Submodularity on a tree: Unifying $L^\natural$-convex and bisubmodular functions

Vladimir Kolmogorov

University College London
v.kolmogorov@cs.ucl.ac.uk

Abstract

We introduce a new class of functions that can be minimized in polynomial time in the value oracle model. These are functions f satisfying $f(\boldsymbol{x}) + f(\boldsymbol{y}) \ge f(\boldsymbol{x} \sqcap \boldsymbol{y}) + f(\boldsymbol{x} \sqcup \boldsymbol{y})$ where the domain of each variable x_i corresponds to nodes of a rooted binary tree, and operations $\sqcap, \sqcup$ are defined with respect to this tree. Special cases include previously studied $L^\natural$-convex and bisubmodular functions, which can be obtained with particular choices of trees. We present a polynomial-time algorithm for minimizing functions in the new class. It combines Murota's steepest descent algorithm for $L^\natural$-convex functions with bisubmodular minimization algorithms.

1 Introduction

Let $f : \mathcal{D} \to \mathbb{R}$ be a function of n variables $\boldsymbol{x} = (x_1, \dots, x_n)$ where $x_i \in D_i$; thus $\mathcal{D} = D_1 \times \ldots \times D_n$. We call elements of D_i *labels*, and the argument of f a *labeling*. Denote $V = \{1, \dots, n\}$ to be the set of nodes. We will consider functions f satisfying

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \ge f(\boldsymbol{x} \sqcap \boldsymbol{y}) + f(\boldsymbol{x} \sqcup \boldsymbol{y}) \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{D} \tag{1}$$

where binary operations $\sqcap, \sqcup : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ (expressed component-wise via operations $\sqcap, \sqcup : D_i \times D_i \to D_i$) are defined below.

There are several known cases in which function f can be minimized in polynomial time in the value oracle model. The following two cases will be of particular relevance:

- $L^\natural$*-convex functions*[1]: $D_i = \{0, 1, \dots, K_i\}$ where $K_i \ge 0$ is integer, $a \sqcap b = \lfloor \frac{a+b}{2} \rfloor$, $a \sqcup b = \lceil \frac{a+b}{2} \rceil$. Property (1) is then called *discrete midpoint convexity* [32].
- *Bisubmodular functions*: $D_i = \{-1, 0, +1\}$, $a \sqcup b = \mathtt{sign}(a + b)$, $a \sqcap b = |ab|\mathtt{sign}(a + b)$.

In this paper we introduce a new class of functions which includes the two classes above as special cases. We assume that labels in each set D_i are nodes of a tree T_i with a designated root $r_i \in D_i$. Define a partial order $\preceq$ on D_i as follows: $a \preceq b$ if a is an ancestor of b, i.e. a lies on the path from b to r_i ($a, b \in D_i$). For two labels $a, b \in D_i$ let $\mathcal{P}[a \to b]$ be unique path from a to b in T_i, $\rho(a, b)$ be the number of edges in this path, and $\mathcal{P}[a \to b, d]$ for integer $d \ge 0$ be the d-th node of this path so that $\mathcal{P}[a \to b, 0] = a$ and $\mathcal{P}[a \to b, \rho(a, b)] = b$. If $d > \rho(a, b)$ then we set by definition $\mathcal{P}[a \to b, d] = b$.

[1] Pronounced as "L-natural convex".

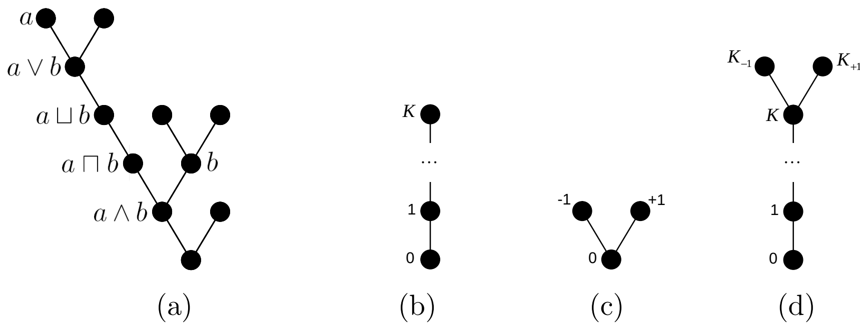

Figure 1: **Examples of trees. Roots are always at the bottom.** (a) Illustration of the definition of $a \sqcap b$, $a \sqcup b$, $a \wedge b$ and $a \vee b$. (b) A tree for $L^\natural$-convex functions. (c) A tree for bisubmodular functions. (d) A tree for which a weakly tree-submodular function can be minimized efficiently (see section 4).

With this notation, we can now define $a \sqcap b$, $a \sqcup b$ as the unique pair of labels satisfying the following two conditions: (1) $\{a \sqcap b, a \sqcup b\} = \{\mathcal{P}[a \to b, \lfloor \frac{d}{2} \rfloor], \mathcal{P}[a \to b, \lceil \frac{d}{2} \rceil]\}$ where $d = \rho(a, b)$, and (2) $a \sqcap b \preceq a \sqcup b$ (Figure 1(a)). We call functions f satisfying condition (1) with such choice of $(\mathcal{D}, \sqcap, \sqcup)$ *strongly tree-submodular*. Clearly, if each T_i is a chain with nodes $0, 1, \ldots, K$ and 0 being the root (Figure 1(b)) then strong tree-submodularity is equivalent to $L^\natural$-convexity. Furthermore, if each T_i is the tree shown in Figure 1(c) then strong tree-submodularity is equivalent to bisubmodularity.

The main result of this paper is the following

Theorem 1. *If each tree T_i is binary, i.e. each node has at most two children, then a strongly tree-submodular function f can be minimized in time polynomial in n and* $\max_i |D_i|$.

Weak tree-submodularity We will also study alternative operations on trees, which we denote as $\wedge$ and $\vee$. For labels $a, b \in D_i$ we define $a \wedge b$ as their highest common ancestor, i.e. the unique node on the path $\mathcal{P}[a \to b]$ which is an ancestor of both a and b. The label $a \vee b$ is defined as the unique label on the path $\mathcal{P}[a \to b]$ such that the distance between a and $a \vee b$ is the same as the distance between $a \wedge b$ and b (Figure 1(a)).

We say that function f is *weakly tree-submodular* if it satisfies

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \wedge \boldsymbol{y}) + f(\boldsymbol{x} \vee \boldsymbol{y}) \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{D} \tag{2}$$

We will show that strong tree-submodularity (1) implies weak tree-submodularity (2), which justifies the terminology. If all trees are chains shown in Figure 1(b) ($D_i = \{0, 1, \ldots, K\}$ with 0 being the root) then $\wedge$ and $\vee$ correspond to the standard operations "meet" and "join" (min and max) on an integer lattice. It is well-known that in this case weakly tree-submodular functions can be minimized in time polynomial in n and K [39, 32]. In section 4 we give a slight generalization of this result; namely, we allow trees shown in Figure 1(d).

1.1 Related work

Studying operations $\langle \sqcap, \sqcup \rangle$ that give rise to tractable optimization problems received a considerable attention in the literature. Some known examples of such operations are reviewed below. For simplicity, we assume that domains D_i (and operations $\langle \sqcap, \sqcup \rangle$) are the same for all nodes: $D_i = D$ for some finite set D.

Submodular functions on lattices The first example that we mention is the case when D is a distributive lattice and $\sqcap, \sqcup$ are the meet and joint operations on this lattice. Functions that satisfy (1) for this choice of D and $\sqcap, \sqcup$ are called *submodular functions* on D; it is well-known that they can be minimized in strongly polynomial time [18, 37, 19].

Recently, researchers considered submodular functions on non-distributive lattices. It is known that a lattice is non-distributive if it contains as a sublattice either the pentagon $\mathcal{N}_5$ or the diamond $\mathcal{M}_3$. Krokhin and Larose [27] proved tractability for the pentagon case, using nested applications of a submodular minimization algorithm. The case of the diamond was considered by Kuivinen [28], who proved pseudo-polynomiality of the problem. The case of general non-distributive lattices is still open.

$L^\natural$-convex functions The concept of $L^\natural$-convexity was introduced by Fujishige and Murota [16] as a variant of L-convexity by Murota [30]. $L^\natural$-convexity is equivalent to the combination of submodularity and integral convexity [13] (see [32] for details).

The fastest known algorithm for minimizing $L^\natural$-convex functions is the *steepest descent* algorithm of Murota [31, 32, 33]. Murota proved in [33] that algorithm's complexity is $O(n \min\{K, n \log K\} \cdot \mathtt{SFM(n)})$ where $K = \max_i |D_i|$ and $\mathtt{SFM}(n)$ is the complexity of a submodular minimization algorithm for a function with n variables. The analysis of Kolmogorov and Shioura [22] improved the bound to $O(\min\{K, n \log K\} \cdot \mathtt{SFM(n)})$. In section 2 we review Murota's algorithm (or rather its version without scaling that has complexity $O(K \cdot \mathtt{SFM}(n))$.)

Note, the class of $L^\natural$-convex functions is a subclass of submodular functions on a totally ordered set $D = \{0, 1, \ldots, K\}$.

Bisubmodular functions Bisubmodular functions were introduced by Chandrasekaran and Kabadi as rank functions of *(poly-)pseudomatroids* [7, 21]. Independently, Bouchet [3] introduced the concept of Δ-matroids which is equivalent to pseudomatroids. Bisubmodular functions and their generalizations have also been considered by Qi [35], Nakamura [34], Bouchet and Cunningham [4] and Fujishige [15].

It has been shown that some submodular minimization algorithms can be generalized to bisubmodular functions. Qi [35] showed the applicability of the ellipsoid method. Fujishige and Iwata [17] developed a weakly polynomial combinatorial algorithm for minimizing bisubmodular functions with complexity $O(n^5 \mathtt{EO} \log M)$ where $\mathtt{EO}$ is the number of calls to the evaluation oracle and M is an upper bound on function values. McCormick and Fujishige [29] presented a strongly combinatorial version with complexity $O(n^7 \mathtt{EO} \log n)$, as well as a $O(n^9 \mathtt{EO} \log^2 n)$ fully combinatorial variant that does not use divisions. The algorithms in [29] can also be applied for minimizing a bisubmodular function over a *signed ring family*, i.e. a subset $\mathcal{R} \subseteq \mathcal{D}$ closed under $\sqcap$ and $\sqcup$.

Valued constraint satisfaction and multimorphisms Our paper also fits into the framework of *Valued Constraint Satisfaction Problems* (VCSPs) [11]. In this framework we are given a *language* Γ, i.e. a set of cost functions $f : D^m \to \mathbb{R}_+ \cup \{+\infty\}$ where D is a fixed discrete domain and f is a function of arity m (different functions $f \in \Gamma$ may have different arities). A Γ-*instance* is any function $f : D^n \to \mathbb{R}_+ \cup \{+\infty\}$ that can be expressed as a finite sum of functions from Γ:

$$f(x_1, \ldots, x_n) = \sum_{t \in T} f_t(x_{i(t,1)}, \ldots, x_{i(t,m_t)})$$

where T is a finite set of terms, $f_t \in \Gamma$ is a function of arity m_t, and $i(t, k)$ are indexes in $\{1, \ldots, n\}$. A finite language Γ is called *tractable* if any Γ-instance can be minimized in polynomial time, and *NP-hard* if this minimization problem is NP-hard. These definitions are extended to infinite languages Γ as follows: Γ is called tractable if any finite subset $\Gamma' \subset \Gamma$ is tractable, and NP-hard if there exists a finite subset $\Gamma' \subset \Gamma$ which is NP-hard.

Classifying the complexity of different languages has been an active research area. A major open question in this line of research is the *Dichotomy Conjecture* of Feder and Vardi (formulated for the *crisp* case), which states that every constraint language is either tractable or NP-hard [14]. So far such dichotomy results have been obtained for some special cases, as described below.

A significant progress has been made in the **crisp** case, i.e. when Γ only contains functions $f : D^m \rightarrow \{0, +\infty\}$. The problem is then called *Constraint Satisfaction* (CSP). The dichotomy is known to hold for languages with a 2-element domain (Schaefer [36]), languages with a 3-element domain (Bulatov [6]), conservative languages[2] (Bulatov [5]), and languages containing a single relation without sources and sinks (Barto *et al.* [1]). All dichotomy theorems above have the following form: if all functions in Γ satisfy a certain condition given by one or more *polymorphisms* then the language is tractable, otherwise it is NP-hard.

For general VCSPs the dichotomy has been shown to hold for Boolean languages, i.e. languages with a 2-element domain (Cohen *et al.* [11]), conservative languages (Kolmogorov and Živný [23, 24, 25], who generalized previous results by Deineko *et al.* [12] and Takhanov [38]), and $\{0, 1\}$-valued languages with a 4-element domain (Jonsson *et al.* [20]). In these examples tractable subclasses are characterized by one or more *multimorphisms*, which are generalizations of polymorphisms. A multimorphism of arity k over D is a tuple $\langle \mathtt{OP}_1, \ldots, \mathtt{OP}_k \rangle$ where $\mathtt{OP}_i$ is an operation $D^k \rightarrow D$. Language Γ is said to admit multimorphism $\langle \mathtt{OP}_1, \ldots, \mathtt{OP}_k \rangle$ if every function $f \in \Gamma$ satisfies

$$f(\boldsymbol{x}_1) + \ldots + f(\boldsymbol{x}_k) \geq f(\mathtt{OP}_1(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)) + \ldots + f(\mathtt{OP}_k(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k))$$

for all labelings $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ with $f(\boldsymbol{x}_1) < +\infty$, ..., $f(\boldsymbol{x}_k) < +\infty$. (The pair of operations $\langle \sqcap, \sqcup \rangle$ used in (1) is an example of a binary multimorphism.) The tractable classes mentioned above (for $|D| > 2$) are characterized by *complementary pairs of STP and MJN* multimorphisms [24] (that generalized *symmetric tournament pair (STP)* multimorphisms [10]), and *1-defect chain* multimorphisms [20] (that generalized tractable weak-tree submodular functions in section 4 originally introduced in [26]).

To make further progress on classifying complexity of VCSPs, it is important to study which multimorphisms lead to tractable optimisation problems. Operations $\langle \sqcap, \sqcup \rangle$ and $\langle \wedge, \vee \rangle$ introduced in this paper represent new classes of such multimorphisms: to our knowledge, previously researchers have not considered multimorphisms defined on trees.

Combining multimorphisms Finally, we mention that some constructions, namely *Cartesian products* and *Malt'stev products*, can be used for obtaining new tractable classes of binary multimoprhisms from existing ones [27]. Note, Krokhin and Larose [27] formulated these constructions only for lattice multimorphisms $\langle \sqcap, \sqcup \rangle$, but the proof in [27] actually applies to arbitrary binary multimorphisms $\langle \sqcap, \sqcup \rangle$.

2 Steepest descent algorithm

It is known that for $L^\natural$-convex functions local optimality implies global optimality [32]. We start by generalizing this result to strongly tree-submodular functions. Let us define the following "local"

[2] A crisp language Γ is called conservative if it contains all unary cost functions $f : D \rightarrow \{0, +\infty\}$ [5]. A general-valued language is called conservative if it contains all unary cost functions $f : D \rightarrow \mathbb{R}_+$ [23, 24, 25].

neighborhoods of labeling $\boldsymbol{x} \in \mathcal{D}$:

$$\begin{aligned} \texttt{NEIB}(\boldsymbol{x}) &= \{\boldsymbol{y} \in \mathcal{D} \mid \rho(\boldsymbol{x}, \boldsymbol{y}) \leq 1\} \\ \texttt{INWARD}(\boldsymbol{x}) &= \{\boldsymbol{y} \in \texttt{NEIB}(\boldsymbol{x}) \mid \boldsymbol{y} \preceq \boldsymbol{x}\} \\ \texttt{OUTWARD}(\boldsymbol{x}) &= \{\boldsymbol{y} \in \texttt{NEIB}(\boldsymbol{x}) \mid \boldsymbol{y} \succeq \boldsymbol{x}\} \end{aligned}$$

where $\boldsymbol{u} \preceq \boldsymbol{v}$ means that $u_i \preceq v_i$ for all $i \in V$, and $\rho(\boldsymbol{x}, \boldsymbol{y}) = \max_{i \in V} \rho(x_i, y_i)$ is the l_∞-distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. Clearly, the restriction of f to $\texttt{INWARD}(\boldsymbol{x})$ is a submodular function, and the restriction of f to $\texttt{OUTWARD}(\boldsymbol{x})$ is bisubmodular assuming that each tree T_i is binary[3].

Proposition 2. *Suppose that* $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\} = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$. *Then* $\boldsymbol{x}$ *is a global minimum of* f.

Proof. First, let us prove that $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{NEIB}(\boldsymbol{x})\}$. Let $\boldsymbol{x}^*$ be a minimizer of f in $\texttt{NEIB}(\boldsymbol{x})$, and denote $\mathcal{D}^* = \{\boldsymbol{y} \in \mathcal{D} \mid y_i \in \mathcal{D}_i^* = \{x_i, x_i^*\}\} \subseteq \texttt{NEIB}(\boldsymbol{x})$. We treat set $\mathcal{D}_i^*$ as a tree with root $x_i \sqcap x_i^*$. Clearly, the restriction of f to $\mathcal{D}^*$ is an $L^\natural$-convex function under the induced operations $\sqcap$, $\sqcup$. It is known that for $L^\natural$-convex functions optimality of $\boldsymbol{x}$ in sets $\{\boldsymbol{y} \in \mathcal{D}^* \mid \boldsymbol{y} \preceq \boldsymbol{x}\}$ and $\{\boldsymbol{y} \in \mathcal{D}^* \mid \boldsymbol{y} \succeq \boldsymbol{x}\}$ suffices for optimality of $\boldsymbol{x}$ in $\mathcal{D}^*$ [32, Theorem 7.14], therefore $f(\boldsymbol{x}) \leq f(\boldsymbol{x}^*)$. This proves that $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{NEIB}(\boldsymbol{x})\}$.

Let us now prove that $\boldsymbol{x}$ is optimal in $\mathcal{D}$. Suppose not, then there exists $\boldsymbol{y} \in \mathcal{D}$ with $f(\boldsymbol{y}) < f(\boldsymbol{x})$. Among such labelings, let us choose $\boldsymbol{y}$ with the minimum distance $\rho(\boldsymbol{x}, \boldsymbol{y})$. We must have $\boldsymbol{y} \notin \texttt{NEIB}(\boldsymbol{x})$, so $\rho(\boldsymbol{x}, \boldsymbol{y}) \geq 2$. Clearly, $\rho(\boldsymbol{x}, \boldsymbol{x} \sqcup \boldsymbol{y}) \leq \rho(\boldsymbol{x}, \boldsymbol{y}) - 1$ and $\rho(\boldsymbol{x}, \boldsymbol{x} \sqcap \boldsymbol{y}) \leq \rho(\boldsymbol{x}, \boldsymbol{y}) - 1$. Strong tree-submodularity and the fact that $f(\boldsymbol{y}) < f(\boldsymbol{x})$ imply that the cost of at least one of the labelings $\boldsymbol{x} \sqcup \boldsymbol{y}$, $\boldsymbol{x} \sqcap \boldsymbol{y}$ is smaller than $f(\boldsymbol{x})$. This contradicts to the choice of $\boldsymbol{y}$. $\square$

Suppose that each tree T_i is binary. The proposition shows that a greedy technique for computing a minimizer of f would work. We can start with an arbitrary labeling $\boldsymbol{x} \in \mathcal{D}$, and then apply iteratively the following two steps in some order:

(1) Compute minimizer $\boldsymbol{x}^{\texttt{in}} \in \arg\min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\}$ by invoking a submodular minimization algorithm, replace $\boldsymbol{x}$ with $\boldsymbol{x}^{\texttt{in}}$ if $f(\boldsymbol{x}^{\texttt{in}}) < f(\boldsymbol{x})$.

(2) Compute minimizer $\boldsymbol{x}^{\texttt{out}} \in \arg\min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$ by invoking a bisubmodular minimization algorithm, replace $\boldsymbol{x}$ with $\boldsymbol{x}^{\texttt{out}}$ if $f(\boldsymbol{x}^{\texttt{out}}) < f(\boldsymbol{x})$.

The algorithm stops if neither step can decrease the cost. Clearly, it terminates in a finite number of steps and produces an optimal solution. We will now discuss how to obtain a polynomial number of steps. We denote $K = \max_i |D_i|$.

2.1 $L^\natural$-convex case

For $L^\natural$-convex functions the *steepest descent* algorithm described above was first proposed by Murota [31, 32, 33], except that in step 2 a submodular minimization algorithm was used. Murota's algorithm actually computes both of $\boldsymbol{x}^{\texttt{in}}$ and $\boldsymbol{x}^{\texttt{out}}$ for the same $\boldsymbol{x}$ and then chooses a better one by comparing costs $f(\boldsymbol{x}^{\texttt{in}})$ and $f(\boldsymbol{x}^{\texttt{out}})$. A slight variation was proposed by Kolmogorov and Shioura [22], who allowed an arbitrary order of steps. Kolmogorov and Shioura also established a tight bound on the number of steps of the algorithm by proving the following theorem.

[3] If label x_i has less than two children in T_i then variable's domain after restriction will be a strict subset of $\{-1, 0, +1\}$. Therefore, we may need to use a bisubmodular minimization algorithm over a signed ring familiy $\mathcal{R} \subseteq \{-1, 0, +1\}^n$ [29].

Theorem 3 ([22])**.** *Suppose that each tree T_i is a chain. For a labeling $\boldsymbol{x} \in \mathcal{D}$ define*

$$\rho^-(\boldsymbol{x}) = \min\{\rho(\boldsymbol{x},\boldsymbol{y}) | \boldsymbol{y} \in OPT^-[\boldsymbol{x}]\}, OPT^-[\boldsymbol{x}] = \operatorname{argmin}\{f(\boldsymbol{y}) | \boldsymbol{y} \in \mathcal{D}, \boldsymbol{y} \preceq \boldsymbol{x}\} \tag{3a}$$
$$\rho^+(\boldsymbol{x}) = \min\{\rho(\boldsymbol{x},\boldsymbol{y}) | \boldsymbol{y} \in OPT^+[\boldsymbol{x}]\}, OPT^+[\boldsymbol{x}] = \operatorname{argmin}\{f(\boldsymbol{y}) | \boldsymbol{y} \in \mathcal{D}, \boldsymbol{y} \succeq \boldsymbol{x}\} \tag{3b}$$

(a) Applying step (1) or (2) to labeling $\boldsymbol{x} \in \mathcal{D}$ does not increase $\rho^-(\boldsymbol{x})$ and $\rho^+(\boldsymbol{x})$.

(b) If $\rho^-(\boldsymbol{x}) \geq 1$ then applying step (1) to $\boldsymbol{x}$ will decrease $\rho^-(\boldsymbol{x})$ by 1.

(c) If $\rho^+(\boldsymbol{x}) \geq 1$ then applying step (2) to $\boldsymbol{x}$ will decrease $\rho^+(\boldsymbol{x})$ by 1.

In the beginning of the algorithm we have $\rho^-(\boldsymbol{x}) \leq K$ and $\rho^+(\boldsymbol{x}) \leq K$, so the theorem implies that after at most K calls to step (1) and K calls to step (2) we get $\rho^-(\boldsymbol{x}) = \rho^+(\boldsymbol{x}) = 0$. The latter condition means that $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\} = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$, and thus, by proposition 2, $\boldsymbol{x}$ is a global minimum of f.

2.2 General case

We now show that the bound $O(K)$ on the number of steps is also achievable for general strongly tree-submodular functions. We will establish it for the following version of the steepest descent algorithm:

S0 Choose an arbitrary labeling $\boldsymbol{x}^\circ \in \mathcal{D}$ and set $\boldsymbol{x} := \boldsymbol{x}^\circ$.

S1 Compute minimizer $\boldsymbol{x}^{\texttt{in}} \in \arg\min\{f(\boldsymbol{y}) | \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\}$. If $f(\boldsymbol{x}^{\texttt{in}}) < f(\boldsymbol{x})$ then set $\boldsymbol{x} := \boldsymbol{x}^{\texttt{in}}$ and repeat step S1, otherwise go to step S2.

S2 Compute minimizer $\boldsymbol{x}^{\texttt{out}} \in \arg\min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$. If $f(\boldsymbol{x}^{\texttt{out}}) < f(\boldsymbol{x})$ then set $\boldsymbol{x} := \boldsymbol{x}^{\texttt{out}}$ and repeat step S2, otherwise terminate.

Note, one could choose x_i° to be the root of tree T_i for each node $i \in V$, then step `S1` would be redundant.

Theorem 4. *(a) Step* `S1` *is performed at most K times. (b) Each step* `S2` *preserves the following property:*

$$f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\} \tag{4}$$

(c) Step `S2` *is performed at most K times. (d) Labeling $\boldsymbol{x}$ produced upon termination of the algorithm is a minimizer of f.*

Proof. For a labeling $\boldsymbol{x} \in \mathcal{D}$ denote $\mathcal{D}^-[\boldsymbol{x}] = \{\boldsymbol{y} \in \mathcal{D} \mid \boldsymbol{y} \preceq \boldsymbol{x}\}$. We will treat domain $\mathcal{D}^-[\boldsymbol{x}]$ as the collection of chains with roots r_i and leaves x_i. Let $\rho^-(\boldsymbol{x})$ be the quantity defined in (3a). There holds

$$f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\} \qquad \Leftrightarrow \qquad \rho^-(\boldsymbol{x}) = 0 \tag{5}$$

Indeed, this equivalence can be obtained by applying proposition 2 to function f restricted to $\mathcal{D}^-[\boldsymbol{x}]$.

(a) When analyzing the first stage of the algorithm, we can assume without loss of generality that $\mathcal{D} = \mathcal{D}^-[\boldsymbol{x}^\circ]$, i.e. each tree T_i is a chain with the root r_i and the leaf x_i°. Indeed, removing the rest of the tree will not affect the behaviour of steps `S1`. With such assumption, function f becomes $L^\natural$-convex. By theorem 3(b), steps `S1` will terminate after at most K steps.

(b,c) Property (4) (or equivalently $\rho^-(\boldsymbol{x}) = 0$) clearly holds after termination of steps `S1`. Let $\boldsymbol{z}$ be the labeling upon termination of steps `S2`. When analyzing the second stage of the algorithm, we can assume without loss of generality that $\mathcal{D} = \mathcal{D}^-[\boldsymbol{z}]$, i.e. each tree T_i is a chain with the root r_i and the leaf z_i. Indeed, removing the rest of the tree will not affect the behaviour of steps S2. Furthermore, restricting f to $\mathcal{D}^-[\boldsymbol{z}]$ does not affect the definition of $\rho^-(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{D}^-[\boldsymbol{z}]$.

By theorem 3(a), steps S2 preserve $\rho^-(\boldsymbol{x}) = 0$; this proves part (b). Part (c) follows from theorem 3(c).

(d) When steps S2 terminate, we have $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$. Combining this fact with condition (4) and using proposition 2 gives that upon algorithm's termination $\boldsymbol{x}$ is a minimizer of f.

□

3 Translation submodularity

In this section we derive an alternative definition of strongly tree-submodular functions. As a corollary, we will obtain that strong tree submodularity (1) implies weak tree submodularity (2).

Let us introduce another pair of operations on trees. Given labels $a, b \in D_i$ and an integer $d \geq 0$, we define

$$a \uparrow^d b = \mathcal{P}[a \to b, d] \wedge b \qquad a \downarrow_d b = \mathcal{P}[a \to b, \rho(a \uparrow^d b, b)]$$

In words, $a \uparrow^d b$ is obtained as follows: (1) move from a towards b by d steps, stopping if b is reached earlier; (2) keep moving until the current label becomes an ancestor of b. $a \downarrow_d b$ is the label on the path $\mathcal{P}[a \to b]$ such that the distances $\rho(a, a \downarrow_d b)$ and $\rho(a \uparrow^d b, b)$ are the same, as well as distances $\rho(a, a \uparrow^d b)$ and $\rho(a \downarrow_d b, b)$. Note, binary operations $\uparrow^d, \downarrow_d \colon D_i \times D_i \to D_i$ (and corresponding operations $\uparrow^d, \downarrow_d \colon \mathcal{D} \times \mathcal{D} \to \mathcal{D}$) are in general non-commutative. One exception is $d = 0$, in which case $\uparrow^d, \downarrow_d$ reduce to the commutative operations defined in the introduction: $\boldsymbol{x} \uparrow^0 \boldsymbol{y} = \boldsymbol{x} \wedge \boldsymbol{y}$ and $\boldsymbol{x} \downarrow_0 \boldsymbol{y} = \boldsymbol{x} \vee \boldsymbol{y}$.

For fixed labels $a, b \in D_i$ it will often be convenient to rename nodes in $\mathcal{P}[a \to b]$ to be consecutive integers so that $a \wedge b = 0$ and $a \leq 0 \leq b$. Then we have $a = -\rho(a, a \wedge b)$, $b = \rho(a \wedge b, b)$ and

$$a \uparrow^d b = \max\{0, \min\{a + d, b\}\} \qquad a \downarrow_d b = a + b - (a \uparrow^d b)$$

Theorem 5. *(a) If f is strongly tree-submodular then for any $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ and integer $d \geq 0$ there holds*

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \uparrow^d \boldsymbol{y}) + f(\boldsymbol{x} \downarrow_d \boldsymbol{y}) \tag{6}$$

(b) If (6) *holds for any $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ and $d \geq 0$ then f is strongly tree-submodular.*

Note, this result is well-known for $L^\natural$-convex functions [32, section 7.1], i.e. when all trees are chains shown in Figure 1(b); inequality (6) was then written as $f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f((\boldsymbol{x} + d \cdot \mathbf{1}) \wedge \boldsymbol{y}) + f(\boldsymbol{x} \vee (\boldsymbol{y} - d \cdot \mathbf{1}))$, and was called *translation submodularity*. In fact, translation submodularity is one of the key properties of $L^\natural$-convex functions, and was heavily used, for example, in [22] for proving theorem 3.

Setting $d = 0$ in theorem 5(a) gives

Corollary 6. *A strongly tree-submodular function f is also weakly tree-submodular, i.e.* (1) *implies* (2).

A proof of parts (b) and (a) of theorem 6 is given in sections 3.1 and 3.2 respectively. In both proofs we always implicitly assume that for each $i \in V$ labels in $\mathcal{P}[x_i \to y_i]$ are renamed to be consecutive integers with $x_i \wedge y_i = 0$ and $x_i \leq 0 \leq y_i$.

3.1 Proof of theorem 5(b)

We prove inequality (1) for $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ using induction on $\rho_1(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i \in V} \rho(x_i, y_i)$. The base case $\rho_1(\boldsymbol{x}, \boldsymbol{y}) = 0$, or $\boldsymbol{x} = \boldsymbol{y}$, is trivial; suppose that $\rho_1(\boldsymbol{x}, \boldsymbol{y}) \geq 1$. Denote $d_{\max} = \rho(\boldsymbol{x}, \boldsymbol{y}) \geq 1$ and $d = \lfloor d_{\max}/2 \rfloor \geq 0$. Two cases are possible.

Case 1 $d_{\max}$ is even. We can assume without loss of generality that there exists $k \in V$ such that $y_k - x_k = d_{\max}$ and $|x_k| \geq y_k$. (If there is no such k, we can simply swap $\boldsymbol{x}$ and $\boldsymbol{y}$; inequality (1) will be unaffected since operations $\sqcap, \sqcup$ are commutative, and $\rho(\boldsymbol{x}, \boldsymbol{y})$, $\rho_1(\boldsymbol{x}, \boldsymbol{y})$ will not change.) Consider labelings $\boldsymbol{x}', \boldsymbol{y}' \in \mathcal{D}$ defined as follows:

$$y'_i = \begin{cases} y_i - 1 & \text{if } y_i - x_i = d_{\max},\ |x_i| \geq y_i \\ y_i & \text{otherwise} \end{cases} \qquad\qquad x'_i = x_i \sqcup y'_i$$

for each $i \in V$. We claim that

$$\text{(a)}\quad \boldsymbol{x} \sqcap \boldsymbol{y}' = \boldsymbol{x} \sqcap \boldsymbol{y} \qquad\qquad \text{(b)}\quad \boldsymbol{x} \sqcup \boldsymbol{y}' = \boldsymbol{x}'$$
$$\text{(c)}\quad \boldsymbol{x}' \uparrow^{d} \boldsymbol{y} = \boldsymbol{y}' \qquad\qquad \text{(d)}\quad \boldsymbol{x}' \downarrow_{d} \boldsymbol{y} = \boldsymbol{x} \sqcup \boldsymbol{y}$$

Indeed, for each node $i \in V$ one of the following holds:

- $y_i - x_i \leq d_{\max} - 1$. Then $y'_i = y_i$, $x'_i = x_i \sqcup y_i$, so (a) and (b) hold for node i. We also have $y_i - x'_i = y_i - (x_i \sqcup y_i) \leq \lceil (y_i - x_i)/2 \rceil \leq \lceil (d_{\max} - 1)/2 \rceil \leq d$, which implies (c) and (d).
- $y_i - x_i = d_{\max}$ and $|x_i| < y_i$. Then $y'_i = y_i$, $x'_i = x_i \sqcup y_i = (x_i + y_i)/2$, $y_i - x'_i = d$; as above, this implies (a)-(d).
- $y_i - x_i = d_{\max}$ and $|x_i| \geq y_i$. Then $y'_i = y_i - 1$, $x'_i = x_i \sqcup y'_i = \lfloor (x_i + y_i - 1)/2 \rfloor = (x_i + y_i)/2 - 1 = y'_i - d$. Checking that (a)-(d) hold is straightforward.

We have $y'_k = y_k - 1$, and so $\rho_1(\boldsymbol{x}, \boldsymbol{y}') < \rho_1(\boldsymbol{x}, \boldsymbol{y})$. Therefore,

$$f(\boldsymbol{x}) + f(\boldsymbol{y}') \geq f(\boldsymbol{x} \sqcap \boldsymbol{y}') + f(\boldsymbol{x} \sqcup \boldsymbol{y}') \qquad\qquad f(\boldsymbol{x}') + f(\boldsymbol{y}) \geq f(\boldsymbol{x}' \uparrow^{d} \boldsymbol{y}) + f(\boldsymbol{x}' \downarrow_{d} \boldsymbol{y})$$

where the first inequality follows from the induction hypothesis and the second one follows from (6). Summing these inequalities and subtracting $f(\boldsymbol{x}') + f(\boldsymbol{y}')$ from both sides using (a)-(d) gives (1).

Case 2 $d_{\max}$ is odd. By swapping $\boldsymbol{x}$ and $\boldsymbol{y}$, if necessary, we can assume without loss of generality that there exists $k \in V$ such that $y_k - x_k = d_{\max}$ and $|x_k| < y_k$. (Note, we cannot have $y_i - x_i = d_{\max}$ and $|x_i| = y_i$ since $d_{\max}$ is odd). Consider labelings $\boldsymbol{x}', \boldsymbol{y}' \in \mathcal{D}$ defined as follows:

$$x'_i = \begin{cases} x_i + 1 & \text{if } y_i - x_i = d_{\max},\ |x_i| < y_i \\ x_i & \text{otherwise} \end{cases} \qquad\qquad y'_i = x'_i \sqcap y_i$$

for each $i \in V$. We claim that

$$\text{(a)}\quad \boldsymbol{x}' \sqcap \boldsymbol{y} = \boldsymbol{y}' \qquad\qquad \text{(b)}\quad \boldsymbol{x}' \sqcup \boldsymbol{y} = \boldsymbol{x} \sqcup \boldsymbol{y}$$
$$\text{(c)}\quad \boldsymbol{x} \uparrow^{d} \boldsymbol{y}' = \boldsymbol{x} \sqcap \boldsymbol{y} \qquad\qquad \text{(d)}\quad \boldsymbol{x} \downarrow_{d} \boldsymbol{y}' = \boldsymbol{x}'$$

Indeed, for each node $i \in V$ one of the following holds:

- $y_i - x_i \leq d_{\max} - 1$. Then $x'_i = x_i$, $y'_i = x_i \sqcap y_i$, so (a) and (b) hold for node i. We also have $y'_i - x_i = (x_i \sqcap y_i) - x_i \leq \lceil (y_i - x_i)/2 \rceil \leq \lceil (d_{\max} - 1)/2 \rceil \leq d$, which implies (c) and (d).

- $y_i - x_i = d_{\max}$ and $|x_i| > y_i$. Then $x'_i = x_i$, $y'_i = x_i \sqcup y_i = \lceil (x_i + y_i)/2 \rceil \le 0$, so (a) and (b) hold for node i. (c) and (d) hold since $y'_i \le 0$.

- $y_i - x_i = d_{\max}$ and $|x_i| < y_i$. Then $x'_i = x_i + 1$, $y'_i = x'_i \sqcap y_i = \lfloor (x_i + y_i - 1)/2 \rfloor$. Checking that (a)-(d) hold is straightforward.

We have $x'_k = x_k + 1$, and so $\rho_1(\boldsymbol{x}', \boldsymbol{y}) < \rho_1(\boldsymbol{x}, \boldsymbol{y})$. Therefore,

$$f(\boldsymbol{x}') + f(\boldsymbol{y}) \ge f(\boldsymbol{x}' \sqcap \boldsymbol{y}) + f(\boldsymbol{x}' \sqcup \boldsymbol{y}) \qquad f(\boldsymbol{x}) + f(\boldsymbol{y}') \ge f(\boldsymbol{x} \uparrow^d \boldsymbol{y}') + f(\boldsymbol{x} \downarrow_d \boldsymbol{y}')$$

where the first inequality follows from the induction hypothesis and the second one follows from (6). Summing these inequalities and subtracting $f(\boldsymbol{x}') + f(\boldsymbol{y}')$ from both sides using (a)-(d) gives (1).

3.2 Proof of theorem 5(a)

We say that the triplet $(\boldsymbol{x}, \boldsymbol{y}, d)$ is *valid* if $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ and $d \in [0, \rho(\boldsymbol{x}, \boldsymbol{y})]$. We denote $\boldsymbol{z} = \boldsymbol{x} \uparrow^d \boldsymbol{y}$; we have $x_i \le 0 \le z_i \le y_i$. Let us introduce a partial order $\preceq$ over valid triplets as the lexicographical order with variables $(y_1 - x_1, \ldots, y_n - x_n, -d)$. Note, the last component $-d$ is the least significant. We use induction on this partial order. The induction base is trivial: if the first n components are zeros then $\boldsymbol{x} = \boldsymbol{y}$ so (6) is an equality, and if the last component is minimal (i.e. $d = \rho(\boldsymbol{x}, \boldsymbol{y})$) then $\boldsymbol{x} \uparrow^d \boldsymbol{y} = \boldsymbol{y}$ and $\boldsymbol{x} \downarrow_d \boldsymbol{y} = \boldsymbol{x}$, so (6) is again an equality. Suppose that $\boldsymbol{x} \ne \boldsymbol{y}$ and $d \le \rho(\boldsymbol{x}, \boldsymbol{y}) - 1$.

Consider integer $d' \ge d$, and denote $\boldsymbol{y}' = \boldsymbol{x} \uparrow^{d'+1} \boldsymbol{y}$ and $\delta_i = y'_i - z_i \ge 0$ for $i \in V$. Suppose that $\delta_i \in \{0, 1\}$ for all nodes $i \in V$. (This holds, for example, if $d' = d$.) Denote $\boldsymbol{x}' = \boldsymbol{x} \downarrow_d \boldsymbol{y}'$, then $x'_i = x_i + y'_i - (x_i \uparrow^d y'_i) = x_i + y'_i - z_i = x_i + \delta_i$. We claim that

$$\begin{array}{llll} \text{(a)} & \boldsymbol{x} \uparrow^d \boldsymbol{y}' = \boldsymbol{x} \uparrow^d \boldsymbol{y} & \text{(b)} & \boldsymbol{x} \downarrow_d \boldsymbol{y}' = \boldsymbol{x}' \\ \text{(c)} & \boldsymbol{x}' \uparrow^{d'} \boldsymbol{y} = \boldsymbol{y}' & \text{(d)} & \boldsymbol{x}' \downarrow_{d'} \boldsymbol{y} = \boldsymbol{x} \downarrow_d \boldsymbol{y} \end{array} \tag{7}$$

In order to prove it, let us consider node i. Property (a) follows from the fact that $y'_i \ge x_i \uparrow^d y_i$. Property (b) is the definition of $\boldsymbol{x}'$. To prove (c), consider two possible cases:

- $\delta_i = 0$, so $x'_i = x_i$ and $y'_i \equiv x_i \uparrow^{d'+1} y_i = x_i \uparrow^d y_i$. The latter condition and the fact $d' + 1 > d$ imply that $x_i + d \ge y_i$, therefore $x'_i + d' \ge y_i = y'_i$. This leads to (c).

- $\delta_i = 1$. If $y'_i = y_i$ then condition (c) is straightforward (it follows from $x'_i \uparrow^{d'} y_i \ge y'_i$). Suppose that $y'_i < y_i$, then from definition of y'_i we have $x_i + d' + 1 \le y'_i$, or $x'_i + d' \le y'_i$. This leads to (c).

Finally, properties (c) and (d) are equivalent since

$$\begin{aligned} x'_i + y_i - y'_i - [x_i \downarrow_d y_i] &= [x_i + y'_i - (x_i \uparrow^d y'_i)] + y_i - y'_i - [x_i + y_i - (x_i \uparrow^d y_i)] \\ &= (x_i \uparrow^d y_i) - (x_i \uparrow^d y'_i) = 0 \end{aligned}$$

Now suppose that in addition to conditions $\delta_i \in \{0, 1\}$ there holds $\boldsymbol{x}' \ne \boldsymbol{x}$ and $\boldsymbol{y}' \ne \boldsymbol{y}$. Then we have $(\boldsymbol{x}, \boldsymbol{y}', d) \prec (\boldsymbol{x}, \boldsymbol{y}, d)$ and $(\boldsymbol{x}', \boldsymbol{y}, d') \prec (\boldsymbol{x}, \boldsymbol{y}, d)$, so by the induction hypothesis

$$f(\boldsymbol{x}) + f(\boldsymbol{y}') \ge f(\boldsymbol{x} \uparrow^d \boldsymbol{y}') + f(\boldsymbol{x} \downarrow_d \boldsymbol{y}') \qquad f(\boldsymbol{x}') + f(\boldsymbol{y}) \ge f(\boldsymbol{x}' \uparrow^{d'} \boldsymbol{y}) + f(\boldsymbol{x}' \downarrow_{d'} \boldsymbol{y})$$

Summing these inequalities and subtracting $f(\boldsymbol{x}') + f(\boldsymbol{y}')$ from both sides using (a)-(d) gives (6).

Let us describe cases when the argument above can be applied; such cases can be eliminated from consideration. First, suppose that $y_j - z_j \ge 2$ for some node $j \in V$, then there exists $d' \ge d$

such that the labeling $\boldsymbol{y}' = \boldsymbol{x} \uparrow^{d'+1} \boldsymbol{y}$ has at least one node $j \in V$ with $y'_j \in [z_j+1, y_j-1]$. Let us choose the minimum integer d' that has this property. Then $\delta_i \in \{0,1\}$ for all nodes $i \in V$, since $\delta_i \geq 2$ would contradict to the minimality of chosen d'. We also have $y'_j \neq y_j$ and $x'_j \neq x_j$ (since $x'_j - x_j = \delta_j = 1$), so the conditions above are satisfied. Therefore, from now on we assume without loss of generality that $y_i - z_i \in \{0,1\}$ for all nodes $i \in V$.

We can also take $d' = d$. Condition $\delta_i \in \{0,1\}$ is then satisfied for all nodes. Therefore, we can assume without loss of generality that either $\boldsymbol{x}' = \boldsymbol{x}$ or $\boldsymbol{y}' = \boldsymbol{y}$ where $\boldsymbol{y}' = \boldsymbol{x} \uparrow^{d+1} \boldsymbol{y}$, $\boldsymbol{x}' = \boldsymbol{x} \downarrow_d \boldsymbol{y}'$, otherwise the induction argument above could be applied. Suppose that $\boldsymbol{x}' = \boldsymbol{x}$. This is equivalent to $\boldsymbol{x} \uparrow^d \boldsymbol{y}' = \boldsymbol{y}'$, or to the following condition for all nodes $i \in V$: either $x_i + d < 0$ or $y_i - x_i \geq d$. It can be checked that $\boldsymbol{x} \uparrow^{d+1} \boldsymbol{y} = \boldsymbol{x} \uparrow^d \boldsymbol{y}$ and $\boldsymbol{x} \downarrow_{d+1} \boldsymbol{y} = \boldsymbol{x} \downarrow_d \boldsymbol{y}$. Furthermore, $(\boldsymbol{x}, \boldsymbol{y}, d+1) \prec (\boldsymbol{x}, \boldsymbol{y}, d)$, so (6) follows by the induction hypothesis. We thus assume from now on that $\boldsymbol{y}' = \boldsymbol{y}$.

Equations below summarize definitions and assumptions made so far:

$$\begin{aligned} z_i &= x_i \uparrow^d y_i && \text{(8a)} \\ y'_i &= x_i \uparrow^{d+1} y_i = y_i && \text{(8b)} \\ x'_i &= x_i \downarrow_d y_i && \text{(8c)} \\ \delta_i &= y_i - z_i = x'_i - x_i \in \{0,1\} && \text{(8d)} \end{aligned}$$

Let S be the set of nodes $i \in V$ with $\delta_i = 1$. It is straighforward to check that

$$\begin{aligned} i \in S \quad &\Rightarrow \quad x_i + d = z_i = y_i - 1 && \text{(8e)} \\ i \in V - S \quad &\Rightarrow \quad x_i \uparrow^d y_i = y_i \quad \text{and} \quad x_i \downarrow_d y_i = x_i = x'_i && \text{(8f)} \end{aligned}$$

If S is empty then $\boldsymbol{x} \uparrow^d \boldsymbol{y} = \boldsymbol{y}$, $\boldsymbol{x} \downarrow_d \boldsymbol{y} = \boldsymbol{x}$, so inequality (6) is trivial. Thus, we can assume that S is non-empty. Suppose that S contains two distinct nodes i and j. Let us modify labelings $\boldsymbol{x}'$ and $\boldsymbol{y}'$ as follows: for node j set $x'_j = x_j$, $y'_j = z_j$. It is straightforward to check that conditions (7) for $d' = d$ still hold. Furthermore, $x'_i > x_i$, $y'_j < y_j$, so $(\boldsymbol{x}, \boldsymbol{y}', d) \prec (\boldsymbol{x}, \boldsymbol{y}, d)$ and $(\boldsymbol{x}', \boldsymbol{y}, d) \prec (\boldsymbol{x}, \boldsymbol{y}, d)$. Applying the argument described above gives (6).

We are left with the case when S contains a single node j. We will consider 5 possible subcases. In 4 of them, we will do the following: (i) specify new labelings $\boldsymbol{x}'$ and $\boldsymbol{y}'$ with $x'_i, y'_i \in [x_i, y_i]$ for each node i; (ii) specify four identities involving $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}', \boldsymbol{y}'$ such that the right-hand sides contain expressions $\boldsymbol{x}', \boldsymbol{y}', \boldsymbol{x} \uparrow^d \boldsymbol{y}, \boldsymbol{x} \downarrow_d \boldsymbol{y}$, and the left-hand sides contain expressions of the form $\boldsymbol{x} \diamond_1 \boldsymbol{y}'$, $\boldsymbol{x} \,\overline{\diamond}_1\, \boldsymbol{y}'$, $\boldsymbol{x}' \diamond_2 \boldsymbol{y}$, $\boldsymbol{x}' \,\overline{\diamond}_2\, \boldsymbol{y}$ where $\diamond_k$ is one of the operations $\sqcap, \sqcup, \uparrow^{d_k}, \downarrow_{d_k}$ and $\overline{\diamond}_k$ is the corresponding "symmetric" operation. This will describe how to prove (6): we would need to sum two inequalities

$$f(\boldsymbol{x}) + f(\boldsymbol{y}') \geq f(\boldsymbol{x} \diamond_1 \boldsymbol{y}') + f(\boldsymbol{x} \,\overline{\diamond}_1\, \boldsymbol{y}') \qquad f(\boldsymbol{x}') + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \diamond_2 \boldsymbol{y}') + f(\boldsymbol{x} \,\overline{\diamond}_2\, \boldsymbol{y}')$$

that hold either by strong tree-submodularity or by the induction hypothesis, then use provided identities to prove (6). Checking the identities and the applicability of the induction hypothesis in the case of operations $\uparrow^{d_k}$, $\downarrow_{d_k}$ is mechanical, and we omit it.

<u>**Case 1**</u> $z_j = x_j + d \geq 1$ (implying $d \geq 1$). The identities are

$$\begin{array}{llll} \text{(a)} & \boldsymbol{x} \uparrow^{d-1} \boldsymbol{y}' = \boldsymbol{x}' & \text{(b)} & \boldsymbol{x} \downarrow_{d-1} \boldsymbol{y}' = \boldsymbol{x} \downarrow_d \boldsymbol{y} \\ \text{(c)} & \boldsymbol{x}' \sqcup \boldsymbol{y} = \boldsymbol{x} \uparrow^d \boldsymbol{y} & \text{(d)} & \boldsymbol{x}' \sqcap \boldsymbol{y} = \boldsymbol{y}' \end{array} \qquad (9)$$

and labelings $\boldsymbol{x}', \boldsymbol{y}'$ are defined as follows:

- if $i = j$ set $x'_j = y_j - 2$, $y'_j = y_j - 1$;

- otherwise if $x_i + d = y_i > 0$ set $x'_i = y'_i = y_i - 1$.
- otherwise (if $y_i = 0$ or $x_i + d > y_i$) set $x'_i = y'_i = y_i$.

The remainder is devoted to the case $z_j = x_j + d = 0$. Note that we must have $y_j = 1$.

<u>**Case 2**</u> $d \geq 1$, $z_j = x_j + d = 0$ and there exists node $k \in V - \{j\}$ with $x_k = 0$, $y_k > 0$. Then

$$\begin{array}{llll} \text{(a)} & \boldsymbol{x}' \uparrow^d \boldsymbol{y} = \boldsymbol{x} \uparrow^d \boldsymbol{y} & \text{(b)} & \boldsymbol{x}' \downarrow_d \boldsymbol{y} = \boldsymbol{y}' \\ \text{(c)} & \boldsymbol{x} \sqcup \boldsymbol{y}' = \boldsymbol{x}' & \text{(d)} & \boldsymbol{x} \sqcap \boldsymbol{y}' = \boldsymbol{x} \downarrow_d \boldsymbol{y} \end{array} \tag{10}$$

$\boldsymbol{x}'$, $\boldsymbol{y}'$ are defined as follows:

- if $i = j$ set $x'_j = x_j$, $y'_j = x_j + 1$;
- otherwise if $i = k$ set $x'_k = y'_k = x_k + 1 = 1$;
- otherwise set $x'_i = y'_i = x_i$.

<u>**Case 3**</u> $d \geq 1$, $z_j = x_j + d = 0$ and there is no node $k \in V - \{j\}$ with $x_k = 0$, $y_k > 0$. The identities are

$$\begin{array}{llll} \text{(a)} & \boldsymbol{x}' \uparrow^{d-1} \boldsymbol{y} = \boldsymbol{x} \uparrow^d \boldsymbol{y} & \text{(b)} & \boldsymbol{x}' \downarrow_{d-1} \boldsymbol{y} = \boldsymbol{y}' \\ \text{(c)} & \boldsymbol{x} \sqcup \boldsymbol{y}' = \boldsymbol{x} \downarrow_d \boldsymbol{y} & \text{(d)} & \boldsymbol{x} \sqcap \boldsymbol{y}' = \boldsymbol{x}' \end{array} \tag{11}$$

$\boldsymbol{x}'$, $\boldsymbol{y}'$ are defined as follows:

- if $i = j$ set $x'_j = x_i + 1$, $y'_j = x_i + 2$;
- otherwise if $x_i < 0$ set $x'_i = y'_i = x_i + 1$;
- otherwise (if $x_i = y_i = 0$) set $x'_i = y'_i = 0$.

Note, to verify identities (11) for node j, one should consider cases $d = 1$ and $d \geq 2$ separately.

<u>**Case 4**</u> $d = 0$ (implying $x_j = 0$, $y_j = 1$) and there exists node $k \in V - \{j\}$ with $x_k < 0$. Then

$$\begin{array}{llll} \text{(a)} & \boldsymbol{x} \uparrow^0 \boldsymbol{y}' = \boldsymbol{x}' & \text{(b)} & \boldsymbol{x} \downarrow_0 \boldsymbol{y}' = \boldsymbol{x} \downarrow_0 \boldsymbol{y} \\ \text{(c)} & \boldsymbol{x}' \sqcup \boldsymbol{y} = \boldsymbol{y}' & \text{(d)} & \boldsymbol{x}' \sqcap \boldsymbol{y} = \boldsymbol{x} \uparrow^0 \boldsymbol{y} \end{array} \tag{12}$$

$\boldsymbol{x}'$, $\boldsymbol{y}'$ are defined as follows:

- if $i = j$ set $x'_j = 0$, $y'_j = 1$;
- otherwise if $x_i < 0$, $y_i = 0$ set $x'_i = y'_i = -1$;
- otherwise (if $x_i = y_i = 0$) set $x'_i = y'_i = 0$.

<u>**Case 5**</u> $d = 0$ (implying $x_j = 0$, $y_j = 1$) and there is no node $k \in V - \{j\}$ with $x_k < 0$. Thus, $x_i = y_i = 0$ for all $i \in V - \{j\}$. There holds $\boldsymbol{x} \uparrow^0 \boldsymbol{y} = \boldsymbol{x}$, $\boldsymbol{x} \downarrow_0 \boldsymbol{y} = \boldsymbol{y}$, so inequality (6) is trivial.

4 Weakly tree-submodular functions

In this section we consider functions f that satisfy condition (2), but not necessarily condition (1). It is well-known [39, 32] that such functions can be minimized efficiently if all trees T_i are chains rooted at an endpoint and $\max_i |D_i|$ is polynomially bounded. The algorithm utilizes Birkhoff's representation theorem [2] which says that there exists a *ring family* $\mathcal{R}$ such that there is an isomorphism between sets $\mathcal{D}$ and $\mathcal{R}$ that preserves operations $\wedge$ and $\vee$. (A subset $\mathcal{R} \subseteq \{0,1\}^m$ is a ring family if it is closed under operations $\wedge$ and $\vee$.) It is known that submodular functions over a ring family can be minimized in polynomial time, which implies the result. Note that the number of variables will be $m = O(\sum_i |D_i|)$.

Another case when f satisfying (2) can be minimized efficiently is when f is bisubmodular, i.e. all trees are as shown in Figure 1(c). Indeed, in this case the pairs of operations $\langle \sqcap, \sqcup \rangle$ and $\langle \wedge, \vee \rangle$ coincide.

An interesting question is whether there exist other classes of weakly tree-submodular functions that can be minimized efficiently. In this section we provide one rather special example. We consider the tree shown in Figure 1(d). Each T_i has nodes $\{0, 1, \ldots, K, K_{-1}, K_{+1}\}$ such that 0 is the root, the parent of k for $k = 1, \ldots, K$ is $k-1$, and the parent of K_{-1} and K_{+1} is K.

In order to minimize function f for such choice of trees, we create $K+1$ variables $y_{i0}, y_{i1}, \ldots, y_{iK}$ for each original variable $x_i \in D_i$. The domains of these variables are as follows: $\tilde{D}_{i0} = \ldots = \tilde{D}_{iK-1} = \{0,1\}$, $\tilde{D}_{iK} = \{-1, 0, +1\}$. Each domain is treated as a tree with root 0 and other nodes being the children of 0; this defines operations $\wedge$ and $\vee$ for domains $\tilde{D}_{i0}, \ldots \tilde{D}_{iK-1}, \tilde{D}_{iK}$. The domain $\tilde{\mathcal{D}}$ is set as the Cartesian product of individual domains over all nodes $i \in V$. Note, a vector $\boldsymbol{y} \in \tilde{\mathcal{D}}$ has $n(K+1)$ components.

For a labeling $\boldsymbol{x} \in \mathcal{D}$ let us define labeling $\boldsymbol{y} = \psi(\boldsymbol{x}) \in \tilde{\mathcal{D}}$ as follows:

$$\begin{array}{lcl} x_i = k \in \{0, 1, \ldots, K\} & \Rightarrow & y_{i0} = \ldots = y_{ik-1} = 1, \ y_{ik} = \ldots = y_{iK} = 0 \\ x_i = K_{-1} & \Rightarrow & y_{i0} = \ldots = y_{iK-1} = 1, \ y_{iK} = -1 \\ x_i = K_{+1} & \Rightarrow & y_{i0} = \ldots = y_{iK-1} = 1, \ y_{iK} = +1 \end{array}$$

It is easy to check that mapping $\psi : \mathcal{D} \to \tilde{\mathcal{D}}$ is injective and preserves operations $\wedge$ and $\vee$. Therefore, $\mathcal{R} = \mathtt{Im}\ \psi$ is a *signed ring family*, i.e. a subset of $\tilde{\mathcal{D}}$ closed under operations $\wedge$ and $\vee$. It is known [29] that bisubmodular functions over ring families can be minimized in polynomial time, leading to

Proposition 7. *Functions that are weakly tree-submodular with respect to trees shown in Figure 1(d) can be minimized in time polynomial in n and* $\max_i |D_i|$.

5 Conclusions and discussion

We introduced two classes of functions (strongly tree-submodular and weakly tree-submodular) that generalize several previously studied classes. For each class, we gave new examples of trees for which the minimization problem is tractable.

Our work leaves a natural open question: what is the complexity of the problem for more general trees? In particular, can we minimize efficiently strongly tree-submodular functions if trees are non-binary, i.e. if some nodes have three or more children? Note that the algorithm in section 2 and its analysis are still valid, but it is not clear whether the minimization procedure in step `S2` can be implemented efficiently. Also, are there trees besides the one shown in Figure 1(d) for which weakly tree-submodular functions can be minimized efficiently?

More generally, can one characterize for which operations $\langle \sqcap, \sqcup \rangle$ the minimization problem is tractable? Currently known tractable examples are distributive lattices, some non-distributive lattices [27, 28], operations on trees introduced in this paper, and combinations of the above operations obtained via Cartesian product and Malt'sev product [27]. Are there tractable cases that cannot be obtained via lattice and tree-based operations?

References

[1] L. Barto, M. Kozik and T. Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). *SIAM Journal on Computing*, 38(5):1782–1802, 2009.

[2] Garrett Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.

[3] A. Bouchet. Greedy algorithm and symmetric matroids. *Math. Programming*, 38:147–159, 1987.

[4] A. Bouchet and W. H. Cunningham. Delta-matroids, jump systems and bisubmodular polyhedra. *SIAM J. Discrete Math.*, 8:17–32, 1995.

[5] A. A. Bulatov. Tractable Conservative Constraint Satisfaction Problems. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 321–330, 2003.

[6] A. A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set *Journal of the ACM*, 53(1):66–120, 2006.

[7] R. Chandrasekaran and Santosh N. Kabadi. Pseudomatroids. *Discrete Math.*, 71:205–217, 1988.

[8] David Cohen, Martin Cooper, and Peter Jeavons. A complete characterization of complexity for boolean constraint optimization problems. In *Principles and Practice of Constraint Programming*, number 3258 in Lecture Notes in Computer Science, pages 212–226, 2004.

[9] David Cohen, Martin Cooper, Peter Jeavons, and Andrei Krokhin. Soft constraints: complexity and multimorphsims. In *Principles and Practice of Constraint Programming*, number 2833 in Lecture Notes in Computer Science, pages 244–258, 2003.

[10] David A. Cohen, Martin C. Cooper, and Peter G. Jeavons. Generalising submodularity and horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theoretical Computer Science*, 401:36–51, 2008.

[11] David A. Cohen, Martin C. Cooper, Peter G. Jeavons, and Andrei A. Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170:983–1016, 2006.

[12] V. Deineko, P. Jonsson, M. Klasson and A. Krokhin. The approximability of Max CSP with fixed-value constraints. *Journal of the ACM*, 55(4), 2008.

[13] P. Favati and F. Tardella. Convexity in nonlinear integer programming. *Ricerca Operativa*, 53:3–44, 1990.

[14] T. Feder and M. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

[15] S. Fujishige. *Submodular Functions and Optimization*. North-Holland, 1991.

[16] S. Fujishige and K. Murota. Notes on L-/M-convex functions and the separation theorems. *Math. Program.*, 88:129–146, 2000.

[17] Satoru Fujishige and Satoru Iwata. Bisubmodular function minimization. *SIAM J. Discrete Math.*, 19(4):1065–1073, 2006.

[18] M. Grötschel, L. Lovász and A. Schrijver. Geometric Algorithms and Combinatorial Optimization. *Springer Heidelberg*, 1988.

[19] S. Iwata, L. Fleischer and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48:761–777, 2001.

[20] P. Jonsson, F. Kuivinen and J. Thapper. Min CSP on Four Elements: Moving Beyond Submodularity. Tech. rep. *arXiv:1102.2880*, February 2011.

[21] Santosh N. Kabadi and R. Chandrasekaran. On totally dual integral systems. *Discrete Appl. Math.*, 26:87–104, 1990.

[22] V. Kolmogorov and A. Shioura. New algorithms for convex cost tension problem with application to computer vision. *Discrete Optimization*, 6(4):378–393, 2009.

[23] V. Kolmogorov and S. Živný. The complexity of conservative finite-valued CSPs. Tech. rep. *arXiv:1008.1555v1*, August 2010.

[24] V. Kolmogorov and S. Živný. Generalising tractable VCSPs defined by symmetric tournament pair multimorphisms. Tech. rep. *arXiv:1008.3104v1*, August 2010.

[25] V. Kolmogorov. A dichotomy theorem for conservative general-valued CSPs. Tech. rep. *arXiv:1008.4035v1*, August 2010.

[26] V. Kolmogorov. Submodularity on a tree: Unifying $L^\natural$-convex and bisubmodular functions. Tech. rep. *arXiv:1007.1229v2*, July 2010.

[27] A. Krokhin and B. Larose. Maximizing supermodular functions on product lattices, with application to maximum constraint satisfaction. *SIAM Journal on Discrete Mathematics*, 22(1):312–328, 2008.

[28] F. Kuivinen. On the Complexity of Submodular Function Minimisation on Diamonds. Tech. rep. *arXiv:0904.3183v1*, April 2009.

[29] S. Thomas McCormick and Satoru Fujishige. Strongly polynomial and fully combinatorial algorithms for bisubmodular function minimization. *Math. Program., Ser. A*, 122:87–120, 2010.

[30] K. Murota. Discrete convex analysis. *Math. Program.*, 83:313–371, 1998.

[31] K. Murota. Algorithms in discrete convex analysis. *IEICE Transactions on Systems and Information*, E83-D:344–352, 2000.

[32] K. Murota. *Discrete Convex Analysis*. SIAM Monographs on Discrete Mathematics and Applications, Vol. 10, 2003.

[33] K. Murota. On steepest descent algorithms for discrete convex functions. *SIAM J. Optimization*, 14(3):699–707, 2003.

[34] M. Nakamura. A characterization of greedy sets: universal polymatroids (I). In *Scientific Papers of the College of Arts and Sciences*, volume 38(2), pages 155–167. The University of Tokyo, 1998.

[35] Liqun Qi. Directed submodularity, ditroids and directed submodular flows. *Mathematical Programming*, 42:579–599, 1988.

[36] T. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*, pages 216–226, 1978.

[37] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Combin. Theory Ser. B*, 80:346–355, 2000.

[38] Rustem Takhanov. A Dichotomy Theorem for the General Minimum Cost Homomorphism Problem. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, pages 657–668, 2010.

[39] Donald M. Topkis. Minimizing a submodular function on a lattice. *Operations Research*, 26(2):305–321, 1978.