

Efficient Minimization of Higher Order Submodular Functions using Monotonic Boolean Functions

Srikumar Ramalingam¹ Chris Russell^{2&3} Ľubor Ladický⁴ Philip H.S. Torr⁵

¹University of Utah, USA

²Alan Turing Institute, UK

³University of Edinburgh, UK

⁴ETH Zurich, Switzerland

⁵University of Oxford, Oxford, UK

Abstract. Submodular function minimization is a key problem in a wide variety of applications in machine learning, economics, game theory, computer vision, and many others. The general solver has a complexity of $O(n^3 \log^2 n \cdot E + n^4 \log^{O(1)} n)$ where E is the time required to evaluate the function and n is the number of variables [32]. On the other hand, many computer vision and machine learning problems are defined over special subclasses of submodular functions that can be written as the sum of many submodular cost functions defined over cliques containing few variables. In such functions, the pseudo-Boolean (or polynomial) representation [3] of these subclasses are of degree (or order, or clique size) k where $k \ll n$. In this work, we develop efficient algorithms for the minimization of this useful subclass of submodular functions. To do this, we define novel mapping that transform submodular functions of order k into quadratic ones. The underlying idea is to use auxiliary variables to model the higher order terms and the transformation is found using a carefully constructed linear program. In particular, we model the auxiliary variables as monotonic Boolean functions, allowing us to obtain a compact transformation using as few auxiliary variables as possible. The transformed quadratic function can be efficiently minimized using the standard max-flow algorithm with a time complexity of $O((n + m)^3)$ where m is the total number of auxiliary variables involved in transforming all the higher order terms to quadratic ones. Specifically, we show that our approach for fourth order function requires only 2 auxiliary variables in contrast to 30 or more variables used in existing approaches. In the general case, we give an upper bound for the number of auxiliary variables required to transform a function of order k using Dedekind number, which is substantially lower than the existing bound of 2^{2^k} .

Keywords: submodular functions, quadratic pseudo-Boolean functions, monotonic Boolean functions, Dedekind number, max-flow/mincut algorithm

1 Introduction

Many optimization problems in several domains such as operations research, computer vision, machine learning, and computational biology involve submodular function minimization. Submodular functions (See Definition 1) are discrete analogues of convex functions [33]. Examples of such functions include cut capacity functions, matroid rank functions and entropy functions. Submodular function minimization techniques may be broadly classified into two categories: algorithms for general submodular functions and efficient and customized algorithms for subclasses of submodular functions. This paper falls under the second category.

General solvers: The role of submodular functions in optimization was first discovered by Edmonds when he gave several important results on the related poly-matroids [10]. Grötschel, Lovász, and Schrijver first gave a polynomial-time algorithm for minimization of submodular function using ellipsoid method

[17]. Recently several combinatorial and strongly polynomial algorithms [13,22,24,45,36] have been developed based on the work of Cunningham [9]. The current best strongly polynomial algorithm for minimizing general submodular functions [32] has a run-time complexity of $O(n^3 \log^2 n \cdot E + n^4 \log^{O(1)} n)$, where E is the time taken to evaluate the function, and n is the number of variables. Weakly polynomial time algorithms with a smaller dependence on n also exist. For example, Lee et al. [32] shows a method with a run-time complexity of $O(n^2 \log n M \cdot E + n^3 \log^{O(1)} n M)$, where M is the maximum absolute value of the function values.

Specialized solvers: Higher order submodular functions are useful in modeling many computer vision and machine learning problems [26,31,21]. Such problems typically involve millions of pixels making the use of general solvers highly infeasible. Further, each pixel may take multiple discrete values and the conversion of such a problem to a Boolean one introduces further variables. On the other hand, the cost functions for many such optimization algorithms belong to a small subclass of submodular functions. The goal of this paper is to provide an efficient approach for minimizing these subclasses of submodular functions using a max-flow algorithm.

Notations: Let \mathbb{B} denote the Boolean set $\{0, 1\}$ and \mathbb{R} the set of reals. Let the vector $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{B}^n$, and $\mathbf{V} = \{1, 2, \dots, n\}$ be the set of indices of \mathbf{x} . We introduce a *set representation* to denote the labelings of \mathbf{x} . Let $S_4 = \{1, 2, 3, 4\}$ and let \mathcal{P} be the power set of S_4 . For example, a labeling $\{x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1\}$ is denoted by the set $\{1, 3, 4\}$. For a subset $A \subseteq V$, let us denote by $\mathbf{1}^A \in \mathbb{B}^n$ its characteristic vector, i.e.

$$\mathbf{1}_j^S = \begin{cases} 1 & \text{if } j \in A, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Definition 1. Submodular functions map $f : \mathbb{B}^n \rightarrow \mathbb{R}$ and satisfy the following condition:

$$f(X) + f(Y) \geq f(X \vee Y) + f(X \wedge Y), \quad (2)$$

where X and Y are elements of \mathbb{B}^n and the symbols \vee and \wedge denote union and intersection of sets respectively.

In this paper, we use a pseudo-Boolean polynomial representation for denoting submodular functions.

Definition 2. Pseudo-Boolean functions (PBF) take a Boolean vector as argument and return a real number, i.e. $f : \mathbb{B}^n \rightarrow \mathbb{R}$ [3]. These can be uniquely expressed as multi-linear polynomials, i.e. for all f there exists a unique set of real numbers $\{a_S : S \in \mathbb{B}^n\}$:

$$f(x_1, \dots, x_n) = \sum_{S \subseteq V} a_S \left(\prod_{j \in S} x_j \right), \quad a_S \in \mathbb{R}, \quad (3)$$

where a_\emptyset is said to be the constant term.

The term *order* refers to the maximum degree of the polynomial. A submodular function of second order involving Boolean variables can be easily represented using a graph such that the minimum cut, computed using a max-flow algorithm, also efficiently minimizes the function. However, max-flow algorithms can not exactly minimize non-submodular functions or some submodular ones of an order greater than 3 [49]. There is a long history of research in solving subclasses of submodular functions both exactly and efficiently using max-flow algorithms [1,28,18,48,38]. In this paper, we propose a linear programming formulation that is capable of answering this question: given any pseudo Boolean function, it can derive a quadratic submodular formulation of the same cost or a *closest* quadratic submodular function (i.e., say

under L_1 norm), if an exact derivation does not exist. The problem of using a linear program (LP) for expressing a given function using other functions (with AVs) was already established in [6]. Compared to the existing results, we also provide a smaller LP for submodular functions and also show that we need only fewer AVs compared to existing methods.

Definition 3. \mathcal{F}^k denotes a class of pseudo-Boolean functions of order k such that every function $f(\mathbf{x}) \in \mathcal{F}^k$ satisfies the submodularity property given in Definition 2.

It was first shown in [18] that any function in \mathcal{F}^2 can be minimized exactly using a max-flow algorithm. Billionnet and Minoux [1] showed that any function in \mathcal{F}^3 can be transformed into a function in \mathcal{F}^2 using additional variables. While transforming a given higher order function to a function in \mathcal{F}^2 , we use additional variables that we refer to as *auxiliary variables* (AV). In the course of this paper, you will see that these AVs are often more difficult to handle than variables in the original function and our algorithms are driven by the quest to understand the role of these auxiliary variables and to eliminate the unnecessary ones.

Kolmogorov [27] improved the complexity of Iwata's capacity scaling algorithm [23] for special functions which are represented as a sum of submodular terms. This is the first line of research that does not use auxiliary variables to handle higher order terms. The formulation of Kolmogorov also closely resembles the approach of Cooper [7], who used a linear program with an exponential number of constraints for solving the minimization of the submodular function. It was shown that we can have an algorithm that can be parallelized for minimizing decomposable submodular functions, which can be decomposed into sum of simple submodular functions. In [35], it was shown that the algorithm converges linearly, and they also provide upper and lower bounds on the rate of convergence.

Recently, Zivny et al. [49] made substantial progress in characterizing the class of functions that can be transformed to \mathcal{F}^2 . Their most notable result is to show that not all functions in \mathcal{F}^4 can be transformed to a function in \mathcal{F}^2 . This result stands in strong contrast to the third order case that was positively resolved more than two decades earlier [1]. Using Theorem 5.2 from [37] it is possible to decompose a given submodular function in \mathcal{F}^4 into 10 different groups \mathcal{G}_i , $i = \{1..10\}$, where each \mathcal{G}_i is shown in Table 1. Zivny et al. showed that one of these groups (\mathcal{G}_{10}) can not be expressed using any function in \mathcal{F}^2 employing any number of AVs. Most of these results were obtained by mapping the submodular function minimization to a valued constraint satisfaction problem.

1.1 Problem Statement and main contributions

Largest subclass of submodular functions: We are interested in transforming a given function in \mathcal{F}^k into a function in \mathcal{F}^2 using AVs. As such a transformation is not possible for all submodular functions of order four or more [49], our goal is to implicitly map the largest subclass \mathcal{F}_2^k that can be transformed into \mathcal{F}^2 . This distinction between the two classes \mathcal{F}_2^k and \mathcal{F}^k will be crucial in the remainder of the paper (see Figure 1).

Definition 4. The class \mathcal{F}_2^k is the largest subclass of \mathcal{F}^k such that every function $f(\mathbf{x}) \in \mathcal{F}_2^k$ has an equivalent quadratic function $h(\mathbf{x}, \mathbf{z}) \in \mathcal{F}^2$ using AVs $\mathbf{z} = z_1, z_2, \dots, z_m \in \mathbb{B}^m$ satisfying the following condition:

$$f(\mathbf{x}) = \min_{\mathbf{z} \in \mathbb{B}^m} h(\mathbf{x}, \mathbf{z}), \quad \forall \mathbf{x}. \quad (4)$$

In this paper, we are interested in developing an algorithm to transform every function in this class \mathcal{F}_2^k to a function in \mathcal{F}^2 .

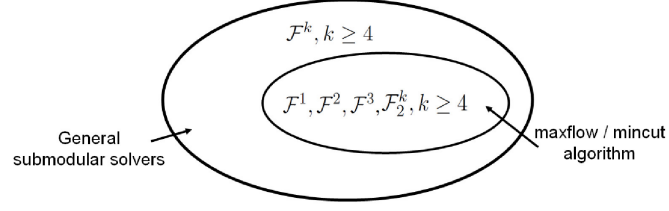


Fig. 1. All the function in the classes $\mathcal{F}^1, \mathcal{F}^2, \mathcal{F}^3$ and $\mathcal{F}_2^k, k \geq 2$ can be transformed to functions in \mathcal{F}^2 and minimized using the maxflow/mincut algorithm.

Efficient transformation of higher order functions: We propose a linear programming algorithm to transform higher order submodular functions to quadratic ones using monotonic Boolean functions (MBF [8]). This framework provides several advantages. First we show that the state of an AV in a minimum cost labeling is equivalent to an MBF defined over the original variables. This provides an upper bound on the number of AVs given by the Dedekind number [29], which is defined as the total number of MBFs over a set of n binary variables. In the case of fourth order functions, there are 168 such functions. Using the properties of MBFs and the nature of these AVs in our transformation, we prove that these 168 AVs can be replaced by two AVs.

Minimal use of AVs: One of our goals is to use a minimum number (m) of AVs in performing the transformation of (4). Although, given a fixed choice of \mathcal{F}_2^k , reducing the value of m does not change the complexity of the resulting min/cut algorithm asymptotically, it is crucial in several machine learning and computer vision problems. In general, most image based labeling problems involve millions of pixels and in typical problems, the number of fourth order priors is linearly proportional to the number of pixels. Such problems may be infeasible for large values of m . It was shown that the transformation of functions in \mathcal{F}_2^4 can be achieved using about 30 auxiliary variables [50]. On the other hand, we show that we can transform the same class of functions using only 2 additional nodes. Note that this reduction is applicable to every fourth order term in the function. A typical vision problem may involve functions having 10000 \mathcal{F}_2^4 terms for an image of size 100×100 . Under these parameters, our algorithm will use 20000 AVs, whereas the existing approach [50] would use as many as 300000 AVs. In several practical problems, this improvement will make a significant difference in the running time of the algorithm.

For a function in \mathcal{F}_2^k , the maximum number of AVs required is given by 2^{2^k} [6]. We show that one can transform the function using substantially fewer number of AVs given by Dedekind number. In section 3.1, we show that the Dedekind number is substantially lower than 2^{2^k} . In [6], an LP based approach was used to obtain the bound of 2^{2^k} . We also use an LP-based approach, however the use of monotonic Boolean functions enables us to improve this bound to Dedekind number. The idea of reducing the number of AVs in an LP formulation has been done in other contexts [46]. In [46], a combinatorial structure commonly referred to as *gadgets* were computed using linear programming. This enables the transformation of constraints from one optimization problem to another. In this work, we show that we can transform a function with several AVs to a function involving much fewer AVs using a linear programming approach.

1.2 Limitations of Current Approaches and Open Problems

Decomposition of submodular functions: Many existing algorithms for transforming higher order functions target the minimization of a single k -variable k^{th} order function. However, the transformation framework is incomplete without showing that a given n -variable submodular function of k^{th} order can be decomposed into several individual k -variable k^{th} order sub-functions. Billionnet proved that it is possible

to decompose a function in \mathcal{F}^3 involving several variables into 3-variable functions in \mathcal{F}^3 [1]. To the best of our knowledge, the decomposition of fourth or higher order functions is still an open problem and it will remain a hard problem due to the following reasoning. In [16], it was proven that testing a membership of a function f with n variables in \mathcal{F}^4 is NP-complete. It is easy to test the submodularity of a fourth order function with 4 variables. Thus if a function f with n variables is decomposed into several 4-variable fourth order functions and if each of these individual 4-variable functions are submodular, then the function f is submodular. This seems to be most possible case when we know that a function is submodular. Thus it is very unlikely to know that a function is submodular and not know its decomposition. Given this, it is likely that specialized solvers based on max-flow algorithms may never solve the general class of submodular functions. However, this decomposition problem is not a critical issue in machine learning and vision problems. This is because the higher order priors from natural statistics already occur in different sub-functions of k nodes - in other words, the decomposition is known a priori. This paper only focuses on the transformation of a single k -variable function in \mathcal{F}^k . As mentioned above, the solution to this problem is still sufficient to solve large functions with hundreds of nodes and higher order priors in applications.

Non-Boolean problems: The results in this paper are applicable only to set or pseudo-Boolean functions. Many real world problems involve variables that can take multiple discrete values. Ishikawa showed that it is possible to transform a multi-label second order function to a Boolean second order function using Boolean variables to encode multi-label variables [20]. To denote a single multi-label variable with l labels, l Boolean variables were used. Ishikawa's method considered functions with convex priors, a class of functions that is slightly more restricted than general submodular functions. Schlesinger and Flach later showed that it is possible to transform general submodular multi-label functions of second order to Boolean second order functions [44]. This approach used $l - 1$ Boolean variables to encode an l -label multi-label variable. Ramalingam et al. [39] generalized this work for transforming multi-label higher order functions to Boolean second-order functions. In [39], the transformation does not preserve submodularity for fourth or higher order functions [39]. Zivny et al. [49] proved that it is not possible to have a submodularity preserving transformation for fourth or higher order functions.

Excess AVs: The complexity of an efficient max-flow algorithm is $O((n+m)^3)$ where n is the number of variables in the original higher order function and m is the number of AVs. Typically in imaging problems, the number of higher order terms is of $O(n)$ and the order k is less than 10. Thus the minimization of the function corresponding to an entire image with $O(n)$ higher order terms will still have a complexity of $O((n+n)^3)$. However when m becomes at least quadratic in n , for example, if a higher-order term is defined over every triplet of variables in V , the complexity of the max-flow algorithm will exceed that of a general solver being $O((n+n^3)^3)$. Thus in applications involving a very large number of higher order terms, a general solver may be more appropriate.

2 Preliminaries

Definition 5. The (discrete) derivative of a function $f(x_1, \dots, x_n)$ with respect to x_i is given by:

$$\frac{\delta f}{\delta x_i}(x_1, \dots, x_n) = f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) - f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n). \quad (5)$$

Definition 6. The second discrete derivative of a function $\Delta_{i,j}(\mathbf{x})$ is given by

$$\begin{aligned} \Delta_{i,j}(\mathbf{x}) &= \frac{\delta}{\delta x_j} \frac{\delta f}{\delta x_i}(x_1, \dots, x_n) \\ &= \left(f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_{j-1}, 1, x_{j+1}, \dots, x_n) - f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_{j-1}, 1, x_{j+1}, \dots, x_n) \right) \\ &\quad - \left(f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_{j-1}, 0, x_{j+1}, \dots, x_n) - f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_{j-1}, 0, x_{j+1}, \dots, x_n) \right). \end{aligned} \quad (6)$$

Note that it follows from the definition of submodular functions (2), that their second derivative is always non-positive for all \mathbf{x} .

3 Transforming functions in \mathcal{F}_2^n to \mathcal{F}^2

Consider the following submodular function $f(\mathbf{x}) \in \mathcal{F}_2^n$ represented as a multi-linear polynomial:

$$f(\mathbf{x}) = \sum_{S \in \mathbb{B}^n} a_S \left(\prod_{j \in S} x_j \right), \quad a_S \in \mathbb{R}. \quad (7)$$

Let us consider a function $h(\mathbf{x}, \mathbf{z}) \in \mathcal{F}^2$ where \mathbf{z} is a set of AVs used to model functions in \mathcal{F}_2^n . Any general function in \mathcal{F}^2 can be represented as a multi-linear polynomial (consisting of linear and bi-linear terms involving all variables):

$$h(\mathbf{x}, \mathbf{z}) = \sum_i a_i x_i - \sum_{i,j:i>j} a_{i,j} x_i x_j + \sum_l a_l z_l - \sum_{l,m:l>m} a_{l,m} z_l z_m - \sum_{i,l} a_{i,l} x_i z_l. \quad (8)$$

The negative signs in front of the bi-linear terms ($x_i x_j, z_l z_m$) emphasize that their coefficients ($-a_{ij}, -a_{il}, -a_{lm}$) must be non-positive if the function is submodular. We are seeking a function h such that:

$$f(\mathbf{x}) = \min_{\mathbf{z} \in \mathbb{B}^n} h(\mathbf{x}, \mathbf{z}), \forall \mathbf{x}. \quad (9)$$

Here the function $f(\mathbf{x})$ is known. We are interested in computing the coefficients (\mathbf{a}), and in determining the number of auxiliary variables required to express a function as a pairwise submodular function. The problem is challenging due to the inherent instability and dependencies within the problem – different choices of parameters cause auxiliary variables to take different states. To explore the space of possible solutions fully, we must characterize what states an AV takes.

3.1 Auxiliary Variables as Monotonic Boolean Functions

Definition 7. A monotonic (increasing) Boolean function (MBF) $m : \mathbb{B}^n \rightarrow \mathbb{B}$ takes a Boolean vector as argument and returns a Boolean, s.t if $y_i \leq x_i, \forall i \implies m(\mathbf{y}) \leq m(\mathbf{x})$.

Lemma 1. Let $z_s(\mathbf{x})$ be a function that takes an argument \mathbf{x} and returns a Boolean as shown below:

$$z_s(\mathbf{x}) = \arg \min_{z_s} \left(\min_{\mathbf{z}'} h(\mathbf{x}, \mathbf{z}', z_s) \right), \quad (10)$$

where $h(\mathbf{x}, \mathbf{z}', z_s)$ is a submodular function defined in Equation (8) and satisfying Equation (9). The function $z_s(\mathbf{x})$ that maps a Boolean vector \mathbf{x} to the Boolean state of z_s is an MBF (See Definition 7), where \mathbf{z}' is the set of all auxiliary variables except z_s .

Proof. We consider a current labeling \mathbf{x} with an induced labeling of $z_s = z_s(\mathbf{x})$. We first note

$$h'(\mathbf{x}, z_s) = \min_{\mathbf{z}'} h(\mathbf{x}, \mathbf{z}', z_s) \quad (11)$$

is a submodular function i.e. it satisfies (2). We now consider *increasing* the value of \mathbf{x} , that is given a current labeling \mathbf{x} we consider a new labeling $\mathbf{x}^{(i)}$ such that

$$x_j^{(i)} = \begin{cases} 1 & \text{if } j = i \\ x_j & \text{otherwise.} \end{cases} \quad (12)$$

We wish to prove

$$z_s(\mathbf{x}^{(i)}) \geq z_s(\mathbf{x}) \quad \forall \mathbf{x}, i. \quad (13)$$

Note that if $z_s(\mathbf{x}) = 0$ or $x_i = 1$ this result is trivial. This leaves the case: $z_s(\mathbf{x}) = 1$ and $x_i = 0$. It follows from (6) that:

$$\begin{aligned} h'(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, 0) - h'(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, 0) &\geq \\ h'(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, 1) - h'(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, 1). \end{aligned} \quad (14)$$

Using Equation (10), we derive the following from our hypothesis $z_s(\mathbf{x}) = 1$ and $x_i = 0$:

$$h'(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, 0) \geq h'(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, 1). \quad (15)$$

Hence by replacing $h'(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, 0)$ with $h'(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, 1)$ in Equation (14), we have

$$\begin{aligned} h'(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, 0) - h'(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, 0) &\geq \\ h'(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, 1) - h'(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, 0). \end{aligned} \quad (16)$$

This implies the following:

$$h'(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, 0) \geq h'(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, 1). \quad (17)$$

Therefore $z_s(\mathbf{x}^{(i)}) = 1$ as per the Equation (10). Repeated application of the statement gives $y_i \leq x_i, \forall i \implies z_s(\mathbf{y}) \leq z_s(\mathbf{x})$ as required \square

Definition 8. The Dedekind number $M(n)$ is the number of MBFs of n variables. Finding a closed-form expression for $M(n)$ is known as the Dedekind problem [25,29].

The Dedekind number of known values are shown below: $M(1) = 3$, this corresponds to the set of functions:

$$M_1(x_1) \in \{\mathbf{0}, \mathbf{1}, x_1\}, \quad (18)$$

where $\mathbf{0}$ and $\mathbf{1}$ are the functions that take any input and return 0 or 1 respectively. $M(2) = 6$ corresponding to the set of functions:

$$M_2(x_1, x_2) = \{\mathbf{0}, \mathbf{1}, x_1, x_2, x_1 \vee x_2, x_1 \wedge x_2\}. \quad (19)$$

Similarly, $M(3) = 20$, $M(4) = 168$, $M(5) = 7581$, $M(6) \approx 7.8 \times 10^6$, $M(7) \approx 2.4 \times 10^{12}$, and $M(8) \approx 5.6 \times 10^{23}$.

Theorem 1. On transforming the largest graph-representable subclass of k^{th} order function to pairwise Boolean function, the upper bound on the maximal number of required ANs is given by the Dedekind number $D(k)$.

Proof. The proof is straightforward. Consider a general multinomial, of similar form to Equation (7) with more than $D(k)$ AVs. It follows from Lemma 1 that at least 2 of the AVs must correspond to the same MBF, and always take the same values. Hence, all references to one of these AV in the pseudo-Boolean representation can be replaced with references to the other, without changing the associated costs. Repeated application of this process will leave us with a solution with at most $D(k)$ AVs. \square

Although this upper bound is large for even small values of k , it is much tighter than the existing upper bound of $S(k) = 2^{2^k}$ [6] (also see Proposition 24 in [51]).

Lemma 2. *Let $D(k)$ denote the Dedekind number for all positive values of k . Given $S(k) = 2^{2^k}$ and for even values of k , we have:*

$$S(k) \geq 2^{\sum_{i \in \{0,1,\dots,k\} \setminus \{\frac{k}{2}-1, \frac{k}{2}\}} \binom{k}{i}} D(k). \quad (20)$$

When k is odd, we have:

$$S(k) \geq 2^{\sum_{i \in \{0,1,\dots,k\} \setminus \{\frac{k-1}{2}, \frac{k+1}{2}\}} \binom{k}{i}} D(k). \quad (21)$$

Proof. For even small values of $k = \{3, \dots, 8\}$ the upper bound using Dedekind's number is much tighter compared to $S(k)$: ($M(3) = 20, S(3) = 256$), ($M(4) = 168, S(4) = 65536$), ($M(5) = 7581, S(5) \approx 4.29 \times 10^9$), ($M(6) \approx 7.8 \times 10^6, S(6) \approx 1.85 \times 10^{19}$), ($M(7) \approx 2.4 \times 10^{12}, S(7) \approx 3.4 \times 10^{38}$), and ($M(8) \approx 5.6 \times 10^{23}, S(8) \approx 1.156 \times 10^{77}$). For $k > 8$, $D(k)$ remains unknown, and the development of a closed form solution remains an active area of research.

Several upper bounds have been derived for $D(k)$ and we use the following bound by Hansel [19,25] to prove our result.

$$D(k) \leq 3^{\binom{k}{\lfloor \frac{k}{2} \rfloor}}, \quad (22)$$

$$D(k) \leq 2^{\log_2(3) \binom{k}{\lfloor \frac{k}{2} \rfloor}}. \quad (23)$$

The proof is given for two different cases depending on whether k is even or odd. First let us consider the case when k is even.

$$\binom{k}{\lfloor \frac{k}{2} \rfloor} = \binom{k}{\frac{k}{2}}. \quad (24)$$

We can obtain the following:

$$\binom{k}{\frac{k}{2}} = \frac{k \times (k-1) \dots (k - \frac{k}{2})}{1 \times 2 \dots (\frac{k}{2})} = \frac{k \times (k-1) \dots (k - (\frac{k}{2} - 1))}{1 \times 2 \dots (\frac{k}{2} - 1)} = \binom{k}{(\frac{k}{2} - 1)}. \quad (25)$$

Using binomial theorem we know that

$$\sum_{i \in \{0,1,\dots,k\}} \binom{k}{i} = 2^k. \quad (26)$$

Using Equations (25) and (26) we have the following Equation:

$$2^k = \sum_{i \in \{0,1,\dots,\frac{k}{2}-2, \frac{k}{2}+1,\dots,k\}} \binom{k}{i} + 2 \binom{k}{\frac{k}{2}}. \quad (27)$$

Since $\log_2(3) < 2$, it is easy to observe the following:

$$2^k \geq \sum_{i \in \{0,1,\dots,\frac{k}{2}-2, \frac{k}{2}+1,\dots,k\}} \binom{k}{i} + \log_2(3) \binom{k}{\frac{k}{2}}. \quad (28)$$

Taking both sides to the exponent of 2, we have the following:

$$2^{2^k} \geq 2^{\sum_{i \in \{0,1,\dots,\frac{k}{2}-2,\frac{k}{2}+1,\dots,k\}} \binom{k}{i} + \log_2(3) \binom{k}{\frac{k}{2}}} \quad (29)$$

$$2^{2^k} \geq 2^{\sum_{i \in \{0,1,\dots,\frac{k}{2}-2,\frac{k}{2}+1,\dots,k\}} \binom{k}{i}} 2^{\log_2(3) \binom{k}{\frac{k}{2}}} \quad (30)$$

$$S(k) \geq 2^{\sum_{i \in \{0,1,\dots,k\} \setminus \{\frac{k}{2}-1,\frac{k}{2}\}} \binom{k}{i}} D(k). \quad (31)$$

This implies that $S(k)$ is significantly larger than $D(k)$. Let us consider the case when k is odd.

$$\binom{k}{\lfloor \frac{k}{2} \rfloor} = \binom{k}{\frac{k-1}{2}}. \quad (32)$$

It is well known that:

$$\binom{k}{\frac{k+1}{2}} = \frac{k \times (k-1) \dots (k - \frac{k-1}{2}) (k - \frac{k+1}{2})}{1 \times 2 \dots (\frac{k-1}{2}) (\frac{k+1}{2})} = \binom{k}{\frac{k-1}{2}} \frac{k-1}{k+1} = \binom{k}{\frac{k-1}{2}} \left(1 - \frac{2}{k+1}\right). \quad (33)$$

Using Equations (33) and (26) we have the following Equation:

$$2^k = \sum_{i \in \{0,1,\dots,\frac{k-3}{2},\frac{k+3}{2},\dots,k\}} \binom{k}{i} + \left(1 + 1 - \frac{2}{k+1}\right) \binom{k}{\frac{k-1}{2}}. \quad (34)$$

Since $\log_2(3) < (1 + 1 - \frac{2}{k+1})$ for $k > 8$, it is easy to observe the following:

$$2^k \geq \sum_{i \in \{0,1,\dots,\frac{k-3}{2},\frac{k+3}{2},\dots,k\}} \binom{k}{i} + \log_2(3) \binom{k}{\frac{k-1}{2}}. \quad (35)$$

By lifting both sides to the power of 2, we have the following relation:

$$2^{2^k} \geq 2^{\sum_{i \in \{0,1,\dots,\frac{k-3}{2},\frac{k+3}{2},\dots,k\}} \binom{k}{i} + \log_2(3) \binom{k}{\frac{k-1}{2}}} \quad (36)$$

$$2^{2^k} \geq 2^{\sum_{i \in \{0,1,\dots,\frac{k-3}{2},\frac{k+3}{2},\dots,k\}} \binom{k}{i}} 2^{\log_2(3) \binom{k}{\frac{k-1}{2}}} \quad (37)$$

$$S(k) \geq 2^{\sum_{i \in \{0,1,\dots,k\} \setminus \{\frac{k-1}{2},\frac{k+1}{2}\}} \binom{k}{i}} D(k). \quad (38)$$

□

We observe that $S(k)$ is significantly larger than $D(k)$ when k is odd.

In [52], the problem of improving this upper bound was mentioned as an open problem. In some sense, both these upper bounds are not practically feasible for even small values of k . This number is prohibitive because we are looking for an exact transformation that preserves submodularity. By using auxiliary variables, we can also transform a given higher order function to a non-submodular one using much fewer variables [21,12,15]. In section 5, we will further tighten the bound for fourth order functions.

Note that this representation of AVs as MBF is over-complete, for example if the MBF of a auxiliary variable z_i is the constant function $z_i(\mathbf{x}) = 1$ we can replace $\min_{\mathbf{z}, z_i} h(\mathbf{x}, \mathbf{z}, z_i)$ with the simpler (i.e. one containing less auxiliary variables) function $\min_{\mathbf{z}} h(\mathbf{x}, \mathbf{z}, 1)$.

Given any function f in \mathcal{F}_2^k , the equivalent pairwise form $f' \in \mathcal{F}^2$ can be found by solving a linear program. The construction of the linear program is given in the following section.

4 The Linear Program

A sketch of the formulation can be given as follows: In general, the presence of AVs of indeterminate state, given a labeling \mathbf{x} makes the minimizing an LP non-convex and challenging to solve directly. Instead of optimizing this problem containing AVs of unspecified state, we create an auxiliary variable associated with every MBF. Hence given any labeling \mathbf{x} the state of every auxiliary variable is fixed a priori, making the problem convex. We show how the constraints that a particular AV must conform to a given MBF can be formulated as linear constraints, and that consequently the problem of finding the closest member of \mathcal{F}^2 to any pseudo Boolean function is a linear program.

This program will make use of the max-flow linear program formulation to guarantee that the minimum cost labeling of the AVs corresponds to their MBFs. To do this we must first rewrite the cost of Equation (8) in a slightly different form. We write:

$$\begin{aligned} f(\mathbf{x}, \mathbf{z}) = & c_\emptyset + \sum_i c_{i,s} (1 - x_i) + \sum_i c_{t,i} x_i + \sum_{i,j:i>j} c_{i,j} x_i (1 - x_j) \\ & + \sum_l c_{l,s} (1 - z_l) + \sum_l c_{t,l} z_l + \sum_{l,m:l>m} c_{l,m} z_l (1 - z_m) + \sum_{i,l} c_{i,l} x_i (1 - z_l). \end{aligned} \quad (39)$$

where c_\emptyset is a constant that may be either positive or negative and all other c are non-negative values referred to as the *capacity* of an edge. By [11], this form is equivalent to that of (8), in that any function that can be written in form (8), can also be written as (39) and visa versa.

4.1 The Max-flow Linear Program

Under the assumption that \mathbf{x} is fixed, we are interested in finding a minimum of the Equation:

$$\begin{aligned} f_{\mathbf{x}}(\mathbf{z}) = & c_\emptyset + \sum_i c_{i,s} (1 - x_i) + \sum_i c_{t,i} x_i + \sum_{i,j:i>j} c_{i,j} x_i (1 - x_j) \\ & + \sum_l c_{l,s} (1 - z_l) + \sum_l c_{t,l} z_l + \sum_{l,m:l>m} c_{l,m} z_l (1 - z_m) + \sum_{i,l} c_{i,l} x_i (1 - z_l) \\ = & d_{\mathbf{x},\emptyset} + \sum_l d_{\mathbf{x},l,s} (1 - z_l) + \sum_l d_{\mathbf{x},t,l} z_l + \sum_{l,m:l>m} d_{\mathbf{x},l,m} z_l (1 - z_m) \end{aligned} \quad (40)$$

where

$$d_{\mathbf{x},\emptyset} = c_\emptyset + \sum_{i:x_i=0} c_{i,s} + \sum_{i:x_i=1} c_{t,i} + \sum_{i,j:i>j \wedge x_i=1 \wedge x_j=0} c_{i,j} \quad (41)$$

$$d_{\mathbf{x},l,s} = c_{l,s} + \sum_{i:x_i=1} c_{i,l}, \quad d_{\mathbf{x},t,l} = c_{t,l} \text{ and } d_{\mathbf{x},l,m} = c_{l,m}. \quad (42)$$

Then the minimum cost of Equation (39) may be found by solving its dual max-flow program. Writing $\nabla_{\mathbf{x},s}$ for flow from the sink, and $\nabla_{\mathbf{x},t}$ for flow to the sink, we seek

$$\max \nabla_{\mathbf{x},s} + d_{\mathbf{x},\emptyset}, \quad (43)$$

subject to the constraints that

$$\begin{aligned} & f_{\mathbf{x},ij} - d_{\mathbf{x},ij} \leq 0, & \forall (i,j) \in E \\ & \sum_{j:(j,i) \in E} f_{\mathbf{x},ji} - \sum_{j:(i,j) \in E} f_{\mathbf{x},ij} \leq 0, & \forall i \neq s, t \\ & \nabla_{\mathbf{x},s} + \sum_{j:(j,s) \in E} f_{\mathbf{x},js} - \sum_{j:(s,j) \in E} f_{\mathbf{x},sj} \leq 0 \\ & \nabla_{\mathbf{x},t} + \sum_{j:(j,t) \in E} f_{\mathbf{x},jt} - \sum_{j:(t,j) \in E} f_{\mathbf{x},tj} \leq 0 \\ & f_{\mathbf{x},ij} \geq 0, & (i,j) \in E \end{aligned} \quad (44)$$

where E is the set of all ordered pairs $(l, m) : \forall l > m, (s, l) : \forall l$ and $(l, t) : \forall t$, and $f_{\mathbf{x}, i, j}$ corresponds to the flow through the edge (i, j) .

We will not use this exact LP formulation, but instead rely on the fact that $f_{\mathbf{x}}(\mathbf{z})$ is a *minimal cost labeling* if and only if there exists a flow satisfying constraints (44) such that

$$f_{\mathbf{x}}(\mathbf{z}) - \nabla_{\mathbf{x}, s} - d_{\mathbf{x}, \emptyset} \leq 0. \quad (45)$$

4.2 Choice of MBF as a set of linear constraints

We are seeking minima of a quadratic pseudo Boolean function of the form (39), where \mathbf{x} is the variables we are interested in minimizing and \mathbf{z} the auxiliary variables. As previously mentioned, formulations that allow the state of the auxiliary variable to vary tend to result in non-convex optimization problems. To avoid such difficulties, we specify as the location of minima of \mathbf{z} as a set of hard constraints. We want that:

$$\min_{\mathbf{z}} f_{\mathbf{x}}(\mathbf{z}) = f_{\mathbf{x}}([m_1(\mathbf{x}), m_2(\mathbf{x}), \dots, m_{D(k)}(\mathbf{x})]) \quad \forall \mathbf{x}. \quad (46)$$

where $f_{\mathbf{x}}$ is defined as in (40), and $m_1, \dots, m_{D(k)}$ are the set of all possible MBFs defined over \mathbf{x} . By setting all of the capacities $d_{i, j}$ to 0, it can be seen that a solution satisfying (46) must exist. It follows from the reduction described in Lemma 1, and that all functions that can be expressed in a pairwise form can also be expressed in a form that satisfies these restrictions.

We enforce condition (46) by the set of linear constraints (44) and (45) for all possible choices of \mathbf{x} . Formally we enforce the condition

$$f_{\mathbf{x}}([m_1(\mathbf{x}), \dots, m_{D(k)}(\mathbf{x})]) - \nabla_{\mathbf{x}, s} - d_{\mathbf{x}, \emptyset} \leq 0. \quad (47)$$

Substituting in (40) we have 2^k sets of conditions, namely,

$$\sum_l d_{\mathbf{x}, l, s} (1 - m_l(\mathbf{x})) + \sum_l d_{\mathbf{x}, t, l} (1 - m_l(\mathbf{x})) + \sum_{l, m: l > m} d_{\mathbf{x}, l, m} m_l(\mathbf{x}) (1 - m_m(\mathbf{x})) - \nabla_{\mathbf{x}, s} \leq 0, \quad (48)$$

subject to the set of constraints (44) for all \mathbf{x} . Note that we make use of the max-flow formulation, and not the more obvious min-cut formulation, as this remains a linear program even if we allow the capacity of edges \mathbf{d}^1 to vary.

Submodularity Constraints We further require that the quadratic function is submodular or equivalently, the capacity of all edges $c_{i, j}$ be non-negative. This can be enforced by the set of linear constraints that

$$c_{i, j} \geq 0, \quad \forall i, j. \quad (49)$$

4.3 Finding the nearest submodular Quadratic Function

We now assume that we have been given an arbitrary function $g(\mathbf{x})$ to minimize, that may or may not lie in \mathcal{F}^k . We are interested in finding the closest possible function in \mathcal{F}^2 to it. To find the closest function to it (under the L_1 norm), we minimize:

¹ In itself \mathbf{d} is just a notational convenience, being a sum of coefficients in \mathbf{c} .

$$\min_{\mathbf{c}} \sum_{\mathbf{x} \in \mathbb{B}^k} \left| g(\mathbf{x}) - \min_{\mathbf{z}} f(\mathbf{x}, \mathbf{z}) \right| = \quad (50)$$

$$\min_{\mathbf{c}} \sum_{\mathbf{x} \in \mathbb{B}^k} \left| g(\mathbf{x}) - f(\mathbf{x}, \mathbf{m}(\mathbf{x})) \right| = \quad (51)$$

$$\begin{aligned} \min_{\mathbf{c}} \sum_{\mathbf{x} \in \mathbb{B}^k} \left| g(\mathbf{x}) - \left(c_{\emptyset} + \sum_i c_{i,s} (1 - x_i) + \sum_i c_{t,i} x_i + \sum_{i,j:i>j} c_{i,j} x_i (1 - x_j) \right. \right. \\ \left. \left. + \sum_l c_{l,s} (1 - m_l(\mathbf{x})) + \sum_l c_{t,l} (1 - m_l(\mathbf{x})) + \sum_{l,m:l>m} c_{l,m} m_l(\mathbf{x}) (1 - m_m(\mathbf{x})) \right. \right. \\ \left. \left. + \sum_{i,l} c_{i,l} x_i (1 - m_l(\mathbf{x})) \right) \right| \end{aligned} \quad (52)$$

where $\mathbf{m}(\mathbf{x}) = [m_1(\mathbf{x}), \dots, m_{D(k)}(\mathbf{x})]$ is the vector of all MBFs over \mathbf{x} , and subject to the family of constraints set out in the previous subsection. Note that expressions of the form $\sum_i |g_i|$ can be written as $\sum_i h_i$ subject to the linear constraints $h_i > g_i$ and $h_i > -g_i$ and this is a linear program. \square

4.4 Discussion

Several results follow from the linear program described in the previous section. In particular, if we consider a function g of the same form as Equation (3) such that

$$\min_{\mathbf{c}} \sum_{\mathbf{x} \in \mathbb{B}^k} \left| g(\mathbf{x}) - \min_{\mathbf{z}} f(\mathbf{x}, \mathbf{z}) \right| = 0. \quad (53)$$

exactly defines a linear polytope for any choice of $|\mathbf{x}| = k$, and this result holds for any choice of basis functions.

Of equal note, the convex-concave procedure [47] is a generic move-making algorithm that finds local optima by successively minimizing a sequence of convex (i.e. tractable) upper-bound functions that are tight at the current location (\mathbf{x}'). [34] showed how this could be similarly done for quadratic Boolean functions, by decomposing them into submodular and supermodular components. The work [30] showed that any function could be decomposed into a quadratic submodular function, and an additional overestimated term. Nevertheless, this decomposition was not optimal, and they did not suggest how to find an optimal overestimation. The optimal overestimation which lies in \mathcal{F}^2 for a cost function defined over a clique \mathbf{g} may be found by solving the above LP subject to the additional requirements:

$$g(\mathbf{x}) \leq \min_{\mathbf{z}} f(\mathbf{x}, \mathbf{z}), \quad \forall \mathbf{x} \neq \mathbf{x}' \quad (54)$$

$$g(\mathbf{x}') \geq \min_{\mathbf{z}} f(\mathbf{x}', \mathbf{z}). \quad (55)$$

Efficiency concerns: As we consider larger cliques, it becomes less computationally feasible to use the techniques discussed in this section, at least without pruning the number of auxiliary variables considered. As previously mentioned, constant AVs and AVs that corresponds to that of a single variable in x i.e. $z_l = x_i$ can be safely discarded without loss of generality. In the following section, we show that a function in \mathcal{F}_2^4 can be represented by only two AVs, rather than 168 as suggested by the Dedekind number. However, in the general case a minimal form representation eludes us. As a matter of pragmatism, it may be useful to attempt to solve the LP of the previous section without making use of any AV, and to successively introduce new variables, until a minimum cost solution is found.

5 Tighter Bounds: Transforming functions in \mathcal{F}_2^4 to \mathcal{F}^2

Consider the following submodular function $f(x_1, x_2, x_3, x_4) \in \mathcal{F}^4$ represented as a multi-linear polynomial:

$$f(x_1, x_2, x_3, x_4) = a_0 + \sum_i a_i x_i + \sum_{i>j} a_{ij} x_i x_j + \sum_{i>j>k} a_{ijk} x_i x_j x_k + a_{1234} x_1 x_2 x_3 x_4, \quad \Delta_{ij}(\mathbf{x}) \leq 0 \quad (56)$$

where $i, j, k \in S_4$ and $\Delta_{ij}(\mathbf{x})$ is the discrete second derivative of $f(\mathbf{x})$ with respect to x_i and x_j .

Consider a function $h(x_1, x_2, x_3, x_4, z_s) \in \mathcal{F}^2$ where z_s is an AV used to model functions in \mathcal{F}_2^4 . In general, we need several AVs to transform a function in \mathcal{F}_2^4 to a function in \mathcal{F}_2 . Any general function in \mathcal{F}^2 using one AV, can be represented as a multi-linear polynomial (consisting of linear and bilinear terms involving all five variables):

$$h(x_1, x_2, x_3, x_4, z_s) = b_0 + \sum_i b_i x_i - \sum_{i>j} b_{ij} x_i x_j + (g_s - \sum_{i=1}^4 g_{s,i} x_i) z_s, \quad b_{ij} \geq 0, g_{s,i} \geq 0, i, j \in S_4. \quad (57)$$

The negative signs in front of the bilinear terms ($x_i x_j, z_s x_i$) emphasize that their coefficients ($-b_{ij}, -g_{s,i}$) must be non-positive to ensure submodularity. We have the following condition from Equation (4), given in page 3:

$$f(x_1, x_2, x_3, x_4) = \min_{z_s \in \mathbb{B}} h(x_1, x_2, x_3, x_4, z_s), \quad \forall \mathbf{x}. \quad (58)$$

Here the coefficients ($a_i, a_{ij}, a_{ijk}, a_{ijkl}$) in the function $f(\mathbf{x})$ are known. We wish to compute the coefficients ($b_i, b_{ij}, g_s, g_{s,n}$) where $i, j \in V, i \neq j, n \in S_4$. If we were given $(g_s, g_{s,i})$ then from Equations (57) and (58) we would have:

$$z_s = \begin{cases} 1 & \text{if } g_s - \sum_{i=1}^4 g_{s,i} x_i < 0, \\ 0 & \text{otherwise.} \end{cases} \quad (59)$$

Our main result is to prove that any function $h \in \mathcal{F}_2^4$ can be transformed to a function $h'(x_1, x_2, x_3, x_4, z_t, z_r) \in \mathcal{F}^2$ involving only two auxiliary variables z_t and z_r as stated in Theorem 2.

Let \mathcal{A} be the family of sets corresponding to labelings of \mathbf{x} such that: $z_s = 0 = \arg \min_{z_s} h(\mathbf{x}, z_s)$. In the same way let \mathcal{B} be the family of sets corresponding to labelings of \mathbf{x} such that: $z_s = 1 = \arg \min_{z_s} h(\mathbf{x}, z_s)$. These sets \mathcal{A} and \mathcal{B} partition \mathbf{x} , as defined below:

Definition 9. A partition divides \mathcal{P} into sets \mathcal{A} and \mathcal{B} such that $\mathcal{A} = \{\mathcal{S}(\mathbf{x}) : 0 = \arg \min_{z \in \mathbb{B}} h(\mathbf{x}, z), \mathbf{x} \in \mathbb{B}^4\}$ and $\mathcal{B} = \mathcal{P} \setminus \mathcal{A}$. Note that $\emptyset \in \mathcal{A}$. Here $\mathcal{S}(\mathbf{x})$ denotes the set corresponding to \mathbf{x} .

In the rest of the paper, we say that the AV z_s is associated with $[\mathcal{A}, \mathcal{B}]$ or denote it by $z_s : [\mathcal{A}, \mathcal{B}]$. We illustrate the concept of a *partition* in Figure 2.

A few partitions that play a key role in our transformation are referred to as forward, backward, and intermediate partitions.

Definition 10. The forward reference partition $[\mathcal{A}_f, \mathcal{B}_f]$ takes the form:

$$B \in \mathcal{B}_f \iff |B| \geq 3, \mathcal{A}_f = \mathcal{P} \setminus \mathcal{B}_f \quad (60)$$

The backward reference partition $[\mathcal{A}_b, \mathcal{B}_b]$ is shown below:

$$B \in \mathcal{B}_b \iff |B| \geq 2, \mathcal{A}_b = \mathcal{P} \setminus \mathcal{B}_b \quad (61)$$

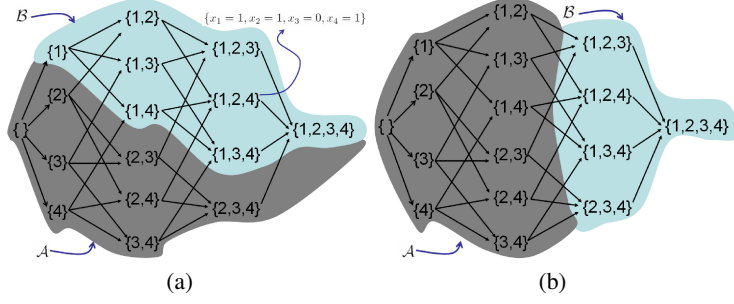


Fig. 2. We show some examples of partitions using Hasse diagrams. Here, we use set representation for denoting the labelings of (x_1, x_2, x_3, x_4) . For example the set $\{1, 2, 4\}$ is equivalent to the labeling $\{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1\}$. In (a), $\mathcal{A} = \{\{\}, \{2\}, \{3\}, \{4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{2, 3, 4\}\}$ and $\mathcal{B} = \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, S_4\}$. (a) and (b) are examples of partitions. Any arbitrary AV must be associated with one of these 168 partitions as given by the Dedekind number $D(k)$.

Figure 3(a) and (b) show the forward and backward partitions respectively.

We consider a set of 18 partitions as **intermediate partitions** $[\mathcal{A}_i, \mathcal{B}_i]$ as shown in Figure 4. There are 6 intermediate partitions where there are five sets in \mathcal{B}_i that have cardinality 2 (one such partition is shown in Figure 4(a)). There are 12 intermediate partitions where there are four sets in \mathcal{B}_i that have cardinality 2 (one such partition is shown in Figure 4(b)). One may expect more intermediate partitions by considering all possible different sets in \mathcal{B}_i having cardinality 2. However, we will see later that such partitions are not necessary for transforming a function in \mathcal{F}_2^4 to a function in \mathcal{F}^2 .

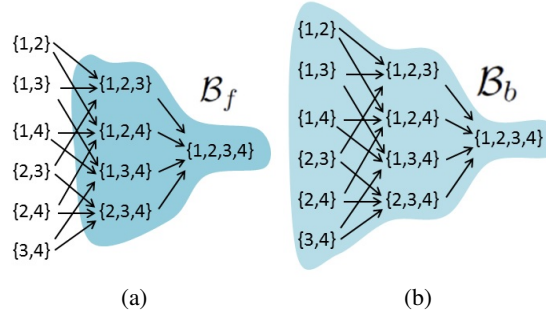


Fig. 3. The two reference partitions, referred to as forward and backward, are shown.

The basic idea in our work is to replace several AVs using the minimum number of AVs without changing the values of the function at their respective minima.

Definition 11. We say that a function $h(\mathbf{x}, \mathbf{z})$ can be transformed to another function $h'(\mathbf{x}, \mathbf{z}')$ where $\mathbf{z} \neq \mathbf{z}'$ if the following condition is satisfied:

$$\min_{\mathbf{z}} h(\mathbf{x}, \mathbf{z}) = \min_{\mathbf{z}'} h'(\mathbf{x}, \mathbf{z}'), \quad \forall \mathbf{x} \quad (62)$$

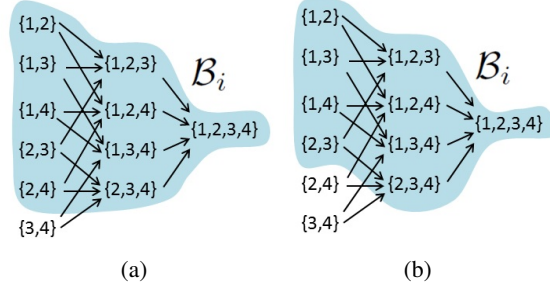


Fig. 4. We have a total of 18 intermediate partitions. In (a), we show one of the 6 intermediate partitions where five sets in \mathcal{B}_i have cardinality 2. We denote this as $\mathcal{I}(34)$, where the index refers to the only set that does not have cardinality 2. In (b), we show one of the 12 intermediate partitions where four sets in \mathcal{B}_i have cardinality 2. We denote this as $\mathcal{I}(24, 34)$, where the indices refer to the sets that do not have cardinality 2.

where \mathbf{z} and \mathbf{z}' are vectors of auxiliary variables with different partitions. The cardinality of \mathbf{z} need not be equal to the cardinality of \mathbf{z}' .

Through a sequence of transformations of the above form, we start with a general function $h(\mathbf{x}, \mathbf{z})$ and finally compute a function $h'(\mathbf{x}, z_s, z_t)$ with only two AVs in reference partitions.

Lemma 3. Let $z_a : [\mathcal{A}_s, \mathcal{B}_s]$ and $z_b : [\mathcal{A}_s, \mathcal{B}_s]$ be two AVs that have the same partition then $h(\mathbf{x}, z_a, z_b) \in \mathcal{F}^2$ can be transformed to some function $h'(\mathbf{x}, z_s) \in \mathcal{F}^2$ involving only one AV z_s .

Proof. According to the Equation 62, we can transform a function $h(\mathbf{x}, z_a, z_b)$ to $h'(\mathbf{x}, z_s)$ if it satisfies the following condition:

$$\min_{z_a, z_b} h(\mathbf{x}, z_a, z_b) = \min_{z_s} h'(\mathbf{x}, z_s), \quad \forall \mathbf{x} \quad (63)$$

Since the AVs z_a and z_b take the same partition $[\mathcal{A}_s, \mathcal{B}_s]$ their Boolean values are equal for different configurations of \mathbf{x} . Thus we can replace all instances of z_a and z_b with z_s . \square

Theorem 2. Any function $f(\mathbf{x})$ in \mathcal{F}_2^4 can be transformed to some function $h(\mathbf{x}, z_f, z_s)$ in \mathcal{F}^2 where z_f correspond to the forward partition and z_s can either be the backward partition or one of the 18 intermediate partitions.

Proof. Using Theorem 5.2 from [37] we can decompose a given submodular function in \mathcal{F}^4 into functions in 10 different groups $\mathcal{G}_i, i = \{1..10\}$ where each \mathcal{G}_i is shown in Table 1. As shown in [50] the functions in \mathcal{G}_{10} does not belong to \mathcal{F}_2^4 . It was also shown that any submodular function that has any functions from group \mathcal{G}_{10} does not belong to \mathcal{F}_2^4 according to Theorem 16(3) in [50]. Thus all the functions in \mathcal{F}_2^k should be composed of functions in the groups $\mathcal{G}_i, i \in \{1, \dots, 9\}$.

The number of distinct terms in each group \mathcal{G}_i is given in Table 1. Overall, there are 31 distinct functions in the groups $\mathcal{G}_i, i \in \{1, \dots, 9\}$. The terms in the first group \mathcal{G}_1 has only second degree terms. Hence, the functions in this group does not require any AVs. The terms in the next 7 groups $\mathcal{G}_i, i \in \{2, \dots, 8\}$ can each be represented by a single AV, which can be either z_f or z_b . Here z_f and z_b denote AVs in the forward and backward partitions respectively. The 6 terms in \mathcal{G}_9 can be represented using two AVs z_f and z_i , where z_f and z_i correspond to forward and intermediate reference partitions (denoted by $\mathcal{I}(k, l)$ in Figure 4(a)) respectively. It is important to note that the functions in \mathcal{G}_9 involve interaction between z_f and z_i , i.e., there exists a bilinear term $z_f z_i$ in \mathcal{G}_9 .

We prove the result by considering two cases.

Group \mathcal{G}_i	$ \mathcal{G}_i $	$f(\mathbf{x})$	$\min_{z_1, z_2} h(\mathbf{x}, z_1, z_2)$ where $h \in \mathcal{F}^2$
$\mathcal{G}_1(i, j)$	6	$-x_i x_j$	$-x_i x_j$
$\mathcal{G}_2(i, j, k)$	4	$-x_i x_j x_k$	$\min_{z_f} (2 - x_i - x_j - x_k) z_f$
\mathcal{G}_3	1	$-x_i x_j x_k x_l$	$\min_{z_f} (3 - x_i - x_j - x_k - x_l) z_f$
\mathcal{G}_4	1	$-x_i x_j x_k x_l + x_i x_j x_k + x_i x_j x_l + x_i x_k x_l + x_j x_k x_l - x_i x_j - x_i x_k - x_i x_l - x_j x_k - x_j x_l - x_k x_l$	$\min_{z_b} (1 - x_i - x_j - x_k - x_l) z_b$
$\mathcal{G}_5(i, j, k)$	4	$x_i x_j x_k x_l - x_i x_j x_k - x_i x_l - x_j x_l - x_k x_l$	$\min_{z_b} (2 - x_i - x_j - x_k - 2x_l) z_b$
$\mathcal{G}_6(i, j, k)$	4	$x_i x_j x_k - x_i x_j - x_i x_k - x_j x_k$	$\min_{z_b} (1 - x_i - x_j - x_k) z_b$
$\mathcal{G}_7(i)$	4	$x_i x_j x_k x_l - x_i x_j x_k - x_i x_j x_l - x_i x_k x_l$	$\min_{z_f} (3 - 2x_i - x_j - x_k - x_l) z_f$
\mathcal{G}_8	1	$2x_i x_j x_k x_l - x_i x_j x_k - x_i x_j x_l - x_i x_k x_l - x_j x_k x_l$	$\min_{z_f} (2 - x_i - x_j - x_k - x_l) z_f$
$\mathcal{G}_9(i, j)$	6	$x_i x_j x_k x_l - x_i x_j - x_i x_k x_l - x_j x_k x_l$	$\min_{z_f, z_i} ((2 - x_k - x_l) z_f + (1 - x_i - x_j) z_i - z_f z_i)$
\mathcal{G}_{10}	6	$-x_i x_j x_k x_l + x_i x_k x_l + x_j x_k x_l - x_i x_k - x_i x_l - x_j x_k - x_j x_l - x_k x_l$	$f(\mathbf{x}) \notin \mathcal{F}_2^4$ as shown in [50]

Table 1. The above table is adapted from Figure 2 of [51] where $\{i, j, k, l\} = S_4$. Each group \mathcal{G}_i has several terms depending on the values of $\{i, j, k, l\}$. The number of distinct terms in each group is given by $|\mathcal{G}_i|$. Since the groups \mathcal{G}_4 and \mathcal{G}_8 involve all four variables and are symmetric, they contain one function each. z_f and z_b correspond to AVs for forward and backward partitions. z_i corresponds to one of the intermediate partitions denoted by $\mathcal{I}(kl)$ in Figure 4(a). For each group \mathcal{G}_i , we also use an index (\cdot) in the first column to identify a specific function from others in its group.

Absence of \mathcal{G}_9 functions: In the first case, we consider functions that can be expressed as a sum of functions in the first 8 groups $\mathcal{G}_i, i = \{1..8\}$. In other words, we study the scenario where we express the function without using any function from \mathcal{G}_9 . Let us denote such a function as $f_0(\mathbf{x})$ that can be expressed as a sum of 25 functions from the 8 groups $\mathcal{G}_i, i = \{1..8\}$ as shown below:

$$f_0(\mathbf{x}) = \alpha_1 \mathcal{G}_1(i, j) + \dots + \alpha_{25} \mathcal{G}_8 \quad (64)$$

The only AVs involved in all the functions are z_f and z_b . Using Lemma 3 we can obtain a function that uses only two variables z_f and z_b as shown below:

$$f_0(\mathbf{x}) = g(\mathbf{x}) + \min_{z_f, z_b} (g_f(\mathbf{x}) z_f + g_b(\mathbf{x}) z_b), \quad (65)$$

where $g(\mathbf{x})$, $g_f(\mathbf{x})$ and $g_b(\mathbf{x})$ are functions involving \mathbf{x} . This implies that any function that can be expressed without any function from \mathcal{G}_9 can be expressed using the only forward and backward partitions.

Presence of \mathcal{G}_9 functions: Let us consider an arbitrary function $f(\mathbf{x})$ in \mathcal{F}_2^4 that is expressed as a sum of functions from these 31 groups including functions from \mathcal{G}_9 :

$$f(\mathbf{x}) = \alpha_1 \mathcal{G}_1(i, j) + \alpha_2 \mathcal{G}_2(i, j) + \dots + \alpha_{31} \mathcal{G}_9(k, l) \quad (66)$$

In Table 2, we show that the sum of two functions can always be represented using only two auxiliary variables. In Tables 3, 4 and 5 we show the sum of functions with 3, 4 and 5 terms respectively. Different combinations of functions lead to functions that can always be expressed with only 2 auxiliary variables.

Without loss of generality, we have avoided the repetition for all possible indices by treating them using the set $\{i, j, k, l\} = \{1, 2, 3, 4\}$. We have already proved the case where \mathcal{G}_9 is absent. Thus, the tables only show summations that involve at least one function from \mathcal{G}_9 .

In some cases, we do not show the sums of functions with real coefficients $(\alpha, \beta, \gamma, \delta, \eta)$ to demonstrate the special scenarios where the combination of two functions involving intermediate partition z_i can be transformed to a function that involves only z_f and z_b . In such cases, we can do the following sequentially:

$$\alpha_1 f_1(\mathbf{x}) + \alpha_2 f_2(\mathbf{x}) = \alpha_1 (f_1(\mathbf{x}) + f_2(\mathbf{x})) + (\alpha_2 - \alpha_1) f_2(\mathbf{x}), \quad \alpha_1 \leq \alpha_2 \quad (67)$$

Let $\beta = \alpha_2 - \alpha_1$ and let $f_3(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$ as per the Table 2. Now we can further use Table 2 on $\alpha_1 f_3(\mathbf{x}) + \beta f_2(\mathbf{x})$ to generate other functions.

As we see in Table 1, the function $\mathcal{G}_9(i, j)$ uses two auxiliary variables z_f and $z_i \in \mathcal{I}(k, l)$. As we observe in Table 2, on adding the function $\mathcal{G}_9(i, j)$ with other functions we have the following scenarios:

1. The coefficient of z_i is unaltered and we only change the coefficients of z_f . This happens in 6 of the additions in Table 2 as given by $(\mathcal{G}_9(i, j), \mathcal{G}_1(i, j))$, $(\mathcal{G}_9(i, j), \mathcal{G}_2(j, k, l))$, $(\mathcal{G}_9(i, j), \mathcal{G}_7(k))$ and $(\mathcal{G}_9(i, j), \mathcal{G}_8)$.
2. We obtain a function that can be expressed with only one z_f . This happens in 4 of the additions in Table 2 as given by $(\mathcal{G}_9(i, j), \mathcal{G}_2(i, j, k))$, $(\mathcal{G}_9(i, j), \mathcal{G}_3)$, $(\mathcal{G}_9(i, j), \mathcal{G}_7(i))$, and $(\mathcal{G}_9(i, j), \mathcal{G}_9(k, l))$.
3. We obtain a function that can be expressed with only one z_b . This happens in 2 of the additions shown in Table 2 as given by $(\mathcal{G}_9(i, j), \mathcal{G}_4)$ and $(\mathcal{G}_9(i, j), \mathcal{G}_6(j, k, l))$.
4. We obtain a function that can be expressed with z_f and z_b . This happens in one of the additions shown in Table 2 as given by $(\mathcal{G}_9(i, j), \mathcal{G}_5(i, j, k))$.
5. We obtain a function with z_f and z_i , whose coefficients are changed. This happens in 3 of the additions shown in Table 2 as given by $(\mathcal{G}_9(i, j), \mathcal{G}_5(j, k, l))$, $(\mathcal{G}_9(i, j), \mathcal{G}_6(i, j, k))$, and $(\mathcal{G}_9(i, j), \mathcal{G}_9(i, k))$.

There are 6 functions in group \mathcal{G}_9 . However, as shown in Table 2, additions involving $\mathcal{G}_9(i, j)$ and $\mathcal{G}_9(k, l)$ produce functions involving only one auxiliary variable z_f . In other words, sum of functions involving $\mathcal{G}_9(i, k)$ can sometimes be represented using functions from the first 8 groups (\mathcal{G}_1 to \mathcal{G}_8). Out of the 6 functions in \mathcal{G}_9 only two of them are necessary at a time. Without loss of generality, we rewrite the $f(\mathbf{x})$ using a maximum of 2 functions in group \mathcal{G}_9 as shown below:

$$f(\mathbf{x}) = \alpha_1 \mathcal{G}_1(i, j) + \dots + \alpha_{26} \mathcal{G}_9(i, j) + \alpha_{27} \mathcal{G}_9(i, k). \quad (68)$$

The remaining four terms in \mathcal{G}_9 are not necessary due to the following reasons:

- $\mathcal{G}_9(i, l)$ is not necessary because its addition to $\mathcal{G}_9(i, j)$ and $\mathcal{G}_9(i, k)$ will lead to a function involving only z_f and z_b as per the second last row of Table 3.
- Any function $\mathcal{G}_9(j, k)$ is not necessary because its addition to $\mathcal{G}_9(i, j)$ and $\mathcal{G}_9(i, k)$ will lead to a function involving only z_f and z_b as per the last row of Table 3.
- Any function $\mathcal{G}_9(j, l)$ is not necessary because its addition to $\mathcal{G}_9(i, k)$ can be represented using a function that involves only z_f as per the last row of Table 2.
- Any function $\mathcal{G}_9(k, l)$ is not necessary because its addition to $\mathcal{G}_9(i, j)$ can be represented using a function that involves only z_f as per the last row of Table 2.

We observed that we need a maximum of two functions from \mathcal{G}_9 to represent any function in \mathcal{F}_2^4 . So there are two possibilities for $f(\mathbf{x})$ and we denote them as $f1(\mathbf{x})$ and $f2(\mathbf{x})$ depending on whether we use one or two of the functions from \mathcal{G}_9 as shown below:

$$f1(\mathbf{x}) = \alpha \mathcal{G}_9(i, j) + \beta \mathcal{G}_5(i, k, l) + \gamma \mathcal{G}_5(j, k, l) + \delta \mathcal{G}_6(i, j, k) + \eta \mathcal{G}_6(i, j, l) + \sigma \min_{z_f} (g_f(\mathbf{x}) z_f) + g(\mathbf{x}), \quad (69)$$

$$f2(\mathbf{x}) = \alpha \mathcal{G}_9(i, j) + \beta \mathcal{G}_9(i, k) + \gamma \mathcal{G}_5(j, k, l) + \delta \mathcal{G}_6(i, j, k) + \eta \min_{z_f} (g_f(\mathbf{x}) z_f) + g(\mathbf{x}). \quad (70)$$

We have represented $f1$ and $f2$ using 7 and 6 terms respectively. We show that we do not need any other functions for representing $f1$ and $f2$:

- \mathcal{G}_1 : We can represent them using $g(\mathbf{x})$.
- \mathcal{G}_2 : These functions involve z_f and we can represent them using $\min_{z_f} g_f(\mathbf{x}, z_f)$ according to Lemma 3.
- \mathcal{G}_3 : These functions involve z_f and we can represent them using $\min_{z_f} g_f(\mathbf{x}, z_f)$ according to Lemma 3.
- \mathcal{G}_4 : By adding this function to $\mathcal{G}_9(i, j)$ we obtain functions in \mathcal{G}_6 . The generated \mathcal{G}_6 functions can be subsequently added to any functions in $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$. If there is no \mathcal{G}_9 term, then we can represent the function using z_f and z_b as explained earlier in the case of $f0$.
- \mathcal{G}_5 : In the case of $f1$, some functions in \mathcal{G}_5 can be added to $\mathcal{G}_9(i, j)$ to obtain \mathcal{G}_6 and \mathcal{G}_8 . The generated functions can be subsequently added to any $\mathcal{G}_9(i, j)$ terms. If there is no $\mathcal{G}_9(i, j)$ term, then we can represent the function using z_f and z_b as explained earlier in the case of $f0$. The functions $\mathcal{G}_5(i, k, l)$ and $\mathcal{G}_5(j, k, l)$ alter the coefficients of z_i . For now, we keep these functions of \mathcal{G}_5 as separate terms in function $f1(\mathbf{x})$.
In the case of $f2$, some functions in \mathcal{G}_5 can be added to $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$ to obtain \mathcal{G}_6 and \mathcal{G}_8 terms. The generated functions can be subsequently added to any $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$ terms. If there is no $\mathcal{G}_9(i, j)$ term, then we can represent the function using z_f and z_b as explained earlier in the case of $f0$. On adding the function $\mathcal{G}_5(j, k, l)$ to $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$ we produce functions that alter the coefficients of z_i . For now, we keep this function in \mathcal{G}_5 to represent the original function $f2(\mathbf{x})$.
- \mathcal{G}_6 : In the case of $f1$, some functions in \mathcal{G}_6 can be added to $\mathcal{G}_9(i, j)$ to obtain functions in \mathcal{G}_5 , which can be subsequently added to any $\mathcal{G}_9(i, j)$ terms. If there is no $\mathcal{G}_9(i, j)$ term, then we can represent the function using z_f and z_b as explained earlier in the case of $f0$. The functions $\mathcal{G}_6(i, j, k)$ and $\mathcal{G}_6(i, j, l)$ produce functions that alter the coefficients of z_i . For now, we keep these two functions of \mathcal{G}_6 as separate terms in $f1(\mathbf{x})$.
In the case of $f2$, some functions in \mathcal{G}_6 can be added to $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$ to obtain \mathcal{G}_5 , which can be subsequently added to any $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$ terms. If there is no $\mathcal{G}_9(i, j)$ term, then we can represent the function using z_f and z_b as explained earlier in the case of $f0$. On adding the function $\mathcal{G}_6(i, j, k)$ to $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$ we produce functions that alter the coefficients of z_i . For now, we keep this function $\mathcal{G}_6(i, j, k)$ as separate term in $f2(\mathbf{x})$.
- \mathcal{G}_7 : Some functions in \mathcal{G}_7 can be added to $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$ to generate functions in \mathcal{G}_8 , which can be subsequently added to any $\mathcal{G}_9(i, j)$ or $\mathcal{G}_9(i, k)$ terms. In other cases, the functions in \mathcal{G}_7 only modify the coefficients of z_f terms that can be represented by the function $\min_{z_f} (g_f(\mathbf{x})z_f)$.
- \mathcal{G}_8 : This function can be represented by $\min_{z_f} (g_f(\mathbf{x})z_f)$ since it only has one AVz_f .

Using Table 5 we rewrite $f1$ as given below:

$$f1(\mathbf{x}) = g(\mathbf{x}) + \min_{z_f, z_i} ((\alpha(2 - x_k - x_l) + \sigma g(\mathbf{x}))z_f + \quad (71)$$

$$\begin{aligned} & (\alpha(1 - x_i - x_j) + \\ & \beta(2 - x_i - 2x_j - x_k - x_l) + \\ & \gamma(2 - 2x_i - x_j - x_k - x_l) + \\ & \delta(1 - x_i - x_j - x_k) + \\ & \eta(1 - x_i - x_j - x_l))z_i - \\ & \alpha z_f z_i) \\ & = g(\mathbf{x}) + \min_{z_f, z_i} (g'_f(\mathbf{x})z_f + g_i(\mathbf{x})z_i - \alpha z_f z_i) \end{aligned}$$

(72)

$f_1(\mathbf{x}), \alpha \in \mathbb{R}^+$	$f_2(\mathbf{x}), \beta \in \mathbb{R}^+$	$\min_{z_1, z_2} h(\mathbf{x}, z_1, z_2),$ where $h(\mathbf{x}, z_1, z_2) = f_1(\mathbf{x}) + f_2(\mathbf{x}), \quad \forall \mathbf{x}$	AVs
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_1(i, j)$	$-\beta x_i x_j + \alpha \min_{z_f, z_i} ((2 - x_k - x_l)z_f + (1 - x_i - x_j)z_i - z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(kl)$
$\mathcal{G}_9(i, j)$	$\mathcal{G}_2(i, j, k)$	$-x_i x_j + \min_{z_f} (3 - x_i - x_j - 2x_k - x_l)z_f$	z_f
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_2(j, k, l)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l) + \beta(2 - x_j - x_k - x_l))z_f + \alpha(1 - x_i - x_j)z_i - \alpha z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(kl)$
$\mathcal{G}_9(i, j)$	\mathcal{G}_3	$-x_i x_j + \mathcal{G}_2(i, k, l) + \mathcal{G}_2(j, k, l) = -x_i x_j + \min_{z_f} ((2 - x_i - x_k - x_l) + (2 - x_j - x_k - x_l))z_f$	z_f
$\mathcal{G}_9(i, j)$	\mathcal{G}_4	$\mathcal{G}_6(i, j, k) + \mathcal{G}_6(i, j, l) - x_k x_l = -x_k x_l + \min_{z_b} (2 - 2x_i - 2x_j - x_k - x_l)z_b$	z_b
$\mathcal{G}_9(i, j)$	$\mathcal{G}_5(i, j, k)$	$\mathcal{G}_8 + \mathcal{G}_6(i, j, l) - x_k x_l = -x_k x_l + \min_{z_f} (2 - x_i - x_j - x_k - x_l)z_f + \min_{z_b} (1 - x_i - x_j - x_l)z_b$	$z_f,$ z_b
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_5(j, k, l)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l)z_f + (\alpha(1 - x_i - x_j) + \beta(2 - 2x_i - x_j - x_k - x_l))z_i - \alpha z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(kl)$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_6(i, j, k)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l)z_f + (\alpha(1 - x_i - x_j) + \beta(1 - x_i - x_j - x_k))z_i - \alpha z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(kl)$
$\mathcal{G}_9(i, j)$	$\mathcal{G}_6(j, k, l)$	$\mathcal{G}_5(i, k, l) - x_k x_l = -x_k x_l + \min_{z_b} (2 - x_i - 2x_j - x_k - x_l)z_b$	z_b
$\mathcal{G}_9(i, j)$	$\mathcal{G}_7(i)$	$\mathcal{G}_8 - x_i x_j + \mathcal{G}_2(i, k, l) = -x_1 x_2 + \min_{z_f} ((2 - x_i - x_j - x_k - x_l) + (2 - x_i - x_k - x_l))z_f$	z_f
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_7(k)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l) + \beta(3 - x_i - x_j - 2x_k - x_l))z_f + \alpha(1 - x_i - x_j)z_i - \alpha z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(kl)$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_8$	$\min_{z_f, z_i} (\alpha(2 - x_k - x_l) + \beta(2 - x_i - x_j - x_k - x_l))z_f + \alpha(1 - x_i - x_j)z_i - \alpha z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(kl)$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_9(i, k)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l) + \beta(2 - x_j - x_l))z_f + (\alpha(1 - x_i - x_j) + \beta(1 - x_i - x_k))z_i - \alpha z_f z_i - \beta z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(kl, jl)$
$\mathcal{G}_9(i, j)$	$\mathcal{G}_9(k, l)$	$\mathcal{G}_8 - x_i x_j - x_k x_l = -x_i x_j - x_k x_l + \min_{z_f} ((2 - x_i - x_j - x_k - x_l)z_f$	z_f

Table 2. We show the sum of a function $\mathcal{G}_9(i, j)$ with any other function in Table 1 can be expressed using two auxiliary variables. Here the index set $\{i, j, k, l\} = S_4$ denotes the four distinct integers $S_4 = \{1, 2, 3, 4\}$.

$f_1(\mathbf{x}), \alpha \in \mathbb{R}^+$	$f_2(\mathbf{x}), \beta \in \mathbb{R}^+$	$f_3(\mathbf{x}), \gamma \in \mathbb{R}^+$	$\min_{z_1, z_2} h(\mathbf{x}, z_1, z_2),$ $h(\mathbf{x}, z_1, z_2) = f_1(\mathbf{x}) + f_2(\mathbf{x}) + f_3(\mathbf{x}), \quad \forall \mathbf{x}$	AVs
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_5(i, k, l)$	$\gamma \mathcal{G}_5(j, k, l)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l)z_f + (\alpha(1 - x_i - x_j) + \beta(2 - x_i - 2x_j - x_k - x_l) + \gamma(2 - 2x_i - x_j - x_k - x_l))z_i - \alpha z_f z_i)$	$z_i \in \mathcal{I}(kl)^{z_f}$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_6(i, j, k)$	$\gamma \mathcal{G}_6(i, j, l)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l)z_f + (\alpha(1 - x_i - x_j) + \beta(1 - x_i - x_j - x_k) + \gamma(1 - x_i - x_j - x_l))z_i - \alpha z_f z_i)$	$z_i \in \mathcal{I}(kl)^{z_f}$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_5(j, k, l)$	$\gamma \mathcal{G}_6(i, j, l)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l)z_f + (\alpha(1 - x_i - x_j) + \beta(2 - 2x_i - x_j - x_k - x_l) + \gamma(1 - x_i - x_j - x_l))z_i - \alpha z_f z_i)$	$z_i \in \mathcal{I}(kl)^{z_f}$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_9(i, k)$	$\gamma \mathcal{G}_5(j, k, l)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l) + \beta(2 - x_j - x_l))z_f + (\alpha(1 - x_i - x_j) + \beta(1 - x_i - x_k) + \gamma(2 - 2x_i - x_j - x_k - x_l))z_i - \alpha z_f z_i - \beta z_f z_i)$	$z_i \in \mathcal{I}(kl, jl)^{z_f}$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_9(i, k)$	$\gamma \mathcal{G}_6(i, j, k)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l) + \beta(2 - x_j - x_l))z_f + (\alpha(1 - x_i - x_j) + \beta(1 - x_i - x_k) + \gamma(1 - x_i - x_j - x_k))z_i - \alpha z_f z_i - \beta z_f z_i)$	$z_i \in \mathcal{I}(kl, jl)^{z_f}$
$\mathcal{G}_9(i, j)$	$\mathcal{G}_9(i, k)$	$\mathcal{G}_9(i, l)$	$-x_j x_k x_l + \mathcal{G}_8 + \mathcal{G}_5(j, k, l) = -x_j x_k x_l + \min_{z_f} (2 - x_i - x_j - x_k - x_l)z_f + \min_{z_b} (2 - 2x_i - x_j - x_k - x_l)z_b$	z_f, z_b
$\mathcal{G}_9(i, j)$	$\mathcal{G}_9(i, k)$	$\mathcal{G}_9(j, k)$	$\mathcal{G}_7(l) + \mathcal{G}_8 + \mathcal{G}_6(i, j, k) = \min_{z_f} ((3 - x_i - x_j - x_k - 4x_l) + (2 - x_i - x_j - x_k - x_l))z_f + \min_{z_b} (1 - x_i - x_j - x_k)z_b$	z_f, z_b

Table 3. We show the sum of any three functions from Table 1 can be expressed using two auxiliary variables. Here the index set $\{i, j, k, l\} = S_4$ denotes the four distinct integers $S_4 = \{1, 2, 3, 4\}$.

$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$	$f_4(\mathbf{x})$	$\min_{z_1, z_2} h(\mathbf{x}, z_1, z_2),$ $h(\mathbf{x}, z_1, z_2) = f_1(\mathbf{x}) + f_2(\mathbf{x}) + f_3(\mathbf{x}) + f_4(\mathbf{x}), \quad \forall \mathbf{x}$	AVs
$\mathcal{G}_9(i, j)$	$\mathcal{G}_5(i, k, l)$	$\mathcal{G}_5(j, k, l)$	$\mathcal{G}_5(i, j, k)$	$(\mathcal{G}_9(i, j) + \mathcal{G}_5(i, j, k)) +$ $(\mathcal{G}_5(i, k, l) + \mathcal{G}_5(j, k, l)) =$ $(\mathcal{G}_8 + \mathcal{G}_6(i, j, l) + \mathcal{G}_1(k, l)) +$ $(\mathcal{G}_5(i, k, l) + \mathcal{G}_5(j, k, l))$	$z_f,$ z_b
$\mathcal{G}_9(i, j)$	$\mathcal{G}_6(i, j, k)$	$\mathcal{G}_6(i, j, l)$	$\mathcal{G}_6(j, k, l)$	$(\mathcal{G}_9(i, j) + \mathcal{G}_6(j, k, l)) +$ $(\mathcal{G}_6(i, j, k) + \mathcal{G}_6(i, j, l)) =$ $(\mathcal{G}_5(i, j, k) + \mathcal{G}_1(k, l)) +$ $(\mathcal{G}_6(i, j, k) + \mathcal{G}_6(i, j, l))$	$z_f,$ z_b
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_5(i, k, l)$	$\gamma \mathcal{G}_5(j, k, l)$	$\delta \mathcal{G}_6(i, j, k)$	$\min_{z_f, z_i} (\alpha(2 - x_k - x_l)z_f +$ $(\alpha(1 - x_i - x_j) +$ $\beta(2 - x_i - 2x_j - x_k - x_l) +$ $\gamma(2 - 2x_i - x_j - x_k - x_l) +$ $\delta(1 - x_i - x_j - x_k))z_i -$ $\alpha z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(k, l)$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_6(i, j, k)$	$\gamma \mathcal{G}_6(i, j, l)$	$\delta \mathcal{G}_5(i, k, l)$	$\min_{z_f, z_i} (\alpha(2 - x_k - x_l)z_f +$ $(\alpha(1 - x_i - x_j) +$ $\beta(1 - x_i - x_j - x_k) +$ $\gamma(1 - x_i - x_j - x_l) +$ $\delta(2 - x_i - 2x_j - x_k - x_l))z_i -$ $\alpha z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(k, l)$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_9(i, k)$	$\gamma \mathcal{G}_5(j, k, l)$	$\delta \mathcal{G}_6(i, j, k)$	$\min_{z_f, z_i} ((\alpha(2 - x_k - x_l) + \beta(2 - x_j - x_l))z_f +$ $(\alpha(1 - x_i - x_j) +$ $\beta(1 - x_i - x_k) +$ $\gamma(2 - 2x_i - x_j - x_k - x_l) +$ $\delta(1 - x_i - x_j - x_k))z_i -$ $\alpha z_f z_i - \beta z_f z_i)$	$z_f,$ $z_i \in \mathcal{I}(k, l)$

Table 4. We show the sum of any four functions from Table 1 can be expressed using two auxiliary variables. Here the index set $\{i, j, k, l\} = S_4$ denotes the four distinct integers $S_4 = \{1, 2, 3, 4\}$.

$f_1(\mathbf{x}),$ $\alpha \in \mathbb{R}^+$	$f_2(\mathbf{x}),$ $\alpha \in \mathbb{R}^+$	$f_3(\mathbf{x}),$ $\alpha \in \mathbb{R}^+$	$f_4(\mathbf{x}),$ $\alpha \in \mathbb{R}^+$	$f_5(\mathbf{x}),$ $\alpha \in \mathbb{R}^+$	$\min_{z_1, z_2} (f_1(\mathbf{x}) + f_2(\mathbf{x}) + f_3(\mathbf{x}) + f_4(\mathbf{x}) + f_5(\mathbf{x})),$ $\forall \mathbf{x}$
$\alpha \mathcal{G}_9(i, j)$	$\beta \mathcal{G}_5(i, k, l)$	$\gamma \mathcal{G}_5(j, k, l)$	$\delta \mathcal{G}_6(i, j, k)$	$\eta \mathcal{G}_6(i, j, l)$	$\min_{z_f, z_i} (\alpha(2 - x_k - x_l)z_f +$ $(\alpha(1 - x_i - x_j) +$ $\beta(2 - x_i - 2x_j - x_k - x_l) +$ $\gamma(2 - 2x_i - x_j - x_k - x_l) +$ $\delta(1 - x_i - x_j - x_k) +$ $\eta(1 - x_i - x_j - x_l))z_i -$ $\alpha z_f z_i)$

Table 5. We show the sum of five functions from Table 1 can be expressed using two auxiliary variables z_f and $z_i \in \mathcal{I}(k, l)$. Here the index set $\{i, j, k, l\} = S_4$ denotes the four distinct integers $S_4 = \{1, 2, 3, 4\}$.

Using the last row of Table 4 we rewrite $f2$ as given below:

$$\begin{aligned}
 f2(\mathbf{x}) &= g(\mathbf{x}) + \min_{z_f, z_i} ((\alpha(2 - x_k - x_l) + \beta(2 - x_j - x_l) + g_f(\mathbf{x}))z_f + \\
 &\quad (\alpha(1 - x_i - x_j) + \\
 &\quad \beta(1 - x_i - x_k) + \\
 &\quad \gamma(2 - 2x_i - x_j - x_k - x_l) + \\
 &\quad \delta(1 - x_i - x_j - x_k))z_i - \\
 &\quad \alpha z_f z_i - \beta z_f z_i) \\
 &= g(\mathbf{x}) + \min_{z_f, z_i} (g'_f(\mathbf{x})z_f + g_i(\mathbf{x})z_i - (\alpha + \beta)z_f z_i)
 \end{aligned} \tag{73}$$

(74)

It is shown that $f1$ and $f2$ need only two AVs z_f and z_s . In the case of $f0$, z_s is a backward partition. In the case of $f1$ and $f2$, z_s belongs to one of the 18 intermediate partitions. \square

6 Linear Programming solution

For a given function $f(x_1, x_2, x_3, x_4)$ in \mathcal{F}_2^4 , our goal is to compute a function $h(\mathbf{x}, \mathbf{z})$ in \mathcal{F}^2 . Theorem 2 shows that we need only two AVs (z_f, z_s) . Here z_f corresponds to the forward reference partition. The AV z_s is either the backward partition or one of the 18 intermediate reference partitions. Unfortunately, we do not know which one of these 19 partitions is required before we do the transformation. In what follows, we will show the transformation assuming that we know the specific partition for z_s . Note that z_b is a special case of z_i and we do not use the bilinear term $z_f z_s$ when $z_s = z_b$. In order to handle this condition we use a Boolean variable that takes the value 0 when the intermediate partition is the backward reference partition and 1 otherwise:

$$\delta(z_s) = \begin{cases} 0 & \text{if } z_i \in [\mathcal{A}_b, \mathcal{B}_b], \\ 1 & \text{otherwise.} \end{cases} \tag{75}$$

The required function $h(\mathbf{x}, \mathbf{z})$ is the following:

$$h(\mathbf{x}, z_f, z_s) = b_0 + \sum_i b_i x_i - \sum_{i>j} b_{ij} x_i x_j + (g_f - \sum_{i=1}^4 g_{f,i} x_i) z_f + (g_s - \sum_{i=1}^4 g_{s,i} x_i) z_s - \delta(z_s) j_{fs} z_f z_s, \tag{76}$$

such that $b_{ij}, g_{f,i}, g_{s,i}, j_{fs} \geq 0$ and $i, j \in S_4$. As we know the partitions of (z_f, z_s) , we know their Boolean values for all labelings of \mathbf{x} . We need the coefficients $(b_i, b_{ij}, j_{fs}, g_f, g_s, g_{f,i}, g_{s,i}), i \in S_4$ to compute $h(x_1, x_2, x_3, x_4, z_f, z_s)$. These coefficients satisfy both submodularity constraints (that the coefficients of all bilinear terms $(x_i x_j, x_i z_f, x_j z_s, z_f z_s)$ are less than or equal to zero) and those imposed by the reference partitions. First we list the submodularity conditions below:

$$\underbrace{\begin{pmatrix} b_{ij} \\ g_{f,i} \\ g_{s,i} \\ j_{fs} \end{pmatrix}^T}_{\mathcal{S}_p} \geq \mathbf{0}, i, j \in S_4, i \neq j, \tag{77}$$

where $\mathbf{0}$ refers of a vector composed of 0's of appropriate length. Next we list the conditions which guarantee $f(\mathbf{x}) = \min_{z_f, z_s} h(\mathbf{x}, z_f, z_s)$, $\forall \mathbf{x}$. Let $\eta(S)$ be the value of $z_f z_s$ for $S \in \mathcal{P}$. This can be obtained using the partitions of z_f and z_s .

$$\eta(S) = \begin{cases} 1 & \text{if } S \in (\mathcal{B}_f \cap \mathcal{B}_s) \\ 0 & \text{otherwise.} \end{cases} \quad (78)$$

Let us denote the value of AV_{z_s} for different subsets of S_4 as given below:

$$z_s^S = \begin{cases} 1 & \text{if } S \in \mathcal{B}_s, \\ 0 & \text{if } S \in \mathcal{A}_s. \end{cases} \quad (79)$$

Let \mathcal{G} and \mathcal{H} denote values of functions f and h respectively:

$$\mathcal{G} = f(\mathbf{1}_1^S, \mathbf{1}_2^S, \mathbf{1}_3^S, \mathbf{1}_4^S), \quad \forall S \in \mathcal{P} \quad (80)$$

$$\mathcal{H} = h(\mathbf{1}_1^S, \mathbf{1}_2^S, \mathbf{1}_3^S, \mathbf{1}_4^S, 0, 0) + (g_f - \sum_{i=1}^4 g_{f,i} \mathbf{1}_i^S) z_f^S + (g_s - \sum_{i=1}^4 g_{s,i} \mathbf{1}_i^S) z_s^S - \delta(z_s) \eta(S) j_{fs} \quad (81)$$

As a result we have the following 16 linear Equations (N.B. there are $2^4(16)$ different S):

$$\mathcal{G} = \mathcal{H}, \quad \forall S \in \mathcal{P} \quad (82)$$

We already know the partition of (z_f, z_s) and their appropriate values a priori. The following constraints ensure that z_f and z_s behave according to their associated partitions.

$$\underbrace{\begin{pmatrix} g_f - \sum_{i=1}^4 g_{f,i} \mathbf{1}_i^S \\ g_s - \sum_{i=1}^4 g_{s,i} \mathbf{1}_i^D \end{pmatrix}}_{\mathcal{G}_g} \geq \mathbf{0}, S \in \mathcal{A}_f, D \in \mathcal{A}_s \quad (83)$$

$$\underbrace{\begin{pmatrix} g_f - \sum_{i=1}^4 g_{f,i} \mathbf{1}_i^S - \delta(z_s) \eta(S) j_{fs} \\ g_s - \sum_{i=1}^4 g_{s,i} \mathbf{1}_i^D - \delta(z_s) \eta(D) j_{fs} \end{pmatrix}}_{\mathcal{G}_l} \leq \mathbf{0}, S \in \mathcal{B}_f, D \in \mathcal{B}_s. \quad (84)$$

We need to compute the coefficients $(b_{ij}, g_f, g_{f,i}, g_s, g_{s,i}, j_{fs})$ that satisfy the Equations (77), (82), and (84). This is equivalent to finding a feasible point in a linear programming problem:

$$\min \text{ const} \quad (85)$$

$$\text{s.t. } \mathcal{S}_p \geq \mathbf{0}, \mathcal{G} = \mathcal{H}, \mathcal{G}_g \geq \mathbf{0}, \mathcal{G}_l \leq \mathbf{0} \quad (86)$$

In the above LP formulation we assumed that we know the partition of AVs z_f and z_s . However, z_s can be one of the 19 partitions. Before we do the transformation it is not easy to know which one of the 19 partitions is necessary. So we solve the LP 19 times and iterate over all the 19 partitions to identify the necessary one. For the correct partition, will be able to find a solution that satisfies all the constraints.

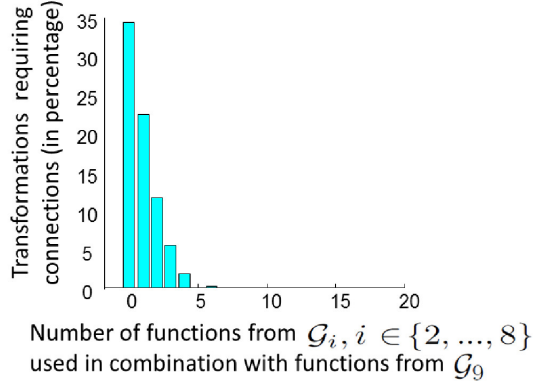


Fig. 5. We generated submodular functions using non-weighted sum of functions from groups $\mathcal{G}_i, i = \{2, \dots, 9\}$. The x -axis denotes the number of functions chosen from $\mathcal{G}_i, i = \{2, \dots, 8\}$ in generating the submodular functions and the y -axis gives the percentage of transformations requiring z_i in intermediate partition.

7 Experiments

The functions in the class \mathcal{F}_2^4 can be transformed to functions in \mathcal{F}^2 using 25 AVs according to existing results [49]. We show that this transformation can be done using only two AVs using a linear program. In Matlab, the transformation takes around 0.03 seconds and it can be further improved using efficient C++ implementation.

In our experiments as shown in Figure 5, we generated submodular functions using non-weighted sum of functions from \mathcal{G}_9 and functions from groups $\mathcal{G}_i, i = \{2, \dots, 8\}$. We do not consider functions from group \mathcal{G}_1 since they do not require any AVs. The number of functions $n_{\mathcal{G}}$ used from groups $\mathcal{G}_i, i = \{2, \dots, 8\}$ is increased from 0 to 19. For each value of $n_{\mathcal{G}}$, we generated 1000 functions and the non-negative weights are randomly generated in the interval $[0, 1]$. We observed that as we increase $n_{\mathcal{G}}$, the generated submodular functions were less likely to use intermediate AVs. This also concurs with Table 2, that many combinations of \mathcal{G}_9 with other functions can be represented using functions in the first 8 groups (\mathcal{G}_1 to \mathcal{G}_8) that do not require any AVs in intermediate partition.

8 Discussion and open problems

The reduction of higher order functions to quadratic ones will be beneficial for developing efficient minimization algorithms. These techniques can be broadly classified into two types: submodularity-preserving [1,28,18,48,38,14,40,50,42] and general techniques [41,12,15,2,21]. This paper belongs to the submodular-preserving class of algorithms where higher order submodular functions are transformed to quadratic submodular functions using AVs. The general techniques are usually employed in association with roof-duality approaches for minimizing non-submodular functions [4,3,5,43]. The general techniques also employ AVs and these AVs need not be MBFs. The existing upper bound for general reduction techniques is given by $G(k) = 2^{k-2}(k-3)+1$ for a k th order function. We show the comparison between the AVs used in general techniques and submodularity-preserving techniques in Table 6.

Note that the upper bound for the number of AVs required for submodularity-preserving transformation is much higher than for general reduction techniques. We have improved the upper bound for submodular functions from 2^{2^k} to Dedekind number $D(k)$. In the case of fourth order functions we have

Type	Degree	3	4	5	6	7	8
General	Ishikawa [21]	1	5	17	49	129	321
Submodularity Preserving Dedekind [25]							
		1	2	7581	$\approx 7.8 \times 10^6$	$\approx 2.4 \times 10^{12}$	$\approx 5.6 \times 10^{23}$

Table 6. Comparison of the number of AVs used for general versus submodularity-preserving techniques. The Dedekind number $D(k)$ is unknown for $k > 8$.

further improved the upper bound from 168 ($D(4)$) to 2.

Acknowledgments: Srikumar Ramalingam would like to thank Mitsubishi Electric Research Laboratories (MERL) for the support. L'ubor Ladický is funded by Max Planck Center for Learning Systems Fellowship. Philip H.S. Torr would like to acknowledge the financial support provided by ERC grant ERC-2012-AdG 321162-HELIOS, EPSRC/MURI grant ref EP/N019474/1, EPSRC grant EP/M013774/, and EPSRC Programme Grant Seebibyte EP/M013774/1.

References

1. A. Billionnet and M. Minoux. Maximizing a supermodular pseudo-boolean function: a polynomial algorithm for supermodular cubic functions. *Discrete Appl. Math.*, 12(1):1–11, 1985.
2. E. Boros and A. Gruber. On quadratization of pseudo-boolean functions. In *ISAIM*, 2012.
3. E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Appl. Math.*, 123(1-3):155–225, 2002.
4. E. Boros and P.L. Hammer. A max-flow approach to improved roof-duality in quadratic 0-1 minimization. *Technical Report RRR 15-1989, RUTCOR, Rutgers University*, 1989.
5. E. Boros, P.L. Hammer, R. Sun, and G. Tavares. A max-flow approach to improved lower bounds for quadratic unconstrained binary optimization (qubo). *Discrete Optimization*, 5(2):501–529, 2008.
6. D.A. Cohen, M.C. Cooper, P. Creed, and S. Zivny P.G. Jeavons. An algebraic theory of complexity for discrete optimization. In *SIAM Journal of Computing*, volume 42(5), pages 1915–193, 2013.
7. M.C. Cooper. Minimization of locally defined submodular functions by optimal soft arc consistency. *Constraints*, 13(4):437–458, 2008.
8. Y. Crama and P.L. Hammer. *Boolean Functions: Theory, Algorithms and Applications*. Cambridge University Press, 2011.
9. W.H. Cunningham. On submodular function minimization. *Combinatorica*, 5:185–192, 1985.
10. J. Edmonds. Submodular functions, matroids and certain polyhedra. *Calgary International Conference on Combinatorial Structures and their applications*, page 69–87, 1969.
11. M.L. Fisher, G.L. Nemhauser, and L.A. Wolsey. An analysis of approximation for maximizing submodular setfunctions-i. *Mathematical Programming Studies*, 8:73–87, 1978.
12. A. Fix, A. Gruber, E. Boros, and R. Zabih. A graph cut algorithm for higher-order markov random fields. In *ICCV*, pages 1020 – 1027, 2011.
13. L. Fleischer and S. Iwata. A push-relabel framework for submodular function minimization and applications to parametric optimization. *Discrete Applied Mathematics*, 131(2):311–322, 2001.
14. D. Freedman and P. Drineas. Energy minimization via graph cuts: Settling what is possible. In *CVPR*, volume 2, pages 939–946, 2005.
15. A.C. Gallagher, D. Batra, and D. Parikh. Inference for order reduction in markov random fields. In *CVPR*, pages 1857 – 1864, 2011.
16. G. Gallo and B. Simeone. On the supermodular knapsack problem. *Mathematical Programming: Series A and B*, 45(2):295–309, 1989.
17. M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.
18. P. L. Hammer. Some network flow problems solved with pseudo-boolean programming. *Operations Research*, 13:388–399, 1965.
19. G. Hansel. Sur le nombre des fonctions booléennes monotones de n variables. *C.R. Acad. Sci. Paris*, 1966.

20. H. Ishikawa. Exact optimization for Markov random fields with convex priors. *PAMI*, 25:1333–1336, 2003.
21. H. Ishikawa. Transformation of general binary mrf minimization to the first-order case. *PAMI*, 13(6):1234 – 1249, 2011.
22. S. Iwata. A fully combinatorial algorithm for submodular function minimization. *Journal of Combinatorial Theory, Series B*, 84(2):203–212, 2002.
23. S. Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM J. Computing*, 2337:1–8, 2003.
24. S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.
25. D. Kleitman. On dedekind's problem: The number of monotone boolean functions. *Proc. Amer. Math. Soc.*, 1969.
26. P. Kohli, M. P. Kumar, and P. H. S. Torr. P^3 & beyond: Solving energies with higher order cliques. In *CVPR*, pages 1 – 8, 2007.
27. V. Kolmogorov. Minimizing a sum of submodular functions. *Discrete Applied Mathematics*, 160(15):2246–2258, 2012.
28. V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *PAMI*, 26(2), 2004.
29. A.D. Korshunov. The number of monotone boolean functions. *Problemy Kibernet* 38:5-108, 1981.
30. L. Ladický, C. Russell, P. Kohli, and P. Torr. Graph cut based inference with co-occurrence statistics. In *European Conference on Computer Vision*, pages 239–253. springer, 2010.
31. X. Lan, S. Roth, D. P. Huttenlocher, and M. J. Black. Efficient belief propagation with learned higher-order Markov random fields. In *ECCV*, pages 269–282, 2006.
32. Y.T. Lee, A. Sidford, and S.C. Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science*, 2015.
33. L. Lovasz. Submodular functions and convexity. *Mathematical Programming - The State of the Art*, pages 235–257, 1983.
34. M. Narasimhan and J.A. Bilmes. A submodular-supermodular procedure with applications to discriminative structure learning. In *Uncertainty in Artificial Intelligence*, pages 404–412, 2005.
35. R. Nishihara, S. Jegelka, and M.I. Jordan. On the convergence rate of decomposable submodular function minimization. In *NIPS*, 2014.
36. J.B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.
37. S. Promislow and V. Young. Supermodular functions on finite lattices. *Order* 22(4), 2005.
38. M. Queyranne. A combinatorial algorithm for minimizing symmetric submodular functions. *SODA*, pages 98–101, 1995.
39. S. Ramalingam, P. Kohli, K. Alahari, and P.H.S. Torr. Exact inference in multi-label crfs with higher order cliques. In *CVPR*, pages 1–8, 2008.
40. J.M.W. Rhys. A selection problem of shared fixed costs and network flows. *Management Science*, pages 200 – 207, 1970.
41. I. G. Rosenberg. Reduction of bivalent maximization to the quadratic case. *Cahiers du Centre d'Etudes de Recherche Operationnelle*, 17:71–74, 1975.
42. C. Rother, P. Kohli, W. Feng, and J. Jia. Minimizing sparse higher order energy functions of discrete variables. In *CVPR*, 2009.
43. C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer. Optimizing binary MRFs via extended roof duality. In *CVPR*, 2007.
44. D. Schlesinger and B. Flach. Transforming an arbitrary minsum problem into a binary one. Technical Report TUD-FI06-01, Dresden University of Technology, 2006.
45. A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory Series B*, 80(2):346–355, 2000.
46. L. Trevisan G.B. Sorkin, M. Sudan, and D.P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal of Computing*, 29(6):2074–2097, 2000.
47. Alan Yuille, Anand Rangarajan, and A. L. Yuille. The concave-convex procedure (cccp). In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.
48. B. Zalesky. Efficient determination of gibbs estimators with submodular energy functions. <http://arxiv.org/abs/math/0304041v1>, 2003.

49. S. Zivny, D.A.Cohen, and P.G. Jeavons. The expressive power of binary submodular functions. *Discrete Applied Mathematics*, 157(15):3347–3358, 2009.
50. S. Zivny and P.G. Jeavons. Classes of submodular constraints expressible by graph cuts. *Proceedings of CP*, pages 112–127, 2008.
51. S. Zivny and P.G. Jeavons. Which submodular functions are expressible using binary submodular functions? *Oxford University Computing Laboratory Research Report CS-RR-08-08*, 2008.
52. S. Zivny and P.G. Jeavons. Classes of submodular constraints expressible by graph cuts. *Constraints*, 15(3):430–452, 2010.