Reflection methods for user-friendly submodular optimization

Stefanie Jegelka, Francis Bach, Suvrit Sra

Reflection methods for user-friendly submodular optimization

Stefanie Jegelka
UC Berkeley
Berkeley, CA, USA

Francis Bach
INRIA - ENS
Paris, France

Suvrit Sra
MPI for Intelligent Systems
Tübingen, Germany

Abstract

Recently, it has become evident that submodularity naturally captures widely occurring concepts in machine learning, signal processing and computer vision. Consequently, there is need for efficient optimization procedures for submodular functions, especially for minimization problems. While general submodular minimization is challenging, we propose a new method that exploits existing decomposability of submodular functions. In contrast to previous approaches, our method is neither approximate, nor impractical, nor does it need any cumbersome parameter tuning. Moreover, it is easy to implement and parallelize. A key component of our method is a formulation of the discrete submodular minimization problem as a continuous best approximation problem that is solved through a sequence of reflections, and its solution can be easily thresholded to obtain an optimal discrete solution. This method solves *both* the continuous and discrete formulations of the problem, and therefore has applications in learning, inference, and reconstruction. In our experiments, we illustrate the benefits of our method on two image segmentation tasks.

1 Introduction

Submodularity is a rich combinatorial concept that expresses widely occurring phenomena such as diminishing marginal costs and preferences for grouping. A set function $F : 2^V \to \mathbb{R}$ on a set V is *submodular* if for all subsets $S, T \subseteq V$, we have $F(S \cup T) + F(S \cap T) \leq F(S) + F(T)$.

Submodular functions underlie the goals of numerous problems in machine learning, computer vision and signal processing [1]. Several problems in these areas can be phrased as submodular optimization tasks: notable examples include graph cut-based image segmentation [9], sensor placement [32], or document summarization [34]. A longer list of examples may be found in [1].

The theoretical complexity of submodular optimization is well-understood: unconstrained minimization of submodular set functions is polynomial-time [21] while submodular maximization is NP-hard. Algorithmically, however, the picture is different. Generic submodular maximization admits efficient algorithms that can attain *approximate* optima with global guarantees; these algorithms are typically based on local search techniques [18, 38]. In contrast, although polynomial-time solvable, submodular function minimization (SFM) which seeks to solve

$$\min_{S \subseteq V} F(S), \tag{1}$$

poses substantial algorithmic difficulties. This is partly due to the fact that one is commonly interested in an exact solution (or an arbitrarily close approximation thereof), and "polynomial-time" is not necessarily equivalent to "practically fast".

Submodular minimization algorithms may be obtained from two main perspectives: *combinatorial* and *continuous*. Combinatorial algorithms for SFM typically use close connections to matroid and

maximum flow methods; the currently theoretically fastest combinatorial algorithm for SFM scales as $O(n^6 + n^5\tau)$, where τ is the time to evaluate the function oracle [40] (for an overview of other algorithms, see e.g., [36]). These combinatorial algorithms are typically nontrivial to implement.

Continuous methods offer an alternative by instead minimizing a *convex extension*. This idea exploits the fundamental connection between a submodular function F and its *Lovász extension* f [35], which is continuous and convex. The SFM problem (1) is then equivalent to

$$\min_{x \in [0,1]^n} f(x). \tag{2}$$

The Lovász extension f is nonsmooth, so we might have to resort to subgradient methods. While a fundamental result of Edmonds [17] demonstrates that a subgradient of f can be computed in $O(n \log n)$ time, subgradient methods can be sensitive to choices of the step size, and can be slow. They theoretically converge at a rate of $O(1/\sqrt{t})$ (after t iterations). The "smoothing technique" of [39] does not in general apply here because computing a smoothed gradient is equivalent to solving the submodular minimization problem. We discuss this issue further in Section 2.

An alternative to minimizing the Lovász extension directly on $[0, 1]^n$ is to consider a slightly modified convex problem. Specifically, the exact solution of the discrete problem $\min_{S \subseteq V} F(S)$ and of its nonsmooth convex relaxation $\min_{x \in [0,1]^n} f(x)$ may be found as a level set $S_0 = \{k \mid x^*_k \geqslant 0\}$ of the unique point x^* that minimizes the strongly convex function [1, 12]:

$$f(x) + \tfrac{1}{2}\|x\|^2. \tag{3}$$

We will refer to the minimization of (3) as the *proximal* problem due to its close similarity to proximity operators used in convex optimization [14]. When F is a cut function, (3) becomes a total variation problem (see, e.g., [11] and references therein) that also occurs in other regularization problems [1]. Two noteworthy points about (3) are: (i) addition of the strongly convex component $\frac{1}{2}\|x\|^2$; (ii) the ensuing removal of the *box-constraints* $x \in [0, 1]^n$. These changes allow us to consider a convex dual which is amenable to smooth optimization techniques.

Typical approaches to generic SFM include Frank-Wolfe methods [19] that have cheap iterations and $O(1/t)$ convergence, but can be quite slow in practice (Section 5); or the minimum-norm-point/Fujishige-Wolfe algorithm [22] that has expensive iterations but finite convergence. Other recent methods are approximate [26]. In contrast to several iterative methods based on convex relaxations, we seek to obtain exact discrete solutions.

To the best of our knowledge, all generic algorithms that use only submodularity are several orders of magnitude slower than specialized algorithms when they exist (e.g., for graph cuts). However, the submodular function is not always generic and given via a black-box, but has known structure. Following [29, 31, 41, 44], we make the assumption that $F(S) = \sum_{i=1}^{r} F_i(S)$ is a sum of sufficiently "simple" functions (see Sec. 3). This structure allows the use of (parallelizable) dual decomposition techniques for the problem in Eq. (2), with [13, 41] or without [31] Nesterov's smoothing technique, or with direct smoothing [44] techniques. But existing approaches typically have two drawbacks: (a) they use smoothing or step-size parameters whose selection may be critical and quite tedious; and (b) they still exhibit slow convergence (see Section 5).

These drawbacks arise from working with formulation (2). Our main insight is that, despite seemingly counter-intuitive, the proximal problem (3) offers a much more user-friendly tool for solving (1) than its natural convex counterpart (2), both in implementation and running time. We approach Problem (3) via its dual. This allows decomposition techniques which combine well with orthogonal projection and reflection methods that (a) exhibit faster convergence, (b) are easily parallelizable, (c) require no extra hyperparameters, and (d) are extremely easy to implement.

The main three algorithms that we consider are: (i) dual block-coordinate descent (equivalently, primal-dual proximal-Dykstra), which was already shown to be extremely efficient for total variation problems [3] that are special cases of Problem (3); (ii) Douglas-Rachford splitting using the careful variant of [5], which for our formulation (Section 4.2) requires no hyper-parameters; and (iii) accelerated projected gradient [6]. We will see these alternative algorithms can offer speedups beyond known efficiencies. Our observations have two implications: first, from the viewpoint of solving Problem (3), they offers speedups for often occurring denoising and reconstruction problems that employ total variation. Second, our experiments suggest that projection and reflection methods can work very well for solving the combinatorial problem (1).

In summary, we make the following contributions:

(1) In Section 3, we cast the problem of minimizing decomposable submodular functions as an orthogonal projection problem and show how existing optimization techniques may be brought to bear on this problem, to obtain fast, easy-to-code and easily parallelizable algorithms. In addition, we show examples of classes of functions amenable to our approach. In particular, for *simple* functions, i.e., those for which minimizing $F(S) - a(S)$ is easy for all vectors[1] $a \in \mathbb{R}^n$, the problem in Eq. (3) may be solved in $O(\log \frac{1}{\varepsilon})$ calls to such minimization routines, to reach a precision ε (Section 2, 3, Appendix B). (2) In Section 5, we demonstrate the empirical gains of using accelerated proximal methods, Douglas-Rachford and block coordinate descent methods over existing approaches: fewer hyperparameters and faster convergence.

2 Review of relevant results from submodular analysis

The relevant concepts we review here are the Lovász extension, base polytopes of submodular functions, and relationships between proximal and discrete problems. For more details, see [1, 21].

Lovász extension and convexity. The power set 2^V may be naturally identified with the vertices of the hypercube, i.e., $\{0,1\}^n$. The Lovász extension f of any set function is defined by linear interpolation, so that for any $S \subset V$, $F(S) = f(1_S)$. It may be computed in closed form once the components of x are sorted: if $x_{\sigma(1)} \geqslant \cdots \geqslant x_{\sigma(n)}$, then $f(x) = \sum_{k=1}^n x_{\sigma(k)} \big[F(\{\sigma(1), \ldots, \sigma(k)\}) - F(\{\sigma(1), \ldots, \sigma(k-1)\})\big]$ [35]. For the graph cut function, f is the total variation.

In this paper, we are going to use two important results: (a) if the set function F is submodular, then its Lovász extension f is convex, and (b) minimizing the set function F is equivalent to minimizing $f(x)$ with respect to $x \in [0,1]^n$. Given $x \in [0,1]^n$, all of its level sets may be considered and the function may be evaluated (at most n times) to obtain a set S. Moreover, for a submodular function, the Lovász extension happens to be the support function of the base polytope $B(F)$ defined as

$$B(F) = \{y \in \mathbb{R}^n \mid \forall S \subset V,\ y(S) \leqslant F(S) \text{ and } y(V) = F(V)\},$$

that is $f(x) = \max_{y \in B(F)} y^\top x$ [17]. A maximizer of $y^\top x$ (and hence the value of $f(x)$), may be computed by the "greedy algorithm", which first sorts the components of w in decreasing order $x_{\sigma(1)} \geqslant \cdots \geqslant x_{\sigma(n)}$, and then compute $y_{\sigma(k)} = F(\{\sigma(1), \ldots, \sigma(k)\}) - F(\{\sigma(1), \ldots, \sigma(k-1)\})$. In other words, a linear function can be maximized over $B(F)$ in time $O(n \log n + n\tau)$ (note that the term $n\tau$ may be improved in many special cases). This is crucial for exploiting convex duality.

Dual of discrete problem. We may derive a dual problem to the discrete problem in Eq. (1) and the convex nonsmooth problem in Eq. (2), as follows:

$$\min_{S \subseteq V} F(S) = \min_{x \in [0,1]^n} f(x) = \min_{x \in [0,1]^n} \max_{y \in B(F)} y^\top x = \max_{y \in B(F)} \min_{x \in [0,1]^n} y^\top x = \max_{y \in B(F)} (y)_-(V), \quad (4)$$

where $(y)_- = \min\{y, 0\}$ applied elementwise. This allows to obtain dual certificates of optimality from any $y \in B(F)$ and $x \in [0,1]^n$.

Proximal problem. The optimization problem (3), i.e., $\min_{x \in \mathbb{R}^n} f(x) + \frac{1}{2}\|x\|^2$, has intricate relations to the SFM problem [12]. Given the unique optimal solution x^* of (3), the maximal (resp. minimal) optimizer of the SFM problem is the set S^* of nonnegative (resp. positive) elements of x^*. More precisely, solving (3) is equivalent to minimizing $F(S) + \mu|S|$ for all $\mu \in \mathbb{R}$. A solution $S_\mu \subseteq V$ is obtained from a solution x^* as $S_\mu^* = \{i \mid x_i^* \geqslant \mu\}$. Conversely, x^* may be obtained from all S_μ^* as $x_k^* = \sup\{\mu \in \mathbb{R} \mid k \in S_\mu^*\}$ for all $k \in V$. Moreover, if x is an ε-optimal solution of Eq. (3), then we may construct $\sqrt{\varepsilon n}$-optimal solutions for all S_μ [1; Prop. 10.5]. In practice, the duality gap of the discrete problem is usually much lower than that of the proximal version of the same problem, as we will see in Section 5. Note that the problem in Eq. (3) provides much more information than Eq. (2), as all μ-parameterized discrete problems are solved.

The dual problem of Problem (3) reads as follows:

$$\min_{x \in \mathbb{R}^n} f(x) + \tfrac{1}{2}\|x\|_2^2 = \min_{x \in \mathbb{R}^n} \max_{y \in B(F)} y^\top x + \tfrac{1}{2}\|x\|_2^2 = \max_{y \in B(F)} \min_{x \in \mathbb{R}^n} y^\top x + \tfrac{1}{2}\|x\|_2^2 = \max_{y \in B(F)} -\tfrac{1}{2}\|y\|_2^2,$$

where primal and dual variables are linked as $x = -y$. Observe that this dual problem is equivalent to finding the orthogonal projection of 0 onto $B(F)$.

[1] Every vector $a \in \mathbb{R}^n$ may be viewed as a modular (linear) set function: $a(S) \triangleq \sum_{i \in S} a(i)$.

Divide-and-conquer strategies for the proximal problems. Given a solution x^* of the proximal problem, we have seen how to get S^*_μ for any μ by simply thresholding x^* at μ. Conversely, one can recover x^* exactly from at most n well-chosen values of μ. A known divide-and-conquer strategy [21, 23] hinges upon the fact that for any μ, one can easily see which components of x^* are greater or smaller than μ by computing S^*_μ. The resulting algorithm makes $O(n)$ calls to the submodular function oracle. In Appendix B, we extend an alternative approach by Tarjan et al. [45] for cuts to general submodular functions and obtain a solution to (3) up to precision ε in $O(\min\{n, \log \frac{1}{\varepsilon}\})$ iterations. This result is particularly useful if our function F is a sum of functions for each of which by itself the SFM problem is easy. Beyond squared ℓ_2-norms, our algorithm equally applies to computing all minimizers of $f(x) + \sum_{j=1}^p h_j(x_j)$ for arbitrary smooth strictly convex functions h_j, $j = 1, \dots, n$.

3 Decomposition of submodular functions

Following [29, 31, 41, 44], we assume that our function F may be decomposed as the sum $F(S) = \sum_{j=1}^r F_j(S)$ of r "simple" functions. In this paper, by "simple" we mean functions G for which $G(S) - a(S)$ can be minimized efficiently for all vectors $a \in \mathbb{R}^n$ (more precisely, we require that $S \mapsto G(S \cup T) - a(S)$ can be minimized efficiently over all subsets of $V \setminus T$, for any $T \subseteq V$ and a). Efficiency may arise from the functional form of G, or from the fact that G has small support. For such functions, Problems (1) and (3) become

$$\min_{S \subseteq V} \sum_{j=1}^r F_j(S) = \min_{x \in [0,1]^n} \sum_{j=1}^r f_j(x) \qquad \min_{x \in \mathbb{R}^n} \sum_{j=1}^r f_j(x) + \tfrac{1}{2}\|x\|_2^2. \tag{5}$$

The key to the algorithms presented here is to be able to minimize $\frac{1}{2}\|x-z\|_2^2 + f_j(x)$, or equivalently, to orthogonally project z onto $B(F_j)$: $\min \frac{1}{2}\|y - z\|_2^2$ subject to $y \in B(F_j)$.

We next sketch some examples of functions F and their decompositions into simple functions F_j. As shown at the end of Section 2, projecting onto $B(F_j)$ is easy as soon as the corresponding submodular minimization problems are easy. Here we outline some cases for which specialized fast algorithms are known.

Graph cuts. A widely used class of submodular functions are graph cuts. Graphs may be decomposed into substructures such as trees, simple paths or single edges. Message passing algorithms apply to trees, while the proximal problem for paths is very efficiently solved by [3]. For single edges, it is solvable in closed form. Tree decompositions are common in graphical models, whereas path decompositions are frequently used for TV problems [3].

Concave functions. Another important class of submodular functions is that of concave functions of cardinality, i.e., $F_j(S) = h(|S|)$ for a concave function h. Problem (3) for such functions may be solved in $O(n \log n)$ time (see [20] and Appendix B). Functions of this class have been used in [26, 28, 44]. Such functions also include covering functions [44].

Hierarchical functions. Here, the ground set corresponds to the leaves of a rooted, undirected tree. Each node has a weight, and the cost of a set of nodes $S \subseteq V$ is the sum of the weights of all nodes in the smallest subtree (including the root) that spans S. This class of functions too admits to solve the proximal problem in $O(n \log n)$ time [24, 25]. Related tree functions have been considered in [27], where the elements v of the ground set are arranged in a tree of height d and each have a weight $w(v)$. Let $\mathrm{desc}(v)$ be the set of descendants of v in the tree. Then $F(S) = \sum_{v \in V} w(v) 1[\mathrm{dec}(v) \cap S \neq \emptyset]$. Jenatton et al. [27] show how to solve the proximal problem for such a function in time $O(nd)$.

Small support. Any general, potentially slower algorithm such as the minimum-norm-point algorithm can be applied if the support of each F_j is only a small subset of the ground set.

3.1 Dual decomposition of the nonsmooth problem

We first review existing dual decomposition techniques for the nonsmooth problem (1). We always assume that $F = \sum_{j=1}^r F_j$, and define $\mathcal{H}^r := \prod_{j=1}^r \mathbb{R}^n \simeq \mathbb{R}^{n \times r}$. We follow [31] to derive a dual formulation (see Appendix A):

Lemma 1. *The dual of Problem* (1) *may be written in terms of variables* $\lambda_1, \dots, \lambda_r \in \mathbb{R}^n$ *as*

$$\max \sum_{j=1}^r g_j(\lambda_j) \qquad \textit{s.t.} \quad \lambda \in \big\{(\lambda_1, \dots, \lambda_r) \in \mathcal{H}^r \mid \sum_{j=1}^r \lambda_j = 0\big\} \tag{6}$$

where $g_j(\lambda_j) = \min_{S \subset V} F_j(S) - \lambda_j(S)$ is a nonsmooth concave function.

The dual is the maximization of a nonsmooth concave function over a convex set, onto which it is easy to project: the projection of a vector y has j-th block equal to $y_j - \frac{1}{r}\sum_{k=1}^{r} y_k$. Moreover, in our setup, functions g_j and their subgradients may be computed efficiently through SFM.

We consider several existing alternatives for the minimization of $f(x)$ on $x \in [0,1]^n$, most of which use Lemma 1. Computing subgradients for any f_j means calling the greedy algorithm, which runs in time $O(n \log n)$. All of the following algorithms require the tuning of an appropriate step size.

Primal subgradient descent (primal-sgd): Agnostic to any decomposition properties, we may apply a standard simple subgradient method to f. A subgradient of f may be obtained from the subgradients of the components f_j. This algorithm converges at rate $O(1/\sqrt{t})$.

Dual subgradient descent (dual-sgd) [31]: Applying a subgradient method to the nonsmooth dual in Lemma 1 leads to a convergence rate of $O(1/\sqrt{t})$. Computing a subgradient requires minimizing the submodular functions F_j individually. In simulations, following [31], we consider a step-size rule similar to Polyak's rule (dual-sgd-P) [7], as well as a decaying step-size (dual-sgd-F), and use discrete optimization for all F_j.

Primal smoothing (primal-smooth) [44]: The nonsmooth primal may be smoothed in several ways by smoothing the f_j individually; one example is $\tilde{f}_j^{\varepsilon}(x_j) = \max_{y_j \in B(F_j)} y_j^\top x_j - \frac{\varepsilon}{2}\|y_j\|^2$. This leads to a function that is $(1/\varepsilon)$-smooth. Computing $\tilde{f}_j^{\varepsilon}$ means solving the proximal problem for F_j. The convergence rate is $O(1/t)$, but, apart from the step size which may be set relatively easily, the smoothing constant ε needs to be defined.

Dual smoothing (dual-smooth): Instead of the primal, the dual (6) may be smoothed, e.g., by entropy [10, 41] applied to each g_j as $\tilde{g}_j^{\varepsilon}(\lambda_j) = \min_{x \in [0,1]^n} f_j(x) + \varepsilon h(x)$ where $h(x)$ is a negative entropy. Again, the convergence rate is $O(1/t)$ but there are two free parameters (in particular the smoothing constant ε which is hard to tune). This method too requires solving proximal problems for all F_j in each iteration.

Dual smoothing with entropy also admits coordinate descent methods [37] that exploit the decomposition, but we do not compare to those here.

3.2 Dual decomposition methods for proximal problems

We may also consider Eq. (3) and first derive a dual problem using the same technique as in Section 3.1. Lemma 2 (proved in Appendix A) formally presents our dual formulation as a best approximation problem. The primal variable can be recovered as $x = -\sum_j y_j$.

Lemma 2. *The dual of Eq. (3) may be written as the* best approximation problem

$$\min_{\lambda, y} \quad \|y - \lambda\|_2^2 \qquad \textit{s.t. } \lambda \in \big\{(\lambda_1, \dots, \lambda_r) \in \mathcal{H}^r \mid \textstyle\sum_{j=1}^{r} \lambda_j = 0\big\}, \qquad y \in \prod_{j=1}^{r} B(F_j). \tag{7}$$

We can actually eliminate the λ_j and obtain the simpler looking dual problem

$$\max_{y} -\frac{1}{2}\Big\|\sum_{j=1}^{r} y_j\Big\|_2^2 \qquad \text{s.t.} \quad y_j \in B(F_j),\ j \in \{1, \dots, r\} \tag{8}$$

Such a dual was also used in [43]. In Section 5, we will see the effect of solving one of these duals or the other. For the simpler dual (8) the case $r = 2$ is of special interest; it reads

$$\max_{y_1 \in B(F_1),\ y_2 \in B(F_2)} -\frac{1}{2}\|y_1 + y_2\|_2^2 \quad \Longleftrightarrow \min_{y_1 \in B(F_1), -y_2 \in -B(F_2)} \|y_1 - (-y_2)\|_2. \tag{9}$$

We write Problem (9) in this suggestive form to highlight its key geometric structure: it is, like (7), a *best approximation problem*: i.e., the problem of finding the closest point between the polytopes $B(F_1)$ and $-B(F_2)$. Notice, however, that (7) is very different from (9)—the former operates in a product space while the latter does not, a difference that can have impact in practice (see Section 5). We are now ready to present algorithms that exploit our dual formulations.

4 Algorithms

We describe a few competing methods for solving our smooth dual formulations. We describe the details for the special 2-block case (9); the same arguments apply to the block dual from Lemma 2.

4.1 Block coordinate descent or proximal-Dykstra

Perhaps the simplest approach to solving (9) (viewed as a minimization problem) is to use a block coordinate descent (BCD) procedure, which in this case performs the alternating projections:

$$y_1^{k+1} \leftarrow \operatorname{argmin}_{y_1 \in B(F_1)} \|y_1 - (-y_2^k)\|_2^2; \qquad y_2^{k+1} \leftarrow \operatorname{argmin}_{y_2 \in B(F_2)} \|y_2 - (-y_1^{k+1})\|_2. \quad (10)$$

The iterations for solving (8) are analogous. This BCD method (applied to (9)) is equivalent to applying the so-called proximal-Dykstra method [14] to the primal problem. This may be seen by comparing the iterates. Notice that the BCD iteration (10) is nothing but alternating projections onto the convex polyhedra $B(F_1)$ and $B(F_2)$. There exists a large body of literature studying method of alternating projections—we refer the interested reader to the monograph [15] for further details.

However, despite its attractive simplicity, it is known that BCD (in its alternating projections form), can converge arbitrarily slowly [5] depending on the relative orientation of the convex sets onto which one projects. Thus, we turn to a potentially more effective method.

4.2 Douglas-Rachford splitting

The Douglas-Rachford (DR) splitting method [16] includes algorithms like ADMM as a special case [14]. It avoids the slowdowns alluded to above by replacing alternating projections with alternating "reflections". Formally, DR applies to convex problems of the form [4, 14]

$$\min_x \quad \phi_1(x) + \phi_2(x), \quad (11)$$

subject to the qualification $\operatorname{ri}(\operatorname{dom}\phi_1) \cap \operatorname{ri}(\operatorname{dom}\phi_2) \neq \varnothing$. To solve (11), DR starts with some z_0, and performs the three-step iteration (for $k \geq 0$):

$$1.\ x_k = \operatorname{prox}_{\phi_2}(z_k); \qquad 2.\ v_k = \operatorname{prox}_{\phi_1}(2x_k - z_k); \qquad 3.\ z_{k+1} = z_k + \gamma_k(v_k - z_k), \quad (12)$$

where $\gamma_k \in [0, 2]$ is a sequence of scalars that satisfy $\sum_k \gamma_k(2 - \gamma_k) = \infty$. The sequence $\{x_k\}$ produced by iteration (12) can be shown to converge to a solution of (11) [4; Thm. 25.6].

Introducing the *reflection operator*

$$R_\phi := 2\operatorname{prox}_\phi - \mathrm{I},$$

and setting $\gamma_k = 1$, the DR iteration (12) may be written in a more symmetric form as

$$x_k = \operatorname{prox}_{\phi_2}(z_k), \qquad z_{k+1} = \tfrac{1}{2}[R_{\phi_1}R_{\phi_2} + \mathrm{I}]z_k, \quad k \geq 0. \quad (13)$$

Applying DR to the duals (7) or (9), requires first putting them in the form (11), either by introducing extra variables or by going back to the primal, which is unnecessary. This is where the special structure of our dual problem proves crucial, a recognition that is subtle yet remarkably important.

Instead of applying DR to (9), consider the closely related problem

$$\min_y \quad \delta_1(y) + \delta_2^-(y), \quad (14)$$

where δ_1, δ_2^- are indicator functions for $B(F_1)$ and $-B(F_2)$, respectively. Applying DR directly to (14) does not work because usually $\operatorname{ri}(\operatorname{dom}\delta_1) \cap \operatorname{ri}(\operatorname{dom}\delta_2) = \varnothing$. Indeed, applying DR to (14) generates iterates that diverge to infinity [5; Thm. 3.13(ii)]. Fortunately, even though the DR iterates for (14) may diverge, Bauschke et al. [5] show how to extract convergent sequences from these iterates, which actually solve the corresponding best approximation problem; for us this is nothing but the dual (9) that we wanted to solve in the first place. Theorem 3, which is a simplified version of [5; Thm. 3.13], formalizes the above discussion.

Theorem 3. *[5] Let $\mathcal{A}$ and $\mathcal{B}$ be nonempty polyhedral convex sets. Let $\Pi_\mathcal{A}$ ($\Pi_\mathcal{B}$) denote orthogonal projection onto $\mathcal{A}$ ($\mathcal{B}$), and let $R_\mathcal{A} := 2\Pi_\mathcal{A} - \mathrm{I}$ (similarly $R_\mathcal{B}$) be the corresponding reflection operator. Let $\{z_k\}$ be the sequence generated by the DR method* (13) *applied to* (14)*. If $\mathcal{A} \cap \mathcal{B} \neq \varnothing$, then $\{z_k\}_{k\geq 0}$ converges weakly to a fixed-point of the operator $T := \frac{1}{2}[R_\mathcal{A}R_\mathcal{B} + \mathrm{I}]$; otherwise $\|z_k\|_2 \to \infty$. The sequences $\{x_k\}$ and $\{\Pi_\mathcal{A}\Pi_\mathcal{B}z_k\}$ are bounded; the weak cluster points of either of the two sequences*

$$\{(\Pi_\mathcal{A}R_\mathcal{B}z_k, x_k)\}_{k\geq 0} \quad \{(\Pi_\mathcal{A}x_k, x_k)\}_{k\geq 0}, \quad (15)$$

are solutions best approximation problem $\min_{a,b}\|a - b\|$ such that $a \in \mathcal{A}$ and $b \in \mathcal{B}$.

The key consequence of Theorem 3 is that we can apply DR with impunity to (14), and extract from its iterates the optimal solution to problem (9) (from which recovering the primal is trivial). The most important feature of solving the dual (9) in this way is that absolutely no stepsize tuning is required, making the method very practical and user friendly (see also Appendix D).

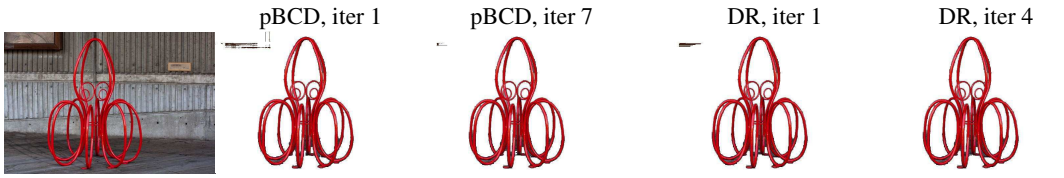

Figure 1: Segmentation results for the slowest and fastest projection method, with smooth (ν_s) and discrete (ν_d) duality gaps. Note how the background noise disappears only for small duality gaps.

5 Experiments

We empirically compare the proposed projection methods[2] to the (smoothed) subgradient methods discussed in Section 3.1. For solving the proximal problem, we apply block coordinate descent (BCD) and Douglas-Rachford (DR) to Problem (8) if applicable, and also to (7) (BCD-para, DR-para). In addition, we use acceleration to solve (8) or (9) [6]. The main iteration cost of all methods except for the primal subgradient method is the orthogonal projection onto polytopes $B(F_j)$, and therefore the number of iterations is a suitable criterion for comparisons. The primal subgradient method uses the greedy algorithm in each iteration, which runs in $O(n \log n)$. However, as we will see, its convergence is so slow to counteract any benefit that may arise from not using projections. We do not include Frank-Wolfe methods here, since FW is equivalent to a subgradient descent on the primal and converges correspondingly slowly.

As benchmark problems, we use (i) graph cut problems for segmentation, or MAP inference in a 4-neighborhood grid-structured MRF, and (ii) concave functions similar to those used in [44], but together with graph cut functions. The segmentation problems (i) are set up in a fairly standard way on a 4-neighbor grid graph, with unary potentials derived from Gaussian Mixture Models of color features. The weight of graph edge (i, j) is a function of $\exp(-\|y_i - y_j\|^2)$, where y_i is the RGB color vector of pixel i. The functions in (i) decompose as sums over vertical and horizontal paths. All horizontal paths are independent and can be solved together in parallel, and similarly all vertical paths. The functions in (ii) are constructed by extracting regions R_j via superpixels [33] and, for each R_j, defining the function $F_j(S) = |S||R_j \setminus S|$. We use 200 and 500 regions. The problems have size 640×427. Hence, for (i) we have $r = 640 + 427$ (but solve it as $r = 2$) and for (ii) $r = 640 + 427 + 500$ (solved as $r = 3$).

For algorithms working with formulation (7), we compute an improved smooth duality gap of a current primary solution $x = -\sum_j y_j$ as follows: find $y' \in \operatorname{argmax}_{y \in B(F)} x^\top y$ (then $f(x) = x^\top y'$) and find an improved x' by minimizing $\min_z z^\top y' + \frac{1}{2}\|z\|^2$ subject to the constraint that z has the same ordering as x [1]. The constraint ensures that $(x')^\top y' = f(x')$. This is an isotonic regression problem and can be solved in time $O(n)$ using the "pool adjacent violators" algorithm [1]. The gap is then $f(x') + \frac{1}{2}\|x'\|^2 - (-\frac{1}{2}\|y'\|^2)$.

For computing the discrete gap, we find the best level set S_i of x and, using $y' = -x$, compute $\min_i F(S_i) - y'_-(V)$.

Two functions ($r = 2$). Figure 2 shows the duality gaps for the discrete and smooth (where applicable) problems for two instances of segmentation problems. The algorithms working with the proximal problems are much faster than the ones directly solving the nonsmooth problem. In particular DR converges extremely fast, faster even than BCD which is known to be a state-of-the-art algorithms for this problem [3]. This, in itself, is a new insight for solving TV. We also see that the discrete gap shrinks faster than the smooth gap, i.e., the optimal discrete solution does not require to solve the smooth problem to extremely high accuracy. Figure 1 illustrates example results for different gaps.

More functions ($r > 2$). Figure 3 shows example results for four problems of sums of concave and cut functions. Here, we can only run DR-para. Overall, BCD, DR-para and the accelerated gradient method perform very well.

[2]Code and data corresponding to this paper are available at *https://sites.google.com/site/mloptstat/drsubmod*

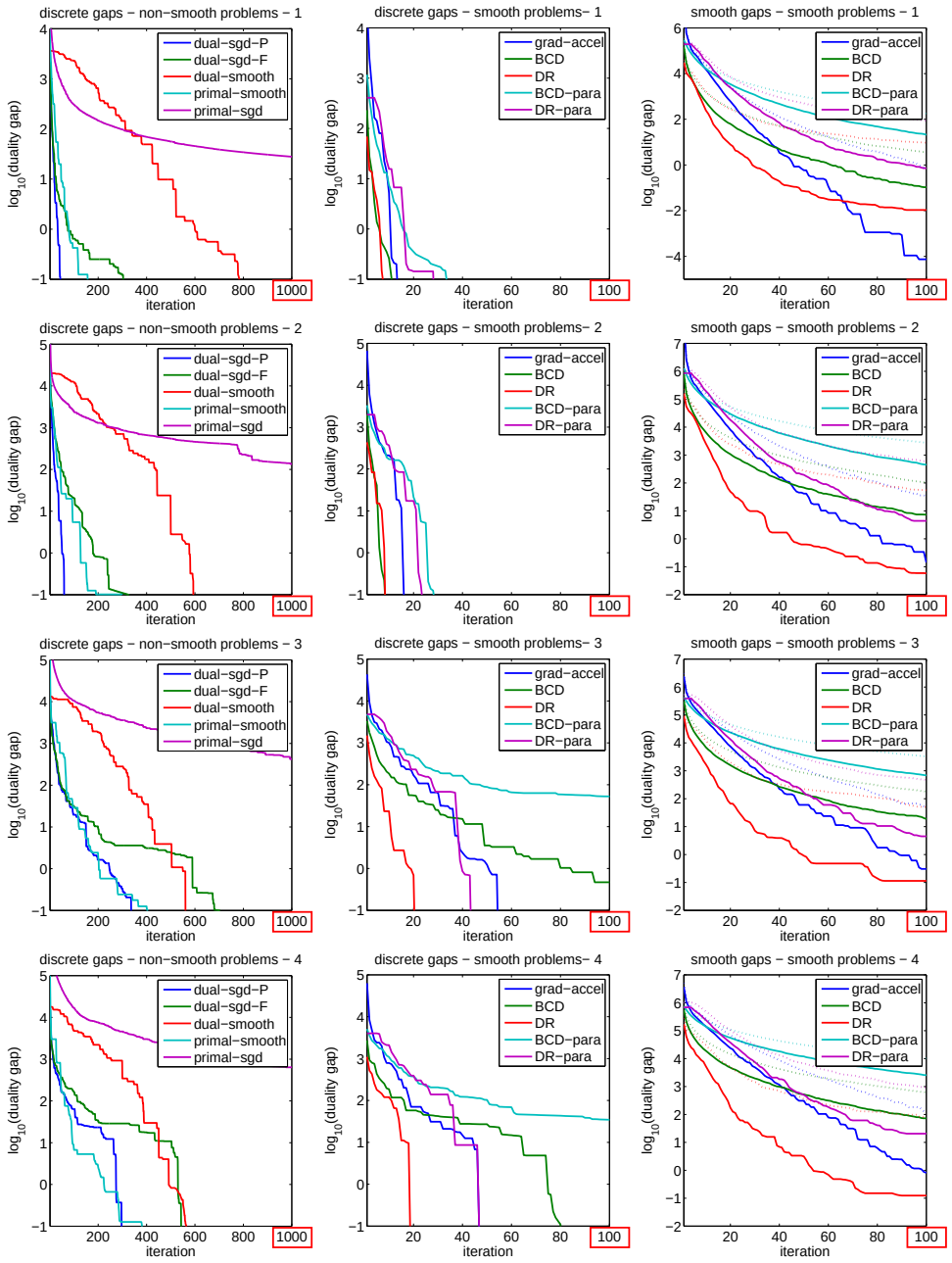

Figure 2: Comparison of convergence behaviors. Left: discrete duality gaps for various optimization schemes for the nonsmooth problem, from 1 to 1000 iterations. Middle: discrete duality gaps for various optimization schemes for the smooth problem, from 1 to 100 iterations. Right: corresponding continuous duality gaps. From top to bottom: four different images.

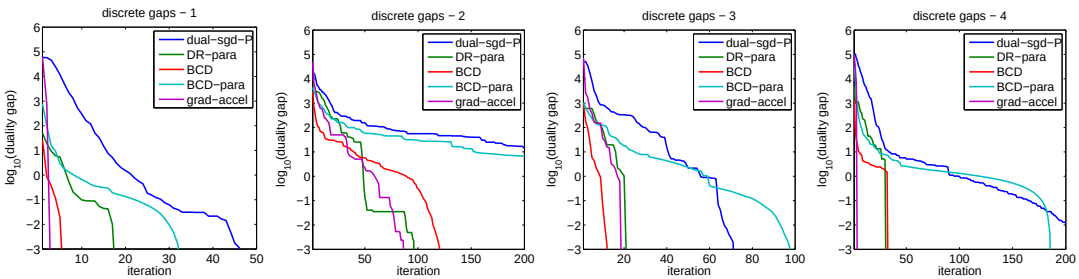

Figure 3: Convergence behavior for graph cut plus concave functions.

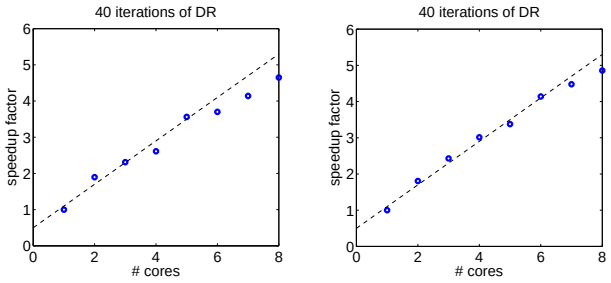

Figure 4: Speedup due to parallel processing for two instances.

	Maxflow	DR 1-thread	DR 2-thread	DR 4-thread
image 1	0.39	1.61 (4.13)	0.93 (2.39)	0.65 (1.66)
image 2	0.32	1.74 (5.45)	0.99 (3.10)	0.69 (2.16)
image 3	0.40	3.45 (8.61)	1.93 (4.82)	1.31 (3.27)
image 4	0.38	3.38 (8.88)	1.90 (5.00)	1.29 (3.38)
average	0.37	2.55	1.44	0.98

Table 1: Running times (in seconds) for the optimized C++ Maxflow code of [8, 9, 30] and our DR for graph cut using one or multiple threads. The last row is the average, and the numbers in parentheses indicate the factor relative to the Maxflow time.

Parallel speedups If we aim for parallel methods, then again DR outperforms BCD. Figure 4 (right) shows the speedup gained from parallel processing for $r = 2$. Using 8 cores, we obtain a 5-fold speed-up.

Running time compared to graph cuts Table 1 shows the running times of our DR method (implemented in Matlab/C++) and the Maxflow code of [8, 9, 30] (using the wrapper [2]) for the four graph cut (segmentation) instances above on a MacBook Air with a 2 GHz Intel Core i7. The running times are averages over 5 repetitions. DR was run for 10, 10, 21, and 20 iterations, respectively.

DR is by a factor of 2-9 slower than the specialized code. Given that, as opposed to the combinatorial algorithm, DR solves the full regularization path, is parallelizable, generic and straightforwardly extends to a variety of functions, this is remarkable.

In summary, our experiments suggest that projection methods can be extremely useful for solving the combinatorial submodular minimization problem. Of the tested methods, DR, cyclic BCD and accelerated gradient perform very well. For parallelism, applying DR on (9) converges much faster than BCD on the same problem.

6 Conclusion

We have presented a novel approach to submodular function minimization based on the equivalence with a best approximation problem. The use of reflection methods avoids any hyperparameters and reduce the number of iterations significantly, suggesting the suitability of reflection methods for combinatorial problems. Given the natural parallelization abilities of our approach, it would be interesting to perform detailed empirical comparisons with existing parallel implementations of graph cuts (e.g., [42]). Moreover, a generalization beyond submodular functions of the relationships between combinatorial optimization problems and convex problems would enable the application of our framework to other common situations such as multiple labels (see, e.g., [31]).

Acknowledgments. This research was in part funded by the Office of Naval Research under contract/grant number N00014-11-1-0688, by NSF CISE Expeditions award CCF-1139158, by DARPA XData Award FA8750-12-2-0331, and the European Research Council (SIERRA project), as well as gifts from Amazon Web Services, Google, SAP, Blue Goji, Cisco, Clearstory Data, Cloudera, Ericsson, Facebook, General Electric, Hortonworks, Intel, Microsoft, NetApp, Oracle, Samsung, Splunk, VMware and Yahoo!. We would like to thank Martin Jaggi, Simon Lacoste-Julien and Mark Schmidt for discussions.

References

[1] F. Bach. Learning with submodular functions: A convex optimization perspective. *Arxiv preprint arXiv:1111.6453v2*, 2013.

[2] Shai Bagon. Matlab wrapper for graph cut, 2006.

[3] A. Barbero and S. Sra. Fast Newton-type methods for total variation regularization. In *Int. Conference on Machine Learning (ICML)*, 2011.

[4] H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 2011.

[5] H. H. Bauschke, P. L. Combettes, and D. R. Luke. Finding best approximation pairs relative to two closed convex sets in Hilbert spaces. *Journal of Approximation Theory*, 127(2):178–192, 2004.

[6] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[7] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[8] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26 (9):1124–1137, 2004.

[9] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[10] B.Savchynskyy, S.Schmidt, J.H.Kappes, and C.Schnörr. Efficient MRF energy minimization via adaptive diminishing smoothing. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012.

[11] A. Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20(1):89–97, 2004.

[12] A. Chambolle and J. Darbon. On total variation minimization and surface evolution using parametric maximum flows. *Int. Journal of Comp. Vision*, 84(3):288–307, 2009.

[13] F. Chudak and K. Nagano. Efficient solutions to relaxations of combinatorial problems with submodular penalties via the Lovász extension and non-smooth convex optimization. In *ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2007.

[14] P. L. Combettes and J.-C. Pesquet. Proximal Splitting Methods in Signal Processing. In *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pages 185–212. Springer, 2011.

[15] F. R. Deutsch. *Best Approximation in Inner Product Spaces*. Springer Verlag, first edition, 2001.

[16] J. Douglas and H. H. Rachford. On the numerical solution of the heat conduction problem in 2 and 3 space variables. *Trans. of the American Mathematical Society*, 82:421–439, 1956.

[17] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial optimization - Eureka, you shrink!*, pages 11–26. Springer, 2003.

[18] U. Feige, V. S. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

[19] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[20] S. Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, pages 186–196, 1980.

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

[22] S. Fujishige and S. Isotani. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7:3–17, 2011.

[23] H. Groenevelt. Two algorithms for maximizing a separable concave function over a polymatroid feasible region. *European Journal of Operational Research*, 54(2):227–236, 1991.

[24] D.S. Hochbaum and S.-P. Hong. About strongly polynomial time algorithms for quadratic optimization over submodular constraints. *Mathematical Programming*, pages 269–309, 1995.

[25] S. Iwata and N. Zuiki. A network flow approach to cost allocation for rooted trees. *Networks*, 44:297–301, 2004.

[26] S. Jegelka, H. Lin, and J. Bilmes. On fast approximate submodular minimization. In *Neural Information Processing Systems (NIPS)*, 2011.

[27] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for hierarchical sparse coding. *Journal of Machine Learning Research*, pages 2297–2334, 2011.

[28] P. Kohli, L. Ladický, and P. Torr. Robust higher order potentials for enforcing label consistency. *Int. Journal of Comp. Vision*, 82, 2009.

[29] V. Kolmogorov. Minimizing a sum of submodular functions. *Discrete Applied Mathematics*, 160(15), 2012.

[30] V. Kolmogorov and R. Zabih. What energy functions can be minimized by graph cuts? *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

[31] N. Komodakis, N. Paragios, and G. Tziritas. MRF energy minimization and beyond via dual decomposition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 33(3):531–552, 2011.

[32] A. Krause and C. Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology*, 2(4), 2011.

[33] A. Levinshtein, A. Stere, K.N. Kutulakos, D.J. Fleet, S.J. Dickinson, and K. Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 31(12), 2009.

[34] H. Lin and J. Bilmes. A class of submodular functions for document summarization. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL/HLT)*, 2011.

[35] L. Lovász. Submodular functions and convexity. *Mathematical programming: the state of the art, Bonn*, pages 235–257, 1982.

[36] S. T. McCormick. Submodular function minimization. *Discrete Optimization*, 12:321–391, 2005.

[37] O. Meshi, T. Jaakkola, and A. Globerson. Convergence rate analysis of MAP coordinate minimization algorithms. In *Neural Information Processing Systems (NIPS)*, 2012.

[38] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions–I. *Mathematical Programming*, 14(1):265–294, 1978.

[39] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103 (1):127–152, 2005.

[40] J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.

[41] B. Savchynskyy, S. Schmidt, J. Kappes, and C. Schnörr. A study of Nesterov's scheme for Lagrangian decomposition and MAP labeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[42] A. Shekhovtsov and V. Hlavác. A distributed mincut/maxflow algorithm combining path augmentation and push-relabel. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, 2011.

[43] P. Stobbe. *Convex Analysis for Minimizing and Learning Submodular Set functions*. PhD thesis, California Institute of Technology, 2013.

[44] P. Stobbe and A. Krause. Efficient minimization of decomposable submodular functions. In *Neural Information Processing Systems (NIPS)*, 2010.

[45] R. Tarjan, J. Ward, B. Zhang, Y. Zhou, and J. Mao. Balancing applied to maximum network flow problems. In *European Symp. on Algorithms (ESA)*, pages 612–623, 2006.

A Derivations of Dual Problems

A.1 Proof of Lemma 1

Proof. To derive the non-smooth dual problem, we follow [31] and use Lagrangian duality:

$$\begin{aligned}
\min_{x\in[0,1]^n} f(x) &= \min_{x\in[0,1]^n} \sum_{j=1}^r f_j(x) = \min_{x_1,\dots,x_r\in[0,1]^n} \sum_{j=1}^r f_j(x_j) \text{ such that } x_1=\dots=x_r \\
&= \min_{x\in\mathbb{R}^n,\ x_1,\dots,x_r\in[0,1]^n} \max_{(\lambda_j)} \sum_{j=1}^r f_j(x_j) + \sum_{j=1}^r \lambda_j^\top (x-x_j) \\
&= \max_{\sum_{j=1}^r \lambda_j=0} \sum_{j=1}^r \min_{x_j\in[0,1]^n} \{f_j(x_j) - \lambda_j^\top x_j\} \\
&= \max_{\sum_{j=1}^r \lambda_j=0} \sum_{j=1}^r \max_{y_j\in B(F_j)} (y_j-\lambda_j)_-(V) = \max_{\sum_{j=1}^r \lambda_j=0} \sum_{j=1}^r g_j(\lambda_j),
\end{aligned}$$

where $g_j(\lambda_j) = \min_{A\subset V} F_j(A) - \lambda_j(A)$ is a nonsmooth concave function, which may be computed efficiently through submodular function minimization. □

A.2 Proof of Lemma 2

Proof. The proof follows a similar saddle-point approach.

$$\begin{aligned}
\min_{x\in\mathbb{R}^n} f(x) + \tfrac{1}{2}\|x\|_2^2 &= \min_{x\in\mathbb{R}^n} \sum_{j=1}^r f_j(x) + \tfrac{1}{2}\|x\|_2^2 \\
&= \min_{x_1,\dots,x_r\in\mathbb{R}^n} \sum_{j=1}^r \Big\{ f_j(x_j) + \frac{1}{2r}\|x_j\|_2^2 \Big\} \text{ such that } x_1=\dots=x_r \\
&= \min_{x\in\mathbb{R}^n,\ x_1,\dots,x_r\in\mathbb{R}^n} \max_{\lambda_j} \sum_{j=1}^r \big\{ f_j(x_j) + \frac{1}{2r}\|x_j\|_2^2 + \lambda_j^\top (x-x_j) \big\} \\
&= \max_{\sum_{j=1}^r \lambda_j=0} \sum_{j=1}^r \min_{x_j\in\mathbb{R}^n} \Big\{ f_j(x_j) - \lambda_j^\top x_j + \tfrac{1}{2r}\|x_j\|_2^2 \Big\} \\
&= \max_{\sum_{j=1}^r \lambda_j=0} \sum_{j=1}^r \min_{x_j\in\mathbb{R}^n} \Big\{ \max_{y_j\in B(F_j)} x_j^\top y_j - \lambda_j^\top x_j + \tfrac{1}{2r}\|x_j\|_2^2 \Big\} \\
&= \max_{\sum_{j=1}^r \lambda_j=0} \sum_{j=1}^r \max_{y_j\in B(F_j)} -\tfrac{r}{2}\|y_j-\lambda_j\|_2^2 \\
&= \max_{\sum_{j=1}^r \lambda_j=0} \max_{y_j\in B(F_j)} -\frac{r}{2}\sum_{j=1}^r \|y_j-\lambda_j\|_2^2. \qquad (16)
\end{aligned}$$

Writing (16) as a minimization problem and ignoring constants completes the proof. □

B Divide-and-conquer algorithm for parametric submodular minimization

B.1 Description of the algorithm

The optimal solution x^* of our proximal problem $\min_{x\in\mathbb{R}^n} f(x) + \|x\|^2$ indicates the minimizers of $F(S) - \lambda|S|$ for all $\lambda \in \mathbb{R}$. Those minimizers form a chain $S_\emptyset \subset S_1 \subset \ldots \subset S_k = V$. The solutions are the level sets of the optimal solution x^*.

Here, we extend the approach of Tarjan et al. [45] for parametric max-flow to all submodular functions and all monotone strictly convex functions beyond the square functions used in the main paper. More precisely, we consider a submodular function F defined on $V = \{1,\dots,n\}$ and n differentiable strictly convex functions h_i such that their Fenchel-conjugates h_i^* have full domain, for

$i \in \{1, \ldots, n\}$. The functions h_i^* are then differentiable. We consider the following problem:

$$\min_{x \in \mathbb{R}^n} f(x) + \sum_{i=1}^n h_i(x_i) = \min_{x \in \mathbb{R}^n} \max_{y \in B(F)} y^\top x + \sum_{i=1}^n h_i(x_i) \quad (17)$$

$$= \max_{y \in B(F)} \min_{x \in \mathbb{R}^n} y^\top x + \sum_{i=1}^n h_i(x_i) \quad (18)$$

$$= \max_{y \in B(F)} -\sum_{i=1}^n h_i^*(-y_i). \quad (19)$$

The optimality conditions are

1. $y \in B(F)$,
2. $y^\top x = f(x)$,
3. $-y_i = h_i'(x_i) \Leftrightarrow x_i = (h_i^*)'(-y_i)$.

Let $\tau(V)$ be the time for minimizing the submodular function $F(S) + a(S)$ on the ground set V (for any $a \in \mathbb{R}^n$). For our complexity analysis, we make the assumption that minimizing the (contracted) function $F^{S,a}(T) \triangleq F(S \cup T) - F(S) + a(T)$ on the smaller ground set $U \subseteq V \setminus S$ (for any $a \in \mathbb{R}^n$, $S \subseteq V$, $U \subseteq V \setminus S$) takes time at most $\frac{|U|}{|V|}\tau(V)$. This is a reasonable assumption, because it essentially says that $\tau(V)$ grows at least linearly in the size of V. To our knowledge, even fast algorithms for special submodular functions take at least linear time.

We will also use the notation $F(S \mid T) \triangleq F(S \cup T) - F(S)$. For recursions, we use the *restriction* $F_S : 2^S \to \mathbb{R}$, $F_S(T) = F(T)$ of F to S and the *contraction* $F^S : 2^{V \setminus S} \to \mathbb{R}$, $F^S(T) = F(T \mid S)$ of F on S.

Algorithm 1: Recursive Divide-and-Conquer

`SplitInterval` ($\lambda_{\min}$, $\lambda_{\max}$, V, F, i)
if i *even* **then**
  `// unbalanced split`
  $\lambda \leftarrow \operatorname{argmin}_\lambda \sum_i h_i(\lambda) - \lambda F(V)$
  $A \leftarrow \operatorname{argmin}_{T \subseteq V} F(T) + \sum_{i \in T} h_i'(\lambda)$
  if $S = \emptyset$ *or* $S = V$ **then**
    return $x = \lambda \mathbf{1}_V$
  end
else
  `// balanced split`
  $\lambda \leftarrow (\lambda_{\min} + \lambda_{\max})/2$
  $S \leftarrow \operatorname{argmin}_{T \subseteq V} F(T) + \sum_{i \in T} h_i'(\lambda)$
  if $S = \emptyset$ **then**
    $x \leftarrow$ `SplitInterval` ($\lambda_{\min}$, λ, V, F, $i+1$)
    return x
  end
  if $S = V$ **then**
    $x \leftarrow$ `SplitInterval` (λ, $\lambda_{\max}$, V, F, $i+1$)
    return x
  end
end
`//` $S \neq \emptyset$ `and` $S \neq V$
$x_S \leftarrow$ `SplitInterval` ($\lambda_{\min}$, λ, S, F^A, $i+1$)
$x_{V \setminus S} \leftarrow$ `SplitInterval` (λ, $\lambda_{\max}$, $V \setminus S$, F_S, $i+1$)
return $[x_S, x_{V \setminus S}]$

Algorithm 1 is a divide-and-conquer algorithm. In each recursive call, it takes an interval $[\lambda_{\min}, \lambda_{\max}]$ in which all components of the optimal solution lie and either (a) shortens the search

interval for any break point, (b) finds the optimal (constant) value of x on a range of elements, or (c) recursively splits the problem into a set S and $V \setminus S$ with corresponding ranges for the values of x^* and finds the optimal values of x on the two subsets.

B.2 Review of related results

The goal of this appendix is to show Proposition 4 below. We first start by reviewing existing results regarding separable problems on the base polyhedron (see [1] for details).

It is known that if $y \in B(F)$, then $y_k \in \big[F(V) - F(V\backslash\{k\}), F(\{k\})\big]$; thus, the optimal solution x is such that $x_k \in \big[(h_k^*)'(-F(\{k\})), (h_k^*)'(F(V\backslash\{k\}) - F(V))\big]$. We therefore set the initial search range to

$$\lambda_{\min} = \min_{k\in V} (h_k^*)'(-F(\{k\})) \quad \text{and} \quad \lambda_{\max} = \max_{k\in V} (h_k^*)'(F(V\backslash\{k\}) - F(V)).$$

The algorithm relies on the following facts (see [1] for a proof). For all three propositions, we assume that the h_i are strictly convex, continuously differentiable functions on $\mathbb{R}$ such that $\sup_{\lambda\in\mathbb{R}} h'(\lambda) = +\infty$ and $\inf_{\lambda\in\mathbb{R}} h'(\lambda) = -\infty$.

Proposition 1 (Monotonicity of optimizing sets)**.** *Let $\alpha < \beta$ and S^α be any minimizer of $F(S) + h'(\alpha)(S)$ and S^β any minimizer of $F(S) + h'(\beta)(S)$. Then $S^\beta \subseteq S^\alpha$.*

Proposition 2 (Characterization of x^*)**.** *The coordinates x_j^* ($j \in V$) of the unique optimal solution x^* of Problem 17 are*

$$x_j^* = \max\{\lambda \mid j \in S^\lambda\},$$

where S^λ is any minimizer of $F(S) + h'(\lambda)(S)$.

Propositions 1 and 2 imply that the level sets of x^* form a chain $\emptyset = S_0 \subset S_1 \subset \ldots \subset S_k = V$ of maximal minimizers for the critical values of λ (which are the entries of x^*). (Each $S_i = S^\lambda$ for some $\lambda = x_j^*$.)

Proposition 3 (Splits)**.** *Let $T = S_i$ be a level set of x^* and let $y \in \mathbb{R}^T, z \in \mathbb{R}^{V\setminus T}$ be the minimizers of the subproblems*

$$y = \operatorname*{argmin}_x f_T(x) + \sum_{i\in T} h_i(x_i)$$
$$z = \operatorname*{argmin}_x f^T(x) + \sum_{i\notin T} h_i(x_i)$$

Then $x_j^* = y_j$ for $j \in T$ and $x_j^* = z_j$ for $j \in V \setminus T$.

The algorithm uses Proposition 3 recursively.

Proof. Let λ be the value in x^* defining $S_i = S^\lambda$. It is easy to see that the restriction F_T and the contraction F^T are both submodular. Hence, Propositions 1 and 2 hold for them.

Since the restriction on T is equivalent to the original function for any $S \subseteq T$, $F(S) + h(\lambda)(S) = F_T(S) + h_T(\lambda)(S)$ for any $S \subseteq T$. With this, Propositions 1 and 2 imply that for any $\alpha > \lambda$, $F(S) + h(\lambda)(S) = F_T(S) + h_T(\lambda)(S)$ and therefore $x_j^* = y_j$ for $j \in T$.

Similarly, for any $S \in V \setminus T$, it holds that $F(S \cup T) + h(\lambda)(S \cup T) = F^T(S) + (h'(\lambda))^T(S) + F(T) + h'(\lambda)(T)$. Due to the monotonicity property of the optimizing sets, $S^\alpha \supseteq T$ for all $\alpha < \lambda$, and therefore the maximal minimizer U^α of $F^T(S) + (h'(\alpha))^T(S)$ satisfies $U^\alpha \cup T = S^\alpha$ (the terms $F(T) + h'(\lambda)(T)$ are constant with respect to U). Hence Poposition 2 implies that $x_j^* = z_j$ for $j \in V \setminus T$. □

These propositions imply that there is a set of at most n values of $\lambda = \alpha$ that define the level sets S^α of the optimal solution x^*. If we know these break point values, then we know x^*. Algorithm 1 interleaves an unbalanced split strategy that may split the seach interval in an unbalanced way but converges in $O(n)$ recursive calls, and a balanced split strategy that always halves the search intervals but is not finitely convergent.

B.3 Proof of convergence

We now prove the convergence rate for Algorithm 1.

Proposition 4. *The minimum of $f(x) + \sum_{i=1}^n h_i(x_i)$ may be obtained up to coordinate-wise accuracy ϵ within*

$$O\left(\min\{n, \log \tfrac{1}{\epsilon}\}\right) \tag{20}$$

submodular function minimizations. If $h_i(x_i) = \frac{1}{2}x_i^2$, then $\epsilon = \frac{\Delta_{\min}}{n^2 \ell_0}$ is sufficient to recover the exact solution, where $\Delta_{\min} = \min\{|F(S \mid T)| \mid S \subseteq V \setminus T, F(S \mid T) \neq 0\}$ and ℓ_0 is the length of the initial interval $[\lambda_{\min}, \lambda_{\max}]$.

Proof. The proof relies on Propositions 1, 2 and 3.

We first argue for the correctness of the balanced splitting strategy. Propositions 1 and 2 imply that for any $\lambda \in \mathbb{R}$, if S is a minimizer of $F(S) + h'(\lambda)(S)$, then the unique minimum of $f(x) + \sum_{i=1}^n h_i(x_i)$ satisfies that for all $k \in S$, $x_k \geqslant h_k'(\lambda)$ and for all $k \in V \backslash S, x_k \leqslant h_k'(\lambda)$. In particular, if $S = \varnothing$, then this means that for all $k \in V$, $x_k \leqslant h_k'(\lambda)$. Similarly, if $S = V$, then for all $k \in V$, $x_k \geqslant h_k'(\lambda)$. The limits of the interval are set accordingly. The correctness of the recursive call follows from Proposition 3.

In each iteration, the size of the search interval $[\lambda_{\min}, \lambda_{\max}]$ for any break point is at least halved. Hence, within d recursions, the length of each interval is at most $2^{-d}\ell_0$.

The choice of λ in the unbalanced splitting strategy corresponds to solving a simplified version of the dual problem. Indeed, by convex duality, the following two problems are dual to each other:

$$\max_y -\sum_i h_i^*(-y_i) \quad \text{s.t. } y(V) = F(V) \tag{21}$$

$$\min_{\lambda \in \mathbb{R}} \sum_{i \in V} h_i(\lambda) - \lambda F(V). \tag{22}$$

Problem (21) replaces the constraint that $y \in B(F)$ by $y(V) = F(V)$, dropping the constraint that $y(S) \leq F(S)$ for all $S \subseteq V$. Testing whether y satisfies all constraints of (19), i.e., $y \in B(F)$ is equivalent to testing whether $F(S) - y(S) \geq 0$. We do this implicitly by our choice of λ: Convex duality implies that the the optimal solutions of Problems (21) and (22) satisfy $y_i = -h_i'(\lambda)$. This holds in particular for the chosen (unique optimal) λ in the algorithm.

Let T be a minimizer of $F(S) + h'(\lambda)(S) = F(S) - y(S)$. If $T = \emptyset$ or $T = V$, then $y \in B(F)$ and an optimal solution for the full dual problem (19). Hence, y and $x = \lambda \mathbf{1}_V = (h^*)'(-y)$ form a primal/dual optimal pair for (19).

If $\emptyset \subset T \subset V$ and $F(T) - y(T) < 0$, then $y \notin B(F)$, and we perform a split with the same argumentation as above. This splitting strategy is exactly that of [1, 23] and splits at most n times. Hence, this strategy yields the global optimum (to machine precision) in the time of $O(n)$ times solving a submodular minimization on V. If n is large, this may be computationally expensive.

If we only do balanced splits, we end up approaching the break points more and more closely (but typically never exactly). Unbalanced splits always find an exact break point, but with potentially little progress in reducing the intervals. Algorithm 1 thus interleaves both strategies where we store intervals of allowed values for subsets of components of A. At step d there are at most $\min\{n, 2^d\}$ different intervals (as there cannot be more intervals than elements of V). To split these intervals, submodular function minimization problems have to be solved on each of these intervals, with total complexity less than a single submodular function optimization problem on the full set. At each iteration, intervals corresponding to a singleton may be trivially completely solved, and components which are already found are discarded. Hence, at each recursive level, the total computation time is bounded above by $\tau(V)$.

While balanced splits always substantially shrink the intervals, they are not finitely convergent. Unbalanced splits converge after at most n recursions. Following the argumentation of Tarjan et al. [45], who considered the special case of flows, alternating the two types of splits gives the best of both worlds: (a) all components are estimated up to precision $\frac{\ell_0}{2^{d/2}}$, and (b) the algorithm is finitely convergent, and will stop when $\frac{\ell_0}{2^{d/2}}$ is less than the minimal distance between two different components of x.

Finally, we adress the precision for the special case that $h_i(x_i) = \frac{1}{2}x_i^2$ for all $i \in V$. If the interval lengths are smaller than the smallest gap between any two break points (components of x^*), then each interval contains at most one break point and the algorithm converges after at most two unbalanced splits. Hence, we here consider ϵ to be one half times the smallest gap between any two break points. Let $\emptyset = S_0 \subset S_1 \subset \ldots \subset S_k = V$ be the chain of level sets of x^*. By the optimality conditions discussed above for unbalanced splits, any constant part $T = S_i \setminus S_{i-1}$ of x^* takes value $\lambda \mathbf{1} = -y_j \mathbf{1}$ ($j \in T$), where $y(T) = F_{S_{i-1}}(T)$, and hence

$$\lambda = -\frac{F(S_i \setminus S_{i-1}|S_{i-1})}{|S_i \setminus S_{i-1}|}. \tag{23}$$

Therefore, the (absolute) difference between any two such values is loosely lower bounded by

$$\min_i \left| \frac{F(S_i \setminus S_{i-1}|S_{i-1})}{|S_i \setminus S_{i-1}|} - \frac{F(S_{i+1} \setminus S_i|S_i)}{|S_{i+1} \setminus S_i|} \right| \geq \Delta_{\min} \left(\frac{2}{n-1} - \frac{2}{n} \right) \geq \frac{2\Delta_{\min}}{n^2}. \tag{24}$$

This implies $O(\log(\ell_0 n^2/\Delta_{\min}))$ iterations. □

Note that in the case of flows, the algorithm is not exactly equivalent to the flow algorithm of [45], which updates flows directly.

C BCD and proximal Dykstra

We consider the best approximation problem

$$\begin{aligned} \min \quad & \tfrac{1}{2}\|x - y\|_2^2 \\ \text{s.t.} \quad & x \in C_1 \cap C_2 \cap \cdots \cap C_m. \end{aligned}$$

Let us show the details for only the two block case. The general case follows similarly.

Consider the more general problem

$$\min \quad \tfrac{1}{2}\|x - y\|_2^2 + f(x) + h(x). \tag{25}$$

Clearly, this problem contains the two-block best approximation problem as a special case (by setting f and h to be suitable indicator functions). Now introduce two variables z, w that equal x; then the corresponding Lagrangian is

$$L(x, z, w, \nu, \mu) := \tfrac{1}{2}\|x - y\|_2^2 + f(z) + h(w) + \nu^T(x - z) + \mu^T(x - w).$$

From this Lagrangian, a brief calculation yields the dual optimization problem

$$\min \; g(\nu, \mu) := \tfrac{1}{2}\|\nu + \mu - y\|_2^2 + f^*(\nu) + h^*(\mu).$$

We solve this dual problem via BCD, which has the updates

$$\nu_{k+1} = \operatorname{argmin}_\nu \; g(\nu, \mu_k), \qquad \mu_{k+1} = \operatorname{argmin}_\mu \; g(\nu_{k+1}, \mu).$$

Thus, $0 \in \nu_{k+1} + \mu_k - y + \partial f^*(\nu_{k+1})$ and $0 \in \nu_{k+1} + \mu_{k+1} - y + \partial h^*(\mu_{k+1})$. The first optimality condition may be rewritten as

$$y - \mu_k \in \nu_{k+1} + \partial f^*(\nu_{k+1}) \implies \nu_{k+1} = \operatorname{prox}_{f^*}(y - \mu_k) \implies \nu_{k+1} = y - \mu_k - \operatorname{prox}_f(y - \mu_k).$$

Similarly, we second condition yields $\mu_{k+1} = y - \nu_{k+1} - \operatorname{prox}_h(y - \nu_{k+1})$. Now use Lagrangian stationarity

$$x = y - \nu - \mu \implies y - \mu = x + \nu$$

to rewrite BCD using primal and dual variables to obtain the so-called proximal-Dykstra method:

$$\begin{aligned} t_k &\leftarrow \operatorname{prox}_f(x_k + \nu_k) \\ \nu_{k+1} &\leftarrow x_k + \nu_k - t_k \\ x_{k+1} &\leftarrow \operatorname{prox}_h(\mu_k + t_k) \\ \mu_{k+1} &\leftarrow \mu_k + t_k - x_{k+1} \end{aligned}$$

We discussed the more general problem (25) because it contains the smoothed primal as a special case, namely with $y = 0$ in (25), $f = f_1$, and $h = f_2$, we obtain

$$\min \quad f_1(x) + f_2(x) + \tfrac{1}{2}\|x\|_2^2,$$

for which BCD yields the proximal-Dykstra method that was previously used in [3] for two-dimensional TV optimization.

D Recipe: Submodular minimization via reflections

To be precise, we summarize here how to solve Problem (17) via reflections. As we showed above, the dual is of the form

$$\min_{\lambda,y} \quad \|y-\lambda\|_2^2 \quad \text{s.t. } \lambda \in \mathcal{A} = \big\{(\lambda_1,\dots,\lambda_r) \in \mathcal{H}^r \mid \sum\nolimits_{j=1}^r \lambda_j = 0\big\}, \quad y \in \mathcal{B} \triangleq \prod\nolimits_{j=1}^r B(F_j). \tag{26}$$

The vector y consists of r parts $y_j \in B(F_j)$. We first solve the dual by starting with any $z^{(0)} \in \mathcal{H}^r$, and iterate

$$z^{(k+1)} = \tfrac{1}{2}(z^k + R_{\mathcal{A}} R_{\mathcal{B}}(z^{(k)})). \tag{27}$$

Upon convergence to a point z^*, we extract the components

$$y_j = \Pi_{B(F_j)}(z_j^*). \tag{28}$$

The final primal solution is $x = -\sum_j y_j \in \mathbb{R}^n$.