ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-1194

Fixed-Parameter Tractable Optimization Under DNNF Constraints

Frédéric Koriche and **Daniel Le Berre** and **Emmanuel Lonca** and **Pierre Marquis**[1]

Abstract. Minimizing a cost function under a set of combinatorial constraints is a fundamental, yet challenging problem in AI. Fortunately, in various real-world applications, the set of constraints describing the problem structure is much less susceptible to change over time than the cost function capturing user's preferences. In such situations, compiling the set of feasible solutions during an offline step can make sense, especially when the target compilation language renders computationally easier the generation of optimal solutions for cost functions supplied "on the fly", during the online step. In this paper, the focus is laid on Boolean constraints compiled into `DNNF` representations. We study the complexity of the minimization problem for several families of cost functions subject to `DNNF` constraints. Beyond linear minimization which is already known to be tractable in the `DNNF` language, we show that both quadratic minimization and submodular minization are *fixed-parameter tractable* for various subsets of `DNNF`. In particular, the fixed-parameter tractability of constrained submodular minimization is established using a natural parameter capturing the structural dissimilarity between the submodular cost function and the `DNNF` representation.

1 INTRODUCTION

Constraint optimization is a fundamental problem in computer science, which arises in various applications including among others, configuration softwares, recommender systems, and e-commerce. For many, if not most, of these applications, the space of feasible solutions under consideration is of combinatorial nature. In AI, a generic approach for representing such combinatorial tasks is the *valued constraint network* framework [5, 26, 29, 34, 37]. Informally, a valued constraint network consists of a set of discrete variables, a collection of *hard* (or *crisp*) constraints encoding a space of feasible solutions, and a set of *soft* constraints (or *potentials*) specifying preferences over solutions. The problem is to find a feasible solution that minimizes the sum of potentials. This expressive framework has also been studied under different names, such as GAI networks [1, 19] and conditional random fields [25, 31]. However, such expressiveness does not come without a price: the minimization problem for valued constraint networks is NP-hard, which prevents one from ensuring some reasonable performance guarantees.

Fortunately, in many real-world situations, the set of hard constraints representing the problem structure does not often evolve, especially in comparison with soft constraints, capturing user's preferences, which are indeed likely to change with the user. This situation pattern can be exploited using a *knowledge compilation* approach [11]: the set of hard constraints is compiled during an offline phase, in order to improve the time needed for the online generation of user-dependent optimal solutions.

As a matter of example, consider a configurable housing web service. Due to its combinatorial nature, the space of customizable residences is represented implicitly using hard constraints, such as "your home must include at least two bedrooms", or "only houses with at least one large bedroom come with luxury kitchens". Clearly, the set of feasible houses does not depend on any user's requirements like "I want a luxury kitchen", or preferences like "I prefer large bedrooms to small ones". Thus, compiling the set of constraints describing the space of feasible solutions during an offline step is relevant if this compilation step renders computationally easier the generation of feasible, yet non-dominated solution, matching the user's requirements and preferences, which are only known at the online step. Actually, such configuration problems are well-known benchmarks for knowledge compilation. Especially, the car configurators of Renault and Toyota take advantage of knowledge compilation techniques for ensuring guaranteed response times for some key requests.

In this paper, hard constraints are represented as (Boolean) `NNF` circuits. Specifically, we focus on constraints compiled into `DNNF` circuits [8], the subclass of `NNF` circuits for which "and-nodes" do not share any variable. `DNNF` is one of the most succinct `NNF` languages that admits a polynomial time algorithm for the task of determining whether a partial assignment can be extended to a feasible solution. This key property is preserved by subsets of `DNNF` such as, for example, the class `DNF` of disjunctive normal form formulae, the class `SDNNF` of *structured* `DNNF` formulae [28], the language `SDD` of sentential decision diagrams [10], and the language `OBDD` of ordered binary decision diagrams [3]. The choice of `DNNF` is also motivated by existing compilers targeting (a subset of) this language, including `c2d` [9], `sdd` [10], and `Dsharp` [27].

In the following, we consider several families $\mathcal{F}$ of (pseudo-Boolean) cost functions including, in particular, *submodular* functions which have received a great deal of interest in combinatorial optimization [16, 30, 21], with numerous applications in AI.[2] The aim of this study is to identify the complexity of the minimization query **MIN**: given a hard constraint C represented as a formula in a subset $\mathcal{L}$ of `DNNF`, and a cost function f expressed as a sum of potentials from a family $\mathcal{F}$, find (when it exists) a feasible solution of C that minimizes f. Plugged into the valued constraint network setting, our goal is to investigate the tractability of constrained optimization problems for which the soft constraints are defined over $\mathcal{F}$, and the set of hard constraints has been first compiled into a single constraint, represented as a formula C in $\mathcal{L}$. For various subsets $\mathcal{L}$ of `DNNF` and families $\mathcal{F}$ of cost functions, we determine whether the minimization problem, defined over $\mathcal{L}$ and $\mathcal{F}$, is in P, or is NP-hard.

[1] CRIL, Univ. Artois and CNRS, F-62300 Lens, France, email: name@cril.fr

[2] See e.g., `submodularity.org/references.html`

Furthermore, taking advantage of Downey and Fellows' parameterized complexity framework [13], a fine-grained analysis of the NP-hard cases is achieved, leading to the identification of *fixed-parameter tractable* restrictions. In the theory of parameterized complexity, the efficiency of an algorithm is evaluated by considering two measurements: the usual size n of the input, and an additional parameter k. This parameter typically represents a structural dimension of the input such as, for example, tree-likeness when the input is a graph. Fixed-parameter tractable (FPT) algorithms are those for which the running time has the form $p(k)n^{\mathcal{O}(1)}$ for a function p which depends only on k. In our setting, **MIN** is fixed-parameter tractable with respect to k if for every constraint C in $\mathcal{L}$, every sum f of potentials in $\mathcal{F}$, and every fixed value of k, the task of minimizing f subject to C can be solved in time polynomial in the sizes of C and f, with a polynomial degree independent of k. From a practical perspective, a fixed-parameter tractability result for the query **MIN** indicates that the optimization task can be solved efficiently for small values of k, even if the sizes of the circuit C and of the representation of the cost function f are large.

The main tractability result already known for the **MIN** query concerns linear minimization under DNNF constraints [12]. In this case, the scope of each potential reduces to a singleton. So, the very purpose of this study is to investigate the complexity of **MIN** for larger families $\mathcal{F}$ of cost functions. Here, strong negative results in the literature reveal that the quest of extending tractability to nonlinear cost functions is far from easy. Indeed, for the family of quadratic functions, expressed as sums of potentials of arity at most 2, *unconstrained* minimization is NP-hard by an immediate reduction from MIN-2-SAT [7, 35]. For the class of submodular functions, it is well-known that the unconstrained version of the minimization is in P [21, 23]. Yet, its constrained version is generally NP-hard, even in the very restricted case when the set of feasible solutions are described by a single cardinality constraint [32, 33].

Based on [12], the present contribution provides a wider range of results for optimization subject to DNNF constraints, especially in the setting of submodular functions. On the one hand, we show that **MIN** is NP-hard for quadratic submodular functions under OBDD constraints and general submodular functions under tree-structured (alias "acyclic") DNNF constraints. This immediately implies that constrained submodular minimization is intractable under DNNF constraints. On the other hand, we show that **MIN** is FPT for various subsets $\mathcal{L}$ of DNNF and families $\mathcal{F}$ of submodular functions. Here, the key complexity parameter ensuring fixed-parameter tractability is captured by the dissimilarity between the structure of the input cost function and the structure of the DNNF formula. In a nutshell, the take-home message of this study is that *structural compatibility* between the hard constraint and the cost function plays a key role in efficient constrained minimization.

2 THE FRAMEWORK

We begin with some basic notations that will be used throughout the paper. For a positive integer n, we use $[n]$ to denote the set $\{1, \ldots, n\}$. We also use $\mathbb{Q}_+$ to denote the set of nonnegative rationals, and define $\overline{\mathbb{Q}}_+ = \mathbb{Q}_+ \cup \{\infty\}$ with the standard addition operation extended so that $v + \infty = \infty$ for all $v \in \mathbb{Q}_+$.

As usual, propositional representations are defined over a set of Boolean variables $X = \{x_1, \ldots, x_p\}$, the constants $\top$ (true) and $\bot$ (false), and the connectives $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction). A literal is a variable x_i or its negation $\neg x_i$, also denoted $\overline{x}_i$. A term (resp. clause) is a conjunction (resp. disjunction) of literals. For a subset $Y \subseteq X$, a partial assignment $\mathbf{y}$ over Y is a vector in $2^{|Y|}$, which can be represented in an equivalent way by a canonical term over Y, i.e., a consistent term with $|Y|$ literals, each defined on a variable of Y that appears once in the term. We also use $\mathbf{Y}$ to denote the set of all partial assignments over Y, and $\mathbf{X}$ to denote the set $\{0,1\}^n$ of all complete assignments, called *interpretations*.

A Boolean *valued constraint network* (or VCN, for short) P consists of a set $X = \{x_1, \ldots, x_p\}$ of Boolean variables, and a set $\mathcal{C} = \{C_1, \ldots, C_m\}$ of *valued constraints*. Each valued constraint $C_i = (Y_i, f_i)$ in $\mathcal{C}$ is defined by a *scope* $Y_i \subseteq X$ and a *cost function* $f_i : \mathbf{Y}_i \to \overline{\mathbb{Q}}_+$. The *arity* of a valued constraint C_i is given by the cardinality $|Y_i|$ of its scope. If the range of f_i is $\{0, \infty\}$, then C_i is called a *hard* or *crisp* constraint. Otherwise, C_i is called a *soft* constraint, or *potential*. We mention in passing that the range of potentials may include ∞, in order to specify users' requirements. For a valued constraint C_i and an assignment $\mathbf{y} \in \mathbf{Y}$, such that $Y_i \subseteq Y$, we use $C_i(\mathbf{y})$ to denote the value of $f_i(\mathbf{y}_i)$, where $\mathbf{y}_i$ is the projection of $\mathbf{y}$ onto Y_i. The total cost of any interpretation $\mathbf{x}$ in P is given by $P(\mathbf{x}) = \sum_{i=1}^{m} C_i(\mathbf{x})$, A *feasible solution* of P is any interpretation $\mathbf{x}$ such that $P(\mathbf{x}) < \infty$, and a *minimal solution* of P is any feasible solution with minimum cost. The problem P is called *feasible* if it admits at least one feasible solution, and *infeasible* otherwise.

We shall frequently use two key operations on valued constraints: conditioning and restriction. For a cost function $f_i : \mathbf{Y}_i \to \overline{\mathbb{Q}}_+$ and a partial assignment $\mathbf{z} \in \mathbf{Z}$ (where $Z \subseteq Y_i$), the *conditioning* of f_i on $\mathbf{z}$ is the function $f_i|\mathbf{z} : \mathbf{Y}_i \to \overline{\mathbb{Q}}_+$ where $(f_i|\mathbf{z})(\mathbf{y}_i) = f_i(\mathbf{y}_i)$ if $\mathbf{y}_i$ is consistent with $\mathbf{z}$, and $(f_i|\mathbf{z})(\mathbf{y}_i) = \infty$ otherwise. By extension, the *conditioning* of a constraint $C_i = (Y_i, f_i)$ on $\mathbf{z}$ is the constraint denoted $C_i|\mathbf{z}$ with scope Y_i and function $f_i|\mathbf{z}$. For a set of constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$ and a set of variables $Y \subseteq X$, the *restriction* of $\mathcal{C}$ to Y, denoted $\mathcal{C}_Y$, is the subset of constraints $\{C_i \in \mathcal{C} \mid Y_i \subseteq Y\}$.

The main motivation of this paper is to analyze the complexity of minimization problems in which the set of crisp constraints has been compiled into a single constraint that admits a polynomial-time algorithm for deciding whether a partial assignment can be extended to a feasible solution. Specifically, we study the complexity of Boolean VCNs including a single hard constraint defined over a propositional language $\mathcal{L}$, and a set of soft constraints defined over a constraint family $\mathcal{F}$. In this setting, any VCN can be viewed as a triple (X, C, f) where C is the hard constraint and f is the cost function, represented by a set $\{C_i\}_{i=1}^{m}$ of soft constraints.

Definition 1. **MIN**$[\mathcal{L}, \mathcal{F}]$ is the following optimization problem:

- **Input:** a valued constraint network $P = (X, C, f)$, where $C \in \mathcal{L}$ and $f \in \mathcal{F}$;
- **Output:** $\text{argmin}_{\mathbf{x} \in \mathbf{X}}\, C(\mathbf{x}) + f(\mathbf{x})$, i.e. a minimal solution of P if P is feasible, and $\bot$ if P is infeasible.

3 REPRESENTING HARD CONSTRAINTS

All languages $\mathcal{L}$ examined in this study are *complete* with respect to propositional logic: any hard constraint C can be represented by a propositional circuit in $\mathcal{L}$. In the knowledge compilation literature, such languages have been classified according to their succinctness and polynomial time support for certain queries and transformations [11]. Namely, we say that a language $\mathcal{L}_1$ is *at least as succinct as* a language $\mathcal{L}_2$, written $\mathcal{L}_1 \leq_s \mathcal{L}_2$, if there is a polynomial p such that every formula C_2 of $\mathcal{L}_2$ has an $\mathcal{L}_1$ equivalent C_1, satisfying $|C_1| \leq p(|C_2|)$, where the size $|C|$ of a constraint C expressed as a propositional circuit is given by the number of its arcs. We also say that $\mathcal{L}_1$ is *(strictly) more succinct* than $\mathcal{L}_2$, denoted $\mathcal{L}_1 <_s \mathcal{L}_2$, if $\mathcal{L}_1$

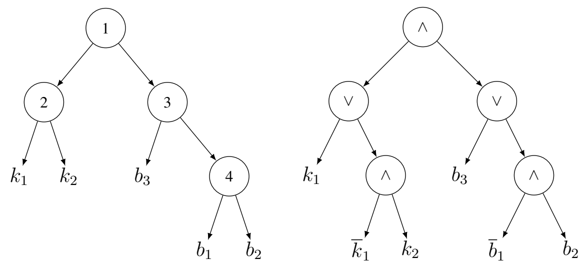

Figure 1: A vtree T (left) and a $\mathtt{DNNF}_T$ formula (right).

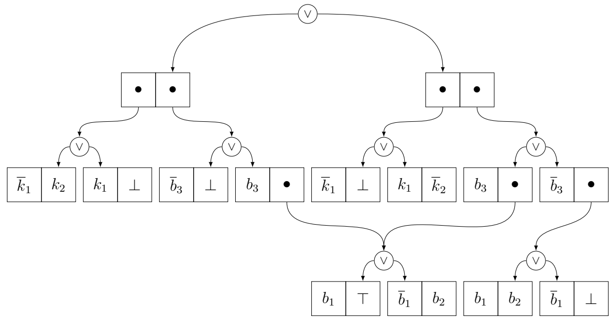

Figure 2: An $\mathtt{SDD}_T$ for housing configuration.

is as at least as succinct as $\mathcal{L}_2$, but the converse is not true. Queries are used to extract information from a circuit without modifying it, while transformations are used to generate a new circuit from one or several circuits. All languages $\mathcal{L}$ of interest here satisfy the query $\mathbf{CO}$, which checks the satisfiability (i.e., the existence of a feasible solution) of any circuit $C \in \mathcal{L}$, and the transformation $\mathbf{CD}$, which maps any circuit $C \in \mathcal{L}$ and any partial assignment $\mathbf{y}$ to an equivalent in $\mathcal{L}$ of $C|\mathbf{y}$, the conditioning of C on $\mathbf{y}$. By satisfying both properties, $\mathcal{L}$ admits a polynomial-time algorithm for deciding whether a partial assignment can be extended, or not, to a feasible solution.

Recall that NNF is the set of rooted, directed acyclic graphs (DAGs) where each leaf node is labeled with $\perp$, $\top$, or a literal over X, and each internal node is labeled with $\wedge$ or $\vee$. For a node N_i in the constraint C, we use $\mathit{Var}(N_i)$ to denote the set of variables labeling the leaf nodes reachable from N_i. We also use $\mathit{Sol}(N_i)$ to denote the set of all assignments $\mathbf{y}_i \in \mathbf{Y}_i$, such that $C_i(\mathbf{y}_i) \neq \infty$, where $Y_i = \mathit{Var}(N_i)$ and C_i is the NNF circuit rooted at N_i. By extension, we write $\mathit{Var}(C)$ (resp. $\mathit{Sol}(C)$) as an abbreviation of $\mathit{Var}(N)$ (resp. $\mathit{Sol}(N)$), where N is the root of C.

The language NNF can be refined by adding conditions to the nodes of the circuits. Of particular interest is the sub-language of *decomposable* NNF formulae [8], defined as follows:

Definition 2 (DNNF). An NNF circuit C is called *decomposable* if for every and-node N in C with children $N_1, \ldots, N_q$, we have $\mathit{Var}(N_i) \cap \mathit{Var}(N_j) = \varnothing$, for all $i, j \in [q]$ with $i \neq j$. The set of all decomposable NNF formulae is denoted DNNF.

DNNF can, in turn, be refined according to structural restrictions over its nodes. To this end, we use the standard notion of vtree introduced in [28]. Formally, a *vtree* is a full, rooted binary tree T whose leaves are in one-to-one correspondence with the variables in X. For a node t in a vtree T, we use $\mathit{Var}(t)$ to denote the set of leaves in the subtree of T rooted at t. We also use t_l and t_r to denote its left child and right child, respectively. A vtree node t is called a *Shannon node* if its left child is a leaf, and a *decomposition node* otherwise. A vtree T is *right-linear* if all its internal nodes are Shannon nodes.

An internal node N_i of an NNF circuit is said to *respect* a vtree T if there is a vtree node $t \in T$, such that $\mathit{Var}(N_i) \subseteq \mathit{Var}(t_l)$ or $\mathit{Var}(N_i) \subseteq \mathit{Var}(t_r)$ for all children N_i of N.

Definition 3 (SDNNF). For a vtree T, $\mathtt{DNNF}_T$ is the set of DNNF circuits for which all and-nodes respect T. The class SDNNF of *structured* DNNF circuits is given by the union of $\mathtt{DNNF}_T$ languages defined over all vtrees T.

All SDNNF circuits considered in this study are defined over *binary* and-nodes. This is not a severe restriction since any SDNNF formula C can be transformed into an equivalent SDNNF formula of size linear in $|C|$, where all and-nodes have exactly two children. By analogy with the terminology used in vtrees, any and-node N_i of C is called a *Shannon node* if at most one of its children is an internal node. Otherwise N_i is called a *decomposition node*. For example, the $\mathtt{DNNF}_T$ circuit of Figure 1 includes one decomposition node (the root), and two Shannon nodes $\overline{k}_1 \wedge k_2$, and $\overline{b}_1 \wedge b_2$.

Two well-known subclasses of SDNNF are the language SDD of *sentential decision diagrams* [10] and the language OBDD of *ordered binary decision diagrams* [3]. We use the term *box* to define any and-node with exactly two children p and s, respectively called *prime* and *sub*, each labeled by a constant, or a literal, or an or-node. An or-node N with children $N_1, \ldots, N_m$ is called a *partition node* if there is a partition $\{Y, Z\}$ of $\mathit{Var}(N)$ such that (i) each child N_i is a box $p_i \wedge s_i$ with $\mathit{Var}(p_i) = Y$ and $\mathit{Var}(s_i) \subseteq Z$, (ii) the primes of any pair of children are mutually exclusive, i.e., $\mathit{Sol}(p_i) \cap \mathit{Sol}(p_j) = \varnothing$ for any $i, j \in [m]$ with $i \neq j$, and (iii) the disjunction of all primes is valid, i.e., $\mathit{Sol}(p_1) \cup \cdots \cup \mathit{Sol}(p_m) = \mathbf{Y}$.

Definition 4 (SDD). For a vtree T, $\mathtt{SDD}_T$ is the language of $\mathtt{DNNF}_T$ circuits rooted at an or-node, such that every and-node is a box respecting T, and every or-node is a partition node. SDD is the union of $\mathtt{SDD}_T$ languages defined over all vtrees T.

Example 1. In Figure 2 is illustrated an $\mathtt{SDD}_T$ formula C where T is the vtree of Figure 1. C represents the hard constraints of a configurable housing application, where k_1 and k_2 capture the type of kitchens (standard or luxury), and b_1, b_2 and b_3 specify the type of bedrooms (small, medium, or large). Any feasible house includes only one kitchen and at least two bedrooms; furthermore, only houses with at least one large bedroom come with luxury kitchens. Formally, C encodes the constraints $k_1 + k_2 = 1$, $b_1 + b_2 + b_3 \geq 2$ and $\overline{k}_2 \vee b_3$.

Based on these notions, $\mathtt{OBDD}_T$ circuits can be viewed as sentential decision diagrams defined over right-linear vtrees [10].

Definition 5 (OBDD). OBDD is the union of $\mathtt{SDD}_T$ languages defined over all right-linear vtrees T.

All aforementioned languages can be further restricted by establishing conditions on the *arcs* of the circuit:

Definition 6 (acy−NNF). An NNF circuit is called *strongly acyclic* if the undirected graph of its DAG is acyclic. acy−NNF is the class of all strongly acyclic NNF circuits.

By extension, $\mathtt{acy{-}DNNF}_T$ is given by acy−NNF $\cap$ $\mathtt{DNNF}_T$, and acy−SDNNF is the union of $\mathtt{acy{-}DNNF}_T$ languages defined over all vtrees T. Recall that the class DNF of disjunctive normal form formulae is a complete propositional language. Since any DNF formula can be represented in linear time as a strongly acyclic $\mathtt{DNNF}_T$ formula defined over any arbitrary vtree T, it follows that $\texttt{acy-DNNF}_T$, and hence acy-SDNNF, are complete propositional languages.

In light of the above definitions, we can observe that both SDD and acy-SDNNF are subsets of SDNNF. One might be tempted to believe that SDD is strictly more succinct than acy-SDNNF, due to the fact that SDD circuits are DAGs and acy-SDNNF circuits are trees. But this cannot be the case, unless the polynomial hierarchy collapses. Indeed, by [10] we know that d-DNNF $\leq_s$ SDD, where d-DNNF is the class of deterministic DNNF formulae. We also know that $\texttt{acy-DNNF}_T \leq_s$ DNF since, as indicated above, any DNF formula can be rewritten in linear time as a strongly acyclic $\texttt{DNNF}_T$ circuit (whatever T). So we cannot have SDD $\leq_s$ acy-SDNNF, because otherwise we would also derive that d-DNNF $\leq_s$ DNF, which is not possible unless the polynomial hierarchy collapses [11].

Thus, from the viewpoint of succinctness, both languages SDD and acy-SDNNF are relevant for compiling hard constraints. Since SDD is a superset of OBDD, it can be used, for example, to encode in polynomial time cardinality constraints [14]. On the other hand, acy-SDNNF is a noteworthy fragment of DNNF which captures SDNNF formulae of *bounded* depth (by simply unfolding them). Furthermore, top-down compilation algorithms [9] can be adapted to directly generate $\texttt{acy-DNNF}_T$ circuits (by deselecting the caching operation). We can also take advantage of bottom-up compilers [28] for computing such representations, because $\texttt{acy-DNNF}_T$ satisfies the $\vee\mathbf{C}$ transformation and the $\wedge\mathbf{BC}$ transformation.

4 REPRESENTING COST FUNCTIONS

Borrowing the terminology of [4], a *valued constraint language* is a set $\mathcal{F}$ of $\overline{\mathbb{Q}}_+$-valued cost functions of possibly different arities. In this study, we use the term *valued constraint family* for referring to $\mathcal{F}$, in order to avoid any confusion with the notion of "constraint language" $\mathcal{L}$ already used to describe hard constraints.

We focus on valued constraint families $\mathcal{F}$ which are closed under addition and conditioning, namely, (i) if f and g are two functions in $\mathcal{F}$, then $f + g \in \mathcal{F}$, and (ii) if $f : \mathbf{X} \to \overline{\mathbb{Q}}_+$ is a function in $\mathcal{F}$, and $\mathbf{z} \in \mathbf{Z}$ is a partial assignment over $Z \subseteq X$, then $f|\mathbf{z} \in \mathcal{F}$.

Various constraint families satisfy these conditions. Notably, let $\texttt{POLY}_k$ be the set of all functions $f : \{0,1\}^j \to \overline{\mathbb{Q}}_+$ of arity $j \in [k]$, and let POLY be the union of all languages $\texttt{POLY}_k$ for $k \in \mathbb{N}$. Any subset $\mathcal{F}$ of POLY is called a *polynomial constraint family*. In particular, $\texttt{POLY}_1$ and $\texttt{POLY}_2$ respectively denote the *linear* family and the *quadratic* family. As usual, soft constraints whose cost function is in $\texttt{POLY}_k$ can be described by weighted polynomials of degree k, that is, weighted sums of canonical terms. For example, if $Y_i = \{x_1, x_2\}$ and f_i is given by the table $\{(00,1),(01,2),(10,3),(11,0)\}$ then the valued binary constraint $C_i = (Y_i, f_i)$ can be represented by the weighted polynomial $(\overline{x}_1 \wedge \overline{x}_2) + 2(\overline{x}_1 \wedge x_2) + 3(x_1 \wedge \overline{x}_2)$.

As emphasized in the introduction of this paper, submodular cost functions take a key part in nonlinear optimization. Formally, a function $f : \mathbf{X} \to \overline{\mathbb{Q}}_+$ is submodular if for all $\mathbf{x}, \mathbf{y} \in \mathbf{X}$ we have $f(\mathbf{x}\wedge\mathbf{y}) + f(\mathbf{x}\vee\mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$, where $\wedge$ and $\vee$ are respectively the bitwise-and operator and the bitwise-or operator over Boolean assignments. It is easy to see that submodularity is preserved under conditioning. It is also worth to recall that any linear function is submodular but the converse is not true. For example, the weighted disjunction $f(\mathbf{x}) = w(x_1 \vee \cdots \vee x_k)$, where $w \in \mathbb{Q}_+$, is a submodular function that cannot be expressed as a linear function. Similarly, the budget function $f(\mathbf{x}) = \min(r, w_1x_1 + \cdots + w_kx_k)$, where $r \in \mathbb{Q}_+$ and $\mathbf{w} \in \mathbb{Q}_+^k$, is submodular but not linear. Let $\texttt{SUB}_k$ be the subset of $\texttt{POLY}_k$ formed by all submodular functions of arity at most k, and let SUB be the union of all $\texttt{SUB}_k$ for $k \in \mathbb{N}$. Any subset $\mathcal{F}$ of SUB is called a *submodular constraint family*. As before, soft constraints defined over $\texttt{SUB}_k$ can be represented by weighted polynomials. For potentials $C_i = (Y_i, f_i)$ defined over the general class SUB, f_i is typically accessed through a *value oracle*, that is, a polynomial time algorithm that maps any input $\mathbf{y}_i \in \mathbf{Y}_i$ to $f(\mathbf{y}_i)$.

Although it is well-known that all polynomial cost functions are expressible by sums of potentials from $\texttt{POLY}_2$ [2], it is also known that arbitrary submodular cost functions are not, in general, expressible by $\texttt{SUB}_2$ [38]. Still, the family $\texttt{SUB}_2$ is very attractive from a computational viewpoint: as quadratic submodular functions can be encoded by cuts in a directed graph, unconstrained minimization over $\texttt{SUB}_2$ can be done in $\mathcal{O}(n^3)$ time, while the current fastest polynomial algorithms for unconstrained minimization over SUB take $\mathcal{O}(n^5\, VO + n^6)$ time, where VO is the time to run the value oracle [23]. Furthermore, several fragments of SUB (e.g., cubic submodular polynomials), can be expressed by $\texttt{SUB}_2$ [38].

For a valued constraint family $\mathcal{F}$ and a cost function f represented by a set $\{(Y_i, f_i)\}_{i=1}^m$ of soft constraints, we say that f *belongs to* $\mathcal{F}$, and write $f \in \mathcal{F}$, if $f_i \in \mathcal{F}$ for all $i \in [m]$. The *size* of f is defined as $|f| = \sum_{i=1}^m |f_i|$. For polynomial families $\mathcal{F} \subseteq \texttt{POLY}_k$, the size $|f_i|$ of each cost function f_i is given by the number of canonical terms in its weighted polynomial representation.

With each cost function f represented by a set $\{(Y_i, f_i)\}_{i=1}^m$ of soft constraints, we associate a hypergraph $\mathcal{H}_f$ capturing the *structure* of f. The set of vertices of $\mathcal{H}_f$ is $Var(f) = \bigcup_{i=1}^m Y_i$, and the set of hyperedges of $\mathcal{H}_f$ is $\{Y_i\}_{i=1}^m$. The size $|\mathcal{H}_f|$ of $\mathcal{H}_f$ is given by the sum of sizes of its hyperedges, that is, $|H_f| = \sum_{i=1}^m |Y_i|$.

In general, a constraint family $\mathcal{F}$ imposes very few restrictions on the structure of a cost function $f \in \mathcal{F}$. For example, if $\mathcal{F}$ is the class $\texttt{SUB}_2$, then $\mathcal{H}_f$ can "a priori" be any subgraph of the complete graph over $Var(f)$. In order to highlight the structural relationships between the cost function f and the hard constraint C, we need further definitions that capture the similarity (or dissimilarity) between the structure of f and the structure C.

Definition 7 (Structural Compatibility). Let C be an SDNNF circuit with $Var(C) \subseteq X$, and let Y be a subset of X. Then,

- Y is *compatible with a decomposition (and-)node* $N = N_l \wedge N_r$ of C if $Y \cap Var(N) \neq \varnothing$ implies that either $Y \subseteq Var(N_l)$ or $Y \subseteq Var(N_r)$, but not both.
- Y is *compatible with an or-node* N of C if $Y \cap Var(N) \neq \varnothing$ implies $Y \subseteq Var(N)$.

We say that Y is *weakly* compatible with C, denoted $Y \sim C$, if Y is compatible with every decomposition node of C. Alternatively, Y is *strongly* compatible with C, denoted $Y \sim^* C$, if Y is compatible with every decomposition node and every or-node of C. By extension, a cost function f is *weakly* (resp. *strongly*) *compatible* with C, if $Y_i \sim C$ (resp. $Y_i \sim^* C$) for every scope $Y_i \in \mathcal{H}_f$.

Intuitively, the weak compatibility property states that if the scope Y_i of some potential (Y_i, f_i) shares variables with a decomposition node $N = N_l \wedge N_r$, then Y_i must be covered by the variables of *exactly one* child of N. The strong compatibility property additionally states that if Y_i shares variables with an or-node N, then Y_i must be covered by *all* variables in N. To this point, we can easily see that if Y_i is a singleton set, then it is guaranteed to be strongly compatible with any node of C. Note that these compatibility properties do not imply any restriction on the Shannon and-nodes of C. The derived dissimilarity measure defined below is thus independent of the number of Shannon nodes in the hard constraint.

Example 2. Using again the housing configuration scenario of Example 1, consider the cost function $f(\mathbf{k},\mathbf{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2 + q_3 b_3)$, such that $p, r \in \mathbb{Q}_+$ and $q \in \mathbb{Q}_+^3$ are cost values; the first term of f is a weighted disjunction indicating that the same cost p is assigned to any nonempty set of kitchens, and the second term of f is a budget potential indicating that a maximum penalty r is assigned to the prices of bedrooms. Based on the above terminology, f is cubic submodular, and strongly compatible with the acy-DNNF_T constraint of Figure 1. However, f is not strongly compatible with the SDD constraint of Figure 2, since the scope $\{b_1, b_2, b_3\}$ associated with the second term in f is not compatible with, for instance, the or-node $b_1 \vee \bar{b}_1 b_2$. On the other hand, the quadratic submodular cost function $g(\mathbf{k},\mathbf{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2) + q_3 b_3$ is strongly compatible with this SDD constraint.

Definition 8 (Structural Dissimilarity). Let C be an SDNNF constraint with $Var(C) \subseteq X$, and let Y be a subset of X. Then,

- the *weak* (resp. *strong*) *dissimilarity* between Y and C, denoted $\delta(Y, C)$ resp. $\delta^*(Y, C)$), is the minimum number of variables that must be removed from Y in order to yield a subset that is weakly (resp. strongly) compatible with C:

$$\delta(Y, C) = \min_{Z \subseteq Y \mid Y \setminus Z \sim C} |Z|, \quad \text{and} \quad \delta^*(Y, C) = \min_{Z \subseteq Y \mid Y \setminus Z \sim^* C} |Z|$$

- By extension, the *weak* (resp. *strong*) *dissimilarity* between a cost function f and C is the sum of the weak (resp. strong) dissimilarities between each of its scopes and C:

$$\delta(f, C) = \sum_{Y \in \mathcal{H}_f} \delta(Y, C), \quad \text{and} \quad \delta^*(f, C) = \sum_{Y \in \mathcal{H}_f} \delta^*(Y, C)$$

For example, the strong dissimilarity between the cost function $f(\mathbf{k},\mathbf{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2 + q_3 b_3)$ and the SDNNF constraint C of Figure 1 is 1. The proof of the next proposition is left in Appendix.

Proposition 1. *Let C be an SDNNF constraint with $Var(C) \subseteq X$, and f be a cost function with $Var(f) \subseteq X$. Then $\delta(f, C)$ and $\delta^*(f, C)$ can both be evaluated in $\mathcal{O}(|C|\,|H_f|)$ time.*

5 COMPLEXITY RESULTS

As mentioned in the introduction, the constrained minimization problem **MIN**[DNNF, POLY_1] is solvable in linear time [12]. Our complexity results can be summarized by three key theorems which establish the fixed-parameter tractability of the minimization query **MIN**[$\mathcal{L}, \mathcal{F}$] for several DNNF languages $\mathcal{L}$ and families $\mathcal{F}$ of nonlinear cost functions. In what follows, we assume that the input cost function f is defined over the whole set of variables X. This is not an important restriction since any f with $Var(f) \subseteq X$ can be extended to X by adding to f "zero potentials" of the form $(\{x_i\}, 0)$, for $x_i \in X \setminus Var(f)$, where 0 is the zero constant function.

5.1 Quadratic Minimization under DNNF

We start by examining the problem of minimizing quadratic cost functions under DNNF_T constraints. Recall here that the problem of minimizing any sum of quadratic potentials subject to $C = \top$ is NP-hard [35]. The next result states that quadratic minimization subject to DNNF is fixed-parameter tractable with respect to the number of (valued) binary constraints in the cost function.

Theorem 1. **MIN**[DNNF, POLY_2] *is* FPT *with respect to the number k of binary scopes in $\mathcal{H}_f$.*

Proof Given a quadratic cost function f, let Y be the set of variables $\bigcup\{Y_i \in H_f : |Y_i| = 2\}$, and $Z = X \setminus Y$. Let g and h denote the restrictions of f to Y and Z, respectively. We thus have $f = g + h$, where g is defined over quadratic (and possibly linear) potentials, and h is only defined over linear potentials. Now, consider any DNNF circuit C over X. Recall that $C(\mathbf{x}) = 0$ if $\mathbf{x} \in Sol(C)$, and $C(\mathbf{x}) = \infty$ otherwise. Furthermore, since $\{Var(g), Var(h)\}$ is a bipartition of X, then using $Y = Var(g)$ and $Z = Var(h)$, it follows that

$$\begin{aligned}
\min_{\mathbf{x} \in \mathbf{X}} \big(f(\mathbf{x}) + C(\mathbf{x})\big) &= \min_{\mathbf{y} \in \mathbf{Y}} \min_{\mathbf{z} \in \mathbf{Z}} \big[g(\mathbf{y}) + h(\mathbf{z}) + C(\mathbf{yz})\big] \\
&= \min_{\mathbf{y} \in \mathbf{Y}} D(\mathbf{y}), \text{ where} \\
D(\mathbf{y}) &= \min_{\mathbf{z} \in \mathbf{Z}} \big[g(\mathbf{y}) + h(\mathbf{z}) + (C|\mathbf{y})(\mathbf{z})\big]
\end{aligned}$$

In the last equality, we used the fact that $C|\mathbf{y} = C(\mathbf{yz})$, where $\mathbf{yz}$ is the concatenation of $\mathbf{y}$ and $\mathbf{z}$.

For any given assignment $\mathbf{y} \in \mathbf{Y}$, the expression $g(\mathbf{y})$ is constant, and hence, $g(\mathbf{y}) + h(\mathbf{z})$ is linear. Moreover, since the conditioning operation (**CD**) can be performed in linear time for the class DNNF [8], the constraint C can be transformed in $\mathcal{O}(|C|)$ time into a DNNF circuit that is equivalent to $C|\mathbf{y}$. This, together with the fact that linear minimization under DNNF can be done in linear time [12], implies that $D(\mathbf{y})$ can be evaluated in $\mathcal{O}(|C|)$ time. Finally, since the number of binary scopes is k, we have $|Y| \leq 2k$, and hence, $|\mathbf{Y}| \leq 4^k$. Therefore, $\min_{\mathbf{y} \in \mathbf{Y}} D(\mathbf{y})$ can be evaluated in $\mathcal{O}(4^k\,|C|)$ time, implying that **MIN**[DNNF, POLY_2] is FPT with respect to k. □

5.2 Submodular Minimization under acy-SDNNF

We now focus on submodular cost functions, and begin with a negative result indicating that even for strongly acyclic (structured) DNNF constraints, the minimization problem is hard.

Proposition 2. **MIN**[acy-SDNNF, SUB] *is* NP*-hard.*

Proof An instance of the Switching Submodular Function Minimization (SSFM) problem [22] consists of two sets $Y = \{y_1, \ldots, y_q\}$ and $Y' = \{y'_1, \ldots, y'_q\}$, and a submodular cost function $f : 2^{Y \cup Y'} \to \mathbb{Q}_+$. Let $\pi : 2^Y \to 2^{Y'}$ be the one-to-one mapping defined by $\pi(Z) = \{y'_i \in Y' \mid y_i \in Z\}$. The problem is to find a bipartition $\{Z_1, Z_2\}$ of Y that minimizes $f(Z_1 \cup \pi(Z_2))$. Let $X = Y \cup Y'$, $p = 2q$, and consider the set of constraints $C = \{y'_i \leftrightarrow \overline{y}_i \mid i \in [q]\}$. For an assignment $\mathbf{x} \in \mathbf{X}$, and a subset $V \subseteq X$, let $Set_V(\mathbf{x})$ be the set of variables in V which are mapped to 1 in $\mathbf{x}$. Based on this notation, $\mathbf{x}$ satisfies all constraints in C if and only if $\{Set_Y(\mathbf{x}), \pi^{-1}(Set_{Y'}(\mathbf{x}))\}$ is a bipartition of Y.

Now, observe that C is a decomposable conjunction of DNF formulae of the form $(y_i \wedge \overline{y'}_i) \vee (\overline{y}_i \wedge y'_i)$. So, C can be encoded into an acy-DNNF_T formula over a vtree T with one Shannon node per index $i \in [q]$, and $q - 1$ decomposition nodes joining those Shannon nodes. Finally, since f is submodular, the SSFM instance (Y, Y', f) can be converted in polynomial time into an equivalent instance of **MIN**[acy-SDNNF, SUB]. This, together with the fact that SSFM is NP-hard, yields the result. □

We now show that if the hard constraint C is in acy-DNNF_T, and if the submodular cost function f is weakly compatible with the vtree T, then the task of minimizing f under C is in P. In order to

Algorithm 1: TDM: Top Down Minimization

Input: A submodular cost function f and an SDNNF circuit C rooted at node N

Output: An assignment $\mathbf{x} \in \mathbf{X}$ that minimizes f if C is consistent, and $\perp$ otherwise

1 **if** $N = \perp$ **then** return $\perp$
2 **if** $N = \top$ **then** return $\text{argmin}_{\mathbf{y} \in \mathbf{Y}} f(\mathbf{y})$, where $Y = Var(f)$
3 **if** $N = \ell$ **then**
4   return $\text{argmin}_{\mathbf{y} \in \mathbf{Y}} f|_{\ell}(\mathbf{y})$, where $Y = Var(f)$
5 **if** $N = \ell \wedge N'$ is a Shannon node **then**
6   return TDM$(f|_{\ell}, N')$
7 **if** $N = N_1 \wedge N_2$ is a decomposition node **then**
8     $\mathbf{y}_0 \leftarrow$ TDM$(f_{Var(f) \setminus Var(N)}, \top)$
9     $\mathbf{y}_i \leftarrow$ TDM$(f_{Var(N_i)}, N_i)$ for each $i \in [2]$
10   return $\mathbf{y}_0 \wedge \mathbf{y}_1 \wedge \mathbf{y}_2$
11 **if** $N = N_1 \vee \ldots \vee N_q$ **then**
12     $\mathbf{y}_i \leftarrow$ TDM(f, N_i) for each $i \in [q]$
13   return $\text{argmin}\{f(\mathbf{y}_i)\}$

prove this result, we use a top-down minimization algorithm (TDM), which iteratively decomposes the cost function f over the nodes of the acyclic SDNNF constraint C. Recall here that $f|_{\ell}$ is the conditioning of f by the literal (or unary partial assignment) ℓ, and f_Y is the restriction of f to the set of variables Y.

Example 3. To illustrate the behavior of the TDM algorithm, consider the strongly acyclic DNNF_T constraint C given in Figure 1, together with $f(\mathbf{k}, \mathbf{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2 + q_3 b_3)$, where $q_3 < r < q_1 < q_2$. As mentioned in Example 2, we know that f is weakly compatible with C. Since the root node N of C is a decomposition node (Line 7), the procedure recursively calls TDM on $f_l(\mathbf{k}) = p(k_1 \vee k_2)$ on the left child $N_l = k_1 \vee (\overline{k}_1 \wedge k_2)$, and $f_r(\mathbf{b}) = \min(r, q_1 b_1 + q_2 b_2 + q_3 b_3)$ on the right child $N_r = b_3 \vee (\overline{b}_1 \wedge b_2)$. For the left child N_l, which is an or-node (Line 11), the procedure recursively calls TDM on $f_l(\mathbf{k})$ subject to k_1, and $f_l(\mathbf{k})$ subject to $(\overline{k}_1 \wedge k_2)$. Now, according to Line 3, the minimizer of $(f_l(\mathbf{k}))_{|k_1}$ is any term in $\{k_1 k_2, k_1 \overline{k}_2\}$ with cost p. According to Line 7, the minimizer of $f_l(\mathbf{k})$ subject to $(\overline{k}_1 \wedge k_2)$ is $\overline{k}_1 k_2$ with cost p. To sum up, the partial assignment returned by TDM on $f_l(\mathbf{k})$ on N_l is any term in $\{k_1 k_2, k_1 \overline{k}_2, \overline{k}_1 k_2\}$. Using similar operations, the minimizer returned by TDM on $f_r(\mathbf{b})$ subject to the constraint of the right child N_r is $\overline{b}_1 \overline{b}_2 b_3$ with cost q_3.

Proposition 3. **MIN**[acy−SDNNF, SUB] *is in* P *if the cost function f is weakly compatible with the hard constraint C.*

Proof Let C be an SDNNF constraint and $f = \{C_i\}_{i=1}^m$ be a sum of submodular potentials which are weakly compatible with C.

We first prove that TDM(f, C) returns a minimizer of f subject to C, if the minimization problem is feasible, and returns $\perp$ otherwise. To this end, we proceed by induction over the structure of the root node N of C. The base cases are straightforward. Namely, if $N = \perp$ (Line 1), then the problem is not feasible, and hence, there is no solution for f under C. If $N = \top$, then the problem is unconstrained, and hence, the solution returned at Line 2 is an unconstrained minimizer of f. If $N = \ell$ is a literal, then any minimizer of f subject to ℓ is a minimizer of $f_{|\ell}$, which is the solution returned at Line 3.

Now, if $N = \ell \wedge N'$ is a Shannon node, let C' be the constraint rooted at N'. Any minimizer of f subject to $\ell \wedge C'$ is a minimizer of $f_{|\ell}$ under C', which is the solution returned at Line 5. For the case when $N = N_1 \wedge N_2$, since all potentials in f are compatible with N, they can be partitioned into three groups, defined over $Var(N_1)$, $Var(N_2)$, and $Var(f) \setminus Var(N)$. It follows that f is the sum of three variable-disjoint functions $f_{Var(N_1)} + f_{Var(N_2)} + f_{Var(f) \setminus Var(N)}$, for which the minimization can be done as at Line 7. Finally, for the case when N is an or-node $N_1 \vee \ldots \vee N_q$, let D_i be the hard constraint rooted at N_i, and D be the disjunction $D_1 \vee \ldots \vee D_q$. Since the minimum of $f(\mathbf{x})$ subject to $D(\mathbf{x}) \neq \infty$ is equal to

$$\min \left(\min_{D_1(x) \neq \infty} f(\mathbf{x}), \cdots, \min_{D_q(\mathbf{x}) \neq \infty} f(\mathbf{x}) \right)$$

it follows that f can be minimized as done at Line 11.

Since acy−SDNNFs are rooted trees, the number of paths in C is bounded by $|C|$. So, TDM(f, C) runs polynomially many times an unconstrained minimization procedure, which is in P for SUB. □

Corollary 1. **MIN**[DNF, SUB] *is in* P.

Proof Follows from Proposition 3, using the fact that any DNF constraint can be transformed in linear time into an acy−SDNNF formula in which every and-node is a Shannon node. □

To summarize, we know that submodular minimization under acy−SDNNF is NP-hard in general, but tractable if the cost function is weakly compatible with the hard constraint. We are now in position to provide a tractable restriction of the general intractability result stated by Proposition 2, using the notion of weak dissimilarity.

Theorem 2. **MIN**[acy−SDNNF, SUB] *is* FPT *with respect to δ.*

Proof Let C be a hard constraint in acy−SDNNF, and $f = \{C_i\}_{i=1}^m$ be a cost function in SUB. For each scope $Y_i \in \mathcal{H}_f$, let Z_i be any minimal subset of Y_i such that $Y_i \setminus Z_i$ is weakly compatible with C. Let $Z = \bigcup_{i=1}^m Z_i$ and $Y = X \setminus Z$. By conditioning both C and f with partial assignments over $\mathbf{Z}$, we have

$$\min_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}) + C(\mathbf{x}) = \min_{\mathbf{z} \in \mathbf{Z}} \left(\min_{\mathbf{y} \in \mathbf{Y}} (f|\mathbf{z})(\mathbf{y}) + (C|\mathbf{z})(\mathbf{y}) \right) \tag{1}$$

Since $C|\mathbf{z}$ can be constructed in time linear in $\mathcal{O}(|C|)$, the task of minimizing $f|\mathbf{z}$ subject to $C|\mathbf{z}$ can be solved in polynomial time. Furthermore, since $|Z| \leq \delta(f, C)$, it follows that $|\mathbf{Z}| \leq 2^{\delta(f,C)}$. Therefore, Eq. 1 can be solved using $2^{\delta(f,C)}$ calls to TDM, which by Proposition 3, takes polynomial time. □

Corollary 2. **MIN**[SDNNF, SUB] *is* FPT *with respect to $d + \delta$, where d is the depth of the* SDNNF *constraint.*

Proof Follows from Theorem 2 and the fact that any SDNNF circuit C of depth d can be transformed into an acy−SDNNF circuit of size $2^d \, |C|$ by simply unfolding C. □

5.3 Submodular Minimization under SDD

The final part of this study is related to submodular minimization under SDD constraints. Again, we begin with a strong negative result indicating that constrained quadratic submodular minimization is NP-hard, even if the hard constraint is given as an OBDD.

Proposition 4. **MIN**[OBDD, SUB_2] *is* NP*-hard.*

Algorithm 2: BUM: Bottom Up Minimization

Input: A submodular cost function f and an SDD circuit C rooted at node N

Output: An assignment $\mathbf{x} \in \mathbf{X}$ that minimizes f if C is consistent, and $\perp$ otherwise

1 **foreach** *node N of C in reverse topological order* **do**
2   **if** $N = \perp$ **then** $f_N \leftarrow \{(\varnothing, \infty)\}$
3   **if** $N = \top$ **then** $f_N \leftarrow \varnothing$
4   **if** $N = \ell$ **then**
5     $f_N \leftarrow \{(Y_i, f_i|_\ell) \mid (Y_i, f_i) \in f \text{ and } Y \cap Var(\ell) \neq \varnothing\}$
6   **if** $N = N_1 \wedge N_2$ **then** $f_N \leftarrow f_{N_1} \cup f_{N_2}$
7   **if** $N = N_1 \vee \ldots \vee N_q$ **then**
8     $\mathbf{y}_i \leftarrow \text{argmin}_{\mathbf{y} \in \mathbf{Y}} f_{N_i}(\mathbf{y})$ for all $i \in [q]$
9     $f_N \leftarrow (Y, \{(\mathbf{y}, f(\mathbf{y}))\})$, where $\mathbf{y} = \text{argmin}_{i=1}^q f(\mathbf{y}_i)$
10 return $\text{argmin}_{\mathbf{x} \in \mathbf{X}} f_N(\mathbf{x}) + f_\top(\mathbf{x})$

Proof In the *minimum graph bisection (*MGB*)* problem, we are given an edge-weighted graph $G = (X, E, \mathbf{w})$, with an even number p of nodes in a set X. The cut function $f : 2^X \to \mathbb{N}$ maps any subset $Y \subseteq X$ into the sum of weights of edges with one end point in Y and one in $X \setminus Y$. The task is to find a subset Y of size $p/2$ that minimizes f. This problem, which is known to be NP-hard [17] even when all weights are equal to 1, can be reduced in polynomial time to **MIN**[OBDD, SUB$_2$]. Indeed, the cut function f is submodular, and can be encoded using the set of potentials $(\{x_i, x_j\}, (x_i \wedge \overline{x}_j) + (\overline{x}_i \wedge x_j))$ for each $\{x_i, x_j\} \in E$. Furthermore, any cardinality constraint $(\sum_{i=1}^p x_i \geq k)$ can be encoded in polynomial time into an OBDD$_<$ circuit (for any ordering $<$), using the technique of [14]. Finally, since OBDD$_<$ satisfies the $\neg\mathbf{C}$ transformation and the $\wedge\mathbf{BC}$ transformation [11], the constraint C given by $\sum_{i=1}^p x_i = p/2$ can be encoded into an OBDD$_<$ circuit, using the fact that C is equivalent to $(\sum_{i=1}^p x_i \geq p/2) \wedge \neg(\sum_{i=1}^p x_i \geq p/2 + 1)$.□

On the other hand, we can show that submodular minimization subject to SDD is in P, provided that the cost function is strongly compatible with the SDD constraint. This result is is established using a bottom-up minimization algorithm (BUM), which first computes a reverse topological order of the input SDD constraint C, and then iteratively simplifies and collects the potentials of f whose scope is covered by the current node N. We use here f_N as an abbreviation of $f_{Var(N)}$, and use $(\varnothing, \infty)$ to denote the infeasible potential, where ∞ is viewed as a constant function. Since $Var(C)$ can be a strict subset of X, f (which by assumption is defined over $Var(f) = X$) can include variables which are not present in C. Thus, a last minimization step is performed over the unconstrained sub-function $f_\top = f_{Var(f) \setminus Var(N)}$, where N is the root of C.

Proposition 5. **MIN**[SDD, SUB] *is in* P *if the input cost function f is strongly compatible with the hard constraint C.*

Proof Let C be an SDD constraint and f be a set of quadratic submodular potentials which are strongly compatible with C. We begin to show that BUM(f, C) returns a minimizer of f subject to C if C is consistent, and returns $\perp$ otherwise.

Let $(N_1, \cdots, N_r)$ of C be a reverse topological ordering of the nodes of C, where $N_r = N$ is the root. For any N_i ($i \in [r]$), let C_i be the constraint rooted at N_i. We prove by induction on each N_i of the ordering $(N_1, \cdots, N_r)$ that

$$\min_{\mathbf{y} \in \mathbf{Y}} (f_Y(\mathbf{y}) + C_i(\mathbf{y})) = \min_{\mathbf{y} \in \mathbf{Y}} f_{N_i}(\mathbf{y}) \text{ where } Y = Var(C_i) \quad (2)$$

Note that the left-hand side of Eq. 2 is a constrained minimization task, while its right-hand side is an unconstrained minimization task. The base cases where $N_i = \perp$, $N_i = \top$, and $N_i = \ell$ are straightforward. Let $N_i = N_j \wedge N_k$ be a Shannon node, where the constraint of N_j is a literal ℓ. Using $g = f_{Var(C_k)}$ and $h = f_Y \setminus g$, we have

$$\min_{\mathbf{y} \in \mathbf{Y}} f_Y(\mathbf{y}) + C_i(\mathbf{y}) = \min_{\mathbf{y} \in \mathbf{Y}} h|_\ell(\mathbf{y}) + g|_\ell(\mathbf{y}) + C_k(\mathbf{y})$$
$$= \min_{\mathbf{y} \in \mathbf{Y}} h|_\ell(\mathbf{y}) + (g(\mathbf{y}) + C_k(\mathbf{y})) = \min_{\mathbf{y} \in \mathbf{Y}} f_{N_j}(\mathbf{y}) + f_{N_k}(\mathbf{y})$$

where the second equality uses the fact that $g|_\ell = g$ (since for any and-node we must have $Var(l) \cap Var(C_k) = \varnothing$), and the last equality follows from Line 4 and induction hypothesis (IH). Using Line 6, the last expression is equal to $\min_{\mathbf{y}} f_N(\mathbf{y})$. Alternatively, suppose that $N_i = N_j \wedge N_k$ is a decomposition node, and let $g = f_{Var(C_j)}$ and $h = f_{Var(C_k)}$. By the weak compatibility property, $\{Var(C_j), Var(C_k)\}$ is a bipartition of $Var(C_i)$. So, $f = g \cup h$. By IH, it follows that

$$\min_{\mathbf{y} \in \mathbf{Y}} f_Y(\mathbf{y}) + C_i(\mathbf{y}) = \min_{\mathbf{y} \in \mathbf{Y}} (g(\mathbf{y}) + C_j(\mathbf{y})) + (h(\mathbf{y}) + C_k(\mathbf{y}))$$
$$= \min_{\mathbf{y} \in \mathbf{Y}} f_{N_j}(\mathbf{y}) + f_{N_k}(\mathbf{y})$$

which is again equal to $\min_{\mathbf{y}} f_N(\mathbf{y})$. Finally, let N_i be an or-node of the form $N_{i_1} \vee \cdots \vee N_{i_q}$. Then, $\min_{\mathbf{y} \in \mathbf{Y}} f_Y(\mathbf{y}) + C_i(\mathbf{y})$ is equal to

$$\min\left(\min_{\mathbf{y} \in \mathbf{Y}} (f_{\mathbf{y} \in \mathbf{Y}}(\mathbf{y}) + C_{i_1}(\mathbf{y})), \cdots, \min_{\mathbf{y} \in \mathbf{Y}} (f_{\mathbf{y} \in \mathbf{Y}}(\mathbf{y}) + C_{i_q}(\mathbf{y}))\right)$$

Suppose w.l.o.g. that the first p nodes of N_i are Shannon nodes. By the strong compatibility property, we know that for each $j \in [p]$, the scopes in $f_{N_{i_j}}$ (Line 4) are covered by $Var(N_i)$. So, by IH, we must have $\min_{\mathbf{y} \in \mathbf{Y}} f_{\mathbf{y} \in \mathbf{Y}}(\mathbf{y}) + C_{i_j}(\mathbf{y}) = \min_{\mathbf{y} \in \mathbf{Y}} f_{N_{i_j}}(\mathbf{y})$. By the weak compatibility property, we know that for each $j \in \{p+1, \cdots, q\}$, the scopes in $f_{N_{i_j}}$ (Line 6) are covered by $Var(C_{i_j}) \subseteq Var(C_i)$. Again, we get that $\min_{\mathbf{y} \in \mathbf{Y}} f_{\mathbf{y} \in \mathbf{Y}}(\mathbf{y}) + C_{i_j}(\mathbf{y}) = \min_{\mathbf{y} \in \mathbf{Y}} f_{N_{i_j}}(\mathbf{y})$. To sum up, it follows that

$$\min_{\mathbf{y} \in \mathbf{Y}} f_Y(\mathbf{y}) + C_i(\mathbf{y}) = \min\left(\min_{\mathbf{y} \in \mathbf{Y}} f_{N_{i_1}}(\mathbf{y}), \cdots, \min_{\mathbf{y} \in \mathbf{Y}} f_{N_{i_q}}(\mathbf{y})\right)$$

which by Line 7 is equal to $\min_{\mathbf{y}} f_N(\mathbf{y})$.

Thus, according to Eq. 2, BUM performs (at most) q unconstrained submodular minimization tasks for each or-node of the constraint C. The number of these optimization tasks is therefore bounded by $|C| + 1$, by taking into account the last step over $f_\top$ (Line 10). Since unconstrained submodular minimization is in P, the result follows.□

Example 4. Let us consider the SDD circuit at Figure 2 together with the cost function $g(\mathbf{k}, \mathbf{b}) = p(k_1 \vee k_2) + \min(r, q_1 b_1 + q_2 b_2) + q_3 b_3$, with $q_3 < q_2 < r < q_1$ and $r < q_2 + q_3$ to illustrate the algorithm BUM. The cost function associated with the box $b_1 \wedge \top$ represented at last "line" of the figure is the set consisting of the two potentials associated with its children b_1 and $\top$, that is respectively $(\{b_1, b_2\}, \min(r, q_1 b_1 + q_2 b_2) \mid q_1)$ and $\varnothing$. Concerning the second box $\neg b_1 \wedge b_2$ of the last line, the associated function is the set consisting of the two potentials $(\{b_1, b_2\}, \min(r, q_1 b_1 + q_2 b_2) \mid \bar{b}_1)$ and $(\{b_1, b_2\}, \min(r, q_1 b_1 + q_2 b_2) \mid b_2)$. Thus, the function g_N associated with the $\vee$ node which is the father of the two boxes is built up from the partial assignment of the variables of $\{b_1, b_2\}$ which minimizes $\min(\min(r, q_1 b_1 + q_2 b_2) \mid q_1, (\min(r, q_1 b_1 + q_2 b_2) \mid \bar{b}_1)$ $+ (\min(r, q_1 b_1 + q_2 b_2) \mid b_2)))$, that is $\bar{b}_1 \wedge b_2$ (remember that $q_2 < q_1$). Thus, we have $g_N = (\{b_1, b_2\}, \min(r, q_1 b_1 + q_2 b_2) \mid \bar{b}_1 b_2)$. Applying this algorithm from the leaves to the root of the SDD circuit leads to the minimal value $p + r$ for the solution $k_1 \wedge \bar{k}_2 \wedge b_1 \wedge b_2 \wedge \bar{b}_3$.

	SDNNF	acy-SDNNF	SDD	OBDD	DNF
$\texttt{POLY}_2$	k	k	k	k	k
SUB	$d+\delta$	δ	δ^*	δ^*	$-$

Table 1: Complexity parameters used in FPT results. Here, k is the number of binary scopes in the cost function, d is the depth of the hard constraint, and $-$ indicates that the problem is in P.

Theorem 3. MIN[SDD, SUB] *is* FPT *with respect to* δ^*.

Proof The result follows by mimicking the proof of Theorem 2. Using Eq. 1, where C is replaced by a hard constraint in SDD, and f by a cost function in SUB, we have $|Z| \leq \delta^*(f, C)$, which in turn implies that $|\mathbf{Z}| \leq 2^{\delta^*(f,C)}$. So, Eq. 1 can be solved using $2^{\delta^*(f,C)}$ calls to BUM, which by Proposition 5, takes polynomial time. □

6 DISCUSSION

In this paper, we have examined the complexity of minimizing quadratic functions and submodular functions, subject to DNNF constraints. The fixed parameter tractable results for these constrained optimization problems are summarized in Table 1. From a practical viewpoint, submodular minimization under SDNNFs (and all subsets of this language) can be efficiently solved if the depth d of the constraint C and the weak dissimilarity of the input query f (with respect to C) are relatively small. On the other hand, submodular minimization under SDD (and hence OBDD) constraints can be efficiently solved if the strong dissimilarity between f and C is small. The result holds here for SDD circuits of *arbitrary* depth. For quadratic submodular minimization, the query **MIN**[SDD, $\texttt{SUB}_2$] can be solved in $\mathcal{O}(|C|\, n^3 2^{\delta(f,C)})$ time using the BUM algorithm.

Related Work. Considerable effort has been made in identifying families of VCNs for which optimization is tractable. Most of the work in this research area has focused on three main approaches, depending on the type of restrictions advocated for deriving tractable cases. The first approach is to identify *structural* properties of VCNs which ensure tractability. For example, the minimization problem is in P if the macro-structure of the network has bounded (hyper)treewidth [20]. In a similar context, several knowledge compilation languages have been defined for compiling the micro-structure of a VCN into a (valued) circuit, from which optimization can be achieved in polynomial time [36, 15, 24]. It is important to emphasize that our work departs from this approach, where *both* hard constraints and soft constraints are compiled during the offline step. In our framework, soft constraints are known only *at the online step and may vary with the user*. The online performance guarantees which are sought prevent one from performing a computationally expensive compilation step each time a new cost function is considered.

The second approach is to identify *algebraic* properties of valued constraints which are sufficiently restrictive to ensure tractability, no matter how constraints are combined in the network. A complete complexity classification of valued constraint languages has been established for Boolean VCNs [4], indicating that the optimization queries are tractable only for very restricted fragments.

Our work is related to the third, *hybrid* approach which concerns both structural and language restrictions. Here, strong negative results in constrained submodular minimization indicate that structural restrictions and language restrictions cannot, in general, be considered separately. Indeed, even if hard constraints are described by a matroid for which linear optimization is in P, and the cost function is submodular, then the corresponding minimization problem is NP-hard and generally not approximable within a constant factor [18, 33]. Tractable classes have been obtained by Cooper and Zivny [5, 6], by appropriately combining restrictions over the network micro-structure and language restrictions over cost functions. Our results also exploit such forms of hybrid restriction, but cover a larger set of hard constraints which are compiled into DNNF circuits.

Perspectives. In light of the present results, an important direction of research is to consider the problem of *maximizing* submodular functions subject to DNNF constraints. While maximization and minimization are equivalent problems for valued constraint languages closed under additive inverse $(-)$, especially for the language of linear cost functions, this is not the case for submodular languages in general. Notably, the problem of (monotone) submodular maximization is NP-hard, but approximable within a constant ratio in the unconstrained case. A key open question is to determine whether such good approximation bounds are preserved under DNNF constraints.

APPENDIX

Proof (of Proposition 1) We first consider $\delta^*(f, C)$. Let Y be an arbitrary subset of X, and let $(N_1, \cdots, N_r)$ be a reverse topological order of the nodes of C. With each N_i in the ordering, we associate a *blocking set* $B(N_i) \subseteq Y$, recursively defined as follows:

1. if N_i is a leaf, then $B(N_i) = \varnothing$;
2. if $N_i = \ell \wedge N_j$ is a Shannon node, then $B(N_i) = B(N_j)$;
3. if $N_i = N_j \wedge N_k$ is a decomposition node, then $B(N_i) = B(N_j) \cup B(N_k) \cup U$, where U is any set of minimal size taken from $\{Y \cap \mathit{Var}(N_j), Y \cap \mathit{Var}(N_k)\}$ if Y is not compatible with N_i, and $U = \varnothing$ otherwise;
4. if $N_i = \bigvee_{j=1}^q N_{i_j}$ is an or-node, then $B(N_i) = \bigcup_{j=1}^q B(N_{i_j}) \cup U$, where $U = Y \setminus B(N_i)$ if Y is not compatible with N_i, and $U = \varnothing$ otherwise.

Obviously, the final set $B(N_r)$ can be obtained in $O(|C|\,|Y|)$ time. Now, we show by induction on the ordering $(N_1, \cdots, N_r)$ that $|B(N_i)| = \delta^*(Y, C_i)$, where C_i is the hard constraint rooted at N_i.

- If N_i is a leaf, then Y is always compatible with N_i, and hence $|B(N_i)| = 0 = \delta^*(Y, C_i)$.
- Similarly, if $N_i = \ell \wedge N_j$ is a Shannon node, then by induction hypothesis (IH) we know that $B(N_j)$ is a blocking set of minimal size for N_j. Since Y is compatible with N_i, it follows that $|B(N_i)| = |B(N_j)| = \delta^*(Y, C_i)$.
- If $N_i = N_j \wedge N_k$ is a decomposition node, then we known by IH that $B(N_j)$ (resp. $B(N_k)$) is a blocking set of minimal size for N_j (resp. N_k). If Y is compatible with N_i, then by taking $B(N_i) = B(N_j) \cup B(N_k)$, it follows that $|B(N_i)| = \delta^*(Y, C_i)$, because $Y \setminus B(N_i)$ is the largest subset of Y strongly compatible with N_i. If Y is not compatible with N_i, we must remove from Y exactly one set between $U_j = Y \cap \mathit{Var}(N_j)$ and $U_k = Y \cap \mathit{Var}(N_k)$. Suppose w.l.o.g. that $|U_j| \leq |U_k|$. By taking $B(N_i) = B(N_j) \cup B(N_k) \cup U_j$, we also have $|B(N_i)| = \delta^*(Y, C_i)$, since $Y \setminus B(N_i)$ is a largest subset of Y that is compatible with C_i.
- If $N_i = \bigvee_{j=1}^q N_{i_j}$ is an or-node, let $V = \bigcup_{j=1}^q B(N_{i_j})$. We know that Y must be compatible with all children of N_i. So, if Y is compatible with N_i, then by IH $|B(N_i)| = |V| = \delta^*(Y, C_i)$. If Y is not compatible with N_i, then we must remove $U = Y \setminus \mathit{Var}(N_i)$ from Y. Therefore, $|B(N_i)| = |U \cup V| = \delta^*(Y, C_i)$.

By summing over all $Y_i \in \mathcal{H}_f$, we get the desired result. The case for $\delta(f, C)$ is similar by simply replacing Rule 4 by $B(N_i) = \bigcup_{j=1}^q B(N_{i_j})$, since f is always weakly compatible with or-nodes. □

REFERENCES

[1] F. Bacchus and A. Grove, 'Graphical models for preference and utility', in *Proc. of UAI '95*, pp. 3–10, (1995).

[2] E. Boros and P. Hammer, 'Pseudo-Boolean Optimization', *Discrete Applied Mathematics*, **123**(1), 155–225, (2002).

[3] R. Bryant, 'Graph-based algorithms for Boolean function manipulation', *IEEE Transactions on Computers*, **C-35**(8), 677–691, (1986).

[4] D. A. Cohen, M. C. Cooper, P. Jeavons, and A. A. Krokhin, 'The complexity of soft constraint satisfaction', *Artif. Intell.*, **170**(11), 983–1016, (2006).

[5] M. Cooper and S. Zivny, 'Hybrid tractability of valued constraint problems', *Artif. Intell.*, **175**(9-10), 1555–1569, (2011).

[6] M. Cooper and S. Zivny, 'Tractable triangles and cross-free convexity in discrete optimisation', *J. Artif. Intell. Res. (JAIR)*, **44**, 455–490, (2012).

[7] N. Creignou, S. Khanna, and M. Suda, *Complexity Classifications of Boolean Constraint Satisfaction Problems*, Monographs on Discrete Mathematics and Applications, SIAM, 2001.

[8] A. Darwiche, 'Decomposable negation normal form', *Journal of the ACM*, **48**(4), 608–647, (2001).

[9] A. Darwiche, 'A compiler for deterministic, decomposable negation normal form', in *Proc. of AAAI*, pp. 627–634, (2002).

[10] A. Darwiche, 'SDD: A new canonical representation of propositional knowledge bases', in *Proc. of IJCAI*, pp. 819–826, (2011).

[11] A. Darwiche and P. Marquis, 'A knowledge compilation map', *J. Artif. Intell. Res. (JAIR)*, **17**, 229–264, (2002).

[12] A. Darwiche and P. Marquis, 'Compiling propositional weighted bases', *Artif. Intell.*, **157**(1-2), 81–113, (2004).

[13] G. Downey and R. Fellows, *Fundamentals of Parameterized Complexity*, Springer, 2013.

[14] N. Eén and N. Sörensson, 'Translating pseudo-boolean constraints into SAT', *JSAT*, **2**(1-4), 1–26, (2006).

[15] H. Fargier and P. Marquis, 'On valued negation normal form formulas', in *Proc. of IJCAI*, ed., Manuela M. Veloso, pp. 360–365, (2007).

[16] S. Fujishige, *Submodular Functions and Optimization*, Elsevier, 2005.

[17] M.R. Garey and D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, 1979.

[18] G. Goel, C. Karande, P. Tripathi, and L. Wang, 'Approximability of combinatorial problems with multi-agent submodular cost functions', in *Proc. of FOCS*, pp. 755–764, (2009).

[19] Ch. Gonzales and P. Perny, 'GAI networks for utility elicitation', in *Proc. of KR*, pp. 224–234, (2004).

[20] G. Gottlob, G. Greco, and F. Scarcello, 'Tractable optimization problems through hypergraph-based structural restrictions', in *Proc. of ICALP*, pp. 16–30, (2009).

[21] S. Iwata, 'Submodular function minimization', *Math. Program.*, **112**(1), 45–64, (2008).

[22] S. Iwata and K. Nagano, 'Submodular function minimization under covering constraints', in *Proc. of FOCS*, pp. 671–680, (2009).

[23] S. Iwata and J. Orlin, 'A simple combinatorial algorithm for submodular function minimization', in *Proc. of SODA*, pp. 1230–1237, (2009).

[24] G. Katsirelos, N. Narodytska, and T. Walsh, 'The weighted grammar constraint', *Annals OR*, **184**(1), 179–207, (2011).

[25] D. Koller and N. Friedman, *Probabilistic Graphical Models*, MIT Press, 2009.

[26] V. Kolmogorov, J. Thapper, and S. Zivny, 'The power of linear programming for general-valued CSPs', *SIAM J. Comput.*, **44**(1), 1–36, (2015).

[27] C. Muise, S. McIlraith, J. Beck, and E. Hsu, 'Dsharp: Fast d-DNNF compilation with sharpSAT', in *Proc. of Canadian Conf. on AI*, pp. 356–361, (2012).

[28] K. Pipatsrisawat and A. Darwiche, 'New compilation languages based on structured decomposability', in *Proc. of AAAI*, pp. 517–522, (2008).

[29] T. Schiex, H. Fargier, and G. Verfaillie, 'Valued constraint satisfaction problems: Hard and easy problems', in *Proc. of IJCAI*, pp. 631–639, (1995).

[30] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, volume B, Springer, 2003.

[31] C. Sutton and A. McCallum, 'An introduction to conditional random fields', *Foundations and Trends in Machine Learning*, **4**(4), 267–373, (2012).

[32] Z. Svikina and E. Tardos, 'Min-max multiway cut', in *Proc. of APPROX*, pp. 207–218, (2004).

[33] Z. Svitkina and L. Fleischer, 'Submodular approximation: Sampling-based algorithms and lower bounds', *SIAM J. Comput.*, **40**(6), 1715–1737, (2011).

[34] J. Thapper and S. Zivny, 'Necessary conditions for tractability of valued csps', *SIAM J. Discrete Math.*, **29**(4), 2361–2384, (2015).

[35] J. Uckelman, Y. Chevaleyre, U. Endriss, and J. Lang, 'Representing utility functions via weighted goals', *Mathematical Logic Quaterly*, **55**(4), 341–361, (2009).

[36] N. Wilson, 'Decision diagrams for the computation of semiring valuations', in *Proc. of IJCAI*, pp. 331–336, (2005).

[37] S. Zivny, *The Complexity of Valued Constraint Satisfaction Problems*, Cognitive Technologies, 2012.

[38] S. Zivny, D. Cohen, and P. Jeavons, 'The expressive power of binary submodular functions', *Discrete Applied Mathematics*, **157**(15), 3347–3358, (2009).