Electronic Colloquium on Computational Complexity, Report No. 95 (2013)

# Welfare Maximization and the Supermodular Degree

Uriel Feige
Weizmann Institute of Science
uriel.feige@weizmann.ac.il

Rani Izsak
Weizmann Institute of Science
ran.izsak@weizmann.ac.il

May 29, 2013

**Abstract**

Given a set of items and a collection of players, each with a nonnegative monotone valuation set function over the items, the welfare maximization problem requires that every item be allocated to exactly one player, and one wishes to maximize the sum of values obtained by the players, as computed by applying the respective valuation function to the bundle of items allocated to the player. This problem in its full generality is $\mathcal{NP}$-hard, and moreover, at least as hard to approximate as set packing. Better approximation guarantees are known for restricted classes of valuation functions.

In this work we introduce a new parameter, the *supermodular degree* of a valuation function, which is a measure for the extent to which the function exhibits supermodular behavior. We design an approximation algorithm for the welfare maximization problem whose approximation guarantee is linear in the supermodular degree of the underlying valuation functions.

## 1 Introduction

The welfare maximization problem (also known as "combinatorial auction") is the following. There is a set of players and a set of indivisible items. Each player has its own (monotone non-decreasing) valuation for any subset of items. The goal is to distribute the items to the players while maximizing social welfare - the sum of values of all players, by their personal valuations. The welfare maximization problem is $\mathcal{NP}$-hard to approximate with any reasonable guarantee. For this reason past research considered restrictions on the class of set functions that may serve as valuation functions for the players. Lehmann, Lehmann and Nisan [15] considered *complement free* functions, which essentially means that a value of a set of items cannot exceed the sum of values of its parts. They presented a hierarchy of classes of complement free functions, and established constant factor approximations for the welfare maximization problem in some cases.[1] Subsequent work established constant factor approximation for all classes of complement free functions. This made it clear that the poor approximation guarantees for the general case must come from complementarities

[1]There was earlier work with related results that used different terminology. For example, Fisher, Nemhauser and Wolsey [10] studied "the m-box problem" which is essentially the welfare maximization problem with monotone submodular valuation functions. Among their results there was a greedy approximation algorithm for a generalized version of this problem with approximation guarantee 1/2.

ISSN 1433-8092

(sets whose value is larger than that of the sum of their parts). Recently, Abraham, Babaioff, Dughmi and Roughgarden [1] considered a restricted class of set functions strictly based on complementarities (in particular, no set is valued less than the sum of its parts). Among other results, they presented an algorithm with an approximation guarantee that is linear in a certain parameter related to the extent of these complementarities.

In the current work, similarly to [1], we express the approximation guarantees as a function of some parameter associated with the underlying set functions. The smaller this parameter is, the better the approximation guarantee. However, we depart from the practice of considering restricted classes of set functions – our parametrization can be applied to any set function. It is most advantageous (in the sense that our approximation guarantees are not bad whereas previous work does not apply to these set functions) when the set functions are basically submodular (offer decreasing marginal value, which is the discrete analog of convexity), but exhibit a limited amount of complementarities. As a simple example of how such functions may arise, consider shopping for shoes. Each additional pair of shoes may have decreasing marginal value, but within a pair of shoes, the left and right shoe are together worth more than the sum of values of each shoe on its own.

Specifically, we introduce two new (as far as we know) complexity measures of set functions. One is the *dependency degree*. Roughly speaking, this is the maximum number of items that may influence the marginal value of any item with respect to any possible subset of other items. The other is the *supermodular degree*. Roughly speaking, this is the dependency degree, taking into account only items that may *increase* the marginal value of an item. That is, "negative dependencies" do not increase the latter complexity measure. In particular, submodular functions might have arbitrary dependency degree, but their supermodular degree is 0. This measure can also be seen, in a sense, as the "degree of complementarity". We design two greedy approximation algorithms for the welfare maximization problem, each with approximation guarantee linear in the maximum of one of these measures over the set functions of the players.

## 2 Preliminaries

We firstly define formally the welfare maximization problem:

**Definition 2.1** *An instance $\mathcal{I}(P, M, v)$ of the welfare maximization problem is the following:*

- *$P$ is a set of $n$ players $1, \ldots, n$.*
- *$M$ is a set of $m$ items $j_1, \ldots, j_m$.*
- *$v$ is a vector of $n$ valuation functions (set functions) $v_1, \ldots, v_n$, where $v_p : 2^M \to \mathbb{R}^+$ is the valuation function associated with the player $p \in P$.[2] For any $p \in P$, $v_p$ is restricted to be monotone non-decreasing and with value $0$ for the empty set (and hence also non-negative).*

*A feasible solution to $\mathcal{I}$ is a mapping $SOL : M \to P$, allocating each of the items to exactly one player. This mapping induces for each player $p \in P$ a set $SOL_p$ of the items mapped to her. The utility/value of a player $p \in P$ is defined as $v_p(SOL_p)$. Our aim is to maximize the social welfare $v(SOL) \overset{def}{=} \sum\limits_{p \in P} v_p(SOL_p)$.*

[2] We use $\mathbb{R}^+$ to indicate the set of all non-negative real numbers (that is, 0 is included).

### 2.1 Types of set functions without complementarities

We recall previously studied types of set functions without complementarities (see for example [15, 8]). Let $S$ be a set and let $f : 2^S \to \mathbb{R}^+$ be a set function.

**Definition 2.2** *We say that $f$ is submodular if for any $S'' \subseteq S' \subseteq S$ and $x \in S \setminus S'$, $f(x \mid S') \leq f(x \mid S'')$.*

**Definition 2.3** *We say that $f$ is fractionally subadditive if for every subset $S' \subseteq S$, subsets $T_i \subseteq S'$ and every coefficients $0 < \alpha_i \leq 1$ such that for any $x \in S'$, $\sum_{i:x \in T_i} \alpha_i \geq 1$, it holds that $f(S') \leq \sum_i \alpha_i f(T_i)$.*

**Definition 2.4** *We say that $f$ is subadditive or complement free if for every $S_1, S_2 \subseteq S$, $f(S_1 \cup S_2) \leq f(S_1) + f(S_2)$.*

### 2.2 Measuring dependencies

In this section we introduce complexity measures capturing dependencies of items in a groundset of a set function. For convenience, we treat the set functions as valuation functions of players of an instance of the welfare maximization problem (for notations only). Let $M$ be a set, let $v_p : 2^M \to \mathbb{R}^+$ be a valuation function of player $p \in P$ and let $j \in M$. We firstly recall the following definition (see for example [15]):

**Definition 2.5** *Let $p \in P$ and let $j \in M$. The marginal valuation function $v_{p,j} : 2^{M \setminus \{j\}} \to \mathbb{R}^+$ is a function mapping each subset $S \subseteq M \setminus \{j\}$ to the marginal value of $j$ given $S$:*

$$v_{p,j}(S) \overset{def}{=} v_p(S \cup \{j\}) - v_p(S) .$$

*We denote the marginal value $v_{p,j}(S)$ also by $v_p(j \mid S)$. For $S' = \{j_1, \ldots, j_{|S'|}\} \subseteq M$ and $S \subseteq M \setminus S'$ we also use either of the notations $v_p(j_1, \ldots, j_{|S'|} \mid S)$ or $v_p(S' \mid S)$ to indicate $v_p(S \cup S') - v_p(S)$.*

**Definition 2.6** *The dependency set of $j$ by $v_p$ is the set of all items $j'$ in $M$ such that there exists $S \subseteq M \setminus \{j\}$ such that $v_p(j \mid S) \neq v_p(j \mid S \setminus \{j'\})$. For each such $j'$, we say that $j$ depends on $j'$ by $p$ and denote it by $j \to_p j'$, or by $j \overset{S}{\to}_p j'$ if we want to explicitly mention the set $S$. We denote the dependency set of $j$ by $v_p$ by $Dep_p(j)$. $p$ or $v_p$ may be omitted in any of the above, when it is clear from the context.*

*The relation '$\to$' is symmetric (see Appendix A), so we may also use the terminology "are dependent" and the notation '$\leftrightarrow$'.*

**Definition 2.7** *The supermodular dependency set of $j$ by $v_p$ is the set of all items $j'$ in $M$ such that there exists $S \subseteq M \setminus \{j\}$ such that $v_p(j \mid S) > v_p(j \mid S \setminus \{j'\})$. Terminology and notations are the same as in Definition 2.6, but with the word "supermodular/ly" or with '$+$' as a superscript. Note that the relation '$\to^+$' is also symmetric (see Appendix A).*

**Definition 2.8** *The dependency degree of $v_p$ is defined as $DD(v_p) \overset{def}{=} \max_{j \in M} |Dep_p(j)|$. The supermodular dependency degree (or simply, the supermodular degree) of $v_p$ is defined as $DD^+(v_p) \overset{def}{=} \max_{j \in M} |Dep_p^+(j)|$. The (supermodular) dependency degree of an instance of the welfare maximization problem is the maximum (supermodular) dependency degree among all valuation functions of the instance.*

Note that any submodular set function has supermodular degree 0. Note also that $DD^+(f) \leq DD(f)$ for any set function $f$.

We also use the following definition:

**Definition 2.9** *Let $f$ be a set function. The (supermodular) dependency graph of $f$ is the following. There is a vertex for each item and an undirected edge for each pair of (supermodularly) dependent items.*

## 2.3 Representing set functions

In the welfare maximization problem, the domain of the valuation functions is exponential in the number of items. We recall two possible approaches to cope with this. The first is using an explicit representation model (specifically, we recall a hypergraph representation; see [5], [4], [1]) and the second is using oracles, representing set functions by supporting queries with respect to them (see for example [3]).

### 2.3.1 A hypergraph representation

Every set function $f$ can be represented in a unique way as a hypergraph in which the vertices are the items, vertices and hyperedges have weights associated with them, and the value $f(S)$ of a subset $S$ of items equals the sum of all weights in the subgraph induced by the corresponding vertices. We observe that two items share a hyperedge in the hypergraph representation if and only if they are dependent, and that sharing a hyperedge of positive weight is a necessary condition but not sufficient for being supermodularly dependent.[3]

**A succinct representation** Let $f : 2^S \to \mathbb{R}^+$ be a set function and let $s \overset{\text{def}}{=} |S|$. In general, a succinct representation of $f$ is any representation that takes space polynomial in $s$. For example, the hypergraph representation is succinct for any set function with dependency degree bounded by $\log s$ (and, of course, for some other set functions, as well).

### 2.3.2 Queries oracles

We recall the definitions of value and demand queries oracle for an underlying set function. Let $f : 2^S \to \mathbb{R}^+$ be a set function.

**Definition 2.10** *Value queries are the following:*
*Input: A subset $S' \subseteq S$.*
*Output: $f(S')$.*

**Definition 2.11** *Demand queries are the following:*
*Input: A cost function $c : S \to \mathbb{R}^+$.*
*Output: A subset $S' \subseteq S$ maximizing $f(S') - \sum_{j \in S'} c(j)$.*

[3]For example, consider the set function $f$ that has value 0 on the empty set and 1 elsewhere. In its hypergraph representation every odd set of items forms a hyperedge of weight 1 and every even set of items forms a hyperedge of weight $-1$. Every two items share a positive hyperedge, but no two items are supermodularly dependent, since $f$ is submodular.

Note that demand queries are strictly stronger than value queries (see [3]).

**Definition 2.12** *Queries oracle for a type of queries for a given set function can answer queries of the respective type with respect to the given set function.*

The notion of an oracle serves as an abstraction for a subroutine that computes an answer to the respective type of queries for a given set function. When we say that an algorithm uses a certain type of oracle, the running time of the algorithm is computed as if each query takes unit time to answer, regardless of the true running time of the underlying subroutine. This abstraction is most justified if for the underlying set function, answers to the respective query can indeed be computed efficiently. We remark that given a succinct hypergraph representation for a set function, one can efficiently answer value queries, whereas answering demand queries might be $\mathcal{NP}$-hard (see for example Theorem 2.4). We also remark that given a succinct hypergraph representation, one can construct the corresponding dependency graph and supermodular dependency graph.

## 2.4 Preliminary observations

### 2.4.1 $\mathcal{APX}$-hardness of the welfare maximization problem

**Proposition 2.1** *The welfare maximization problem is $\mathcal{APX}$-hard even if there are only three players who all have the same valuation function $f$, with $DD^+(f) = 0$ and $DD(f) = 3$.*

**Proof:** It is known that the question of whether a 3-regular graph can be legally 3-colored is $\mathcal{NP}$-hard, and that it is $\mathcal{APX}$-hard to maximize the number of legally colored edges [16]. Given a 3-regular graph $G$, consider a set function $f$ that has $G$ as its hypergraph representation, with all vertices having value 3 and all edges having value $-1$. Associating a color with each player, the allocation that maximizes welfare is the one that minimizes the number of monochromatic edges. The proposition follows. □

### 2.4.2 Hardness of welfare maximization as a function of the dependency degree

Recall that the maximum weighted $k$-set packing problem is the following:

**Definition 2.13** *Let $G = (V, E, w)$ be a weighted $k$-uniform hypergraph (i.e. every hyperedge contains exactly $k$ vertices) with set of vertices $V$, set of undirected edges $E$ and edge weights function $w$. The maximum weighted $k$-set packing problem is to find a set of disjoint edges of maximum weight.*

This problem is known to be $\mathcal{NP}$-hard for any $k > 2$ and is hard for approximation with guarantee $\Omega(\frac{\ln k}{k})$, even in the unweighted case, by a result of Hazan, Safra and Schwartz [14]. The best approximation guarantee known for it currently (as far as we know) is $(k + 1)/2$, by an algorithm of Berman [2]. We show the following:

**Proposition 2.2** *There exists an approximation preserving reduction of the maximum weighted $k$-set packing problem to the welfare maximization problem with dependency degree at most $k - 1$.*

**Proof:** The reduction is Reduction 2.1.

It is easy to verify that any feasible solution for $\mathcal{I}_{SP}$ (the input of Reduction 2.1) has a corresponding feasible solution for $\mathcal{I}$ (the output of Reduction 2.1), with the same value, and vice versa,

**Reduction 2.1** $k$-set packing to welfare maximization with dependency degree at most $k-1$

```
Input: An instance $\mathcal{I}_{SP}(V, E, w)$ of the maximum weighted $k$-set packing problem, with $V = \{v_1, \dots, v_{|V|})\}$ .
Output: An instance $\mathcal{I}(P, M, v)$ of the welfare maximization problem with dependency degree at most $k-1$.
1: For each vertex $v_i \in V$, create an item $i \in M$.
2: For each hyper-edge $e = \{v_{e_1}, \dots, v_{e_k}\} \in E$, create a player $p_e \in P$ with $v_p(S) = w(e)$ for any
   $S$ such that $\{e_1, \dots, e_k\} \subseteq S$ and $v_p(S) = 0$ otherwise.
```

as desired. □

### 2.4.3 An exact algorithm for dependency degree at most 1

**Proposition 2.3** *The welfare maximization problem with dependency degree at most 1 admits an exact polynomial time algorithm.*

The main idea is to use symmetry of the dependency relation in order to reduce an instance of the welfare maximization problem to an instance of maximum weighted matching. The full reduction appears in Appendix C.

### 2.4.4 Demand queries and the dependency degree

**Theorem 2.4** *Given a hypergraph representation of any set function with dependency degree at most 2, demand queries may be answered in polynomial time (in the size of the hypergraph representation). Given a hypergraph representation of a set function with dependency degree at least 3, demand queries are generally $\mathcal{APX}$-hard to answer (with respect to the size of the hypergraph representation).*

**Proof:** Let $f$ be a set function. If $DD(f) \leq 2$, then each item depends on at most two other items. The dependency graph of $f$ is of maximum degree 2 and hence its connected components are either isolated vertices, isolated paths or isolated cycles. On each such component, demand queries may be answered using dynamic programming.

To show hardness for set functions $f$ with $DD(f) \geq 3$, we reduce from the problem of maximum independent set in 3-regular graphs. $f$ is represented by the following hypergraph representation. Give each vertex value 3 and each edge value $-1$. The value of the answer to a demand query in which the cost of each vertex is 2 is equal to the size of the maximum independent set (it is worth taking a vertex only if it does not contribute an edge to the induced subgraph). Due to $\mathcal{APX}$-hardness of maximum independent set in 3-regular graphs, it is $\mathcal{APX}$-hard to answer demand queries, as well. □

## 2.5 Our main results

We view the notion of the supermodular degree and the study of its basic properties as an important contribution of our work. This notion is applicable to any set function, and its relevance to the welfare maximization problem is demonstrated by the following theorem.

**Theorem 2.5** *The welfare maximization problem with supermodular degree at most $d$ admits a polynomial time greedy $\frac{1}{d+2}$-approximation algorithm. This algorithm requires for each valuation function a value queries oracle and a supermodular dependency graph.*

For the dependency degree we have a slightly better approximation guarantee, which is relevant only when the dependency degree and the supermodular degree are equal:

**Theorem 2.6** *The welfare maximization problem with dependency degree at most $d$ admits a greedy $\frac{1}{d+1}$ approximation algorithm. Its running time is polynomial in the number of players and items and in $2^d$. This algorithm requires for each valuation function a value queries oracle and a dependency graph.*

Proposition 2.2 implies that an improvement of our results by a multiplicative factor of more than roughly 2 would improve the current approximation guarantee for weighted $k$-set packing. Additionally, by a hardness result of Blumrosen and Nisan [3] of $\Omega(\log m/m)$ for algorithms requiring only value queries oracle[4], our bounds are tight up to a multiplicative factor of $O(\log m)$, in the sense that general set functions have dependency degree (and supermodular degree) at most $m - 1$.

## 2.6 Related work

The supermodular degree is a complexity measure for set functions, that ranges from 0 (for submodular set functions) to $m - 1$ (where $m$ is the number of items). Our main result is a greedy algorithm for the welfare maximization problem whose approximation guarantee increases linearly with the supermodular degree of the underlying set functions. Such a linear increase is to be expected, given the known reduction from set packing to the welfare maximization problem with single minded bidders, combined with the difficulty of approximating set packing ([14]). As far as we know, the notion of supermodular degree has not appeared in previous work. However, other related notions did appear, and in this section we discuss some of them.

Perhaps the simplest complexity measure for a set function is its support size, namely, the number of items that it depends on. This measure ranges from 1 to $m$. Moreover, simple greedy algorithms approximate the welfare maximization problem with guarantee equal to this measure, and known hardness results for the case of single minded bidders apply here as well. Hence the results that one can prove for the complexity measures of support size and supermodular degree are of a similar nature. However, as the supermodular degree of a function is not greater than its support size, and moreover, it is often much smaller (the gap being most dramatic for submodular functions), we view our results for supermodular degree as significantly more informative than the corresponding results for support size.

Lehmann, Lehmann and Nisan [15] proposed a hierarchy of classes of set functions based on notions of complement-freeness, and initiated a study of the complexity of the welfare maximization problem for these classes. For the lower classes (linear functions, and functions enjoying the *gross substitutes* property) the welfare maximization problem can be solved in polynomial time. For

[4]Our algorithms require also dependency graph / supermodular dependency graph. However, in the instance used by [3], it is trivial to construct both, since all the items supermodularly depend on each other (for infinitely many values of $m$). Alternatively, since the approximation hardness is $\Omega(\log m/m)$, we may simply modify the valuations functions so that every two items are supermodularly dependent, by adding a positive hyperedge that contains all items. Details omitted.

*submodular* functions, a constant approximation guarantee is possible and value queries suffice for this [15, 18] (and somewhat better constants are achievable using demand queries [9]). For XOS (later referred to as *fractionally subadditive* in [8]) and *subadditive* set functions there are approximation algorithms with constant approximation guarantees [6, 17, 8], but they require demand queries (value queries do not suffice [6]). The algorithm given in [15] for submodular functions is greedy, and the greedy algorithm in the current paper can be viewed as an extension of the greedy algorithm of [15] to a setting that is not submodular. The classification of [15] does not distinguish between different classes of functions that are not subadditive, and hence unlike our supermodular degree measure, is applicable only to some classes of set functions, but not to set functions in general. Set functions not lying in the classification of [15] may have any supermodular degree between 1 and $m$ and our algorithms for maximizing welfare distinguish among them in the approximation guarantees that they provide. Lehmann, Lehmann and Nisan [15] do suggest a way of extending their classification to additional functions, as follows. A function $f$ can be called $c$-submodular, if for every (possibly empty) sets $S$ and $T$ and item $x$, the marginal value of $x$ with respect to $S \cup T$ is at most $c$ times larger that the marginal value of $x$ with respect to $S$. (For submodular functions $c = 1$.) It is shown in [15] that the welfare maximization problem can be approximated with guarantee of $c + 1$ when the set functions are $c$-submodular. We note however that even functions on two items need not be $c$-submodular for any finite $c$ (if one of the items has value 0 by itself but positive marginal value together with the other item).

Submodular functions play a central role in the definition of the supermodular degree. However, other classes within the hierarchy of [15] have no special significance in this respect. The supermodular degree does not distinguish between linear functions and arbitrary submodular functions – they both have supermodular degree 0. Functions in the [15] hierarchy which are not submodular may have arbitrarily large supermodular degree.[5]

Conitzer, Sandholm and Santi [5] introduce the representation of set functions via hypergraphs with positive and negative hyperedges (presented in Section 2.3. See also the work of Chevaleyre, Endriss, Estivie and Maudet [4], who defined independently a similar concept). They showed that even if each hyperedge has at most two vertices, the welfare maximization problem is $\mathcal{NP}$-hard. They did not consider approximation algorithms. Abraham, Babaioff, Dughmi and Roughgarden [1] consider supermodular functions which have no negative hyperedges in their hypergraph representation. Among other results, they give an algorithm approximating the welfare maximization problem within a value equal to the maximum cardinality of any hyperedge (which may be smaller than the supermodular degree). They prove that obtaining similar approximation guarantees in the presence of negative hyperedges is $\mathcal{NP}$-hard. This serves as an explanation of why their model forbids negative hyperedges. Our results are to some extent in disagreement with this conclusion of [1]. Given a hypergraph representation of a set function with a given supermodular degree, adding negative hyperedges cannot increase the supermodular degree (in fact, it may cause the supermodular degree to decrease) and hence will not hurt our bounds on the approximation guarantees. This discrepancy between our results and those of [1] is explained by our requirement that set functions are nondecreasing, whereas the hardness of approximation results presented in [1] used set functions that are sometimes decreasing.

[5]For example, partition the set of items into two disjoint sets, $A$ and $B$. Let $f_A$ be a linear function that gives value 1 to each item in $A$ and 0 to each item in $B$. Let $f_B$ be a linear function that gives value 1 to each item in $B$ and 0 to each item in $A$. Let $f$ be defined as $f(S) = \max[f_A(S), f_B(S)]$. This is an XOS function (according to the classification of [15]), but its supermodular degree is $\max[|A|, |B|] - 1$.

## 3 Approximation guarantee linear in supermodular degree

In this section we prove Theorem 2.5. Our result may be seen as an extension of a work of Lehmann, Lehmann and Nisan [15], who presented a greedy 2-approximation algorithm for submodular set functions.

### 3.1 The algorithm

The algorithm is greedy. In a given iteration, for every player $p$ and item $j$, let $D_p^+(j)$ denote the set of items not yet allocated that have supermodular dependency with $j$ with respect to $v_p$. The algorithm computes the player $p$ and item $j$ for which the marginal value for $p$ (given the items that $p$ already has) of the set $j \cup D_p^+(j)$ is maximized, and allocates $j \cup D_p^+(j)$ to $p$. For a full description of the algorithm, see Algorithm 3.1.

**Algorithm 3.1** Greedily Approximate Welfare Maximization with Guarantee Linear in Supermodular Degree

Input:

- An instance $\mathcal{I}(P, M, v)$ of the welfare maximization problem.
- A value queries oracle and a supermodular dependency graph for each of the valuation functions.

Output: A solution with approximation guarantee $\frac{1}{d+2}$, where $d$ is the supermodular degree of $\mathcal{I}$.

```
1:  $Unallocated \leftarrow M$, $Approx \leftarrow \emptyset$
2:  while $Unallocated \neq \emptyset$ do
3:      $MaxMarginalUtility \leftarrow -1$
4:      for all $j \in Unallocated$, $p \in P$ do
5:          if $v_p(j, Dep_p^+(j) \cap Unallocated \mid \{j' \in M \mid (j' \mapsto p) \in Approx\}) > MaxMarginalUtility$ then
6:              $MaxMarginalUtility \leftarrow v_p(j, Dep_p^+(j) \cap Unallocated \mid \{j' \in M \mid (j' \mapsto p) \in Approx\})$
7:              $BestAllocation \leftarrow (\{j\} \cup (Dep_p^+(j) \cap Unallocated) \mapsto p)$
8:              $WinningPlayer \leftarrow p$, $AllocatedItem \leftarrow j$
9:          end if
10:     end for
11:     $Approx \leftarrow Approx \cup BestAllocation$
12:     $Unallocated \leftarrow Unallocated \setminus (AllocatedItem \cup Dep_{WinningPlayer}^+(AllocatedItem))$
13: end while
```

We show Algorithm 3.1 has approximation guarantee $1/(d+2)$, using a hybrid argument. This will prove Theorem 2.5.

**Proof:** [of Theorem 2.5] Let $OPT$ be an optimal solution with value opt and let $APPROX$ be the output of Algorithm 3.1 with value approx. For iteration $i$ of the loop at line 2 of Algorithm 3.1, let $APPROX_i$ be the allocations made at the first $i$ iterations, let $OPT_i$ be the allocations made by $OPT$ for the items that have not yet been allocated and let $HYBRID_i = APPROX_i \cup OPT_i$

be a hybrid solution. Let $HYBRID_i^p$ and $OPT_i^p$ be the items allocated in $HYBRID_i$ and $OPT_i$ (respectively) to player $p \in P$. Note that $HYBRID_0 = OPT$ and $HYBRID_t = APPROX$, where $t$ is the total number of iterations. We prove the following lemma:

**Lemma 3.1** *Let $i$ be an iteration of the loop at line 2 of Algorithm 3.1 and let $p^*$ be the player to whom items are allocated at iteration $i$. Then,*

$$\begin{aligned}&(d+2)(v_{p^*}(APPROX_i^{p^*}) - v_{p^*}(APPROX_{i-1}^{p^*}))\\ &\geq \sum_{p=1}^{n} \left( v_p(OPT_{i-1}^p \mid APPROX_{i-1}^p) - v_p(OPT_i^p \mid APPROX_i^p) \right) \ .\end{aligned}$$

*That is, the value lost by any iteration is bounded by $d+2$ times the value gained by the same iteration.*

**Proof:** Let $x = v_{p^*}(APPROX_i^{p^*}) - v_{p^*}(APPROX_{i-1}^{p^*})$. Roughly speaking, we prove that:

1. For an item allocated to some $p' \neq p^*$ in $OPT$, the loss to the value of $p'$ for not getting the item is at most $x$.
2. Having received items in the current iteration, the loss in marginal value of future items given to $p^*$ is at most $x$.

The first "contributes" to the "damage" up to $(d+1) \cdot x$, since at most $d+1$ items are allocated at each iteration. The second "contributes" up to another $x$, and for any other player $HYBRID_{i-1} = HYBRID_i$. We prove the lemma formally. Let $j_1, \ldots j_{d'}$ be the items allocated at iteration $i$ and let $P'$ be the set of players $p' \neq p^*$ such that at least one of the items $j_1, \ldots, j_{d'}$ is allocated to $p'$ in $OPT_{i-1}$. Let $p' \in P'$ and let $\hat{j}_1, \ldots, \hat{j}_{d''} \in \{j_1, \ldots, j_{d'}\}$ be all the items of $j_1, \ldots, j_{d'}$, allocated to

$p'$ by $OPT_{i-1}$. Then,

$$\begin{aligned}
& v_{p'}\left(OPT_{i-1}^{p'}\middle|APPROX_{i-1}^{p'}\right) - v_{p'}\left(OPT_{i}^{p'}\middle|APPROX_{i}^{p'}\right) \\
&= v_{p'}\left(\{\hat{j}_1,\dots\hat{j}_{d''}\}\middle|OPT_i^{p'} \cup APPROX_{i-1}^{p'}\right) \\
&= \sum_{k=1}^{d''} v_{p'}\left(\hat{j}_k\middle|\bigcup_{k'=k+1}^{d''}\{\hat{j}_{k'}\} \cup OPT_i^{p'} \cup APPROX_{i-1}^{p'}\right) \\
&\leq d'' \cdot \max_{k=1}^{d''} v_{p'}\left(\hat{j}_k\middle|\bigcup_{k'=k+1}^{d''}\{\hat{j}_{k'}\} \cup OPT_i^{p'} \cup APPROX_{i-1}^{p'}\right) \\
&\leq d'' \cdot \max_{k=1}^{d''} v_{p'}\left(\hat{j}_k\middle|\left(\left(\bigcup_{k'=k+1}^{d''}\{\hat{j}_{k'}\} \cup OPT_i^{p'}\right) \cap Dep_{p'}^{+}(\hat{j}_k)\right) \cup APPROX_{i-1}^{p'}\right) \\
&\leq d'' \cdot \max_{k=1}^{d''} v_{p'}\left(\hat{j}_k \cup \left(\left(\bigcup_{k'=k+1}^{d''}\{\hat{j}_{k'}\} \cup OPT_i^{p'}\right) \cap Dep_{p'}^{+}(\hat{j}_k)\right)\middle|APPROX_{i-1}^{p'}\right) \\
&\leq d'' \cdot \max_{k=1}^{d''} v_{p'}\left(\hat{j}_k \cup \left(Dep_{p'}^{+}(\hat{j}_k) \setminus \bigcup_{p\in P} APPROX_{i-1}^{p}\right)\middle|APPROX_{i-1}^{p'}\right) \\
&\leq d'' \cdot v_{p^*}\left(j_1,\dots,j_{d'}\middle|APPROX_{i-1}^{p^*}\right) \\
&= d'' \cdot \left(v_{p^*}\left(APPROX_i^{p^*}\right) - v_{p^*}\left(APPROX_{i-1}^{p^*}\right)\right)
\end{aligned}$$

where, the first equality follows by definitions and by observing that for any player $p \neq p^*$, $APPROX_{i-1}^{p} = APPROX_i^{p}$; the second by definitions. The first inequality is trivial; the second follows by Definition 2.7; the third and fourth by monotonicity; the fifth by line 5 of Algorithm 3.1. The last equality follows by definitions. Since there are only $d' \leq d+1$ items allocated, and since for any player $p \notin P' \cup \{p^*\}$, $HYBRID_i^p = HYBRID_{i-1}^p$, we get,

$$\begin{aligned}
&(d+1)(v_{p^*}(APPROX_i^{p^*}) - v_{p^*}(APPROX_{i-1}^{p^*})) \\
&\geq \sum_{p\in P\setminus\{p^*\}} (v_p(OPT_{i-1}^p \mid APPROX_{i-1}^p) - v_p(OPT_i^p \mid APPROX_i^p)) \ . \qquad (1)
\end{aligned}$$

For player $p^*$, we have by monotonicity $v_{p^*}(HYBRID_i^{p*}) \geq v_{p^*}(HYBRID_{i-1}^{p*})$. Hence,

$$v_{p^*}(OPT_{i-1}^{p^*} \mid APPROX_{i-1}^{p^*}) + v_{p^*}(APPROX_{i-1}^{p^*}) \leq v_{p^*}(OPT_i^{p^*} \mid APPROX_i^{p^*}) + v_{p^*}(APPROX_i^{p^*})$$

and then,

$$v_{p^*}(OPT_{i-1}^{p^*} \mid APPROX_{i-1}^{p^*}) - v_{p^*}(OPT_i^{p^*} \mid APPROX_i^{p^*}) \leq v_{p^*}(APPROX_i^{p^*}) - v_{p^*}(APPROX_{i-1}^{p^*}) \ .$$

This and (1) prove Lemma 3.1. □

We use Lemma 3.1 to complete the proof of Theorem 2.5. Let $x_i$ be the profit of Algorithm 3.1

at iteration $i$. Then,

$$\begin{aligned}
\mathsf{opt} &= \sum_{p=1}^{n} v_p(OPT_0^p \mid APPROX_0^p) \\
&= \sum_{p=1}^{n} \left(v_p(OPT_0^p \mid APPROX_0^p) - v_p(OPT_t^p \mid APPROX_t^p)\right) \\
&= \sum_{p=1}^{n} \sum_{i=0}^{t-1} \left(v_p(OPT_i^p \mid APPROX_i^p) - v_p(OPT_{i+1}^p \mid APPROX_{i+1}^p)\right) \\
&\leq (d+2) \cdot \sum_{i=1}^{t} x_i = (d+2) \cdot \mathsf{approx}
\end{aligned}$$

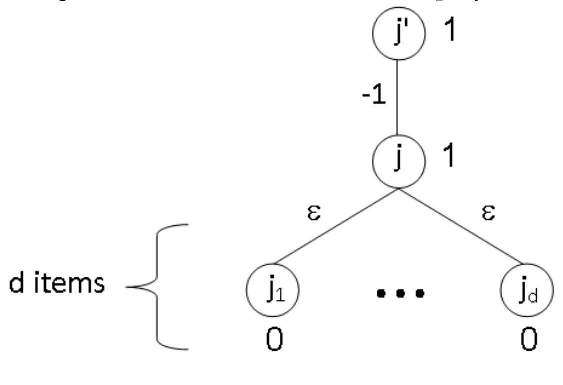
This proves Theorem 2.5. $\square$

## 3.2 A tight example

The following example shows Algorithm 3.1 may return a solution with value arbitrarily close to $\frac{1}{d+2}$, which matches the upper bound we proved in Theorem 2.5.

**Example 3.2** *Let $\mathcal{I}(P, M, v)$ be an instance of the welfare maximization problem with players $P = \{1, 2\}$, items $M = \{j, j_1, \ldots, j_d, j'\}$ and valuation functions $v_1$ as in the hypergraph representation shown in Figure 1 below and $v_2(S) = |S \setminus \{j'\}|$, for any $S \subseteq M$. It is easy to observe the optimal solution gives all the items except $j'$ to player 2 and $j'$ to player 1. The value of this solution is $d+2$. On the other hand, Algorithm 3.1 gives all the items except $j'$ to player 1, and gives $j'$ to an arbitrary player. This solution has value of only $1 + d \cdot \varepsilon$, which tends to 1 as $\varepsilon$ decreases.*

Figure 1: Valuation function of player 1



# 4 Approximation guarantee linear in dependency degree

We discuss two possible alternatives for proving Theorem 2.6. One is adapting Algorithm 3.1, as we briefly discuss in this section and the other is presented in Appendix B. We sketch a possible adaptation of Algorithm 3.1. Recall that Algorithm 3.1 does the following in each iteration: for each player and item, it calculates the marginal value of the item and all its unallocated supermodular dependencies with respect to the allocated items of the player. A player and items with maximum value are selected. The modification here is twofold. Firstly, for each player and item, one considers not only the subset of *all* unallocated dependencies of the item, but *any possible subset* of unallocated dependencies. Secondly, one does not consider the marginal value of an item *together with its dependencies* with respect to the previously allocated items, but *only of the item*, with respect to the subset of dependencies under consideration and the previously allocated items, together. It can be shown that the "damage" for any player, caused by an allocation of a single item to another player, is bounded by the "profit" of the iteration "damaging" it, and also (unlike the case of supermodular dependency) that the player getting the allocation has no "damage" at all in future iterations.

The other alternative, fully presented in Appendix B, is designing a different greedy algorithm. This algorithm has the somewhat surprising property that when considering which items to add to a player, it completely ignores the items that the player already has (despite the fact that these items determine the marginal values for new items).

## 5 Acknowledgements

This research was supported by The Israel Science Foundation (grant No. 621/12). We thank Inbal Talgam-Cohen for her useful comments on an earlier version of this manuscript.

## References

[1] I. Abraham, M. Babaioff, S. Dughmi, and T. Roughgarden. Combinatorial auctions with restricted complements. In *EC*, 2012.

[2] P. Berman. A $d/2$ approximation for maximum weight independent set in $d$-claw free graphs. *Nordic Journal of Computing*, 7:178–184, 2000. Preliminary version in SWAT'00.

[3] L. Blumrosen and N. Nisan. On the computational power of demand queries. *SIAM Journal on Computing*, 39:1372–1391, 2009.

[4] Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet. Multiagent resource allocation in $k$-additive domains: preference representation and complexity. *Annals of Operations Research*, 163:49–62, 2008.

[5] V. Conitzer, T. Sandholm, and P. Santi. Combinatorial auctions with $k$-wise dependent valuations. In *AAAI*, pages 248–254, 2005.

[6] S. Dobzinski, N. Nisan, and M. Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. *Mathematics of Operations Research*, 35:1–13, 2010. Preliminary version in STOC'05.

[7] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards*, 69:125–130, 1965.

[8] U. Feige. On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing*, 39:122–142, 2009. Preliminary version in STOC'06.

[9] U. Feige and J. Vondrák. Approximation algorithms for allocation problems: Improving the factor of $1 - 1/e$. In *FOCS*, pages 667–676, 2006.

[10] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions - II. *Mathematical Programming Study*, 8:73–87, 1978.

[11] H. N. Gabow. *Implementation of algorithms for maximum matching on nonbipartite graphs.* PhD thesis, Stanford University, 1974.

[12] Z. Galil. Efficient algorithms for finding maximal matching in graphs. Technical report, Columbia University, New York, 1983.

[13] Z. Galil, S. Micali, and H. N. Gabow. An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal on Computing*, 15:120–130, 1986.

[14] E. Hazan, S. Safra, and O. Schwartz. On the complexity of approximating $k$-set packing. *Computational Complexity*, 15:20–39, 2006.

[15] B. Lehmann, D. J. Lehmann, and N. Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55:270–296, 2006. Preliminary version in EC'2001.

[16] E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4:133–157, 1994.

[17] M. Schapira S. Dobzinski. An improved approximation algorithm for combinatorial auctions with submodular bidders. In *SODA*, pages 1064–1073, 2006.

[18] J. Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, pages 67–74, 2008.

## A Symmetry of dependency relations

**Lemma A.1 (Symmetry)** *Let $p \in P$ and let $j_1, j_2 \in M$ be such that $j_1 \to_p j_2$. Then $j_2 \to_p j_1$. In other words, the relation '$\to_p$' is symmetric.*

Note that the same is true for the relation $\to^+$ and that the proof is exactly the same.

**Proof:** Let $S$ be such that $j_1 \overset{S}{\to} j_2$. We show that

$$j_2 \xrightarrow{S \setminus \{j_2\} \cup \{j_1\}} j_1 \ .$$

From Definition 2.5, on one hand,

$$v(\{j_1\} \cup S) = v(j_1 \mid S) + v(j_2 \mid S \setminus \{j_2\}) + v(S \setminus \{j_2\}) \ ,$$

and on the other hand,

$$v(\{j_1\} \cup S) = v(j_2 \mid S \setminus \{j_2\} \cup \{j_1\}) + v(j_1 \mid S \setminus \{j_2\}) + v(S \setminus \{j_2\}) \ .$$

By subtracting $v(S \setminus \{j_2\})$, we get:

$$v(j_1 \mid S) + v(j_2 \mid S \setminus \{j_2\}) = v(j_2 \mid S \setminus \{j_2\} \cup \{j_1\}) + v(j_1 \mid S \setminus \{j_2\}) \ .$$

Since $j_1 \xrightarrow{S} j_2$ means $v(j_1 \mid S) \neq v(j_1 \mid S \setminus \{j_2\})$, we get $v(j_2 \mid S \setminus \{j_2\}) \neq v(j_2 \mid S \setminus \{j_2\} \cup \{j_1\})$. The latter is exactly the definition of $j_2 \xrightarrow{S \setminus \{j_2\} \cup \{j_1\}} j_1$, as desired. □

## B A greedy $\frac{1}{d+1}$-approximation algorithm for dependency degree at most $d$

In this appendix we prove Theorem 2.6.

### B.1 The algorithm

The algorithm is Algorithm B.1.

**Algorithm B.1** Greedily Approximate Welfare Maximization with Guarantee Linear in Dependency Degree

Input:

- An instance $\mathcal{I}(P, M, v)$ of the welfare maximization problem.
- A value queries oracle and a dependency graph for each of the valuation functions.

Output: A solution with approximation guarantee $\frac{1}{d+1}$, where $d$ is the dependency degree of $\mathcal{I}$.

```
1: $Unallocated \leftarrow M$, $Approx \leftarrow \emptyset$
2: while $Unallocated \neq \emptyset$ do
3:   $MaxMarginalUtility \leftarrow -1$
4:   for all $j \in Unallocated$, $p \in P$, $S' \subseteq (Dep_p(j) \cap Unallocated)$ do
5:     if $v_p(j \mid S') > MaxMarginalUtility$ then
6:       $MaxMarginalUtility \leftarrow v_p(j \mid S')$
7:       $BestAllocation \leftarrow (\{j\} \cup S' \mapsto p)$
8:       $WinningPlayer \leftarrow p$, $AllocatedItem \leftarrow j$, $AllocatedOptimalDependencies \leftarrow S'$
9:     end if
10:  end for
11:  $Approx \leftarrow Approx \cup BestAllocation$
12:  $Unallocated \leftarrow Unallocated \setminus (AllocatedItem \cup AllocatedOptimalDependencies)$
13:  $Unallocated \leftarrow Unallocated \setminus Dep_{WinningPlayer}(AllocatedItem)$ {Discard also unallocated
     dependencies of $j$}
14: end while
```

Intuitively, Algorithm B.1 promises at each iteration that the most possibly contributing item will have its full contribution in the approximated solution. Thus, any mislocated item in an optimal solution cannot "damage" it in more than the benefit of the iteration "damaging" it. We conclude the approximation guarantee by observing no more than $d+1$ items are allocated at each iteration.

Few remarks are in place.

**Remark B.1** *Algorithm B.1 does not look at all at the value of already allocated items (neither at new inspected items relatively to them nor at the whole sub-allocation). Note that an approach looking* only *at the marginal value with respect to already allocated items without looking on "forward" dependencies does not work.*[6]

**Remark B.2** *Algorithm B.1 discards unallocated dependencies (at line 13). This is to ensure any selected item will have its inspected marginal value. In other words, we ensure that the only dependencies of an item at the rest of the solution (i.e. the unallocated part) will be its optimal dependencies, as inspected at lines 4-10. Of course, because of monotonicity, we may add the "discarded" items to any player we wish (for example to the player we allocated to the rest or each item to its best possibility, in any order). The tight example we will show is tight also for any of these possibilities.*

**Proof:** [of Theorem 2.6] Let $OPT$ be an optimal solution with value $\mathsf{opt}$ and let $APPROX$ be an output of Algorithm B.1 with value $\mathsf{approx}$. Let $t$ be the number of iterations of the "while" loop at line 2 of the run created $APPROX$. We layer $\mathsf{opt}$ and $\mathsf{approx}$ by writing both of them by iterations of Algorithm B.1.

$$\mathsf{opt} = \sum_{i=1}^{t}\sum_{k=1}^{d+1} v_{p_{opt}(i,k)}(j_{i,k} \mid OPT_{p_{opt}(i,k)} \setminus \bigcup_{i'=1}^{i-1}\bigcup_{k'=1}^{d+1}\{j_{i',k'}\} \setminus \bigcup_{k'=1}^{k}\{j_{i,k'}\})$$

$$\mathsf{approx} = \sum_{i=1}^{t}\sum_{k=1}^{d+1} v_{p_{app}(i)}(j_{i,k} \mid APPROX_{p_{app}(i)} \setminus \bigcup_{i'=1}^{i-1}\bigcup_{k'=1}^{d+1}\{j_{i',k'}\} \setminus \bigcup_{k'=1}^{k}\{j_{i,k'}\})$$

where:

- $j_{i,k}$ is the $k^{th}$ item allocated at iteration $i$, where $j_{i,1}$ is the final item assigned at line 7 of this iteration, and the rest are ordered arbitrarily.
- $p_{opt}(i,k)$ is the player to whom the $k^{th}$ item of iteration $i$ is allocated in $OPT$.
- $p_{app}(i)$ is the player to whom all the items of iteration $i$ are allocated in $APPROX$ (all items of any iteration of Algorithm B.1 are allocated to the same player).

Note that for simplicity, equations are written assuming exactly $d+1$ items are allocated at each iteration. The proof is correct also without this assumption.

Let $i \in [1..t]$ and let $Unallocated_i$ be the items of $Unallocated$ at line 4 of Algorithm B.1 at iteration $i$. Then, since $M = \bigcup_{p=1}^{n}(OPT_p) = \bigcup_{p=1}^{n}(APPROX_p)$, we have:

$$Unallocated_i = \bigcup_{p=1}^{n}\left(OPT_p \setminus \bigcup_{i'=1}^{i-1}\bigcup_{k'=1}^{d+1}\{j_{i',k'}\}\right) = \bigcup_{p=1}^{n}\left(APPROX_p \setminus \bigcup_{i'=1}^{i-1}\bigcup_{k'=1}^{d+1}\{j_{i',k'}\}\right)$$

[6]For example, we may have two items and two players, where the first player has set function $f(S) = |S|$ and the second has set function giving $\infty$ to both items together and 0 otherwise. An algorithm that looks only "backward" will allocate both items to player 1 and gain value 2 where an optimal solution has value $\infty$.

Therefore, for all $i \in [1..t]$, $k \in [1..d+1]$,

$$\bigcup_{p=1}^{n} \left( OPT_p \setminus \bigcup_{i'=1}^{i-1} \bigcup_{k'=1}^{d+1} \{j_{i',k'}\} \setminus \bigcup_{k'=1}^{k} \{j_{i,k'}\} \right) \subseteq Unallocated_i \ .$$

Then, by lines 4-10 and 13 of Algorithm B.1, for all $i \in [1..t]$, $k \in [1..d+1]$, $v_{p_{app}(i)}(j_{i,1} \mid APPROX_{p_{app}(i)} \setminus \bigcup_{i'=1}^{i-1} \bigcup_{k'=1}^{d+1} \{j_{i',k'}\} \setminus \{j_{i,1}\}) \geq v_{p_{opt}(i,k)}(j_{i,k} \mid OPT_{p_{opt}(i,k)} \setminus \bigcup_{i'=1}^{i-1} \bigcup_{k'=1}^{d+1} \{j_{i',k'}\} \setminus \bigcup_{k'=1}^{k} \{j_{i,k'}\})$ . Therefore and by invoking (2) and (2) together with monotonicity,

$$\begin{aligned}
&(d+1) \cdot \mathsf{approx} = \\
&(d+1) \cdot \sum_{i=1}^{t} \sum_{k}^{d+1} v_{p_{app}(i)}(j_{i,k} \mid APPROX_{p_{app}(i)} \setminus \bigcup_{i'=1}^{i-1} \bigcup_{k'=1}^{d+1} \{j_{i',k'}\} \setminus \bigcup_{k'=1}^{k} \{j_{i,k'}\}) \\
&\geq (d+1) \cdot \sum_{i=1}^{t} v_{p_{app}(i)}(j_{i,1} \mid APPROX_{p_{app}(i)} \setminus \bigcup_{i'=1}^{i-1} \bigcup_{k'=1}^{d+1} \{j_{i',k'}\} \setminus \{j_{i,1}\}) \\
&\geq \sum_{i=1}^{t} \sum_{k}^{d+1} v_{p_{opt}(i,k)}(j_{i,k} \mid OPT_{p_{opt}(i,k)} \setminus \bigcup_{i'=1}^{i-1} \bigcup_{k'=1}^{d+1} \{j_{i',k'}\} \setminus \bigcup_{k'=1}^{k} \{j_{i,k'}\}) \\
&= \mathsf{opt}
\end{aligned}$$

It is easy to see the running time of Algorithm B.1 is polynomial in $|M|, |P|$ and $2^c$. This proves Theorem 2.6. □

### B.1.1 An example justifying "discarding" items

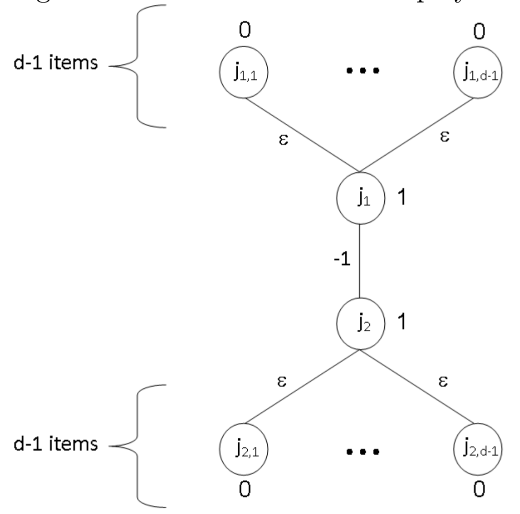
We demonstrate the necessity of "discarding unused dependencies" as in line 13 of Algorithm B.1. Note that another approach is to look also on already allocated items, as described in Section 4.

**Example B.1** *Let $|P| = 2$. Let the set functions be as follows: The valuation function of player 1 will be as in the hypergraph representation in Figure 2. The valuation function of player 2 will be $v_2(S) = |S|$.*

*Intuitively, the idea is to cause Algorithm B.1 to "ruin" a marginal value of an already allocated item, when trying to gain a maximal marginal value for another item. The middle edge with value $-1$ does so, without breaking monotonicity.*

*We analyze the approximation guarantee for this instance for Algorithm B.1 with line 13 omitted (i.e. without "discarding unused dependencies"). On the first iteration the algorithm allocates either $j_1$ or $j_2$ to player 1 with their optimal dependencies. Assume without loss of generality it is $j_1$. The optimal dependencies of $j_1$ are $j_{1,1}, \ldots, j_{1,d-1}$ and the marginal value of $j_1$ with these dependencies is $1 + (d-1) \cdot \varepsilon$. On the second iteration, the algorithm allocates $j_2$ to player 1, together with its optimal dependencies (that has not been allocated yet) $j_{2,1}, \ldots, j_{2,d-1}$. The marginal value of $j_2$ with respect to these dependencies is also $1 + (d-1) \cdot \varepsilon$. But now, the marginal value of $j_1$ with respect to the rest of the items allocated to player 1 (which had not been allocated yet when it was allocated) is only $(d-1) \cdot \varepsilon$. The algorithm terminates after the second iteration with social welfare*

Figure 2: Valuation function of player 1



*$1 + 2(d-1) \cdot \varepsilon$. On the other hand, allocating all the items to player 2 results in a social welfare of $2d$. For small enough $\varepsilon$ the approximation guarantee is arbitrarily close to $1/2d$ which is much worse than the approximation guarantee of $1/(d+1)$ of Algorithm B.1.*

*Note that Algorithm B.1 (the unmodified version) does much better for this instance. It also chooses firstly $j_1$ (without loss of generality) with its optimal dependencies, but then does not inspect anymore any of its dependencies, including $j_2$. Therefore, assuming $\varepsilon$ is small, it allocates $j_{2,1}, \dots, j_{2,d-1}$ to player 2 and gains for them a value of $d-1$ in addition to the marginal value it indeed gained for $j_1$. Thus, the approximated solution's total value Algorithm B.1 gains for this input is $1 + (d-1) \cdot \varepsilon + (d-1) = d + (d-1) \cdot \varepsilon$, which expresses an approximation guarantee of slightly more than $\frac{1}{2}$. Note that allocating $j_2$ to any of the players will do no harm (and allocating it to player 2 will even slightly help); just reinspecting it does the harm.*

## B.2 A tight example

We now show the analysis of Algorithm B.1 is tight.

**Example B.2** *Let $|P| = 2$ and let $|M| = m' \cdot (d+1)$ for some $m' \in \mathbb{N}$. We set an arbitrary ordering on the items and define $m'$ subsets of items $S_1, \dots S_{m'}$; the first set will be the first $d+1$ items, the second one will be the next $d+1$ items and so on. Let $v$ be the following set functions for any $S \subseteq M$:*

- $v_1(S) = \sum_{\substack{i \in [1..m'], \\ S_i \subseteq S}} (1+\varepsilon)$

  *(meaning, $(1+\varepsilon)$ times the number of subsets $S_i$ that are subsets of $S$).*

- $v_2(S) = |S|$.

*It is easy to see the marginal value of any item is maximized, when it is given to player 1 with all its dependencies. Moreover, since at this way only whole subsets are allocated, this is the situation* *at any iteration of Algorithm B.1. Therefore, algorithm B.1 allocates all the items to player* 1 *and gains total value of* $m' \cdot (1+\varepsilon)$. *In contrast, the optimal solution is to allocate all the items to player* 2 *and to gain total value of* $m$. *Thus, the approximation guarantee for this instance is close as we with to* $m/m' = 1/(d+1)$, *and the analysis of Theorem 2.6 is indeed tight.*

## C An exact algorithm for dependency degree at most 1

In this appendix, we present a full reduction (Reduction C.1) of the welfare maximization problem with dependency degree at most 1 to the maximum weighted matching problem.

**Reduction C.1**

Input: An instance $\mathcal{I}(P, M, v)$ of the welfare maximization problem.
Output: An instance $\mathcal{I}_M(V, E, w)$ of the maximum weighted matching problem, with set of vertices $V$, set of undirected edges $E$ and edge weights function $w$, such that each item $j \in M$ is represented by a corresponding vertex $v_j \in V$.
Reduction: For each item $j$, we have a vertex $u_j$. For each player $p$, we have the following. For each item $j$ with $Dep_p(j) = \emptyset$, we have an auxiliary vertex $u_j^p$ and an edge $(u_j, u_j^p)$ with weight $v_p(\{j\})$, representing the possibility of allocating $j$ to $p$. For each pair of items $j, j'$, such that $j \leftrightarrow_p j'$, we have a single auxiliary vertex $u_{j,j'}^p$ and three edges: $(u_j, u_{j,j'}^p)$ with weight $v_p(\{j\})$, representing the possibility of allocating $j$ to $p$ without allocating $j'$ to $p$; $(u_{j'}, u_{j,j'}^p)$ with weight $v_p(\{j'\})$, representing the possibility of allocating $j'$ to $p$ without allocating $j$ to $p$; $(u_j, u_{j'})$ with weight $v_p(\{j, j'\})$, representing the possibility of allocating both $j$ and $j'$ to $p$. Note that both $j$ and $j'$ have no other dependencies, since the dependency degree is at most 1 and by Lemma A.1. Note also that multiedges may be resolved, by choosing, without loss of generality, one edge with maximum weight for each pair of vertices.

The following observation is straightforward:

**Observation C.1**

- *Reduction C.1 is polynomial time computable.*
- *Every feasible solution for* $\mathcal{I}_M$ *induces a feasible solution for* $\mathcal{I}$ *with the same value, that may be computed in polynomial time.*
- *Every feasible solution for* $\mathcal{I}$, *induces a feasible solution for* $\mathcal{I}_M$ *with at least the same value.*

Therefore, we may use Reduction C.1 together with any exact polynomial time maximum weighted matching algorithm (see for example [7, 11, 12, 13]) to derive an exact polynomial time algorithm for the welfare maximization problem with dependency degree at most 1.

**Remark C.1** *Note that Reduction C.1 does not work for* $c > 1$. *This is since we may have a player* $p$ *with items* $j_1, j_2, j_3$, *such that* $j_1 \leftrightarrow j_2 \leftrightarrow j_3$ *(i.e. a "chain"). At this case,* $j_1, j_2$ *and* $j_3$ *may be allocated together to* $p$, *although this allocation does not induce a feasible matching.*