h$ 's in the second half and there is still a 1 corresponding to an  $l$  in the second half. A

sufficient condition to guarantee a 1 on an  $l$  in the second half is that putting 1's on all  $h$ 's in the second half is not enough to match the approximation ratio  $l/h$ . That is,  $\frac{2f(n) \cdot h}{mh} < \frac{l}{h}$ . Since  $f(n) \in o(n)$ , we can choose  $n$  large enough so that this condition is satisfied.

For each of the inputs  $B_3, B_4, \dots, B_{m+2f(n)+1}$ ,  $\mathcal{T}$  must return a subset of  $\mathcal{A}(B_1)$ . At input  $B_{m+2f(n)+1}$ , however,  $\mathcal{T}$  cannot discover the allocation  $\mathcal{A}(B_1)$  because of the Hamming distance restriction. Hence  $\mathcal{T}$  cannot succeed.  $\square$

If we are only interested in preserving a constant factor of the approximation ratio, then Theorem 8 shows that this is possible in the same setting of  $h/l$  constant, and Theorem 5 shows that it is also possible when  $h/l \in \Omega(n)$ . It is not clear whether a negative result can be obtained when  $h/l \in O(n^\alpha)$  for some  $\alpha \in (0, 1)$ .

## 4 Positive Results

In this section, we consider the powers of black-box transformations in downward-closed environments. We show that when values are either high or low, and the ratio between high and low is  $\Omega(n)$ , then there is a monotone transformation that gives a constant approximation to the welfare of any given algorithm pointwise, and therefore also preserves the approximation ratio up to a constant factor. This can be generalized to any constant number of values, and the transformation can be modified so that it also preserves full welfare at a constant fraction of the inputs. While these results are of independent interest, they also serve to demonstrate the limitations of extending the negative results in Section 3. For the strongest possible results, we exhibit transformations that do not query  $\mathcal{F}$  or operate adaptively.

### 4.1 Two values

We begin by showing that when the private valuations take on two values that are far apart, there exists a transformation that preserves a constant fraction of the welfare at each input. This contrasts with the negative result when the values are close to each other (Theorem 3).

**Theorem 5** *Let  $l < h$  be such that  $h/l \in \Omega(n)$ , and assume that the private valuation of each agent is either  $l$  or  $h$ . There exists a polynomial-time, monotone, constant-fraction welfare-preserving transformation.*

*Proof* First we give a high-level intuition of the transformation. A monotone transformation needs to ensure that for any two adjacent inputs, it does not simultaneously occur that a 0 appears on  $h$  and a 1 on  $l$  in the differing position. As such, we would like to use the downward-closedness to “zero out” the  $l$ 's in a given input to avoid the undesirable situation. If the algorithm already returns a 1 on some  $h$  for the input, this can be done while still preserving a constant fraction of the welfare. Otherwise, we look at nearby inputs and take an allocation that would return a 1 on some  $h$  for our input, if such an allocation exists.

We now formally describe the transformation  $\mathcal{T}$ . Given an input  $\mathbf{v}$ ,  $\mathcal{T}$  proceeds as follows:

1. If  $\mathcal{A}(\mathbf{v})$  already has a 1 on  $h$ , “zero out” all the  $l$ ’s, and return that allocation.
2. Else, if some input adjacent to  $\mathbf{v}$  has an allocation that would yield a 1 on  $h$  at  $\mathbf{v}$ , take that allocation and zero out all the  $l$ ’s, and return that allocation. (Pick arbitrarily if there are many such allocations.)
3. Else, if some input of Hamming distance 2 away from  $\mathbf{v}$  has an allocation that would yield a 1 on  $h$  at  $\mathbf{v}$ , take that allocation and zero out all the  $l$ ’s, and return that allocation. (Pick arbitrarily if there are many such allocations.)
4. Else, return  $\mathcal{A}(\mathbf{v})$ .

The transformation takes polynomial time, and it only zeroes out the  $l$ ’s when the allocation already has a 1 on  $h$ . Since  $h/l = \Omega(n)$ , a constant fraction of the welfare is preserved pointwise.

It remains to show that the resulting allocation rule is monotone. Suppose for contradiction that for some neighboring inputs  $\mathbf{v}$  and  $\mathbf{w}$ , at the position where the two inputs differ, there exists a 0 on  $h$  at  $\mathbf{v}$ , and a 1 on  $l$  at  $\mathbf{w}$ . The allocation at  $\mathbf{w}$  cannot have changed in Steps 1, 2, or 3 of the transformation, and  $\mathbf{w}$  has 0 on all the  $h$ ’s since otherwise Step 1 would have been activated on  $\mathbf{w}$ . But then the original allocation at  $\mathbf{v}$  must have 0 on all the  $h$ ’s, except possibly at the position where the two inputs differ, because otherwise Step 2 would have been activated on  $\mathbf{w}$ . At the differing position, however, the original allocation at  $\mathbf{v}$  must have a 0 too, because otherwise Step 1 would have been activated on  $\mathbf{v}$  and the final allocation would have a 1 at this position. Now, the allocation at  $\mathbf{v}$  must have changed in Step 2, because the original allocation at  $\mathbf{v}$  has no 1 on  $h$  and the allocation at  $\mathbf{w}$  would yield a 1 on  $h$  at  $\mathbf{v}$ . The allocation at  $\mathbf{v}$  did not change to the allocation at  $\mathbf{w}$ , however, because otherwise the non-monotonicity would not have occurred. Hence it must have changed to the allocation at some other input with a 1 on  $h$  in a position where  $\mathbf{v}$  and  $\mathbf{w}$  do not differ. But then Step 3 would have been activated on  $\mathbf{w}$ , which gives us the desired contradiction.  $\square$

Note that the transformation in Theorem 5 might preserve full welfare at a very small number of inputs. Indeed, if  $\mathcal{A}$  returns the allocations with all 1’s at every input, then  $\mathcal{T}$  preserves full welfare at only 2 out of the  $2^n$  inputs. Nevertheless, we can improve the transformation so that not only does it preserve a constant fraction of the welfare pointwise, but it also preserves full welfare at a  $1/n$  fraction of the inputs. To this end, we will need to make a slightly stronger assumption that  $h/l > n$ .

**Theorem 6** *Let  $l < h$  be such that  $h/l > n$ , and assume that the private valuation of each agent is either  $l$  or  $h$ . There exists a polynomial-time, monotone, constant-fraction welfare-preserving transformation that preserves the full welfare at a  $1/n$  fraction of the inputs.*

*Proof* We exhibit such a transformation  $\mathcal{T}$ , which is a slight modification of the transformation in Theorem 5.

We call an allocation (implicitly along with an input) an *h-allocation* if it has a 1 on *h* at the input, and an *l-allocation* otherwise. For any allocation (again implicitly along with an input), call another allocation a *higher h-allocation* if it yields strictly more 1's on *h* than the original allocation at the input.

Given any input  $\mathbf{v}$ , the transformation  $\mathcal{T}$  proceeds as follows:

1. If  $\mathcal{A}(\mathbf{v})$  is an *h-allocation*, consider its adjacent inputs. If the allocation at one of these inputs would yield a higher *h-allocation* at  $\mathbf{v}$ , take that allocation provisionally. (Break ties in a consistent manner if there are many such allocations, e.g., by choosing the allocation at the input that differs from  $\mathbf{v}$  at the leftmost position among all corresponding inputs.)
2. Simulate Step 1 for each input  $\mathbf{w}$  at Hamming distance 1 or 2 away from  $\mathbf{v}$  by checking whether any original allocation at a neighbor of  $\mathbf{w}$  would yield a higher *h-allocation* at  $\mathbf{w}$  (if  $\mathcal{A}(\mathbf{w})$  is an *h-allocation*), and taking such an allocation provisionally for  $\mathbf{w}$  if one exists.
3. If the allocation at  $\mathbf{v}$  is an *l-allocation*, consider its adjacent inputs. If the provisional allocation at one of these inputs would yield an *h-allocation* at  $\mathbf{v}$ , take that allocation provisionally. (Break ties in a consistent manner if there are many such allocations.)
4. If the allocation at  $\mathbf{v}$  is still an *l-allocation*, consider the inputs of Hamming distance 2 away from  $\mathbf{v}$ . If the provisional allocation at one of these inputs would yield an *h-allocation* for  $\mathbf{v}$ , take that allocation provisionally. (Break ties in a consistent manner if there are many such allocations.)
5. If the allocation at  $\mathbf{v}$  has improved to a higher *h-allocation* than the original allocation, zero out all the *l*'s. Call the allocation at  $\mathbf{v}$  at this point the *almost-final allocation*.
6. Simulate Steps 1 through 5 for all inputs adjacent to  $\mathbf{v}$  to arrive at an almost-final allocation for each such input.
7. For any 1 on *l* at  $\mathbf{v}$ , zero it out only if it yields a monotonicity conflict with the almost-final allocation at a neighboring input. This is the final allocation at  $\mathbf{v}$ .

The transformation takes polynomial time. The resulting allocation rule is monotone, since any monotonicity conflict is fixed in Step 7. We next show that a constant fraction of the welfare is preserved pointwise.

Consider any input  $\mathbf{v}$ . If  $\mathcal{A}(\mathbf{v})$  is an *h-allocation*, the worst that can happen is that all of the 1's on *l* are zeroed out, but even then half of the welfare is still preserved. Assume that  $\mathcal{A}(\mathbf{v})$  is an *l-allocation*, meaning that Step 1 is not activated on  $\mathbf{v}$ . If the allocation at  $\mathbf{v}$  changes during Steps 1 through 6, the welfare can only increase, so we may assume that the allocation remains the same throughout the first six steps. It suffices to show that the allocation is also unchanged in Step 7.

To this end, suppose for contradiction that the almost-final allocation at a neighboring input  $\mathbf{w}$  has a 0 on *h* in a position where  $\mathbf{v}$  has a 1 on *l*. The provisional allocation at  $\mathbf{w}$  after Step 2 must have 0 on all the *h*'s, except possibly at the position where the two inputs differ, because otherwise Step 3 would have been activated on  $\mathbf{v}$ . At the differing position, however, the provisional allocation at  $\mathbf{w}$  must have a 0 too, because if it has a 1 then the 1 cannot turn into a 0 later. So this allocation is an *l-allocation*.

Now, the allocation at  $\mathbf{w}$  must have changed in Step 3, because it is an  $l$ -allocation and the provisional allocation at  $\mathbf{v}$  would yield a 1 on  $h$  at  $\mathbf{w}$ . The allocation at  $\mathbf{w}$  did not change to the allocation at  $\mathbf{v}$ , however, because otherwise the non-monotonicity in the almost-final allocations would not have occurred. Hence it must have changed to another allocation with a 1 on  $h$  at a position where  $\mathbf{v}$  and  $\mathbf{w}$  do not differ. But then Step 4 would have been activated on  $\mathbf{v}$  and turned the allocation at  $\mathbf{v}$  into an  $h$ -allocation. This contradiction concludes the proof that a constant fraction of the welfare is preserved pointwise.

We now show that at least a  $1/n$  fraction of the inputs obtain weakly better welfare. In particular, for each input that obtains strictly less welfare, we will find a neighbor that obtains strictly better welfare.

An input  $\mathbf{v}$  obtains strictly less welfare only if it has to zero out an  $l$  in Step 7. That means that the almost-final allocation at  $\mathbf{v}$  has a 1 on  $l$ . In particular, the allocation has never been changed in Steps 1 through 6. On the other hand, a neighbor  $\mathbf{w}$  has an almost-final allocation with a 0 on  $h$  in that position. Assume for contradiction that  $\mathbf{w}$  obtains less (or equal) welfare than before. That means that the allocation at  $\mathbf{w}$  has also never been changed to a higher  $h$ -allocation during Steps 1 through 6. Consider the  $n - 1$  positions at which  $\mathbf{v}$  and  $\mathbf{w}$  do not differ. If the allocation at  $\mathbf{v}$  has at least as many 1's on  $h$  among these positions as the allocation at  $\mathbf{w}$ , then  $\mathbf{w}$  could have gotten a higher  $h$ -allocation by taking the allocation at  $\mathbf{v}$ . Else, the allocation at  $\mathbf{v}$  has strictly fewer 1's on  $h$  among these positions than the allocation at  $\mathbf{w}$ , and  $\mathbf{v}$  could have gotten a higher  $h$ -allocation by taking the allocation at  $\mathbf{w}$ . Either way, we have a contradiction.

Hence, every time an input loses a 1 on  $l$ , it can point to a neighbor that got better. Each input that got better can be pointed to at most  $n - 1$  times (since it must have at least one 1 on  $h$  and cannot have a monotonicity conflict in that position). Let  $W$  be the set of inputs that got worse. We have  $|W| \leq (n - 1) \cdot (2^n - |W|)$ , and therefore  $|W| \leq \frac{n-1}{n} \cdot 2^n$ , as desired.  $\square$

If  $h/l > 2n$ , the transformation in Theorem 6 also preserves the expected welfare over the uniform distribution over the  $2^n$  inputs, as we show next.

**Theorem 7** *Let  $l < h$  be such that  $h/l > 2n$ , and assume that the private valuation of each agent is either  $l$  or  $h$ . There exists a polynomial-time, monotone, constant-fraction welfare-preserving transformation that preserves full welfare at a  $1/n$  fraction of the inputs and preserves expected welfare over the uniform distribution over the  $2^n$  inputs.*

*Proof* Consider the transformation in Theorem 6. Every time an input loses a 1 on  $l$ , it can point to a neighbor that got better. The welfare of that neighbor has increased by at least  $h - nl > nl$ . Since each input that got better can be pointed to at most  $n - 1$  times, the expected welfare over the uniform distribution over the  $2^n$  inputs is preserved.  $\square$

Finally, we consider the other extreme case where  $h/l$  is constant. In this case, simply returning a constant allocation already preserves a constant fraction of the

approximation ratio. We focus on the allocation  $\mathcal{A}(ll \dots l)$ , but a similar statement can be obtained for any other constant allocation. The result can also be extended to the case where we have multiple input values, all of which are within a constant factor of each other.

**Theorem 8** *Let  $l < h$  be arbitrary values (possibly depending on  $n$ ), and assume that the private valuation of each agent is either  $l$  or  $h$ . Let  $\mathcal{T}$  be a transformation that returns the constant allocation  $\mathcal{A}(ll \dots l)$  at any input. Then  $\mathcal{T}$  preserves an  $l/h$  fraction of the approximation ratio.*

*Proof* One can check that  $\mathcal{T}(\mathcal{A})(\mathbf{v}) \geq \mathcal{A}(ll \dots l)$  for any input  $\mathbf{v}$ . Moreover, we have that  $OPT(\mathbf{v}) \leq \frac{h}{l} \cdot OPT(ll \dots l)$ , since any allocation at  $\mathbf{v}$  would return at least an  $l/h$  fraction of the welfare when allocated to the input  $ll \dots l$ . Hence

$$\begin{aligned} \text{approx}_{\mathcal{F}}(\mathcal{T}(\mathcal{A})) &= \min_{\mathbf{v}} \frac{\mathcal{T}(\mathcal{A})(\mathbf{v})}{OPT_{\mathcal{F}}(\mathbf{v})} \\ &\geq \frac{l \cdot \mathcal{A}(ll \dots l)}{h \cdot OPT_{\mathcal{F}}(ll \dots l)} \\ &\geq \frac{l}{h} \cdot \text{approx}_{\mathcal{F}}(\mathcal{A}), \end{aligned}$$

as desired.  $\square$

Combining this theorem with Theorem 5, we have that a constant fraction of the approximation ratio can be preserved if either  $h/l$  is constant or  $h/l \in \Omega(n)$ . This means that if we were to obtain a negative result with two values, it would have to be the case that  $h/l$  lies strictly between constant and linear.

## 4.2 Multiple values

In this subsection, we show that we can generalize the transformation in Theorem 5 to the case where we have multiple input values, each pair separated by a ratio of  $\Omega(n)$ . Recall that when some two input values are separated by  $O(n^\alpha)$  for some  $\alpha \in [0, 1)$ , we have from Theorem 3 that it is impossible to preserve a constant fraction of the welfare pointwise. Hence we have a complete picture of constant-fraction welfare-preserving transformations for multiple input values as well.

**Theorem 9** *Let  $k$  be a constant, and let  $a_1, \dots, a_k$  be such that  $a_{i+1}/a_i \in \Omega(n)$  for  $i = 1, \dots, k-1$ . Assume that the private valuation of each agent is one of  $a_1, \dots, a_k$ . There exists a polynomial-time, monotone, constant-fraction welfare-preserving transformation.*

*Moreover, if  $a_{i+1}/a_i > n$  for all  $i$ , then the transformation can be modified so that it also preserves full welfare at a  $\frac{1}{(k-1)(n-1)+1}$  fraction of the inputs.*

*Proof* We first consider the case where there are three input values  $h, m, l$ , and focus only on preserving a constant fraction of the welfare pointwise. It is possible to extend



to any constant number of inputs  $k$  and also preserve full welfare for a  $\frac{1}{(k-1)(n-1)+1}$  fraction of the inputs, and we explain that later.

For any allocation (implicitly along with an input), we call it an  $h$  – allocation if it has a 1 on  $h$  at the input. Otherwise, we call it an  $m$  – allocation if it has a 1 on  $m$  at the input. Finally, we call it an  $l$  – allocation if it is neither an  $h$  – allocation nor an  $m$  – allocation.

We exhibit a transformation  $\mathcal{T}$  that preserves a constant fraction of the welfare pointwise. Given any input  $\mathbf{v}$ , the transformation  $\mathcal{T}$  proceeds as follows:

1. If  $\mathcal{A}(\mathbf{v})$  is an  $l$  – allocation and some input adjacent to  $\mathbf{v}$  has an allocation that would yield an  $m$  – allocation or an  $h$  – allocation at  $\mathbf{v}$ , or if  $\mathcal{A}(\mathbf{v})$  is currently an  $m$  – allocation and some input adjacent to  $\mathbf{v}$  has an allocation that would yield an  $h$  – allocation at  $\mathbf{v}$ , take that allocation provisionally. (If there are many such allocations, take the higher type if possible; otherwise pick arbitrarily.)
2. If the allocation at  $\mathbf{v}$  is currently an  $l$  – allocation, and some input at Hamming distance at most 2 away from  $\mathbf{v}$  has an allocation that would yield an  $m$  – allocation at  $\mathbf{v}$ , take that allocation provisionally. (Pick arbitrarily if there are many such allocations.)
3. If the allocation at  $\mathbf{v}$  is currently *not* an  $h$  – allocation, and some input at Hamming distance at most 3 away from  $\mathbf{v}$  has an allocation that would yield an  $h$  – allocation at  $\mathbf{v}$ , take that allocation provisionally. (Pick arbitrarily if there are many such allocations.)
4. If the allocation at  $\mathbf{v}$  is currently an  $m$  – allocation, and some input at Hamming distance at most 4 away from  $\mathbf{v}$  has an allocation that would yield an  $h$  – allocation at  $\mathbf{v}$ , take that allocation provisionally. (Pick arbitrarily if there are many such allocations.)
5. If the allocation at  $\mathbf{v}$  is currently an  $l$  – allocation, and some input at Hamming distance at most 5 away from  $\mathbf{v}$  has an allocation that would yield an  $h$  – allocation at  $\mathbf{v}$ , take that allocation provisionally. (Pick arbitrarily if there are many such allocations.)
6. If the allocation at  $\mathbf{v}$  is currently an  $h$  – allocation, zero out all the  $m$ ’s and  $l$ ’s. If it is an  $m$  – allocation, zero out all the  $l$ ’s. Return the current allocation  $\mathcal{A}(\mathbf{v})$ .

The transformation runs in polynomial time and since  $h/m, m/l \in \Omega(n)$ , a constant fraction of the welfare is preserved pointwise. We now show that the resulting allocation rule is monotone. Suppose for contradiction that this is not true. There are three cases.

- *Case 1:* For some neighboring inputs  $\mathbf{v}$  and  $\mathbf{w}$ , at the position where the two inputs differ, there exists a 0 on  $h$  at  $\mathbf{v}$ , and a 1 on  $l$  at  $\mathbf{w}$ . The allocation at  $\mathbf{w}$  cannot have changed throughout the transformation, and this allocation is an  $l$  – allocation. The original allocation at  $\mathbf{v}$  must have 0 on all the  $h$ ’s, except possibly at the position where the two inputs differ, because otherwise Step 1 would have been activated on  $\mathbf{w}$ . At the differing position, however, the original allocation at  $\mathbf{v}$  must have a 0 too, because otherwise this allocation would be an  $h$  – allocation and the final allocation would have a 1 in this position. Now, the allocation at  $\mathbf{v}$  must have changed in Step 1, because the original allocation at  $\mathbf{v}$

is an  $l$ - or  $m$ -allocation and the allocation at  $\mathbf{w}$  would yield an  $h$ -allocation. The allocation at  $\mathbf{v}$  did not change to the allocation at  $\mathbf{w}$ , however, because otherwise the non-monotonicity would not have occurred. Hence it must have changed to the allocation of some other input with a 1 on  $h$  or  $m$  in a position where  $\mathbf{v}$  and  $\mathbf{w}$  do not differ. But then Step 2 or 3 would have been activated on  $\mathbf{w}$ , a contradiction.

- *Case 2:* For some neighboring inputs  $\mathbf{v}$  and  $\mathbf{w}$ , at the position where the two inputs differ, there exists a 0 on  $h$  at  $\mathbf{v}$ , and a 1 on  $m$  at  $\mathbf{w}$ . The final allocation at  $\mathbf{w}$  is an  $m$ -allocation, so it has 0 on all the  $h$ 's and  $l$ 's. The original allocation at  $\mathbf{v}$  must have 0 on all the  $h$ 's, except possibly at the position where the two inputs differ, because otherwise Step 1 would have been activated on  $\mathbf{w}$ . At the differing position, however, the original allocation at  $\mathbf{v}$  must have a 0 too, because otherwise this allocation would be an  $h$ -allocation and the final allocation would have a 1 in this position. Now, the 1 on  $m$  in the allocation at  $\mathbf{w}$  occurs at the latest in Step 2, since in later steps an allocation can only turn into an  $h$ -allocation. This means that after Step 3, the allocation at  $\mathbf{v}$  must be an  $h$ -allocation. This allocation is different from the one at  $\mathbf{w}$ , because otherwise the non-monotonicity would not have occurred, so the allocation has a 1 on  $h$  in a position where  $\mathbf{v}$  and  $\mathbf{w}$  do not differ. But then Step 4 would have been activated on  $\mathbf{w}$ , a contradiction.
- *Case 3:* For some neighboring inputs  $\mathbf{v}$  and  $\mathbf{w}$ , at the position where the two inputs differ, there exists a 0 on  $m$  at  $\mathbf{v}$ , and a 1 on  $l$  at  $\mathbf{w}$ . The allocation at  $\mathbf{w}$  cannot have changed throughout the transformation, and this allocation is an  $l$ -allocation. The original allocation at  $\mathbf{v}$  must have 0 on all the  $m$ 's and  $h$ 's, except possibly at the position where the two inputs differ, because otherwise Step 1 would have been activated on  $\mathbf{w}$ . After Step 1, the allocation at  $\mathbf{v}$  is at least an  $m$ -allocation, since it could change to the allocation at  $\mathbf{w}$ . If this allocation has a 1 on  $m$  or  $h$  in a position where  $\mathbf{v}$  and  $\mathbf{w}$  do not differ, the allocation at  $\mathbf{w}$  should have changed to an  $m$ - or  $h$ -allocation in Step 2 or 3. This means that the allocation at  $\mathbf{v}$  has a 1 on  $m$  at the position where the two inputs differ after Step 1, and yet changes to an  $h$ -allocation later. This change occurs at the latest in Step 4, since an  $m$ -allocation is not allowed to change in Step 5. But then Step 5 or an earlier step would have been activated on  $\mathbf{w}$ , a contradiction.

As mentioned, it is possible to extend the transformation to any constant number  $k$  of input values. Suppose that the input values are  $a_1 < a_2 < \dots < a_k$ , and define an  $a_i$ -allocation for each  $i$  analogously to the  $k = 3$  case. The transformation takes  $\frac{k^2+k-2}{2}$  steps.

- $? \rightarrow ?$
- $a_1 \rightarrow a_2$
- $? \rightarrow a_3$
- $a_2 \rightarrow a_3$
- $a_1 \rightarrow a_3$
- $? \rightarrow a_4$
- $a_3 \rightarrow a_4$
- $a_2 \rightarrow a_4$

- $a_1 \rightarrow a_4$
- $\dots$
- $a_{k-1} \rightarrow a_k$

In each step, the transformation considers allocations at inputs within Hamming distance one higher than the previous step. If the change in the type of allocation (e.g., from an  $a_2$ -allocation to an  $a_5$ -allocation) matches the specified change in that step, the transformation executes the change. The question mark (e.g.,  $? \rightarrow a_3$ ) denotes *any* allocation of a lower type than the target allocation. In the first step, if there are several candidate allocations, take one with the highest type possible. Finally, the transformation zeroes out all the input values other than the highest one of the allocation. Since the ratio between any two adjacent input values is at least linear, this transformation preserves a constant fraction of the welfare pointwise.

To show that the transformation is monotone, assume for contradiction that for some neighboring inputs  $\mathbf{v}$  and  $\mathbf{w}$ , at the position where the two inputs differ, there exists a 0 on  $a_j$  at  $\mathbf{v}$ , and a 1 on  $a_i$  at  $\mathbf{w}$ , where  $i < j$ . This means that the final allocation at  $\mathbf{w}$  is an  $a_i$ -allocation, so the allocation at  $\mathbf{w}$  has 0 on all the  $a_{i+1}$ 's,  $a_{i+2}$ 's,  $\dots$ ,  $a_j$ 's throughout the transformation. Therefore the original allocation at  $\mathbf{v}$  must also have 0 on all the  $a_{i+1}$ 's,  $a_{i+2}$ 's,  $\dots$ ,  $a_j$ 's at the positions where the two inputs do not differ; otherwise Step 1 would have been activated on  $\mathbf{w}$ . Now, the allocation at  $\mathbf{w}$  becomes an  $a_i$ -allocation at some point (possibly at the beginning), at the latest in the step  $a_{i-1} \rightarrow a_i$ . So the allocation at  $\mathbf{v}$  must be at least an  $a_j$ -allocation after the step  $? \rightarrow a_j$  (or Step 1, if  $j = 2$ ). If this allocation has a 1 on one of  $a_j, a_{j+1}, \dots, a_n$  at a position where the two inputs do not differ, the allocation at  $\mathbf{w}$  would have changed to at least an  $a_j$ -allocation in a later step. This means that the allocation at  $\mathbf{v}$  has a 1 on  $a_j$  at the position where the two inputs differ after the step  $? \rightarrow a_j$ . However, the allocation later has a 0 in this position, so the allocation must have changed to at least an  $a_{j+1}$ -allocation. But then the allocation at  $\mathbf{w}$  should also have changed to this allocation in a later step instead of staying as an  $a_i$ -allocation. This gives us the desired contradiction.

We can extend the transformation in a similar way as in Theorem 6 so that the transformation also preserves full welfare at a  $\frac{1}{(k-1)(n-1)+1}$  fraction of the inputs. For any allocation (implicitly along with an input), call another allocation a *higher allocation* if it yields either strictly more 1's on  $a_n$  than the original allocation at the input, or an equal number of 1's on  $a_n$  and strictly more 1's on  $a_{n-1}$ , or an equal number of 1's on  $a_n$  and  $a_{n-1}$  and strictly more 1's on  $a_{n-2}$ ,  $\dots$ , or an equal number of 1's on  $a_n, a_{n-1}, \dots, a_3$  and strictly more 1's on  $a_2$ . We add a preprocessing step in which for each input, we check whether the allocation at any neighboring input would yield a higher allocation, and if so, we take that allocation provisionally. We simulate this step for every input of Hamming distance no more than  $\frac{k^2+k-2}{2}$  away, and assume without loss of generality that all inputs start with these provisional allocations. We then perform the  $\frac{k^2+k-2}{2}$  steps as before. At the end, for any input whose allocation has improved to a higher allocation, we zero out all the input values except the highest one to arrive at an *almost-final* allocation of the input. We then zero out a 1 in an almost-final allocation only if it yields a monotonicity conflict with the almost-final allocation at a neighboring input.

The transformation is monotone by construction. To show that at least a  $\frac{1}{(k-1)(n-1)+1}$  fraction of the inputs obtain weakly better welfare, we establish in a similar way as in Theorem 6 that each input that obtains strictly less welfare can point to a neighbor that obtains strictly better welfare. Since each input that obtains better welfare can be pointed to at most  $(k-1)(n-1)$  times, the claim follows. To show that a constant fraction of the welfare is preserved at any input  $\mathbf{v}$ , note that the only case we have to deal with is when the allocation at  $\mathbf{v}$  is an  $a_1$ -allocation that remains the same until it is an almost-final allocation. An argument similar to that for the earlier transformation in this proof shows that such an allocation will also remain the same until the end.  $\square$

## 5 Conclusion

In this paper, we consider black-box transformations in downward-closed single-parameter environments, an important subclass of environments that occur in several settings in mechanism design. We show both positive and negative results on the power of such transformations (see Table 1). Several questions remain, the most important of which is perhaps whether there exists a black-box transformation that preserves the approximation ratio of any algorithm, or an approximation thereof, when there is no Hamming distance restrictions on the transformation.

**Acknowledgments** This work was partially supported by NSF Award CCF-1813188, by the European Research Council (ERC) under grant number 639945 (ACCORD), and by a Stanford Graduate Fellowship. The author would like to thank Tim Roughgarden for helpful discussions and the reviewers for helpful comments.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## References

1. Babaioff, M., Lavi, R., Pavlov, E.: Single-value combinatorial auctions and algorithmic implementation in undominated strategies. *J. ACM* **56**(1), 4 (2009)
2. Bei, X., Huang, Z.: Bayesian incentive compatibility via fractional assignments. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 720–733 (2011)
3. Briest, P., Krysta, P., Vöcking, B.: Approximation techniques for utilitarian mechanism design. *SIAM J. Comput.* **40**(6), 1587–1622 (2011)
4. Cai, Y., Daskalakis, C., Weinberg, S.M.: Understanding incentives: Mechanism design becomes algorithm design. In: Proceedings of the 54th Symposium on Foundations of Computer Science, pp. 618–627 (2013)
5. Chawla, S., Immorlica, N., Lucier, B.: On the limits of black-box reductions in mechanism design. In: Proceedings of the 44th Symposium on Theory of Computing, pp. 435–448 (2012)

6. Dughmi, S., Roughgarden, T.: Black-box randomized reductions in algorithmic mechanism design. *SIAM J. Comput.* **43**(1), 312–326 (2014)
7. Goel, G., Karande, C., Wang, L.: Single-parameter combinatorial auctions with partially public valuations. In: *Proceedings of the 3rd International Symposium on Algorithmic Game Theory*, pp. 234–245 (2010)
8. Hartline, J.D., Kleinberg, R., Malekian, A.: Bayesian incentive compatibility via matchings. *Games Econom. Behav.* **92**, 401–429 (2015)
9. Hartline, J.D., Lucier, B.: Bayesian algorithmic mechanism design. In: *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pp. 301–310 (2010)
10. Huang, Z., Wang, L., Zhou, Y.: Black-box reductions in mechanism design. In: *Proceedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pp. 254–265 (2011)
11. Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. *J. ACM* **58**(6), 25 (2011)
12. Myerson, R.B.: Optimal auction design. *Math. Oper. Res.* **6**(1), 58–73 (1981)
13. Nisan, N., Ronen, A.: Algorithmic mechanism design. *Games Econom. Behav.* **35**(1–2), 166–196 (2001)
14. Pass, R., Seth, K.: On the impossibility of black-box transformations in mechanism design. In: *Proceedings of the 7th International Symposium on Algorithmic Game Theory*, pp. 279–290 (2014)