On the complexity of binary polynomial optimization over acyclic hypergraphs*

Alberto Del Pia† Silvia Di Gregorio‡

Abstract

In this work we advance the understanding of the fundamental limits of computation for Binary Polynomial Optimization (BPO), which is the problem of maximizing a given polynomial function over all binary points. In our main result we provide a novel class of BPO that can be solved efficiently both from a theoretical and computational perspective. In fact, we give a strongly polynomial-time algorithm for instances whose corresponding hypergraph is β-acyclic. We note that the β-acyclicity assumption is natural in several applications including relational database schemes and the lifted multicut problem on trees. Due to the novelty of our proving technique, we obtain an algorithm which is interesting also from a practical viewpoint. This is because our algorithm is very simple to implement and the running time is a polynomial of very low degree in the number of nodes and edges of the hypergraph. Our result completely settles the computational complexity of BPO over acyclic hypergraphs, since the problem is NP-hard on α-acyclic instances. Our algorithm can also be applied to any general BPO problem that contains β-cycles. For these problems, the algorithm returns a smaller instance together with a rule to extend any optimal solution of the smaller instance to an optimal solution of the original instance.

1 Introduction

In *binary polynomial optimization* we seek a binary point that maximizes a given polynomial function. This fundamental problem has a broad range of applications in several areas, including operations research, engineering, computer science, physics, biology, finance, and economics (see e.g., [8, 39, 21]). The generality of this problem is highlighted by the fact that the problem of maximizing *any* given function over all binary points can be reformulated as a binary polynomial optimization problem.

In order to formalize this optimization problem, a hypergraph representation is often used [17]. A *hypergraph* G is a pair (V, E), where V is the *node set* and E is the *edge set*, which is a family of non-empty subsets of V. We remark that the edge set E may contain parallel edges and loops, as opposed to the setting considered in [17, 19, 20]. In the hypergraph representation, each node represents a variable of the given polynomial function, whereas every edge represents a monomial. Therefore, any binary polynomial optimization problem can be formulated as

$$\text{(BPO)} \qquad \begin{aligned} \max \quad & \sum_{v \in V} p_v x_v + \sum_{e \in E} p_e \prod_{v \in e} x_v \\ \text{s.t.} \quad & x \in \{0,1\}^V. \end{aligned}$$

In this formulation, x is the decision vector, and an instance comprises of a hypergraph $G = (V, E)$ together with a *profit* vector $p \in \mathbb{Z}^{V \cup E}$. We remark that a rational profit vector can be scaled to be integral by multiplying it by the least common multiple of the denominators and this transformation leads to a polynomial growth of the instance size (see Remark 1.1 in [12]).

The main goal of this paper is that of advancing the understanding of the fundamental limits of computation for (BPO). In fact, while there are several known classes of binary *quadratic* optimization that are polynomially solvable (see for instance [1, 45, 13, 41, 42]), very few classes of higher degree (BPO) are known to be solvable in polynomial-time. These are instances that have: (i) incidence graph or co-occurrence graph of fixed treewidth [14, 41, 5], or (ii) objective function whose restriction to $\{0,1\}^n$ is supermodular (see Chapter 45 in [48]), or (iii) a highly acyclic structure [20], which we discuss in detail below.

*This research was supported in part by National Science Foundation award CMMI-1634768.

†Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI, USA; Wisconsin Institute for Discovery, University of Wisconsin-Madison, Madison, WI, USA.

‡Machine Learning for Computer Vision, Faculty of Computer Science, Technische Universität Dresden, Dresden, Germany.

Notice that, in the quadratic setting, the hypergraphs representing the instances are actually graphs. It is known that instances over acyclic graphs can be solved in strongly polynomial time [14]. Motivated by this fact, it is natural to analyze the computational complexity of (BPO) in the setting in which the hypergraph G does not contain any cycle. However, for hypergraphs, the definition of cycle is not unique. As a matter of fact, one can define Berge-cycles, γ-cycles, β-cycles, and α-cycles [25, 35]. Correspondingly, one obtains Berge-acyclic, γ-acyclic, β-acyclic, and α-acyclic hypergraphs, in increasing order of generality. The definitions of β-acyclic and α-acyclic hypergraph are given in Sections 1.1 and 1.2, and we refer the reader to [25] for the remaining definitions.

In [20], Del Pia and Khajavirad show that it is possible to solve (BPO) in polynomial-time if the corresponding hypergraph is kite-free β-acyclic. It should be noted that this class of hypergraphs lies between γ-acyclic and β-acyclic hypergraphs. This result is obtained via *linearization,* which is a technique that consists in linearizing the polynomial objective function via the introduction of new variables y_e, for $e \in E$, defined by $y_e = \prod_{v \in e} x_v$. This leads to an extended integer linear programming formulation in the space of the (x, y) variables, which is obtained by replacing the nonlinear constraints $y_e = \prod_{v \in e} x_v$, for all $e \in E$, with the inequalities that describe its convex hull on the unit hypercube [26]. The convex hull of the feasible points is known as the *multilinear polytope*, as defined in [17]. The tractability result in [20] is then achieved by providing a linear programming extended formulation of the multilinear polytope of polynomial size. The linearization technique also led to several other polyhedral results for (BPO), including [17, 15, 19, 18, 10, 20, 16, 32].

A different approach to study binary polynomial optimization involves quadratization techniques [46, 27, 11, 33, 34]. The common idea in the quadratization approaches is to add additional variables and constraints so that the original polynomial can be expressed in a higher dimensional space as a new quadratic polynomial. The reason behind it is that in this way it is possible to exploit the vast literature available for the quadratic case. An alternative approach is to use a different formalism altogether like pseudo-Boolean optimization [30, 31, 14, 8, 7, 6]. Pseudo-Boolean optimization is a more general framework, as in fact the goal is to optimize set functions that admit closed algebraic expressions.

1.1 A strongly polynomial-time algorithm for β-acyclic hypergraphs Our main result is an algorithm that solves (BPO) in strongly polynomial-time whenever the hypergraph corresponding to the instance is β-acyclic.

To formally state our tractability result, we first provide the definition of β-acyclic hypergraph [25]. A hypergraph is *β-acyclic* if it does not contain any β-cycle. A *β-cycle* of length q, for some $q \geq 3$, is a sequence $v_1, e_1, v_2, e_2, \ldots, v_q, e_q, v_1$ such that $v_1, v_2, \ldots, v_q$ are distinct nodes, $e_1, e_2, \ldots, e_q$ are distinct edges, and v_i belongs to e_{i-1}, e_i and no other e_j for all $i = 1, \ldots, q$, where $e_0 = e_q$.

Our algorithm is based on a dynamic programming-type recursion. The idea behind it is to successively remove a nest point from G, until there is only one node left in the hypergraph. In fact, we observe that optimizing the problem becomes trivial when there is only one node left. A node u of a hypergraph is a *nest point* if for every two edges e, f containing u, either $e \subseteq f$ or $f \subseteq e$. Equivalently, the set of the edges containing u is totally ordered. Observe that, in connected graphs with at least two nodes, nest points coincide with leafs. Therefore, nest points can be seen as an extension of the concept of leaf in a graph to the hypergraph setting. Before going forward, we remark that finding a nest point in a hypergraph can be done in strongly polynomial-time by brute force [44]. We denote by τ the number of operations required to find a nest point, which is bounded by a polynomial in $|V|$ and $|E|$. We are now ready to state our main result.

THEOREM 1.1. *There is a strongly polynomial-time algorithm to solve* (BPO)*, provided that the input hypergraph $G = (V, E)$ is β-acyclic. In particular, the number of arithmetic operations performed is $O(|V|(\tau + |E| + |V| \log |E|))$.*

The description of the algorithm and the proof of Theorem 1.1 can be found in Section 2. Theorem 1.1 provides a novel class of (BPO) that can be solved efficiently both from a theoretical and computational perspective. In fact, this class of problems is not contained in the classes (i), (ii), or (iii) for which a polynomial-time algorithm was already known. This can be seen because a laminar hypergraph $G = (V, E)$ with edges $e_1 \subseteq e_2 \subseteq \cdots \subseteq e_m = V$ is β-acyclic and does not satisfy the assumptions in (i). Furthermore, it is simple to see that there exist polynomials whose restriction to $\{0, 1\}^n$ is not supermodular and the corresponding hypergraph is β-acyclic. Finally, it is well-known that the class of β-acyclic hypergraphs significantly extends the class of kite-free β-acyclic hypergraphs.

The concept of β-acyclicity is not interesting only in a theoretical context. To the contrary, this assumption is quite natural in several real world applications. A thorough discussion of this topic is not in the scope of this

paper, where instead we only mention a couple of examples. In the study of relational database schemes, the β-acyclicity assumption is renowned to be advantageous [24]. In fact, a number of basic and desirable properties in database theory turn out to be equivalent to acyclicity. A second example is given by the lifted multicut problem on trees, where the problem can be equivalently formulated via binary polynomial optimization [40]. The goal of the lifted multicut problem is to partition a given graph in a way that minimizes the total cost associated with having different pairs of nodes in different components. This problem has been shown to be very useful in the field of computer vision, in particular when applied to image segmentation [3], object tracking [49], and motion segmentation [36]. Even when the underlying graph is a tree, the lifted multicut problem is NP-hard. However it can be solved in polynomial time when we focus on paths rather than on trees. It is simple to observe that this special case is formulated with a polynomial whose hypergraph is β-acyclic. Lastly, we observe that these β-acyclic hypergraphs can exhibit kites, and therefore do not fit into the previous studies [20].

The interest of Theorem 1.1 also lies in the novelty of the proving technique with respect to the other recent results in the field previously mentioned. In particular, our algorithm does not rely on linear programming, extended formulations, polyhedral relaxations, or quadratization. This in turn leads to two key advantages. First, our algorithm it is very simple to implement. Second, we obtain a *strongly* polynomial time algorithm (as opposed to a *weakly* polynomial time algorithm) and the running time is a polynomial of very low degree in the number of nodes and edges of the hypergraph. These two key points contribute to making our algorithm interesting also from a practical viewpoint. Furthermore, we remark that it is possible to recognize efficiently when (BPO) is represented by a β-acyclic hypergraph [25].

Theorem 1.1 has important implications in polyhedral theory as well. In particular, it implies that one can optimize over the multilinear polytope for β-acyclic hypergraphs in strongly polynomial-time. By the polynomial equivalence of separation and optimization (see, e.g., [12]), for this class of hypergraphs, the separation problem over the multilinear polytope can be solved in polynomial-time.

We remark that our algorithm in Theorem 1.1 can be applied also to hypergraphs that are not β-acyclic. In this case, the algorithm does not return an optimal solution to the given instance. However, it returns a smaller instance together with a rule to construct an optimal solution to the original instance, given an optimal solution to the smaller instance. Therefore, our algorithm can be used as a reduction scheme to decrease the size of a given instance. Via computational experiments, we generate random instances and study the magnitude of this decrease. In particular, the results of our simulations show that the percentage of removed nodes is on average 50% whenever the number of the edges is half the number of nodes. We discuss this topic in Section 3.

1.2 Settling the complexity of (BPO) over acyclic hypergraphs Theorem 1.1 allows us to completely settle the computational complexity of (BPO) over acyclic hypergraphs. More specifically, it can be seen that two hardness results hold for (BPO) when the input hypergraphs belong to the next class of acyclic hypergraphs, in increasing order of generality, that is the one of α-acyclic hypergraphs. Several equivalent definitions of α-acyclic hypergraphs are known (see, e.g., [2, 25, 9]). In the following, we will use the characterization stated in Theorem 1.2 below. This characterization is based on the concept of removing nodes and edges from a hypergraph. When we *remove* a node u from $G = (V, E)$ we are constructing a new hypergraph $G' = (V', E')$ with $V' = V \setminus \{u\}$ and $E' = \{e \setminus \{u\} : e \in E,\ e \neq \{u\}\}$. Observe that when we remove a node we might be introducing loops and parallel edges in the hypergraph. When we *remove* an edge f from $G = (V, E)$, we construct a new hypergraph $G' = (V, E')$, where $E' = E \setminus \{f\}$.

THEOREM 1.2. ([2]) *A hypergraph G is α-acyclic if and only if the empty hypergraph $(\emptyset, \emptyset)$ can be obtained by applying the following two operations repeatedly, in any order:*

1. *if a node v belongs to at most one edge, then remove v;*
2. *if an edge e is contained in another edge f, then remove e.*

We claim that both Simple Max-Cut and Max-Cut can be formulated as special cases of (BPO) where the hypergraphs representing the problems are α-acyclic. It is well-known that both these problems can be formulated as binary quadratic problems [12]. Then, we define the corresponding instance of (BPO) starting from the graph representing the instance of the binary quadratic problem. Namely, we construct the hypergraph by adding to the graph one edge of weight zero that contains all the nodes. Theorem 1.2 implies that such hypergraph is α-acyclic. At this point, it can be seen that the corresponding instance of (BPO) is equivalent to the original quadratic instance. Therefore, the known hardness results of Simple Max-Cut and Max-Cut [28, 50] transfer to this setting,

yielding the following hardness result.

THEOREM 1.3. (BPO) *over α-acyclic hypergraph is strongly NP-hard. Furthermore, it is NP-hard to obtain a r-approximation for* (BPO)*, with $r > \frac{16}{17} \approx 0.94$, unless $P = NP$.*

The reduction just described shows that the statement of Theorem 1.3 holds even if the values of the objective function belong to a restricted subset. The interested reader can find more details in Appendix B. Together, Theorem 1.1 and Theorem 1.3 completely settle the computational complexity of binary polynomial optimization over acyclic hypergraphs.

2 A strongly polynomial-time algorithm for β-acyclic hypergraphs

In this section we present the general algorithm for β-acyclic instances. Our algorithm makes use of a characterization of β-acyclic hypergraphs, which is based on the concept of removing nest points from the hypergraph. We remind the reader that the operation of removing a node is explained in Section 1.2. We are now ready to state this characterization of β-acyclic hypergraphs.

THEOREM 2.1. ([22]) *A hypergraph G is β-acyclic if and only if after removing nest points one by one we obtain the empty hypergraph $(\emptyset, \emptyset)$.*

We observe that Theorem 2.1 does not depend on the particular choice of the nest point to be removed at each step. Theorem 2.1 implies that, for our purposes, it suffices to understand how to reduce an instance of the problem to one obtained by removing a nest point u. In particular, realizing how to update the profit vector is essential. Once we solve the instance of the new problem without u, we decide whether to set the variable corresponding to u to zero or one depending on the values of the variables of the other nodes in the edges containing u, which are given by the solution of the smaller problem.

Before describing the algorithm, we explain some notation that will be used in this section. Let $u \in V$ be a nest point contained in k edges. Without loss of generality, we can assume that these edges are e_1, e_2, ..., e_k and that $e_1 \subseteq e_2 \subseteq \cdots \subseteq e_k$. For simplicity of notation, we denote by e_0 the set $\{u\}$ and by p_{e_0} the profit p_u. Moreover, we clearly have $e_0 \subseteq e_1$. We will divide the subcases to consider based on the sequence of the signs of

$$p_{e_0}, \quad p_{e_0} + p_{e_1}, \quad p_{e_0} + p_{e_1} + p_{e_2}, \quad \dots, \quad p_{e_0} + p_{e_1} + \cdots + p_{e_k}.$$

Note that the number of subcases can be exponential in the number of edges, however we find a compact formula for the optimality conditions, which in turn yields a compact way to construct the new profit vector p' for the hypergraph $G' = (V', E')$ obtained by removing u from G. We say that there is a *flip* in the sign sequence whenever the sign of the sequence changes. More precisely, a flip is *positive* if the sign sequence goes from non-positive to positive and the previous non-zero value of the sequence is negative. Similarly, we say that a flip is *negative* if the sequence goes from non-negative to negative and the previous non-zero value of the sequence is positive. We say that an edge e_i *corresponds to a flip* in the sign sequence, if there is a flip between $\sum_{j=0}^{i-1} p_{e_j}$ and $\sum_{j=0}^{i} p_{e_j}$.

In order to describe the several cases easily, in a compact way, we partition the indices $0, \dots, k$ into four sets $\mathscr{P}$, $\mathscr{N}$, $\mathscr{N}\mathscr{P}$, and $\mathscr{P}\mathscr{N}$. The first two sets are defined by

$$\begin{aligned} \mathscr{P} &:= \{i : i = 1, \dots, k, \ e_i \text{ corresponds to a positive flip}\}, \\ \mathscr{N} &:= \{i : i = 1, \dots, k, \ e_i \text{ corresponds to a negative flip}\}. \end{aligned}$$

If there is at least one flip, the sets $\mathscr{N}\mathscr{P}$, and $\mathscr{P}\mathscr{N}$ are defined as follows:

$$\begin{aligned} \mathscr{N}\mathscr{P} := &\{0, \dots, s-1 : s \text{ is the first flip and } s \in \mathscr{P}\} \\ &\cup \{i : \exists \text{two consecutive flips } s \in \mathscr{N}, t \in \mathscr{P} \text{ s.t. } s+1 \le i \le t-1\} \\ &\cup \{t+1, \dots, k : \text{if } t \text{ is the last flip and } t \in \mathscr{N}\}, \\ \mathscr{P}\mathscr{N} := &\{0, \dots, s-1 : s \text{ is the first flip and } s \in \mathscr{N}\} \\ &\cup \{i : \exists \text{two consecutive flips } s \in \mathscr{P}, t \in \mathscr{N} \text{ s.t. } s+1 \le i \le t-1\} \\ &\cup \{t+1, \dots, k : \text{if } t \text{ is the last flip and } t \in \mathscr{P}\}. \end{aligned}$$

Otherwise, if there is no flip, we define

$$\mathscr{N}\mathscr{P} := \{0, \dots, k : \text{if } p_{e_0} \leq 0\},$$
$$\mathscr{P}\mathscr{N} := \{0, \dots, k : \text{if } p_{e_0} > 0\}.$$

REMARK 1. *We observe that the indices $\{0, 1, \dots, k\}$ cycle between $\mathscr{N}\mathscr{P}$, $\mathscr{P}$, $\mathscr{P}\mathscr{N}$, $\mathscr{N}$ following this order. In fact, if $i \in \mathscr{P}$ then the following indices must be in $\mathscr{P}\mathscr{N}$ until we reach an index that belongs to $\mathscr{N}$. Similarly, if $i \in \mathscr{N}$ the indices after i must belong to $\mathscr{N}\mathscr{P}$ until there is an index in $\mathscr{P}$. Note that it can happen that there is an index in $\mathscr{P}$ and the next index is in $\mathscr{N}$. If this happens, then there are no indices in $\mathscr{P}\mathscr{N}$ between these two indices. Similarly, it may happen that there is an index in $\mathscr{N}$ followed immediately by an index in $\mathscr{P}$. Moreover, the index 0 belongs to either $\mathscr{N}\mathscr{P}$ or $\mathscr{P}\mathscr{N}$.* ◇

EXAMPLE 1. *Let us give an example to clarify the meaning of the sets $\mathscr{P}$, $\mathscr{N}$, $\mathscr{N}\mathscr{P}$, and $\mathscr{P}\mathscr{N}$. Consider a nest point u, contained in the edges e_1, e_2, e_3, e_4, e_5 such that $e_1 \subseteq e_2 \subseteq e_3 \subseteq e_4 \subseteq e_5$. Assume that $p_{e_0} = 3$, $p_{e_1} = -3$, $p_{e_2} = 1$, $p_{e_3} = -2$, $p_{e_4} = 3$, $p_{e_5} = 2$. We can check that $p_{e_0} = 3 > 0$, $p_{e_0} + p_{e_1} = 0$, $p_{e_0} + p_{e_1} + p_{e_2} = 1 > 0$, $p_{e_0} + p_{e_1} + p_{e_2} + p_{e_3} = -1 < 0$, $p_{e_0} + p_{e_1} + p_{e_2} + p_{e_3} + p_{e_4} = 2 > 0$ and finally $p_{e_0} + p_{e_1} + p_{e_2} + p_{e_3} + p_{e_4} + p_{e_5} = 4 > 0$. The indices $0, \dots, 5$ are partitioned in the sets $\mathscr{P}\mathscr{N} = \{0, 1, 2, 5\}$, $\mathscr{N} = \{3\}$, $\mathscr{N}\mathscr{P} = \emptyset$, $\mathscr{P} = \{4\}$. Observe that here there are no indices in $\mathscr{N}\mathscr{P}$ when we go from the negative flip corresponding to e_3 to the next positive flip, which corresponds to e_4.*

Our algorithm acts differently whether all the edges containing the nest point u are loops or not. Let us now consider the case where u is contained not only in loops. In this case, for a vector $x \in \{0,1\}^{V'}$, we define $\varphi(x) \in \{0, 1\}$ that will assign the optimal value to the variable corresponding to the nest point u, given the values of the variables corresponding to the nodes in V'. We denote by $\mu = \mu(x)$ the largest index $i \in \{0, \dots, k\}$, such that $x_v = 1$ for every $v \in e_i \setminus \{u\}$. We then set

$$\varphi(x) := \begin{cases} 1 & \text{if } \mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N} \\ 0 & \text{if } \mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P}. \end{cases}$$

Note that all the edges e that are loops $\{u\}$ satisfy trivially the condition $x_v = 1$ for every $v \in e \setminus \{u\}$, as $e \setminus \{u\} = \emptyset$. In particular, e_0 always satisfies this condition, hence μ is well defined. Given a vector $x \in \{0,1\}^V$, we denote by $\text{drop}_u(x)$ the vector in $\{0,1\}^{V'}$ obtained from x by dropping its entry corresponding to the node u. We then define $\mu(x) := \mu(\text{drop}_u(x))$ and $\varphi(x) := \varphi(\text{drop}_u(x))$.

In our algorithm we decide to keep loops and parallel edges for ease of exposition. An additional reason is that we avoid checking for loops and parallel edges at every iteration. Furthermore, in this way there is a bijection between $\{e \in E : e \neq \{u\}\}$ and E', which will be useful in the arguments below. In order to construct the new profit vector p', it is convenient to give a name to the index of the first edge in $e_0 \subseteq e_1 \subseteq \cdots \subseteq e_k$ that is not equal to $\{u\}$. We denote this index by λ. We remark that when u is not contained only in loops, the index λ is well defined. Next, observe that $p' \in \mathbb{R}^{V' \cup E'}$. We will use an abuse of notation for the indices of p' corresponding to the edges in E' obtained from $e_\lambda, \dots, e_k$ by removing u. We denote these indices by $e_\lambda, \dots, e_k$, even if these edges belong to E. This abuse of notation does not introduce ambiguity because of the bijection between $\{e \in E : e \neq \{u\}\}$ and E' and the fact that $\{e_i \in E : i = \lambda, \dots, k\} \subseteq \{e \in E : e \neq \{u\}\}$. We are now ready to present our algorithm for β-acyclic hypergraphs, which we denote by `Acyclic`(G, p).

Algorithm 1 `Acyclic`(G, p)

Find a nest point u. Let $e_1 \subseteq e_2 \subseteq \cdots \subseteq e_k$ be the edges containing it.
Compute $\mathscr{P}$, $\mathscr{N}$, $\mathscr{N}\mathscr{P}$, $\mathscr{P}\mathscr{N}$.
Construct the hypergraph $G' = (V', E')$ by removing u from G.
if $e_k = \{u\}$ **then**
    Set $x^*_u := \begin{cases} 1 & \text{if } \sum_{i=0}^k p_{e_i} \geq 0 \\ 0 & \text{if } \sum_{i=0}^k p_{e_i} < 0 \end{cases}$.
    if $|V| > 1$ **then**
        Set $p'_t := p_t$ for all $t \in V' \cup E'$.

8: Set $x' := \texttt{Acyclic}(G', p')$.
9: Set $x^*_w := x'_w$ for all $w \in V'$.
10: **else**
11: Find λ.
12: Set $p'_t := p_t$ for all $t \in V' \cup E' \setminus \{e_1, \dots, e_k\}$.
13: Set $p'_{e_i} := \begin{cases} 0 & \text{for every } i \in \mathscr{N}\mathscr{P} \cap \{i : i \geq \lambda\} \\ \sum_{r=0}^{i} p_{e_r} & \text{for every } i \in \mathscr{P} \cap \{i : i \geq \lambda\} \\ p_{e_i} & \text{for every } i \in \mathscr{P}\mathscr{N} \cap \{i : i \geq \lambda\} \\ -\sum_{r=0}^{i-1} p_{e_r} & \text{for every } i \in \mathscr{N} \cap \{i : i \geq \lambda\} \ . \end{cases}$
14: Set $x' := \texttt{Acyclic}(G', p')$.
15: Set $x^*_w := x'_w$ for all $w \in V'$.
16: Set $x^*_u := \varphi(x')$.
17: **return** x^*.

An example of the execution of the algorithm is presented in Appendix A. We next show the correctness of the algorithm.

PROPOSITION 2.1. *The algorithm* $\texttt{Acyclic}(G, p)$ *returns an optimal solution to* (BPO)*, provided that* G *is* β*-acyclic.*

Proof. We prove this proposition by induction on the number of nodes. We start from the base case, that is when $|V| = 1$. It follows that $e = \{u\}$ for all $e \in E$, since $\{e_1, \dots, e_k\} = E$. In this case the algorithm only performs lines 1-5 and line 17. There are only two possible solutions: either $x^*_u = 0$, or $x^*_u = 1$. The algorithm computes the objective corresponding to $x^*_u = 1$. If the objective is non-negative, it sets $x^*_u := 1$, otherwise it sets $x^*_u := 0$. The solution provided by the algorithm is optimal, since we are maximizing.

Next we consider the inductive step, and analyze the correctness of $\texttt{Acyclic}(G, p)$ when it removes a nest point. We define $\text{obj}(\cdot)$ to be the objective value of (BPO) yielded by a binary vector in $\{0,1\}^V$. Let u be the nest point to be removed at a given iteration of the algorithm. We denote by (BPO)$'$ the problem of the form (BPO) over the hypergraph G' and the profits p', defined by $\texttt{Acyclic}(G, p)$. Likewise, let $\text{obj}'(\cdot)$ be the objective value of (BPO)$'$ provided by a vector in $\{0,1\}^{V'}$. By the inductive hypothesis, the vector x' defined in line 8 or 14 is optimal to (BPO)$'$. Our goal is to show that the returned solution x^* is optimal to (BPO).

We consider first the case in which $e_k = \{u\}$, i.e., when all the edges that contain u are loops. This implies that every edge $e \in E$ is either a loop $\{u\}$ or does not contain the node u. Therefore, an optimal solution to (BPO) is obtained by combining an optimal solution to (BPO)$'$ with an optimal solution to the problem represented by the hypergraph $(\{u\}, \{e_1, \dots, e_k\})$ with profits p_u and p_{e_i}, for $i = 1, \dots, k$. By using the same proof of the base case, we can see that line 5 provide the optimal value of x^*_u. Since the vector x' is optimal to (BPO)$'$, we can conclude that the vector x^* returned by the algorithm is optimal.

Next, we consider the case in which e_k is not a loop.

CLAIM 1. *There exists an optimal solution* $\tilde{x}$ *to* (BPO) *such that* $\tilde{x}_u = \varphi(\tilde{x})$.

Proof of Claim 1. To show this, let $\bar{x}$ be an optimal solution to (BPO). If $\bar{x}_u = \varphi(\bar{x})$, then we are done. Thus, assume that $\bar{x}_u = 1 - \varphi(\bar{x})$, and let $\tilde{x}$ be obtained from $\bar{x}$ by setting $\tilde{x}_u := \varphi(\bar{x})$. Note however that $\varphi(\bar{x}) = \varphi(\tilde{x})$, since $\bar{x}_v = \tilde{x}_v$ for all nodes $v \neq u$. Therefore we want to show that $\tilde{x}$ is optimal. The proof splits in two cases: either $\varphi(\tilde{x}) = 0$, or $\varphi(\tilde{x}) = 1$.

Consider the first case $\varphi(\tilde{x}) = 0$. Hence $\bar{x}_u = 1$ and $\tilde{x}_u = 0$. Therefore, it follows that $\text{obj}(\bar{x}) = \text{obj}(\tilde{x}) + \sum_{i=0}^{\mu} p_{e_i}$. By definition of φ, we have $\mu = \mu(\tilde{x}) \in \mathscr{N} \cup \mathscr{N}\mathscr{P}$, thus $\sum_{i=0}^{\mu} p_{e_i} \leq 0$. Then, we obtain that $\text{obj}(\bar{x}) \leq \text{obj}(\tilde{x})$ and $\tilde{x}$ is optimal to (BPO) as well.

Assume now that we are in the second subcase, i.e., $\varphi(\tilde{x}) = 1$. Therefore we have $\bar{x}_u = 0$, $\tilde{x}_u = 1$, and $\text{obj}(\tilde{x}) = \text{obj}(\bar{x}) + \sum_{i=0}^{\mu} p_{e_i}$. Since $\mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N}$, it follows that $\sum_{i=0}^{\mu} p_{e_i} \geq 0$, therefore $\text{obj}(\tilde{x}) \geq \text{obj}(\bar{x})$. Thus, we can conclude that also $\tilde{x}$ is optimal to (BPO). ◇

We remark that, since $e_k \neq \{u\}$, the index λ is well defined and $\lambda \geq 1$. From now on let x be any vector $\{0,1\}^V$ such that $x_u = \varphi(x)$. Let $\mu = \mu(x)$.

Our next main goal is to show the equality

$$\text{obj}(x) = \begin{cases} \text{obj}'(\text{drop}_u(x)), & \text{if } \lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P} \\ \text{obj}'(\text{drop}_u(x)) + \sum_{i=0}^{\lambda-1} p_{e_i}, & \text{if } \lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}. \end{cases} \tag{2.1}$$

We define the sets A and B as follows. If $x_u = 0$ let $A := \emptyset$. Otherwise, that is if $x_u = 1$, we define $A := \{0, 1, \dots, \mu\}$. In order to define B we observe that either $\lambda \leq \mu$ or $\mu = \lambda - 1$. This is because $\lambda - 1$ is the index of the last loop $\{u\}$. We then define $B := \{\lambda, \dots, \mu\}$ if $\lambda \leq \mu$, otherwise we set $B := \emptyset$, if $\mu = \lambda - 1$. In order to prove (2.1), it suffices to check that

$$\sum_{i \in A} p_{e_i} = \begin{cases} \sum_{i \in B} p'_{e_i}, & \text{if } \lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P} \\ \sum_{i \in B} p'_{e_i} + \sum_{i=0}^{\lambda-1} p_{e_i}, & \text{if } \lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}, \end{cases} \tag{2.2}$$

by the definitions of p' and $\text{drop}_u(x)$. In the next claim, we study the value of $\sum_{i \in B} p'_{e_i}$, which is present in (2.2).

Claim 2. *Let $\lambda \leq \mu$. If $\mathscr{P} \cap \{\lambda, \dots, \mu\} = \emptyset$, then*

$$\sum_{i \in B} p'_{e_i} = \begin{cases} 0, & \text{if } \lambda \in \mathscr{N}\mathscr{P} \\ \sum_{i=\lambda}^{\mu} p_{e_i}, & \text{if } \lambda \in \mathscr{P}\mathscr{N} \text{ and } \mu \in \mathscr{P}\mathscr{N} \\ -\sum_{i=0}^{\lambda-1} p_{e_i}, & \text{if } \lambda \in \mathscr{N} \text{ or if } \lambda \in \mathscr{P}\mathscr{N} \text{ and } \mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P}. \end{cases} \tag{2.3}$$

If $\mathscr{P} \cap \{\lambda, \dots, \mu\} \neq \emptyset$, then

$$\sum_{i \in B} p'_{e_i} = \begin{cases} 0, & \text{if } \lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P} \text{ and } \mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P} \\ \sum_{i=0}^{\mu} p_{e_i}, & \text{if } \lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P} \text{ and } \mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N} \\ -\sum_{i=0}^{\lambda-1} p_{e_i}, & \text{if } \lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N} \text{ and } \mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P} \\ \sum_{i=\lambda}^{\mu} p_{e_i}, & \text{if } \lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N} \text{ and } \mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N}. \end{cases} \tag{2.4}$$

Proof of Claim 2. Observe that $\sum_{i \in B} p'_{e_i}$ is not trivially equal to zero, since $\lambda \leq \mu$.

First, we assume that $\mathscr{P} \cap \{\lambda, \dots, \mu\} = \emptyset$. In this case we can easily compute the value of $\sum_{i \in B} p'_{e_i}$. Assume first that $\lambda \in \mathscr{N}\mathscr{P}$. By Remark 1 it is easy to see that $\{\lambda, \dots, \mu\}$ must belong to $\mathscr{N}\mathscr{P}$. Then, by definition of p', it follows that $\sum_{i \in B} p'_{e_i} = 0$. Next, consider the case in which $\lambda \in \mathscr{P}\mathscr{N}$ and $\mu \in \mathscr{P}\mathscr{N}$. From Remark 1, we can conclude that $\{\lambda, \dots, \mu\} \subseteq \mathscr{P}\mathscr{N}$. By definition of p', we can observe that $\sum_{i \in B} p'_{e_i} = \sum_{i=\lambda}^{\mu} p_{e_i}$. Next, assume that $\lambda \in \mathscr{N}$. Since $\lambda \in \mathscr{N}$, it is easy to see that $\{\lambda + 1, \dots, \mu\} \subseteq \mathscr{N}\mathscr{P}$ by Remark 1. Hence by definition of p', we get that $\sum_{i \in B} p'_{e_i} = p'_{e_\lambda} = -\sum_{i=0}^{\lambda-1} p_{e_i}$. Lastly, let $\lambda \in \mathscr{P}\mathscr{N}$ and $\mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P}$. This implies that there must be exactly one index $q \in \mathscr{N} \cap \{\lambda + 1, \dots, \mu\}$. By using the definition of p' we obtain $\sum_{i \in B} p'_{e_i} = \sum_{i=\lambda}^{q-1} p'_{e_i} + p'_q + \sum_{i=q+1}^{\mu} p'_{e_i} = \sum_{i=\lambda}^{q-1} p_{e_i} - \sum_{i=0}^{q-1} p_{e_i} = -\sum_{i=0}^{\lambda-1} p_{e_i}$. This ends the proof of (2.3).

Next, we assume $\mathscr{P} \cap \{\lambda, \dots, \mu\} \neq \emptyset$. We divide $\sum_{i \in B} p'_{e_i}$ in three parts. Let ι_1 be the first index in $\mathscr{P} \cap \{\lambda, \dots, \mu\}$, and let ι_2 be the last index in $\mathscr{P} \cap \{\lambda, \dots, \mu\}$. Note that it is possible that $\iota_1 = \iota_2$. Then, we observe that

$$\sum_{i \in B} p'_{e_i} = \sum_{i=\lambda}^{\iota_1 - 1} p'_{e_i} + \sum_{i=\iota_1}^{\iota_2 - 1} p'_{e_i} + \sum_{i=\iota_2}^{\mu} p'_{e_i} \ . \tag{2.5}$$

Now we study the value of the sums in the right hand side of (2.5).

We start by showing that

$$\sum_{i=\iota_1}^{\iota_2 - 1} p'_{e_i} = 0 \ . \tag{2.6}$$

If it is vacuous, then it is trivially equal to zero. Then we assume that it is not vacuous. Since $\iota_2 \in \mathscr{P}$, the last index in this sum is in $\mathscr{N} \cup \mathscr{N}\mathscr{P}$. Because of the fact that the first index of the sum is in $\mathscr{P}$ and by definition

of p', we can conclude that all the profits in $\sum_{i=\iota_1}^{\iota_2-1} p'_{e_i}$ cancel each other out. Then, (2.6) holds. From now on, in the analysis of (2.5), we will only focus on the values of $\sum_{i=\lambda}^{\iota_1-1} p'_{e_i}$ and $\sum_{i=\iota_2}^{\mu} p'_{e_i}$.

We consider the first of these two sums. We prove that

$$\sum_{i=\lambda}^{\iota_1-1} p'_{e_i} = \begin{cases} 0, & \text{if } \lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P} \\ -\sum_{i=0}^{\lambda-1} p_{e_i}, & \text{if } \lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N} \end{cases}. \tag{2.7}$$

We start with analyzing the case in which $\lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P}$. If $\lambda \in \mathscr{P}$, then $\iota_1 = \lambda$ and the sum is trivially equal to 0. Then we assume that $\lambda \in \mathscr{N}\mathscr{P}$. Since ι_1 is the first index in $\mathscr{P}$ with $\iota_1 \geq \lambda$, Remark 1 implies that all indices $\{\lambda, \dots, \iota_1 - 1\}$ belong to $\mathscr{N}\mathscr{P}$. Therefore, by definition of p', we conclude that $\sum_{i=\lambda}^{\iota_1-1} p'_{e_i} = 0$. Next, let $\lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}$. Here, the indices in $\{\lambda, \dots, \iota_1 - 1\}$ can be in $\mathscr{P}\mathscr{N}$, $\mathscr{N}$, or $\mathscr{N}\mathscr{P}$. Moreover, there must be exactly one index $q \in \mathscr{N} \cap \{\lambda, \dots, \iota_1 - 1\}$. Then, we can see that $\sum_{i=\lambda}^{\iota_1-1} p'_{e_i} = \sum_{i=\lambda}^{q-1} p'_{e_i} + p'_q + \sum_{i=q+1}^{\iota_1-1} p'_{e_i} = \sum_{i=\lambda}^{q-1} p_{e_i} - \sum_{i=0}^{q-1} p_{e_i} = -\sum_{i=0}^{\lambda-1} p_{e_i}$. This concludes the proof of (2.7).

It remains to compute $\sum_{i=\iota_2}^{\mu} p'_{e_i}$. We show that

$$\sum_{i=\iota_2}^{\mu} p'_{e_i} = \begin{cases} 0, & \text{if } \mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P} \\ \sum_{i=0}^{\mu} p_{e_i}, & \text{if } \mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N} \end{cases}. \tag{2.8}$$

Assume that $\mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P}$. Since $\iota_2 \in \mathscr{P}$, there must be an index $q \in \mathscr{N} \cap \{\iota_2 + 1, \dots, \mu\}$. Then, $\sum_{i=\iota_2}^{\mu} p'_{e_i} = p'_{\iota_2} + \sum_{i=\iota_2+1}^{q-1} p'_{e_i} + p'_{e_q} + \sum_{i=q+1}^{\mu} p'_{e_i}$. By using the definition of p' we obtain the following: $\sum_{i=\iota_2}^{\mu} p'_{e_i} = \sum_{i=0}^{\iota_2} p_{e_i} + \sum_{i=\iota_2+1}^{q-1} p_{e_i} - \sum_{i=0}^{q-1} p_{e_i} = 0$. We look at the second case, and we assume that $\mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N}$. Then, by definition of ι_2 and the fact that $\mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N}$, it follows that all indices in $\{\iota_2+1, \dots, \mu\}$ belong to $\mathscr{P}\mathscr{N}$. By the definition of p', we can conclude that $\sum_{i=\iota_2}^{\mu} p'_{e_i} = p'_{\iota_2} + \sum_{i=\iota_2+1}^{\mu} p'_{e_i} = \sum_{i=0}^{\iota_2} p_{e_i} + \sum_{i=\iota_2+1}^{\mu} p_{e_i} = \sum_{i=0}^{\mu} p_{e_i}$. This concludes the proof of (2.8).

We conclude that (2.4) holds, by combining appropriately the different cases of (2.7) and (2.8) into (2.5). $\diamond$

We are now ready to prove (2.1). This proof is divided in two cases, depending on the value of x_u. The first case that we consider is when $x_u = \varphi(x) = 0$. Therefore, we assume that $x_u = \varphi(x) = 0$. As previously observed, we only need to show that (2.2) holds. Since $x_u = 0$, it follows that $A = \emptyset$, hence $\sum_{i \in A} p_{e_i} = 0$. Furthermore, $\varphi(x) = 0$ implies $\mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P}$. We consider the two cases $\mu = \lambda - 1$ and $\lambda \leq \mu$. Consider the case $\mu = \lambda - 1$. Then $\sum_{i \in B} p'_{e_i}$ is vacuous and equal to 0. Furthermore, if $\mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P}$ it means that $\lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P}$. Hence (2.2) holds. Next, we assume that $\lambda \leq \mu$, which implies that B is non-empty. If $\mathscr{P} \cap \{\lambda, \dots, \mu\} = \emptyset$, we get that $\sum_{i \in B} p'_{e_i} = 0$ if $\lambda \in \mathscr{N}\mathscr{P}$ by (2.3). Therefore (2.2) is true. Otherwise, if $\lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}$, then $\sum_{i \in B} p'_{e_i} = -\sum_{i=0}^{\lambda-1} p_{e_i}$ since $\mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P}$. Hence (2.2) holds also in this case. Therefore, we assume that $\mathscr{P} \cap \{\lambda, \dots, \mu\} \neq \emptyset$. We start from the case in which $\lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P}$. From (2.4), we see that $\sum_{i \in B} p'_{e_i} = 0$, as $\mu \in \mathscr{N} \cup \mathscr{N}\mathscr{P}$. This concludes the proof of (2.2) when $\lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P}$. So now consider $\lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}$. From (2.4) we obtain $\sum_{i \in B} p'_{e_i} = -\sum_{i=0}^{\lambda-1} p_{e_i}$ in this case. Hence, we can conclude that (2.2) holds also if $\lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}$.

The remaining case to consider, in order to prove that (2.1) holds for every $x \in \{0,1\}^V$, is when $x_u = \varphi(x) = 1$. Similarly to the previous case, we just need to show that (2.2) holds. Assume $x_u = \varphi(x) = 1$. From $x_u = 1$ we obtain $\sum_{i \in A} p_{e_i} = \sum_{i=0}^{\mu} p_{e_i}$. Because of $\varphi(x) = 1$, we know that $\mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N}$. Once again, we consider the cases $\mu = \lambda - 1$ and $\lambda \leq \mu$. Assume $\mu = \lambda - 1$. Then, we have that $B = \emptyset$ and $\sum_{i \in B} p'_{e_i}$ is equal 0. Moreover, we have $\lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}$, since $\mu \in \mathscr{P} \cup \mathscr{P}\mathscr{N}$. Then, it is easy to see that (2.2) is true. Next, we consider the case in which $\lambda \leq \mu$. We start from situation where $\lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P}$. By using Remark 1, we observe that $\mathscr{P} \cap \{\lambda, \dots, \mu\} \neq \emptyset$. Then, we obtain that $\sum_{i \in B} p'_{e_i} = \sum_{i=0}^{\mu} p_{e_i}$ from (2.4). Hence, (2.2) holds. Assume now $\lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}$. We first observe that it is possible that $\mathscr{P} \cap \{\lambda, \dots, \mu\} = \emptyset$. This can happen only if $\lambda, \mu \in \mathscr{P}\mathscr{N}$. In this case $\sum_{i \in B} p'_{e_i} = \sum_{i=\lambda}^{\mu} p_{e_i}$ by (2.3). It is easy to see that (2.2) holds in this case. So assume instead that $\mathscr{P} \cap \{\lambda, \dots, \mu\} \neq \emptyset$. Then, $\sum_{i \in B} p'_{e_i} = \sum_{i=\lambda}^{\mu} p_{e_i}$ by (2.4). Therefore, (2.2) is true. This concludes the proof of (2.1).

We are finally ready to show that the solution provided by the algorithm is optimal. Let $\tilde{x}$ be an optimal solution to (BPO) such that $\tilde{x}_u = \varphi(\tilde{x})$. We know that it exists by Claim 1. We denote by x^* the solution

returned by the algorithm, which is defined by

$$x^*_w := \begin{cases} x'_w, & \text{if } w \neq u \\ \varphi(x'), & \text{if } w = u \ . \end{cases}$$

It is easy to see that $x^*_u = \varphi(x')$. By the previous argument, it follows that (2.1) holds for both $\tilde{x}$ and x^*. Therefore, $\text{obj}(x^*) = \text{obj}'(x')$ and $\text{obj}(\tilde{x}) = \text{obj}'(\text{drop}_u(\tilde{x}))$, if $\lambda \in \mathscr{N}\mathscr{P} \cup \mathscr{P}$. Similarly, if $\lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}$, we obtain that $\text{obj}(x^*) = \text{obj}'(x') + \sum_{i=0}^{\lambda-1} p_{e_i}$ and $\text{obj}(\tilde{x}) = \text{obj}'(\text{drop}_u(\tilde{x})) + \sum_{i=0}^{\lambda-1} p_{e_i}$. We are now ready to prove that x^* is optimal to (BPO). The optimality of x' to (BPO)$'$ implies that $\text{obj}'(x') \geq \text{obj}'(\text{drop}_u(\tilde{x}))$. This inequality implies $\text{obj}(x^*) \geq \text{obj}(\tilde{x})$ in both cases. Note that if $\lambda \in \mathscr{P}\mathscr{N} \cup \mathscr{N}$ it suffices to add $\sum_{i=0}^{\lambda-1} p_{e_i}$ on both sides of the inequality to see this. Hence, we can conclude that x^* is an optimal solution to (BPO). ∎

We remark that our algorithm is correct even if the profits are allowed to be real numbers. However, for the purposes of the analysis of the algorithm, we chose to consider only the setting in which the profits are all integers.

Next, we show that `Acyclic(`G, p`)` runs in strongly polynomial time. We remark that in this paper we use standard complexity notions in Discrete Optimization, and we refer the reader to the book [47] for a thorough introduction. Our analysis is admittedly crude and provides a loose upper bound of the running time. It could be further improved by paying particular attention to the data structure and to the exact number of operations performed in each step. In our analysis, we choose to store the hypergraph $G = (V, E)$ by its node-edge incidence matrix.

The running time that we exhibit below is in terms of the time needed to find one nest point in G, which is denoted by τ. As mentioned in [44], nest points can be found in polynomial-time by brute force. Once we find one nest point, we also explicitly know the edges that contain it and their order under set inclusion.

PROPOSITION 2.2. *The algorithm* `Acyclic(`G, p`)` *is strongly polynomial, provided that $G = (V, E)$ is a β-acyclic hypergraph. In particular, the number of arithmetic operations performed is $O(|V|(\tau + |E| + |V| \log |E|))$.*

Proof. We first examine the number of arithmetic operations performed by the algorithm. In line 1, there are at most τ operations to find a nest point u and the ordered sequence of edges it belongs to, that is, $e_1 \subseteq e_2 \subseteq \cdots \subseteq e_k$. Line 2 requires $O(|E|)$ operations, between sums and comparisons, to compute the sets $\mathscr{P}$, $\mathscr{N}$, $\mathscr{N}\mathscr{P}$, $\mathscr{P}\mathscr{N}$. In line 3, there are other $O(|E|)$ operations to remove u from G in order to construct the hypergraph G', since it suffices to drop the u-th row from the incidence matrix. We observe that we do not remove the columns of edges that might have become empty. So, the incidence matrix could have some zero columns. Line 4 takes $O(|V|)$ operations. Next, there are $O(|E|)$ sums in the if condition in line 5. Line 6 can be performed in constant time. Then, line 7 requires $O(|V| + |E|)$ operations, and line 9 takes $O(|V|)$ operations. Next, finding λ in line 11 requires $O(|V| \log |E|)$ operations, by performing binary search on the ordered edges and checking the nodes they contain. Consider now the construction of p' in lines 12-13. Line 12 takes $O(|V| + |E|)$ operations. The profits p' for the edges $e_\lambda, \ldots, e_k$ can be constructed with a total number of $O(|E|)$ operations. It remains to consider the operations needed to construct x^* from x', see lines 15-16. Line 15 requires $O(|V|)$ operations. Now consider line 16. Using the definition of the quantity $\varphi(x)$, it can be seen that the definition of x^*_u requires $O(|V| \log |E|)$ operations. In fact it suffices to find $\mu(x')$.

Therefore, each iteration of algorithm performs at most $\tau + O(|E| + |V| \log |E|)$ arithmetic operations. Moreover, we observe that `Acyclic(`G, p`)` performs $|V|$ iterations, thanks to Theorem 2.1. We hence obtain that the total number of arithmetic operations performed by `Acyclic(`G, p`)` is $O(|V|(\tau + |E| + |V| \log |E|))$.

To prove that the algorithm `Acyclic(`G, p`)` is strongly polynomial, it remains to show that any integer produced in the course of the execution of the algorithm has size bounded by a polynomial in $|V| + |E| + \log U$, where U is the largest absolute value of the profits in the instance (see page 362 in [4]). The numbers that are produced by the execution of the algorithm are the profits of the smaller instances. The only arithmetic operations involving the profits are addition and subtraction of the original profits. In particular, this implies that the numbers produced are integers. Moreover, only a polynomial number of operations $p(|V|, |E|)$ occur in the algorithm since its arithmetic running time is polynomial in $|V|$ and $|E|$. Then, any integer obtained at the end of the algorithm must have absolute value less than or equal to $2^{p(|V|,|E|)}U$. Its bit size therefore is less than or equal to $p(|V|, |E|) + \log U$. ∎

We close this section by observing that the overarching structure of our algorithm, where nodes are removed one at a time, resembles that of the *basic algorithm* for pseudo-Boolean optimization, which was first defined in the sixties [30, 31]. Except for this similarity, the two algorithms are entirely different. For example, in the basic algorithm nodes can be removed in any order, but the running time can be exponential. On the other hand, in our algorithm the node to be removed must be a nest point in order for the algorithm to be correct. In particular, this allows us to define the updated profits and it is key in achieving a polynomial running time. In [14] the authors show that, if nodes are removed according to a "k-perfect elimination scheme", the basic algorithm runs in polynomial time for hypergraphs whose co-occurrence graph has fixed treewidth. However, analyzing the laminar hypergraph discussed after the statement of Theorem 1.1 in Section 1.1, it is simple to see that the basic algorithm does not run in polynomial time over β-acyclic hypergraphs, under *any* choice of the node to be removed.

3 Reduction scheme for general hypergraphs

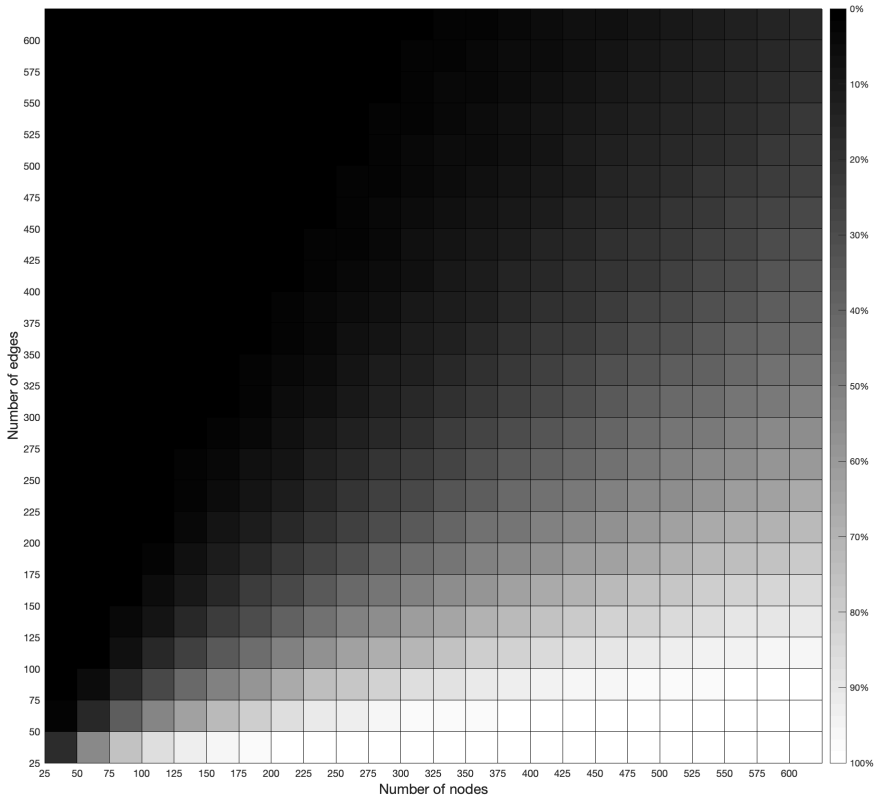

Figure 1: Percentage of removed nodes in hypergraphs as a function of $|V|$ and $|E|$.

We observe that, even if our algorithm is not able to solve instances over hypergraphs that contain β-cycles, it is still possible to use it as a reduction scheme. In particular, we can iteratively remove nest points, which leads to a decrease in the number of nodes, and possibly edges, of the hypergraph until there are no nest points left. If we are able to obtain an optimal solution to the smaller problem, we can then use the rules outlined in the algorithm to construct an optimal solution to the original problem.

In order to better assess if our reduction scheme could be useful in practice, we ran some computational experiments. We studied the reduction scheme on random instances, as it is commonly done in the literature

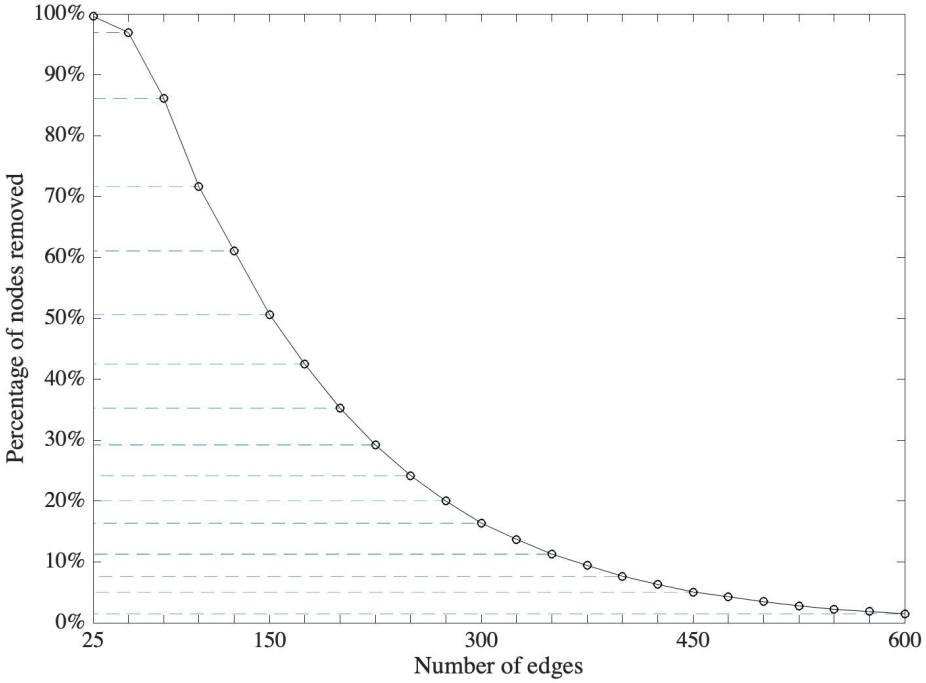

Figure 2: Percentage of removed nodes as a function of $|E|$ when $n = 300$.

[11, 15, 21]. For every instance, we computed the percentage of removed nodes. First, we explain the setting of our experiments. We chose the setting of [15], i.e., we decide the number of nodes $|V|$ and of edges $|E|$ of the hypergraph representing the instance, but we do not make any restriction on the rank of the hypergraph. We recall that the *rank* of a hypergraph is the maximum cardinality of any of its edges. For every edge, its cardinality c is chosen from $\{2, \ldots, |V|\}$ with probability equal to 2^{1-c}. As explained in [15], the purpose of this choice is to model the fact that a random hypergraph is expected to have more edges of low cardinality than high cardinality. Then, once c is fixed, the nodes of the edge are chosen uniformly at random in V with no repetitions. We also make sure that there are no parallel edges in the produced hypergraph. This will be useful in the interpretations of the results, as we explain later in the section. The parameters $|V|$ and $|E|$ have values in the set {25, 50, 75, ..., 600}. For every pair $(|V|, |E|)$ we made 250 repetitions and computed the percentage of removed nodes. Then, we took the average of these percentages. The results of our simulations are shown in Figure 1. The values on the x axis correspond to the number of nodes of the hypergraph, whereas the values on the y axis represent the number of edges. The lighter the cell, the more nodes are removed for instances with those values of n and m. A legend can be found to the right of the grid.

From the results, we noticed that the percentage of the removed nodes is related to the value of the ratio $|E|/|V|$, where $G = (V, E)$ is the hypergraph representing the instance. From Figure 1, it is apparent that the smaller is the ratio $|E|/|V|$, the more effective our algorithm is. In particular, we observe that if $|E|/|V| = 1$, then the average of nodes removed is 16.72%. However, when $|E|/|V| = 1/2$, this percentage is roughly 50%, and if $|E|/|V| = 1/4$ our algorithm removes on average 86% of the original nodes. Additional values can be extracted from Figure 2, which captures the trend of this percentage as a function of $|E|$. In this figure, the number of nodes $|V|$ is set to 300. We see that the reduction scheme is particularly useful whenever $|E|/|V| \leq 1$, i.e., when the number of edges is bounded by the number of nodes. Furthermore, we observe that a large subset of the hypergraphs with $|E|/|V| \leq 1$ have a highly non-trivial structure, since they have a huge connected component with high probability. In fact, the largest connected component of G is of order $|V|$ whenever the fraction $|E|/|V|$ is asymptotic to a constant c such that $c > 1/2$. This follows from [23] once we observe that each edge of a hypergraph connects at least as many nodes as an edge in a graph. We remark that the authors in [23] do not

allow parallel edges, and this is why we introduced this requirement for our instances.

For denser hypergraphs, i.e., hypergraphs with $|E|/|V| > 1$, our procedure does not work as well, and this can be explained by the fact that, for these hypergraphs, it is more unlikely that a node would be able to satisfy the definition of nest point. For non-random instances, it should be noted that the outcome of our reduction scheme depends heavily on the structure of the specific instance.

Lastly, we remark that the reduction scheme can be applied also to quadratic instances, where the corresponding hypergraph is simply a graph. We found that the behavior of the percentages of removed nodes is similar to the one in the hypergraph setting, even if the values of the percentages of removed nodes are generally higher than in the more general case.

References

[1] F. Barahona. A solvable case of quadratic $0-1$ programming. *Discrete Applied Mathematics*, 13(1):23–26, 1986.

[2] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.

[3] T. Beier, C. Pape, N. Rahaman, T. Prange, S. Berg, D. Bock, A. Cardona, G. Knott, S. Plaza, L. Scheffer, U. Koethe, A. Kreshuk, and F. Hamprecht. Multicut brings automated neurite segmentation closer to human performance. *Nature Methods*, 14(2):101–102, 2017.

[4] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[5] D. Bienstock and G. Muñoz. LP formulations for polynomial optimization problems. *SIAM Journal on Optimization*, 28(2):1121–1150, 2018.

[6] E. Boros, Y. Crama, and E. Rodríguez-Heck. Compact quadratizations for pseudo-boolean functions. *Journal of Combinatorial Optimization*, 39:687–707, 2020.

[7] E. Boros and A. Gruber. On quadratization of pseudo-boolean functions. *Preprint, arXiv:1404.6538.*, 2012.

[8] E. Boros and P.L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123:155–225, 2002.

[9] J. Brault-Baron. Hypergraph acyclicity revisited. *ACM Computing Surveys*, 49(3):54:1–54:26, 2016.

[10] C. Buchheim, Y. Crama, and E. Rodríguez-Heck. Berge-acyclic multilinear $0-1$ optimization problems. *European Journal of Operational Research*, 273(1):102–107, 2018.

[11] C. Buchheim and G. Rinaldi. Efficient reduction of polynomial zero-one optimization to the quadratic case. *SIAM Journal on Optimization*, 18(4):1398–1413, 2007.

[12] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. Springer, 2014.

[13] Y. Crama. Concave extensions for non-linear $0-1$ maximization problems. *Mathematical Programming*, 61(1):53–60, 1993.

[14] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-boolean programming revisited. *Discrete Applied Mathematics*, 29:171–185, 1990.

[15] Y. Crama and E. Rodríguez-Heck. A class of valid inequalities for multilinear $0-1$ optimization problems. *Discrete Optimization*, 25:28–47, 2017.

[16] A. Del Pia and S. Di Gregorio. Chvátal rank in binary polynomial optimization. *INFORMS Journal on Optimization*, 2021. doi:10.1287/ijoo.2019.0049.

[17] A. Del Pia and A. Khajavirad. A polyhedral study of binary polynomial programs. *Mathematics of Operations Research*, 42(2):389–410, 2017.

[18] A. Del Pia and A. Khajavirad. The multilinear polytope for acyclic hypergraphs. *SIAM Journal on Optimization*, 28(2):1049–1076, 2018.

[19] A. Del Pia and A. Khajavirad. On decomposability of multilinear sets. *Mathematical Programming, Series A*, 170(2):387–415, 2018.

[20] A. Del Pia and A. Khajavirad. The running intersection relaxation of the multilinear polytope. *Mathematics of Operations Research*, 46(3):1008–1037, 2021.

[21] A. Del Pia, A. Khajavirad, and N. Sahinidis. On the impact of running-intersection inequalities for globally solving polynomial optimization problems. *Mathematical Programming Computation*, 12:165–191, 2020.

[22] D. Duris. Some characterizations of γ and β-acyclicity of hypergraphs. *Information Processing Letters*, 112:617–620, 2012.

[23] P. Erdős and A. Rényi. On the evolution of random graphs. In *Publication Of The Mathematical Institute Of The Hungarian Academy Of Sciences*, pages 17–61, 1960.

[24] R. Fagin. Acyclic database schemes (of various degrees): A painless introduction. In *CAAP*, 1983.

[25] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the Association for Computing Machinery*, 30(3):514–550, 1983.

[26] R. Fortet. Applications de l'algébre de boole en recherche opèrationelle. *Revue Française d'Automatique, Informatique et Recherche Opérationnelle*, 4:17–26, 1960.
[27] D. Freedman and P. Drineas. Energy minimization via graph cuts: Settling what is possible. *CVPR, IEEE Computer Society*, pages 939–946, 2005.
[28] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, pages 237–267, 1976.
[29] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1154, 1995.
[30] P.L. Hammer, I. Rosenberg, and S. Rudeanu. On the determination of the minima of pseudo-boolean functions (in romanian). *Studii si Cercetari Matematice*, 14:359–364, 1963.
[31] P.L. Hammer and S. Rudeanu. *Boolean methods in operations research and related areas*. Springer, Berlin, New York, 1968.
[32] C. Hojny, M.E. Pfetsch, and M. Walter. Integrality of linearizations of polynomials over binary variables using additional monomials. *Manuscript*, 2019. URL: `http://www.optimization-online.org/DB_FILE/2019/11/7481.pdf`.
[33] H. Ishikawa. Higher-order gradient descent by fusion-move graph cut. *ICCV, IEEE*, pages 568–574, 2009.
[34] H. Ishikawa. Transformation of general binary mrf minimization to the first-order case. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(6):1234–1249, 2011.
[35] P. Jégoua and S.N. Ndiayeb. On the notion of cycles in hypergraphs. *Discrete Mathematics*, 309:6535–6543, 2009.
[36] M. Keuper. Higher-order minimum cost lifted multicuts for motion segmentation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4252–4260, 2017.
[37] S. Khot. On the power of unique 2-prover 1-round games. *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 767–775, 2002.
[38] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max-cut and other two-variable csps? *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 146–154, 2004.
[39] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28:58–81, 2014.
[40] J.-H. Lange and B. Andres. On the lifted multicut polytope for trees. In *Pattern Recognition*, volume 12544, pages 360–372. 42nd DAGM German Conference, DAGM GCPR 2020, 2021.
[41] M. Laurent. Sums of squares, moment matrices and optimization over polynomials. In *Emerging Applications of Algebraic Geometry*, volume 149 of *The IMA Volumes in Mathematics and its Applications*, pages 157–270. Springer, 2009.
[42] C. Michini. Tight cycle relaxations for the cut polytope. *To appear in SIAM Journal on Discrete Mathematics*, 2021.
[43] E. Mossel, R. O'Donnell, and K. Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 21–30, 2005.
[44] S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic cnf formulas. *Theoretical Computer Science*, 481:85–99, 2013.
[45] M. Padberg. The Boolean quadric polytope: Some characteristics, facets and relatives. *Mathematical Programming*, 45(1–3):139–172, 1989.
[46] I. Rosenberg. Reduction of bivalent maximization to the quadratic case. *Cahiers Centre Études Recherche Opér.*, 17:71–74, 1975.
[47] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.
[48] A. Schrijver. *Combinatorial Optimization. Polyhedra and Efficiency*. Springer-Verlag, Berlin, 2003.
[49] S. Tang, M. Andriluka, B. Andres, and B. Schiele. Multiple people tracking by lifted multicut and person re-identification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3701–3710, 2017.
[50] L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000.

Appendices

A Example of an execution of `Acyclic`(G, p)

In this appendix we show how `Acyclic`(G, p) works, by running it on an example. We choose a β-acyclic hypergraph G that is not kite-free (see [20]), since no other polynomial-time algorithm is known for instances of this type. We use the notation defined in Section 2. The input hypergraph G, together with the hypergraphs produced by the algorithm throughout its execution, is represented in the Figure 3. The profits of the edges are written next to the label of the corresponding edge. Labels are always outside their edges. For ease of exposition, we will keep the same names for the edges throughout the execution of the algorithm. For example, in $G^{(3)}$ we do not change the names of e_3 and e_4 to e_1 and e_2 respectively.

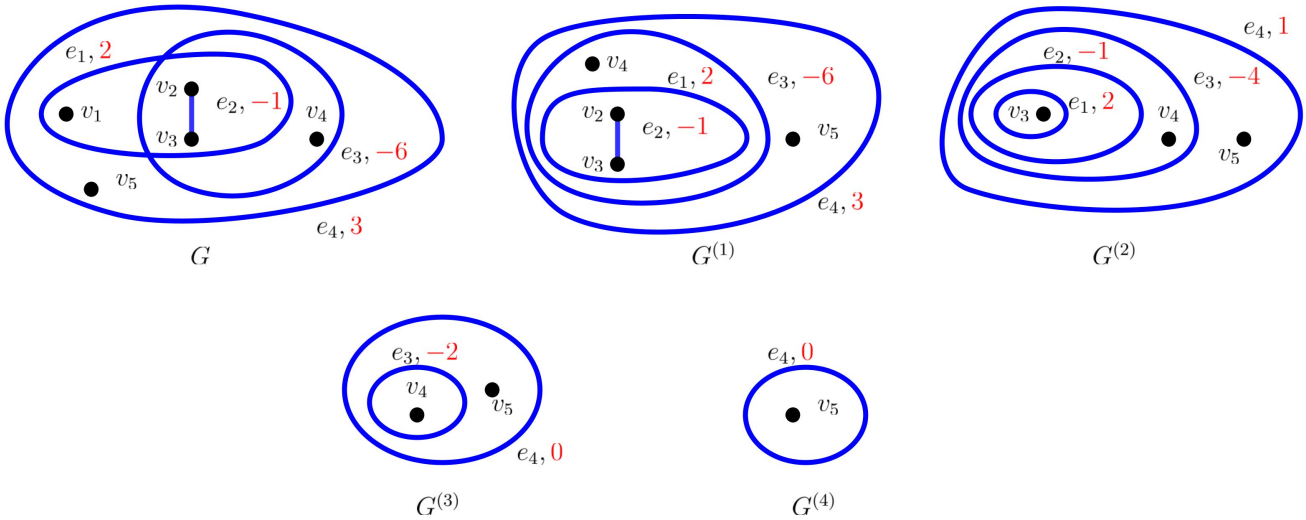

Figure 3: The input hypergraph, and the hypergraphs produced at each iteration by `Acyclic`(G, p).

We define the profit vector as follows: $p_{v_1} = 1$, $p_{v_2} = 3$, $p_{v_3} = 2$, $p_{v_4} = -1$, $p_{v_5} = 1$, $p_{e_1} = 2$, $p_{e_2} = -1$, $p_{e_3} = -6$, $p_{e_4} = 3$.

Iteration 1: observe that v_1 is a nest point of G, as it belongs only to e_1, e_4, and these edges are such that $e_1 \subseteq e_4$. Moreover, $\lambda = 1$. First of all we need to compute the sets $\mathscr{P}$, $\mathscr{N}$, $\mathscr{NP}$, $\mathscr{PN}$. In order to do so, observe that p_{v_1} is non-negative, as well as $p_{v_1} + p_{e_1}$ and $p_{v_1} + p_{e_1} + p_{e_4}$. This means that $\mathscr{PN} = \{0, 1, 4\}$. The node v_1 is removed from G, and the result is the hypergraph $G^{(1)}$ showed in Figure 3. We denote the profits corresponding to $G^{(1)}$ by $p^{(1)}$. Therefore, at this step $p^{(1)}_v := p_v$ for every $v \in \{v_2, v_3, v_4, v_5\}$ and $p^{(1)}_e := p_e$ for every $e \in \{e_1, e_2, e_3, e_4\}$.

Iteration 2: here `Acyclic`$(G^{(1)}, p^{(1)})$ removes v_2. We remark that v_2 is now a nest point for $G^{(1)}$, even if not for G. In fact, in $G^{(1)}$ we have $e_1 \subseteq e_2 \subseteq e_3 \subseteq e_4$. Furthermore, $\lambda = 1$. Here, $p^{(1)}_{v_2}$, $p^{(1)}_{v_2} + p^{(1)}_{e_1}$, $p^{(1)}_{v_2} + p^{(1)}_{e_1} + p^{(1)}_{e_2}$, are non-negative, while $p^{(1)}_{v_2} + p^{(1)}_{e_1} + p^{(1)}_{e_2} + p^{(1)}_{e_3}$ is negative, and $p^{(1)}_{v_2} + p^{(1)}_{e_1} + p^{(1)}_{e_2} + p^{(1)}_{e_3} + p^{(1)}_{e_4}$ is positive. Thus, $\mathscr{PN} = \{0, 1, 2\}$, $\mathscr{N} = \{3\}$, $\mathscr{P} = \{4\}$, $\mathscr{NP} = \emptyset$. Next, we construct $G^{(2)}$. We define the profits $p^{(2)}$ as follows: $p^{(2)}_{e_1} := p^{(1)}_{e_1} = 2$, and $p^{(2)}_{e_2} := p^{(1)}_{e_2} = -1$, however we define $p^{(2)}_{e_3} := -p^{(1)}_{v_2} - p^{(1)}_{e_1} - p^{(1)}_{e_2} = -4$ and $p^{(2)}_{e_4} := p^{(1)}_{v_2} + p^{(1)}_{e_1} + p^{(1)}_{e_2} + p^{(1)}_{e_3} + p^{(1)}_{e_4} = 1$. Moreover, $p^{(2)}_v := p^{(1)}_v$ for every node v of $G^{(2)}$, this means that $p^{(2)}_{v_3} := 2$, $p^{(2)}_{v_4} := -1$, $p^{(2)}_{v_5} := 1$.

Iteration 3: now it's the turn of v_3, which is a nest point of $G^{(2)}$. Here $\lambda = 3$. It is easy to check that all sums $p^{(2)}_{v_3}$, $p^{(2)}_{v_3} + p^{(2)}_{e_1}$, $p^{(2)}_{v_3} + p^{(2)}_{e_1} + p^{(2)}_{e_2}$ are positive, whereas $p^{(2)}_{v_3} + p^{(2)}_{e_1} + p^{(2)}_{e_2} + p^{(2)}_{e_3}$ is negative, and $p^{(2)}_{v_3} + p^{(2)}_{e_1} + p^{(2)}_{e_2} + p^{(2)}_{e_3} + p^{(2)}_{e_4}$ is equal to zero. Therefore $\mathscr{PN} = \{0, 1, 2\}$, $\mathscr{N} = \{3\}$, $\mathscr{NP} = \{4\}$, $\mathscr{P} = \emptyset$. The hypergraph $G^{(3)}$ is constructed by removing v_3 from $G^{(2)}$. Observe that, as we remove v_3, we are also removing e_1 and e_2, since $e_1 = e_2 = \{v_3\}$. Then, we set $p^{(3)}_{e_3} := -p^{(2)}_u - p^{(2)}_{e_1} - p^{(2)}_{e_2} = -3$, $p^{(3)}_{e_4} := 0$. Finally, $p^{(3)}_{v_4} := -1$, $p^{(3)}_{v_5} := 1$. We then iterate on the

smaller hypergraph.

Iteration 4: next, v_4 is a nest point of $G^{(3)}$. Observe that now we have that $\lambda = 4$. Here, $p^{(3)}_{v_4}$, $p^{(3)}_{v_4} + p^{(3)}_{e_3}$, $p^{(3)}_{v_4} + p^{(3)}_{e_3} + p^{(3)}_{e_4}$ are all negative. Hence, $\mathscr{N}\mathscr{P} = \{0, 3, 4\}$. We construct $G^{(4)}$. It is easy to see that $p^{(4)}_{e_4}$ is set equal to zero. Therefore, we define $p^{(4)}_{v_5} := p^{(3)}_{v_5} = 1$.

Iteration 5: we have arrived at the last step of the algorithm. Indeed, v_5 is the only node in $G^{(4)}$. Observe that it is useless to compute $\mathscr{P}$, $\mathscr{N}$, $\mathscr{N}\mathscr{P}$, $\mathscr{P}\mathscr{N}$, and $G^{(5)}$ in this last iteration. So, we skip it. We introduce $e_0 = \{v_5\}$ and let $p^{(4)}_{v_5} := p^{(4)}_{v_5}$, $p^{(4)}_{v_5} := 0$. We check that $p^{(4)}_{v_5} + p^{(4)}_{e_4} = p^{(4)}_{v_5} > 0$. So, we set $x_{v_5} := 1$.

At this point, we are ready to compute x_{v_1}, x_{v_2}, x_{v_3}, and x_{v_4}. We start from computing x_{v_4}. Since $x_{v_5} = 1$, it follows that $\mu = 4$. Recall that $4 \in \mathscr{N}\mathscr{P}$ in iteration number 4. Then, by the definition of φ, we set $x_{v_4} := 0$. Now we look at x_{v_3}. In this case $\mu = 2$. Since $2 \in \mathscr{P}\mathscr{N}$ in iteration number 3, we set $x_{v_3} := 1$. Next, consider x_{v_2}. Similarly to before, $\mu = 2$, since $x_{v_4} = 0$. Again, we have that $2 \in \mathscr{P}\mathscr{N}$ in iteration number 2. Hence, we define $x_{v_2} := 1$. It remains to compute x_{v_1}. We find that $\mu = 1$. Therefore we set $x_{v_1} := 1$, since in $1 \in \mathscr{P}\mathscr{N}$ in the first iteration. Then, an optimal solution of the problem is $x = (x_{v_1}, x_{v_2}, x_{v_3}, x_{v_4}, x_{v_5}) = (1, 1, 1, 0, 1)$.

B Hardness for α-acyclic hypergraphs

In this section we describe the intractability results for (BPO) over α-acyclic instances, thereby showing Theorem 1.3. In order to prove these results, we will use polynomial reductions from Max-Cut and Simple Max-Cut to (BPO). We recall that Max-Cut can be formulated as

$$\begin{aligned} \max \quad & \sum_{\{u,v\} \in E} w_{\{u,v\}}(x_u + x_v - 2x_u x_v) \\ \text{s.t.} \quad & x \in \{0,1\}^V, \end{aligned}$$

where $G = (V, E)$ is the graph representing the instance of Max-Cut and $w \in \mathbb{Z}^E_+$ [12].

Similarly to the β-acyclic case, we apply the idea of removing nodes and edges from a hypergraph. Here, we will use it to show that the instances obtained via the polynomial reductions from Max-Cut and Simple Max-Cut to (BPO) are represented by α-acyclic hypergraphs. Now, we are ready to describe a simple polynomial reduction of Max-Cut to (BPO).

PROPOSITION B.1. *Assume that an instance of Max-Cut is represented by a graph $G' = (V, E')$ and a weight vector $w \in \mathbb{Z}^{E'}_+$. Then, there exists a polynomial-time reduction from Max-Cut to (BPO), where the instance of (BPO) is represented by a hypergraph $G = (V, E)$ with profit vector $p \in \mathbb{Z}^{V \cup E}$ such that:*

(c1) G is α-acyclic;
(c2) all edges in E have cardinality either two or $|V|$;
(c3) all edges $e \in E$ such that $|e| = 2$ have profit $p_e = -2w_e$, all edges $e \in E$ such that $|e| = |V|$ have profit $p_e = 0$, and all nodes $v \in V$ have profit $p_v = \sum_{u \in V : \{u,v\} \in E} w_{\{u,v\}}$;
(c4) every vector in $\{0,1\}^V$ yields the same objective value in the two problems.

Proof. Let I be an instance of Max-Cut. We denote by $G' = (V, E')$ its associated graph, and by w the weight vector for the edges in E'. Let $\bar{e}$ be a new edge defined as $\bar{e} := V$. At this point, we construct an instance J of (BPO). The hypergraph representing the instance is $G = (V, E)$, where $E := E' \cup \{\bar{e}\}$. It is easy to see that it satisfies (c2) by construction. The profit vector of J is $p \in \mathbb{Z}^{V \cup E}$, which is defined as

$$p_i := \begin{cases} \displaystyle\sum_{u \in V : \{u,v\} \in E} w_{\{u,v\}}, & \text{if } i = v \in V \\ -2w_{\{u,v\}}, & \text{if } i = \{u,v\} \in E' \\ 0, & \text{if } i = \bar{e}\ . \end{cases}$$

Clearly the vector p satisfies condition (c3). Furthermore, it is immediate to see that solving I is equivalent to J. In particular, the set of feasible solutions is $\{0,1\}^V$ for both Max-Cut and (BPO). Moreover, the objective value obtained by any binary vector in J coincides with the objective value yielded by the same vector in I. This shows that (c4) holds. It remains to prove that also (c1) is satisfied. Hence, we show that G is α-acyclic. We observe

that we obtain the empty hypergraph $(\emptyset, \emptyset)$ from G by first removing all edges $e \in E'$, and then by removing all nodes. Therefore, by Theorem 1.2 we can conclude that the hypergraph G is α-acyclic. ∎

Next, we show the first hardness result, by reducing Simple Max-Cut to (BPO). Simple Max-Cut is the special case of Max-Cut, in which the weight vector w is restricted to be the vector of all ones. This problem has been shown to be strongly NP-hard in [28]. We observe that the polynomial reduction of Max-Cut to (BPO) works also for Simple Max-Cut, since this is a special case. The following theorem follows from using the polynomial reduction presented in Proposition B.1 applied to Simple Max-Cut.

THEOREM B.1. *Solving (BPO) is strongly NP-hard, even if $G = (V, E)$ is a hypergraph that satisfies conditions (c1), (c2), and*

(c3') all edges $e \in E$ such that $|e| = 2$ have profit $p_e = -2$, all edges $e \in E$ such that $|e| = |V|$ have profit $p_e = 0$, and all nodes $v \in V$ have profit $p_v = |\{e \in E : v \in e,\ |e| = 2\}|$.

Observe that condition (c3') coincides with condition (c3), when we adjust the latter to Simple Max-Cut.

Next, we move on to describing the inapproximability result. We start by defining the concept of r-approximation, for any maximization problem P, where $r \in [0, 1]$. Let ALG be an algorithm that returns a feasible solution to P yielding objective value $ALG(I)$, for every instance I of P. Now, let us fix I. We denote by $l(I)$ the minimum value that the objective function of I can achieve on all feasible points, and by $OPT(I)$ the optimum value of that instance. Then, we say that an algorithm ALG is a *r-approximation* for P if, for every instance I of P, we have that $\frac{ALG(I)-l(I)}{OPT(I)-l(I)} \geq r$. In particular, when P is Max-Cut, we have that $l(I) = 0$ for all instances I.

THEOREM B.2. *If $P \neq NP$, then it is NP-hard to obtain a r-approximation for (BPO), with $r > \frac{16}{17}$, even if the instance of (BPO) satisfies conditions (c1), (c2), (c3), for some vector $w \in \mathbb{Z}_+^E$.*

Proof. In [50] the authors show that if there is an r-approximation algorithm for Max-Cut, for $r > \frac{16}{17}$, then P = NP. Next, assume P $\neq$ NP. For the sake of contradiction let us assume that there exists a r-approximation, with $r > \frac{16}{17}$, for the instances of (BPO) that satisfy conditions (c1), (c2), (c3), for some vector $w \in \mathbb{Z}_+^E$. Let this algorithm be denoted by ALG'. We claim that ALG' is an r-approximation for Max-Cut. In fact, consider an instance I of Max-Cut. Then, we can construct an instance J of (BPO) using Proposition B.1, and apply ALG' to it, since J satisfies conditions (c1), (c2), (c3) by construction. Thanks to condition (c4), it follows that this procedure provides a r-approximation for Max-Cut, with $r > \frac{16}{17}$. However, this is a contradiction, given that we are assuming that P $\neq$ NP. ∎

We observe that the bound on r can be further strengthened if we assume the validity of the Unique Games conjecture, first formulated in [37]. In fact, [38] tells that it is NP-hard to approximate Max-Cut to within a factor greater than $\alpha_{GW} \approx 0.878$, granted that the conjectures P $\neq$ NP, Unique Games and the Majority is Stablest are true. The constant α_{GW} was originally defined in [29], where the authors provide an α_{GW}-approximation algorithm for Max-Cut. Lastly, we observe that the Majority is Stablest conjecture was proved to be correct in [43], and therefore this stronger inapproximability result now only relies on the P $\neq$ NP and the Unique Games conjectures.