Decomposable Submodular Function Minimization Discrete and Continuous

Alina Ene[*] Huy L. Nguyễn[†] László A. Végh[‡]

Abstract

This paper investigates connections between discrete and continuous approaches for decomposable submodular function minimization. We provide improved running time estimates for the state-of-the-art continuous algorithms for the problem using combinatorial arguments. We also provide a systematic experimental comparison of the two types of methods, based on a clear distinction between level-0 and level-1 algorithms.

1 Introduction

Submodular functions arise in a wide range of applications: graph theory, optimization, economics, game theory, to name a few. A function $f : 2^V \to \mathbb{R}$ on a ground set V is *submodular* if $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$ for all sets $X, Y \subseteq V$. Submodularity can also be interpreted as a diminishing returns property.

There has been significant interest in submodular optimization in the machine learning and computer vision communities. The *submodular function minimization* (SFM) problem arises in problems in image segmentation or MAP inference tasks in Markov Random Fields. Landmark results in combinatorial optimization give polynomial-time exact algorithms for SFM. However, the high-degree polynomial dependence in the running time is prohibitive for large-scale problem instances. The main objective in this context is to develop fast and scalable SFM algorithms.

Instead of minimizing arbitrary submodular functions, several recent papers aim to exploit special structural properties of submodular functions arising in practical applications. This paper focuses on the popular model of *decomposable submodular functions*. These are functions that can be written as sums of several "simple" submodular functions defined on small supports.

Some definitions are needed to introduce our problem setting. Let $f : 2^V \to \mathbb{R}$ be a submodular function, and let $n := |V|$. We can assume w.l.o.g. that $f(\emptyset) = 0$. We are interested in solving the *submodular function minimization problem*:

$$\min_{S \subseteq V} f(S). \tag{SFM}$$

For a vector $y \in \mathbb{R}^V$ and a set $S \subseteq V$, we use the notation $y(S) := \sum_{v \in S} y(v)$. The base polytope of a submodular function is defined as

$$B(f) := \{y \in \mathbb{R}^V : y(S) \leq f(S)\ \forall S \subseteq V, y(V) = f(V)\}.$$

One can optimize linear functions over $B(f)$ using the greedy algorithm. The SFM problem can be reduced to finding the minimum norm point of the base polytope $B(f)$ [11].

$$\min \left\{ \frac{1}{2}\|y\|_2^2 \colon y \in B(f) \right\}. \tag{Min-Norm}$$

[*]Department of Computer Science, Boston University, aene@bu.edu

[†]College of Computer and Information Science, Northeastern University, hu.nguyen@northeastern.edu

[‡]Department of Mathematics, London School of Economics, L.Vegh@lse.ac.uk

This reduction is the starting point of convex optimization approaches for SFM. We refer the reader to Sections 44–45 in [29] for concepts and results in submodular optimization, and to [3] on machine learning applications.

We assume that f is given in the decomposition $f(S) = \sum_{i=1}^{r} f_i(S)$, where each $f_i : 2^V \to \mathbb{R}$ is a submodular function. Such functions are called *decomposable* or *Sum-of-Submodular (SoS)* in the literature. In the *decomposable submodular function minimization (DSFM)* problem, we aim to minimize a function given in such a decomposition. We will make the following assumptions.

For each $i \in [r]$, we assume that two oracles are provided: *(i)* a value oracle that returns $f_i(S)$ for any set $S \subseteq V$ in time EO_i; and *(ii)* a quadratic minimization oracle $\mathcal{O}_i(w)$. For any input vector $w \in \mathbb{R}^n$, this oracle returns an optimal solution to (Min-Norm) for the function $f_i + w$, or equivalently, an optimal solution to $\min_{y \in B(f_i)} \|y + w\|_2^2$. We let Θ_i denote the running time of a single call to the oracle $\mathcal{O}_i$, $\Theta_{\max} := \max_{i \in [r]} \Theta_i$ denote the maximum time of an oracle call, $\Theta_{\text{avg}} := \frac{1}{r} \sum_{i \in [r]} \Theta_i$ denote the average time of an oracle call.[4] We let $F_{i,\max} := \max_{S \subseteq V} |f_i(S)|$, $F_{\max} := \max_{S \subseteq V} |f(S)|$ denote the maximum function values. For each $i \in [r]$, the function f_i has an effective support C_i such that $f_i(S) = f_i(S \cap C_i)$ for every $S \subseteq V$.

DSFM thus requires algorithms on two levels. The *level-0* algorithms are the subroutines used to evaluate the oracles $\mathcal{O}_i$ for every $i \in [r]$. The *level-1* algorithm minimizes the function f using the level-0 algorithms as black boxes.

1.1 Prior work

SFM has had a long history in combinatorial optimization since the early 1970s, following the influential work of Edmonds [5]. The first polynomial-time algorithm was obtained via the ellipsoid method [15]; recent work presented substantial improvements using this approach [23]. Substantial work focused on designing strongly polynomial combinatorial algorithms [10, 16, 17, 26, 18, 28]. Still, designing practical algorithms for SFM that can be applied to large-scale problem instances remains an open problem.

Let us now turn to DSFM. Previous work mainly focused on level-1 algorithms. These can be classified as *discrete* and *continuous* optimization methods. The discrete approach builds on techniques of classical discrete algorithms for network flows and for submodular flows. Kolmogorov [22] showed that the problem can be reduced to submodular flow maximization, and also presented a more efficient augmenting path algorithm. Subsequent discrete approaches were given in [2, 8, 9]. Continuous approaches start with the convex programming formulation (Min-Norm). Gradient methods were applied for the decomposable setting in [6, 25, 31].

Less attention has been given to the level-0 algorithms. Some papers mainly focus on theoretical guarantees on the running time of level-1 algorithms, and treat the level-0 subroutines as black-boxes (e.g. [6, 25, 22]). In other papers (e.g. [19, 31]), the model is restricted to functions f_i of a simple specific type that are easy to minimize. An alternative assumption is that all C_i's are small, of size at most k; and thus these oracles can be evaluated by exhaustive search, in 2^k value oracle calls (e.g. [2, 8]). Shanu *et al.* [30] use a block coordinate descent method for level-1, and make no assumptions on the functions f_i. The oracles are evaluated via the Fujishige-Wolfe minimum norm point algorithm [12, 32] for level-0.

Let us note that these experimental studies considered the level-0 and level-1 algorithms as a single "package". For example, Shanu *et al.* [30] compare the performance of their *SoS Min-Norm algorithm* to the continuous approach of Jegelka *et al.* [19] and the combinatorial approach of Arora *et al.* [2]. However, these implementations cannot be directly compared, since they use three different level-0 algorithms: Fujishige-Wolfe in SoS Min-Norm, a general QP solver for the algorithm of [19], and exhaustive search for [2]. For potentials of large support, Fujishige-Wolfe outperforms these other level-0 subroutines, hence the level-1 algorithms in [19, 2] could have compared more favorably using the same Fujishige-Wolfe subroutine.

[4]For flow-type algorithms for DSFM, a slightly weaker oracle assumption suffices, returning a minimizer of $\min_{S \subseteq C_i} f_i(S) + w(S)$ for any given $w \in \mathbb{R}^{C_i}$. This oracle and the quadratic minimization oracle are reducible to each other: the former reduces to a single call to the latter, and one can implement the latter using $O(|C_i|)$ calls to the former (see e.g. [3]).

1.2 Our contributions

Our paper establishes connections between discrete and continuous methods for DSFM, as well as provides a systematic experimental comparison of these approaches. Our main theoretical contribution improves the worst-case complexity bound of the most recent continuous optimization methods [6, 25] by a factor of r, the number of functions in the decomposition. This is achieved by improving the bounds on the relevant condition numbers. Our proof exploits ideas from the discrete optimization approach. This provides not only better, but also considerably simpler arguments than the algebraic proof in [25].

The guiding principle of our experimental work is the clean conceptual distinction between the level-0 and level-1 algorithms, and to compare different level-1 algorithms by using the same level-0 subroutines. We compare the state-of-the-art continuous and discrete algorithms: RCDM and ACDM from [6] with Submodular IBFS from [8]. We consider multiple options for the level-0 subroutines. For certain potential types, we use tailored subroutines exploiting the specific form of the problem. We also consider a variant of the Fujishige-Wolfe algorithm as a subroutine applicable for arbitrary potentials.

Our experimental results reveal the following tradeoff. Discrete algorithms on level-1 require more calls to the level-0 oracle, but less overhead computation. Hence using algorithms such as IBFS on level-1 can be significantly faster than gradient descent, as long as the potentials have fairly small supports. However, as the size of the potentials grow, or we do need to work with a generic level-0 algorithm, gradient methods are preferable. Gradient methods can perform better for larger potentials also due to weaker requirements on the level-0 subroutines: approximate level-0 subroutines suffice for them, whereas discrete algorithms require exact optimal solutions on level-0.

Paper outline. The rest of the paper is structured as follows. The level-1 algorithmic frameworks using discrete and convex optimization are described in Sections 2 and 3, respectively. Section 4 gives improved convergence guarantees for the gradient descent algorithms outlined in Section 3. Section 5 discusses the different types of level-0 algorithms and how they can be used together with the level-1 frameworks. Section 6 presents a brief overview of our experimental results.

2 Discrete optimization algorithms on Level-1

In this section, we outline a level-1 algorithmic framework for DSFM that is based on a combinatorial framework first studied by Fujishige and Zhang [13] for submodular intersection. The submodular intersection problem is equivalent to DSFM for the sum of two functions, and the approach can be adapted and extended to the general DSFM problem with an arbitrary decomposition. We now give a brief description of the algorithmic framework. The Appendix exhibits submodular versions of the Edmonds-Karp and preflow-push algorithms.

Algorithmic framework. For a decomposable function f, every $x \in B(f)$ can be written as $x = \sum_{i=1}^r x_i$, where $\mathrm{supp}(x_i) \subseteq C_i$ and $x_i \in B(f_i)$ (see e.g. Theorem 44.6 in [29]). A natural algorithmic approach is to maintain an $x \in B(f)$ in such a representation, and iteratively update it using the combinatorial framework described below. DSFM can be cast as a maximum network flow problem in a network that is suitably defined based on the current point x. This can be viewed as an analogue of the residual graph in the maxflow/mincut setting, and it is precisely the residual graph if the DSFM instance is a minimum cut instance.

The auxiliary graph. For an $x \in B(f)$ of the form $x = \sum_{i=1}^r x_i$, we construct the following directed auxiliary graph $G = (V, E)$, with $E = \bigcup_{i=1}^r E_i$ and capacities $c : E \to \mathbb{R}_+$. E is a multiset union: we include parallel copies if the same arc occurs in multiple E_i. The arc sets E_i are complete directed graphs (cliques) on C_i, and for an arc $(u, v) \in E_i$, we define $c(u, v) := \min\{f_i(S) - x_i(S) \colon S \subseteq C_i, u \in S, v \notin S\}$. This is the maximum value ε such that $x'_i \in B(f_i)$, where $x'_i(u) = x_i(u) + \varepsilon$, $x'_i(v) = x_i(v) - \varepsilon$, $x'_i(z) = x_i(z)$ for $z \notin \{u, v\}$.

Let $N := \{v \in V \colon x(v) < 0\}$ and $P := \{v \in V \colon x(v) > 0\}$. The algorithm aims to improve the current x by updating along shortest directed paths from N to P with positive capacity; there are several ways to update the solution, and we discuss specific approaches (derived from maximum flow algorithms) in the Appendix. If there exists no such directed path, then we let S denote the set reachable from N on directed paths with positive capacity; thus, $S \cap P = \emptyset$. One can show that S is a minimizer of the function f.

Updating along a shortest path $\mathcal{Q}$ from N to P amounts to the following. Let ε denote the minimum capacity of an arc on $\mathcal{Q}$. If $(u, v) \in \mathcal{Q} \cap E_i$, then we increase $x_i(u)$ by ε and decrease $x_i(v)$ by ε. The crucial technical claim is the following. Let $d(u)$ denote the shortest path distance of positive capacity arcs from u to the set P. Then, an update along a shortest directed path from N to P results in a feasible $x \in B(f)$, and further, all distance labels $d(u)$ are non-decreasing. We refer the reader to Fujishige and Zhang [13] for a proof of this claim.

Level-1 algorithms based on the network flow approach. Using the auxiliary graph described above, and updating on shortest augmenting paths, one can generalize several maximum flow algorithms to a level-1 algorithm of DSFM. In particular, based on the preflow-push algorithm [14], one can obtain a strongly polynomial DSFM algorithm with running time $O(n^2 \Theta_{\max} \sum_{i=1}^{r} |C_i|^2)$. A scaling variant provides a weakly polynomial running time $O(n^2 \Theta_{\max} \log F_{\max} + n \sum_{i=1}^{r} |C_i|^3 \Theta_i)$. We defer the details to the Appendix.

In our experiments, we use the submodular IBFS algorithm [8] as the main discrete level-1 algorithm; the same running time estimate as for preflow-push is applicable. If all C_i's are small, $O(1)$, the running time is $O(n^2 r \Theta_{\max})$; note that $r = \Omega(n)$ in this case.

3 Convex optimization algorithms on Level-1

3.1 Convex formulations for DSFM

Recall the convex quadratic program (Min-Norm) from the Introduction. This program has a unique optimal solution s^*, and the set $S = \{v \in V : s^*(v) < 0\}$ is the unique smallest minimizer to the SFM problem. We will refer to this optimal solution s^* throughout the section.

In the DSFM setting, one can write (Min-Norm) in multiple equivalent forms [19]. For the first formulation, we let $\mathcal{P} := \prod_{i=1}^{r} B(f_i) \subseteq \mathbb{R}^{rn}$, and let $A \in \mathbb{R}^{n \times (rn)}$ denote the following matrix:

$$A := \underbrace{[I_n I_n \dots I_n]}_{r \text{ times}}.$$

Note that, for every $y \in \mathcal{P}$, $Ay = \sum_{i=1}^{r} y_i$, where y_i is the i-th block of y, and thus $Ay \in B(f)$. The problem (Min-Norm) can be reformulated for DSFM as follows.

$$\min \left\{ \frac{1}{2} \|Ay\|_2^2 : y \in \mathcal{P} \right\}. \tag{Prox-DSFM}$$

The second formulation is the following. Let us define the subspace $\mathcal{A} := \{a \in \mathbb{R}^{nr} : Aa = 0\}$, and minimize its distance from $\mathcal{P}$:

$$\min \left\{ \|a - y\|_2^2 : a \in \mathcal{A}, y \in \mathcal{P} \right\}. \tag{Best-Approx}$$

The set of optimal solutions for both formulations (Prox-DSFM) and (Best-Approx) is the set $\mathcal{E} := \{y \in \mathcal{P} : Ay = s^*\}$, where s^* is the optimum of (Min-Norm). We note that, even though the set of solutions to (Best-Approx) are pairs of points $(a, y) \in \mathcal{A} \times \mathcal{P}$, the optimal solutions are uniquely determined by $y \in \mathcal{P}$, since the corresponding a is the projection of y to $\mathcal{A}$.

3.2 Level-1 algorithms based on gradient descent

The gradient descent algorithms of [25, 6] provide level-1 algorithms for DSFM. We provide a brief overview of these algorithms and we refer the reader to the respective papers for more details.

The alternating projections algorithm. Nishihara *et al.* [25] minimize (Best-Approx) using *alternating projections*. The algorithm starts with a point $a_0 \in \mathcal{A}$ and it iteratively constructs a sequence $\left\{(a^{(k)}, x^{(k)})\right\}_{k \geq 0}$ by projecting onto $\mathcal{A}$ and $\mathcal{P}$: $x^{(k)} = \mathrm{argmin}_{x \in \mathcal{P}} \|a^{(k)} - x\|_2$, $a^{(k+1)} = \mathrm{argmin}_{a \in \mathcal{A}} \|a - x^{(k)}\|_2$.

Random coordinate descent algorithms. Ene and Nguyen [6] minimize (Prox-DSFM) using *random coordinate descent*. The RCDM algorithm adapts the random coordinate descent algorithm of Nesterov [24] to (Prox-DSFM). In each iteration, the algorithm samples a block $i \in [r]$ uniformly at random and it updates x_i via a standard gradient descent step for smooth functions. ACDM, the accelerated version of the algorithm, presents a further enhancement using techniques from [7].

3.3 Rates of convergence and condition numbers

The algorithms mentioned above enjoy a *linear convergence rate* despite the fact that the objective functions of (Best-Approx) and (Prox-DSFM) are not strongly convex. Instead, the works [25, 6] show that there are certain parameters that one can associate with the objective functions such that the convergence is at the rate $(1-\alpha)^k$, where $\alpha \in (0,1)$ is a quantity that depends on the appropriate parameter. Let us now define these parameters.

Let $\mathcal{A}'$ be the affine subspace $\mathcal{A}' := \{a \in \mathbb{R}^{nr} : Aa = s^*\}$. Note that the set $\mathcal{E}$ of optimal solutions to (Prox-DSFM) and (Best-Approx) is $\mathcal{E} = \mathcal{P} \cap \mathcal{A}'$. For $y \in \mathbb{R}^{nr}$ and a closed set $K \subseteq \mathbb{R}^{nr}$, we let $d(y, K) = \min\{\|y - z\|_2 : z \in K\}$ denote the distance between y and K. The relevant parameter for the Alternating Projections algorithm is defined as follows.

Definition 3.1 ([25])**.** For every $y \in (\mathcal{P} \cup \mathcal{A}') \setminus \mathcal{E}$, let

$$\kappa(y) := \frac{d(y, \mathcal{E})}{\max\{d(y, \mathcal{P}), d(y, \mathcal{A}')\}}, \qquad \text{and} \qquad \kappa_* := \sup\{\kappa(y) : y \in (\mathcal{P} \cup \mathcal{A}') \setminus \mathcal{E}\}.$$

The relevant parameter for the random coordinate descent algorithms is the following.

Definition 3.2 ([6])**.** For every $y \in \mathcal{P}$, let $y^* := \operatorname{argmin}_p\{\|p - y\|_2 : p \in \mathcal{E}\}$ be the optimal solution to (Prox-DSFM) that is closest to y. We say that the objective function $\frac{1}{2}\|Ay\|_2^2$ of (Prox-DSFM) is *restricted ℓ-strongly convex* if, for all $y \in \mathcal{P}$, we have

$$\|A(y - y^*)\|_2^2 \geq \ell \|y - y^*\|_2^2.$$

We define

$$\ell_* := \sup\left\{\ell : \frac{1}{2}\|Ay\|_2^2 \text{ is restricted } \ell\text{-strongly convex}\right\}.$$

The running time dependence of the algorithms on these parameters is given in the following theorems.

Theorem 3.3 ([25])**.** *Let $(a^{(0)}, x^{(0)} = \operatorname{argmin}_{x \in \mathcal{P}}\|a^{(0)} - x\|_2)$ be the initial solution and let (a^*, x^*) be an optimal solution to (Best-Approx). The alternating projection algorithm produces in*

$$k = \Theta\left(\kappa_*^2 \ln\left(\frac{\|x^{(0)} - x^*\|_2}{\epsilon}\right)\right)$$

iterations a pair of points $a^{(k)} \in \mathcal{A}$ and $x^{(k)} \in \mathcal{P}$ that is ϵ-optimal, i.e.,

$$\|a^{(k)} - x^{(k)}\|_2^2 \leq \|a^* - x^*\|_2^2 + \varepsilon.$$

Theorem 3.4 ([6])**.** *Let $x^{(0)} \in \mathcal{P}$ be the initial solution and let x^* be an optimal solution to (Prox-DSFM) that minimizes $\|x^{(0)} - x^*\|_2$. The random coordinate descent algorithm produces in*

$$k = \Theta\left(\frac{r}{\ell_*} \ln\left(\frac{\|x^{(0)} - x^*\|_2}{\epsilon}\right)\right)$$

iterations a solution $x^{(k)}$ that is ϵ-optimal in expectation, i.e., $\mathbb{E}\left[\frac{1}{2}\|Ax^{(k)}\|_2^2\right] \leq \frac{1}{2}\|Ax^*\|_2^2 + \epsilon$.

The accelerated coordinate descent algorithm produces in

$$k = \Theta\left(r\sqrt{\frac{1}{\ell_*}} \ln\left(\frac{\|x^{(0)} - x^*\|_2}{\epsilon}\right)\right)$$

iterations (specifically, $\Theta\left(\ln\left(\frac{\|x^{(0)} - x^*\|_2}{\epsilon}\right)\right)$ epochs with $\Theta\left(r\sqrt{\frac{1}{\ell_*}}\right)$ iterations in each epoch) a solution $x^{(k)}$ that is ϵ-optimal in expectation, i.e., $\mathbb{E}\left[\frac{1}{2}\|Ax^{(k)}\|_2^2\right] \leq \frac{1}{2}\|Ax^*\|_2^2 + \epsilon$.

3.4 Tight analysis for the condition numbers and running times

We provide a tight analysis for the condition numbers (the parameters κ_* and ℓ_* defined above). This leads to improved upper bounds on the running times of the gradient descent algorithms.

Theorem 3.5. *Let κ_* and ℓ_* be the parameters defined in Definition 3.1 and Definition 3.2. We have $\kappa_* = \Theta(n\sqrt{r})$ and $\ell_* = \Theta(1/n^2)$.*

Using our improved convergence guarantees, we obtain the following improved running time analyses.

Corollary 3.6. *The total running time for obtaining an ϵ-approximate solution*[5] *is as follows.*

- *Alternating projections (AP): $O\left(n^2 r^2 \Theta_{\text{avg}} \ln\left(\frac{\|x^{(0)}-x^*\|_2}{\epsilon}\right)\right)$.*
- *Random coordinate descent (RCDM): $O\left(n^2 r \Theta_{\text{avg}} \ln\left(\frac{\|x^{(0)}-x^*\|_2}{\epsilon}\right)\right)$.*
- *Accelerated random coordinate descent (ACDM): $O\left(n r \Theta_{\text{avg}} \ln\left(\frac{\|x^{(0)}-x^*\|_2}{\epsilon}\right)\right)$.*

We can upper bound the diameter of the base polytope by $O(\sqrt{n}F_{\max})$ [20], and thus $\|x^{(0)} - x^*\|_2 = O(\sqrt{n}F_{\max})$. For integer-valued functions, a ε-approximate solution can be converted to an exact optimum if $\varepsilon = O(1/n)$ [3].

The upper bound on κ_* and the lower bound on ℓ_* are shown in Theorem 4.2. The lower bound on κ_* and upper bound on ℓ_* in Theorem 3.5 follow by constructions in previous work, as explained next. Nishihara *et al.* showed that $\kappa_* \leq nr$, and they give a family of minimum cut instances for which $\kappa_* = \Omega(n\sqrt{r})$. Namely, consider a graph with n vertices and m edges, and suppose for simplicity that the edges have integer capacities at most C. The cut function of the graph can be decomposed into functions corresponding to the individual edges, and thus $r = m$ and $\Theta_{\text{avg}} = O(1)$. Already on simple cycle graphs, they show that the running time of AP is $\Omega(n^2 m^2 \ln(nC))$, which implies $\kappa_* = \Omega(n\sqrt{r})$.

Using the same construction, it is easy to obtain the upper bound $\ell_* = O(1/n^2)$.

4 Tight convergence bounds for the convex optimization algorithms

In this section, we show that the combinatorial approach introduced in Section 2 can be applied to obtain better bounds on the parameters κ_* and ℓ_* defined in Section 3. Besides giving a stronger bound, our proof is considerably simpler than the algebraic one using Cheeger's inequality in [25]. The key is the following lemma.

Lemma 4.1. *Let $y \in \mathcal{P}$ and $s^* \in B(f)$. Then there exists a point $x \in \mathcal{P}$ such that $Ax = s^*$ and $\|x - y\|_2 \leq \frac{\sqrt{n}}{2}\|Ay - s^*\|_1$.*

Before proving this lemma, we show how it can be used to derive the bounds.

Theorem 4.2. *We have $\kappa_* \leq n\sqrt{r}/2 + 1$ and $\ell_* \geq 4/n^2$.*

Proof: We start with the bound on κ_*. In order to bound κ_*, we need to upper bound $\kappa(y)$ for any $y \in (\mathcal{P} \cup \mathcal{A}') \setminus \mathcal{E}$. We distinguish between two cases: $y \in \mathcal{P} \setminus \mathcal{E}$ and $y \in \mathcal{A}' \setminus \mathcal{E}$.

Case I: $y \in \mathcal{P} \setminus \mathcal{E}$**.** The denominator in the definition of $\kappa(y)$ is equal to $d(y, \mathcal{A}') = \|Ay - s^*\|_2/\sqrt{r}$. This follows since the closest point $a = (a_1, \ldots, a_r)$ to y in $\mathcal{A}'$ can be obtained as $a_i = y_i + (s^* - Ay)/r$ for each $i \in [r]$. Lemma 4.1 gives an $x \in \mathcal{P}$ such that $Ax = s^*$ and $\|x - y\|_2 \leq \frac{\sqrt{n}}{2}\|Ay - s^*\|_1 \leq \frac{n}{2}\|Ay - s^*\|_2$. Since $Ax = s^*$, we have $x \in \mathcal{E}$ and thus the numerator of $\kappa(y)$ is at most $\|x - y\|_2$. Thus $\kappa(y) \leq \|x - y\|_2/(\|Ay - s^*\|_2/\sqrt{r}) \leq n\sqrt{r}/2$.

Case II: $y \in \mathcal{A}' \setminus \mathcal{E}$**.** This means that $Ay = s^*$. The denominator of $\kappa(y)$ is equal to $d(y, \mathcal{P})$. For each $i \in [r]$, let $q_i \in B(f_i)$ be the point that minimizes $\|y_i - q_i\|_2$. Let $q = (q_1, \ldots, q_r) \in \mathcal{P}$. Then

[5]The algorithms considered here solve the optimization problem (Prox-DSFM). An ε-approximate solution to an optimization problem $\min\{f(x)\colon x \in P\}$ is a solution $x \in P$ satisfying $f(x) \leq f(x^*) + \varepsilon$, where $x^* \in \text{argmin}_{x \in P} f(x)$ is an optimal solution.

$d(y,\mathcal{P}) = \|y-q\|_2$. Lemma 4.1 with q in place of y gives a point $x \in \mathcal{E}$ such that $\|q-x\|_2 \le \frac{\sqrt{n}}{2}\|Aq-s^*\|_1$. We have $\|Aq-s^*\|_1 = \|Aq-Ay\|_1 \le \sum_{i=1}^r \|q_i-y_i\|_1 = \|q-y\|_1 \le \sqrt{nr}\|q-y\|_2$. Thus $\|q-x\|_2 \le \frac{n\sqrt{r}}{2}\|q-y\|_2$. Since $x \in \mathcal{E}$, we have $d(y,\mathcal{E}) \le \|x-y\|_2 \le \|x-q\|_2 + \|q-y\|_2 \le \left(1+\frac{n\sqrt{r}}{2}\right)\|q-y\|_2 = \left(1+\frac{n\sqrt{r}}{2}\right) d(y,\mathcal{P})$. Therefore $\kappa(p) \le 1+\frac{n\sqrt{r}}{2}$, as desired.

Let us now prove the bound on ℓ_*. Let $y \in \mathcal{P}$ and let $y^* := \operatorname{argmin}_p\{\|p-y\|_2 \colon y \in \mathcal{E}\}$. We need to verify that $\|A(y-y^*)\|_2^2 \ge \frac{4}{n^2}\|y-y^*\|_2^2$. Again, we apply Lemma 4.1 to obtain a point $x \in \mathcal{P}$ such that $Ax = s^*$ and $\|x-y\|_2^2 \le \frac{n}{4}\|Ax-Ay\|_1^2 \le \frac{n^2}{4}\|Ax-Ay\|_2^2$. Since $Ax = s^*$, the definition of y^* gives $\|y-y^*\|_2^2 \le \|x-y\|_2^2$. Using that $Ax = Ay^* = s^*$, we have $\|Ax-Ay\|_2 = \|Ay-Ay^*\|_2$. □

Proof of Lemma 4.1: We give an algorithm that transforms y to a vector $x \in \mathcal{P}$ as in the statement through a sequence of path augmentations in the auxiliary graph defined in Section 2. We initialize $x = y$ and maintain $x \in \mathcal{P}$ (and thus $Ax \in B(f)$) throughout. We now define the set of source and sink nodes as $N := \{v \in V \colon (Ax)(v) < s^*(v)\}$ and $P := \{v \in V \colon (Ax)(v) > s^*(v)\}$. Once $N = P = \emptyset$, we have $Ax = s^*$ and terminate. Note that since $Ax, s^* \in B(f)$, we have $\sum_v (Ax)(v) = \sum_v s^*(v) = f(V)$, and therefore $N = \emptyset$ is equivalent to $P = \emptyset$. The blocks of x are denoted as $x = (x_1, x_2, \dots, x_r)$, with $x_i \in B(f_i)$.

Claim 4.3. *If $N \neq \emptyset$, then there exists a directed path of positive capacity in the auxiliary graph between the sets N and P.*

Proof: We say that a set T is i-tight, if $x_i(T) = f_i(T)$. It is a simple consequence of submodularity that the intersection and union of two i-tight sets are also i-tight sets. For every $i \in [r]$ and every $u \in V$, we define $T_i(u)$ as the unique minimal i-tight set containing u. It is easy to see that for an arc $(u,v) \in E_i$, $c(u,v) > 0$ if and only if $v \in T_i(u)$. We note that if $u \notin C_i$, then $x(u) = f_i(\{u\}) = 0$ and thus $T_i(u) = \{u\}$.

Let S be the set of vertices reachable from N on a directed path of positive capacity in the auxiliary graph. For a contradiction, assume $S \cap P = \emptyset$. By the definition of S, we must have $T_i(u) \subseteq S$ for every $u \in S$ and every $i \in [r]$. Since the union of i-tight sets is also i-tight, we see that S is i-tight for every $i \in [r]$, and consequently, $x(S) = f(S)$. On the other hand, since $N \subseteq S$, $S \cap P = \emptyset$, and $N \neq \emptyset$, we have $x(S) < s^*(S)$. Since $s^* \in B(f)$, we have $f(S) = x(S) < s^*(S) \le f(S)$, a contradiction. We conclude that $S \cap P \neq \emptyset$. □

In every step of the algorithm, we take a shortest directed path $\mathcal{Q}$ of positive capacity from N to P, and update x along this path. That is, if $(u,v) \in \mathcal{Q} \cap E_i$, then we increase $x_i(u)$ by ε and decrease $x_i(v)$ by ε, where ε is the minimum capacity of an arc on $\mathcal{Q}$. Note that this is the same as running the Edmonds-Karp-Dinitz algorithm in the submodular auxiliary graph. Using the analysis of [13], one can show that this change maintains $x \in \mathcal{P}$, and that the algorithm terminates in finite (in fact, strongly polynomial) time. We defer the details to the Appendix.

It remains to bound $\|x-y\|_2$. At every path update, the change in ℓ_∞-norm of x is at most ε, and the change in ℓ_1-norm is at most $n\varepsilon$, since the length of the path is $\le n$. At the same time, $\sum_{v \in N}(s^*(v) - (Ax)(v))$ decreases by ε. Thus, $\|x-y\|_\infty \le \|Ay-s^*\|_1/2$ and $\|x-y\|_1 \le n\|Ay-s^*\|_1/2$. Using the inequality $\|p\|_2 \le \sqrt{\|p\|_1\|p\|_\infty}$, we obtain $\|x-y\|_2 \le \frac{\sqrt{n}}{2}\|Ay-s^*\|_1$, completing the proof. □

5 The level-0 algorithms

In this section, we briefly discuss the level-0 algorithms and the interface between the level-1 and level-0 algorithms.

Two-level frameworks via quadratic minimization oracles. Recall from the Introduction the assumption on the subroutines $\mathcal{O}_i(w)$ that finds the minimum norm point in $B(f_i + w)$ for the input vector $w \in \mathbb{R}^n$ for each $i \in [r]$. The continuous methods in Section 3 directly use the subroutines $\mathcal{O}_i(w)$ for the alternating projection or coordinate descent steps. For the flow-based algorithms in Section 2, the main oracle query is to find the auxiliary graph capacity $c(u,v)$ of an arc $(u,v) \in E_i$ for some $i \in [r]$. This can be easily formulated as minimizing the function $f_i + w$ for an appropriate w with $\operatorname{supp}(w) \subseteq C_i$. As explained at the beginning of Section 3, an optimal solution to (Min-Norm) immediately gives an optimal solution to the SFM problem for the same submodular function. Hence, the auxiliary graph capacity queries can be implemented via single calls to the subroutines $\mathcal{O}_i(w)$. Let us also remark that, while the functions f_i are formally defined on the entire ground set V, their effective support is C_i, and thus it suffices to solve the quadratic minimization problems on the ground set C_i.

Whereas discrete and continuous algorithms require the same type of oracles, there is an important difference between the two algorithms in terms of exactness for the oracle solutions. The discrete algorithms require exact values of the auxiliary graph capacities $c(u, v)$, as they must maintain $x_i \in B(f_i)$ throughout. Thus, the oracle must always return an optimal solution. The continuous algorithms are more robust, and return a solution with the required accuracy even if the oracle only returns an approximate solution. As discussed in Section 6, this difference leads to the continuous methods being applicable in settings where the combinatorial algorithms are prohibitively slow.

Level-0 algorithms. We now discuss specific algorithms for quadratic minimization over the base polytopes of the functions f_i. Several functions that arise in applications are "simple", meaning that there is a function-specific quadratic minimization subroutine that is very efficient. If a function-specific subroutine is not available, one can use a general-purpose submodular minimization algorithm. The works [2, 8] use a *brute force search* as the subroutine for each each f_i, whose running time is $2^{|C_i|}\mathrm{EO}_i$. However, this is applicable only for small C_i's and is not suitable for our experiments where the maximum clique size is quite large. As a general-purpose algorithm, we used the *Fujishige-Wolfe minimum norm point algorithm* [12, 32]. This provides an ε-approximate solution in $O(|C_i|F_{i,\max}^2/\varepsilon)$ iterations, with overall running time bound $O((|C_i|^4 + |C_i|^2\mathrm{EO}_i)F_{i,\max}^2/\varepsilon)$ [4]. The experimental running time of the Fujishige-Wolfe algorithm can be prohibitively large [21]. As we discuss in Section 6, by warm-starting the algorithm and performing only a small number of iterations, we were able to use the algorithm in conjunction with the gradient descent level-1 algorithms.

6 Experimental results

We evaluate the algorithms on energy minimization problems that arise in image segmentation problems. We follow the standard approach and model the image segmentation task of segmenting an object from the background as finding a minimum cost $0/1$ labeling of the pixels. The total labeling cost is the sum of labeling costs corresponding to *cliques*, where a clique is a set of pixels. We refer to the labeling cost functions as clique potentials.

The main focus of our experimental analysis is to compare the running times of the decomposable submodular minimization algorithms. Therefore we have chosen to use the simple hand-tuned potentials that were used in previous work: the *edge-based costs* [2] and the *count-based costs* defined by [30, 31]. Specifically, we used the following clique potentials in our experiments, all of which are submodular:

- **Unary potentials** for each pixel. The unary potentials are derived from Gaussian Mixture Models of color features [27].
- **Pairwise potentials** for each edge of the 8-neighbor grid graph. For each graph edge (i, j) between pixels i and j, the cost of a labeling equals 0 if the two pixels have the same label, and $\exp(-\|v_i - v_j\|^2)$ for different labels, where v_i is the RGB color vector of pixel i.
- **Square potentials** for each 2×2 square of pixels. The cost of a labeling is the square root of the number of neighboring pixels that have different labels, as in [2].
- **Region potentials.** We use the algorithm from [31] to identify regions. For each region C_i, the labeling cost is $f_i(S) = |S||C_i \setminus S|$, where S and $C_i \setminus S$ are the subsets of C_i labeled 0 and 1, respectively, see [30, 31].

We used five image segmentation instances to evaluate the algorithms.[6] The experiments were carried out on a single computer with a 3.3 GHz Intel Core i5 processor and 8 GB of memory; we reported averaged times over 10 trials.

We performed several experiments with various combinations of potentials and parameters. In the *minimum cut experiments*, we evaluated the algorithms on instances containing only unary and

[6]The data is available at `http://melodi.ee.washington.edu/~jegelka/cc/index.html` and `http://research.microsoft.com/en-us/um/cambridge/projects/visionimagevideoediting/segmentation/grabcut.htm`

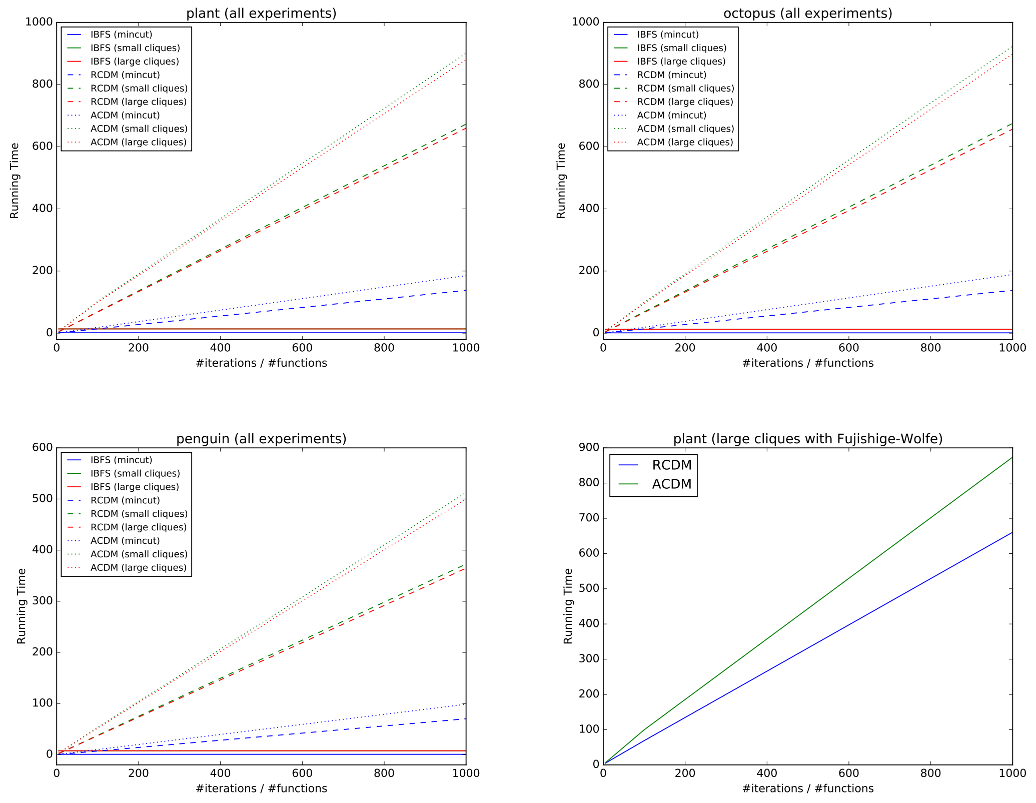

Figure 1: Running times in seconds on a subset of the instances. The results for the other instances are very similar and are deferred to the Appendix. The x-axis shows the number of iterations for the continuous algorithms. The IBFS algorithm is exact, and we display its running time as a flat line. In the first three plots, the running time of IBFS on the small cliques instances nearly coincides with its running time on minimum cut instances. In the last plot, the running time of IBFS is missing since it is computationally prohibitive to run it on those instances.

pairwise potentials; in the *small cliques experiments*, we used unary, pairwise, and square potentials. Finally, the *large cliques experiments* used all potentials above. Here, we used two different level-0 algorithms for the region potentials. Firstly, we used an algorithm specific to the particular potential, with running time $O(|C_i|\log(|C_i|) + |C_i|\mathrm{EO}_i)$. Secondly, we used the general Fujishige-Wolfe algorithm for level-0. This turned out to be significantly slower: it was prohibitive to run the algorithm to near-convergence. Hence, we could not implement IBFS in this setting as it requires an exact solution.

We were able to implement coordinate descent methods with the following modification of Fujishige-Wolfe at level-0. At every iteration, we ran Fujishige-Wolfe for 10 iterations only, but we *warm-started* with the current solution $x_i \in B(f_i)$ for each $i \in [r]$. Interestingly, this turned out to be sufficient for the level-1 algorithm to make progress.

Summary of results. Figure 1 shows the running times for some of the instances; we defer the full experimental results to the Appendix. The IBFS algorithm is significantly faster than the gradient descent algorithms on all of the instances with small cliques. For all of the instances with larger cliques, IBFS (as well as other combinatorial algorithms) are no longer suitable if the only choice for the level-0 algorithms are generic methods such as the Fujishige-Wolfe algorithm. The experimental results suggest that in such cases, the coordinate descent methods together with a suitably modified Fujishige-Wolfe algorithm provides an approach for obtaining an approximate solution.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., feb 1993.

[2] C. Arora, S. Banerjee, P. Kalra, and S. Maheshwari. Generic cuts: An efficient algorithm for optimal inference in higher order MRF-MAP. In *European Conference on Computer Vision*, pages 17–30. Springer, 2012.

[3] F. Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 6(2-3):145–373, 2013.

[4] D. Chakrabarty, P. Jain, and P. Kothari. Provable submodular minimization using Wolfe's algorithm. In *Advances in Neural Information Processing Systems*, pages 802–809, 2014.

[5] J. Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial structures and their applications*, pages 69–87, 1970.

[6] A. R. Ene and H. L. Nguyen. Random coordinate descent methods for minimizing decomposable submodular functions. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[7] O. Fercoq and P. Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

[8] A. Fix, T. Joachims, S. Min Park, and R. Zabih. Structured learning of sum-of-submodular higher order energy functions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3104–3111, 2013.

[9] A. Fix, C. Wang, and R. Zabih. A primal-dual algorithm for higher-order multilabel Markov random fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1138–1145, 2014.

[10] L. Fleischer and S. Iwata. A push-relabel framework for submodular function minimization and applications to parametric optimization. *Discrete Applied Mathematics*, 131(2):311–322, 2003.

[11] S. Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, 5(2):186–196, 1980.

[12] S. Fujishige and S. Isotani. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7(1):3–17, 2011.

[13] S. Fujishige and X. Zhang. New algorithms for the intersection problem of submodular systems. *Japan Journal of Industrial and Applied Mathematics*, 9(3):369, 1992.

[14] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[15] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[16] S. Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM Journal on Computing*, 32(4):833–840, 2003.

[17] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001.

[18] S. Iwata and J. B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2009.

[19] S. Jegelka, F. Bach, and S. Sra. Reflection methods for user-friendly submodular optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[20] S. Jegelka and J. A. Bilmes. Online submodular minimization for combinatorial structures. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 345–352, 2011.

[21] S. Jegelka, H. Lin, and J. A. Bilmes. On fast approximate submodular minimization. In *Advances in Neural Information Processing Systems*, pages 460–468, 2011.

[22] V. Kolmogorov. Minimizing a sum of submodular functions. *Discrete Applied Mathematics*, 160(15):2246–2258, 2012.

[23] Y. T. Lee, A. Sidford, and S. C.-w. Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE Foundations of Computer Science (FOCS)*, 2015.

[24] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[25] R. Nishihara, S. Jegelka, and M. I. Jordan. On the convergence rate of decomposable submodular function minimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 640–648, 2014.

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

[27] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 23(3):309–314, 2004.

[28] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

[29] A. Schrijver. *Combinatorial optimization - Polyhedra and Efficiency*. Springer, 2003.

[30] I. Shanu, C. Arora, and P. Singla. Min norm point algorithm for higher order MRF-MAP inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5365–5374, 2016.

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

[32] P. Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11(1):128–149, 1976.

A Level-1 algorithms based on the network flow approach

In this section, we extend the Fujishige-Zhang approach to the general DSFM problem with an arbitrary number of functions in the decomposition. We illustrate the power of the resulting framework by showing that it can be used to extend the classical maximum flow algorithms for graphs to the DSFM setting: the preflow-push algorithm [14] and its scaling variant. Other maximum flow algorithms can also be adapted using similar analyses.

A.1 Preflow-push algorithm

We proceed as in the preflow-push algorithm for the classical maximum flow problem in graphs. We refer the reader to [1] on network flow algorithms, in particular, to Section 7.6 on the generic preflow-push algorithm.

We maintain a point $x \in B(f)$ with a decomposition $x = \sum_{i=1}^{r} x_i$ into points $x_i \in B(f_i)$. Throughout, we maintain $\mathrm{supp}(x_i) \subseteq V_i$ for all $i \in [r]$. Initially, each x_i is an arbitrary vertex in $B(f_i)$, which we can find by running the greedy algorithm for an arbitrary permutation for f_i. Let

$$N := \{v \in V \colon x(v) < 0\}, \qquad P := \{v \in V \colon x(v) > 0\}.$$

The auxiliary graph. For each $i \in [r]$ and each $u \in V$, we define a set $T_i(u)$ as follows. We say that a set $S \subseteq V$ is (x_i, f_i)*-tight* if $x_i(S) = f_i(S)$. We let $T_i(u)$ be the minimal (x_i, f_i)-tight set that contains u, i.e., the minimal set S such that $u \in S$ and $x_i(S) = f_i(S)$. Since f_i is submodular, there is a unique minimal (x_i, f_i)-tight set $T_i(u)$ containing u (see Lemma A.2). We note that if $u \notin V_i$, then $x_i(u) = f_i(u) = 0$ and $T_i(u) = \{u\}$ hold throughout.

We use the sets $\{T_i(u) \colon i \in [r], u \in V\}$ to define a directed graph $G = (V, E)$, with $E = \bigcup_{i=1}^{r} E_i$ and capacities $c : E \to \mathbb{R}_+$. Let

$$E_i := \{(u, v) \colon u \in V, v \in T_i(u)\} \quad \forall i \in [r].$$

We think of the arcs of E_i as having color i, and we sometimes refer to an arc in E_i as an i-arc. If an arc appears in more than one set E_i, we include multiple copies of the arc with the appropriate colors and capacities.

We define the capacity $c(u, v) = c_i(u, v)$ for an i-arc as the maximum amount ε such that $x_i^\varepsilon \in B(f_i)$, where $x_i^\varepsilon := x_i(u) + \varepsilon$, $x_i^\varepsilon(v) = x_i(v) - \varepsilon$, and $x_i^\varepsilon(w) = x_i(w)$ for all $w \neq u, v$. This can be equivalently written as

$$c_i(u, v) = \min\{f_i(S) - x_i(S) \colon S \subseteq V, u \in S, v \notin S\}. \tag{1}$$

Using the minimality of $T_i(u)$, one can show that $c_i(u, v) > 0$ for every $(u, v) \in E$ (see Lemma A.10).

The resulting graph G can be thought of as an analogue of the residual graph from the max flow - min cut setting, since it is the residual graph for minimum cut instances. The nodes of N are the "sources", and the nodes of P are the "sinks".

In Section A.2.3, we show how the auxiliary graph can be constructed efficiently using the oracles $\mathcal{O}_i$. The auxiliary graph keeps changing during the modifications to the x_i's; the main technical challenge is to keep track of all these changes. However, we do not need to maintain all arcs and capacities at every point of the algorithm. The subroutine TIGHT-UPDATE(u, i) recomputes the set $T_i(u)$ for the current x_i, and thus identifies all arcs of color i going out from u. The subroutine CAPACITY-UPDATE(u, v, i) computes the current capacity of the arc (u, v) of color i. These subroutines are described in Section A.2.3.

Throughout the algorithm, we maintain *distance labels* $d : V \to \mathbb{Z}_+$ with the standard properties of preflow-push algorithms:

$$\begin{aligned} d(v) &= 0 && \forall v \in P, \\ d(u) &\leq d(v) + 1 && \forall (u, v) \in E. \end{aligned} \tag{$\star$}$$

The labels are initialized as $d(v) = 0$ for all $v \in V$. Further, we consider an arbitrary but fixed complete ordering $\prec$ of the ground set V, used for tie-breaking. An arc (u, v) is called *admissible*, if $d(u) = d(v) + 1$.

Algorithm 1 Decomposable Submodular Function Minimization

```
Input: A submodular function $f = \sum_{i=1}^r f_i$ on ground set $V$, with value oracles and oracles $\mathcal{O}_i(w)$ provided.
Output: A set $S \subseteq V$ minimizing $f(S)$.
 1: for $i = 1, \dots, r$ do $x_i \leftarrow$ arbitrary vertex of $B(f_i)$.
 2: $x \leftarrow \sum_{i=1}^r x_i$.
 3: for $v \in V$ do $d(v) \leftarrow 0$.
 4: $N \leftarrow \{v \in V : x(v) < 0\}$; $H \leftarrow \emptyset$.
 5: for $i = 1, \dots, r$ do
 6:     for $u \in V_i$ do TIGHT-UPDATE$(u, i)$.
 7: while $N \setminus H \neq \emptyset$ do pick $u \in N \setminus H$.
 8:     if there is an arc $(u, v) \in E$, $d(u) = d(v) + 1$ then
 9:         Select $(u, v)$ with $v$ minimal for $\prec$, and $i$ s.t. $(u, v) \in E_i$.
10:         $c_i(u, v) \leftarrow$ CAPACITY-UPDATE$(u, v, i)$.
11:         $\varepsilon \leftarrow \min\{c_i(u, v), -x(u)\}$.
12:         $x_i(u) \leftarrow x_i(u) + \varepsilon$.
13:         $x_i(v) \leftarrow x_i(v) - \varepsilon$.
14:         if $\varepsilon = c_i(u, v)$ then
15:             for $(z, v) \in E_i$ do TIGHT-UPDATE$(z, i)$.
16:         if $\varepsilon < c_i(u, v)$ then
17:             for $z\colon (z, v) \in E_i, (z, u) \notin E_i$ do $T_i(z) \leftarrow T_i(z) \cup T_i(u)$.
18:         Update the status of $u$ and $v$ in $N$.
19:     else $t \leftarrow \min\{d(v) + 1 : (u, v) \in E\}$ ;
20:         if $t \leq n$ then $d(u) \leftarrow t$;
21:         else $d(u) \leftarrow n$; $H \leftarrow H \cup \{u\}$.
22: $S \leftarrow \{u \in V : \exists$ path in $G$ from $N$ to $u\}$.
23: return $S$.
```

The overall algorithm is shown in Algorithm 1. At every iteration, a node $u \in N$ is selected, and one of the usual operations of preflow-push algorithms is performed.

- **Relabel.** If $d(u) < n$ with no admissible arc leaving u, then we update
$$d(u) := \min\left\{\min\{d(v) + 1 : (u, v) \in E\}, n\right\}.$$
We add the node u into the set H, if $d(u) = n$ and there are no admissible arcs leaving u. Nodes in H will not be considered again.

- **Push.** If there are outgoing admissible arcs $(u, v) \in E$, then we pick v minimal with respect to the ordering $\prec$. Assume $(u, v) \in E_i$. In this case, we update x_i as follows; we leave all other x_j's unchanged. Let $\varepsilon := \min\{-x(u), c_i(u, v)\}$, and update
$$x_i'(u) := x_i(u) + \varepsilon,$$
$$x_i'(v) := x_i(v) - \varepsilon,$$
$$x_i'(w) := x_i(w) \text{ if } w \neq u, v.$$
As shown in Lemma A.5 below, the distance labels remain valid after the push operation and thus ($\star$) is maintained for the updated graph and vectors x_i.

Termination. The algorithm terminates when no push or relabel operation is possible. At that point, we compute the set of nodes S reachable from N in the auxiliary graph G, and return S as an optimal solution to (SFM). (Note that if $N = \emptyset$, then $S = \emptyset$.)

We show the following running time bound.

Theorem A.1. *Algorithm 1 can be implemented to find the minimizer of $f(S)$ in running time $O(n^2 \Theta_{\max} \sum_{i=1}^r |C_i|^2)$.*

For the complexity bound, the main issue is how to efficiently maintain the graph $G = (V, E)$, and find the minimum label of the neighbors for the label update step. Algorithm 1 provides a simple and

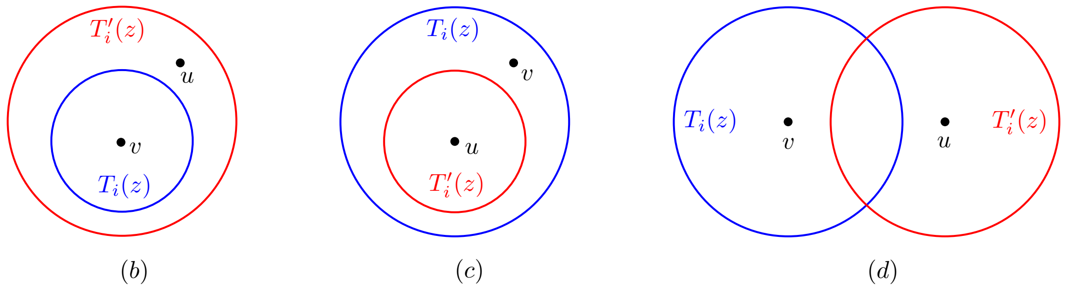

Figure 2: Lemma A.3 cases.

natural way of updating the sets $T_i(u)$. To achieve a better complexity, in the proof we will describe a slightly more complicated, but more efficient variant.

A.2 Analysis

The correctness and running time analysis is performed in the following steps. In Section A.2.1, we describe the changes in the auxiliary graph caused by push operations, and prove that the main invariant ($\star$) is maintained. In Section A.2.2, we show that if the algorithm terminates, it correctly returns an optimal solution. Section A.2.3 describes the implementation of the subroutines TIGHT-UPDATE and CAPACITY-UPDATE via the oracles $\mathcal{O}_i$. The running time bound is given in Section A.2.4. At a high level, it follows the lines of the standard preflow-push analysis, bounding first the number of saturating, then the number of nonsaturating pushes (to be defined in due course). However, due to the complex changes in the auxiliary graph, we need a more careful analysis. The most difficult and technical part of the analysis is bounding the number of saturating pushes (Lemma A.12), which is a significant departure from the analysis of the preflow-push algorithm for graphs.

Let us start by a simple claim on tight sets. The proof is the standard application of submodularity.

Lemma A.2. *If $X, Y \subseteq V$ are (x_i, f_i)-tight sets, then $X \cap Y$ and $X \cup Y$ are also (x_i, f_i)-tight. In particular, for every $u \in V_i$, there exists a unique minimal tight set $T_i(u)$, and $T_i(u) \subseteq V_i$. If $u \notin V_i$, then $T_i(u) = \{u\}$. If $v \in T_i(u)$, then $T_i(v) \subseteq T_i(u)$.*

A.2.1 Evolution of the auxiliary graph

Our first important goal is to show that the labels $d(u)$ remain feasible throughout, i.e, ($\star$) is maintained. Towards this goal, we first need to understand how the minimal tight sets $T_i(z)$ evolve. Clearly, relabel steps have no effect on these sets. Also, a push on an edge $(u, v) \in E_i$ does not affect any of the sets $T_j(z)$ for $j \neq i$. However, $T_i(z)$ may change for some nodes $z \in V_i$, in accordance with the next lemma.

Lemma A.3. *Consider an iteration of the algorithm that updates the solution x to x' by pushing $\varepsilon > 0$ along the arc $(u, v) \in E_i$. Let $\{T_i(z) : z \in V_i\}$ and $\{T_i'(z) : z \in V_i\}$ be the minimal (x_i, f_i)-tight sets and minimal (x_i', f_i)-tight sets, respectively. For any $z \in V_i$, exactly one of the following holds:*

(a) $T_i(z) = T_i'(z)$;

(b) $T_i(z) \subsetneq T_i'(z)$ and $u \in T_i'(z) \setminus T_i(z)$ and $v \in T_i(z)$;

(c) $T_i'(z) \subsetneq T_i(z)$ and $u \in T_i'(z)$ and $v \in T_i(z) \setminus T_i'(z)$;

(d) $u \in T_i'(z) \setminus T_i(z)$ and $v \in T_i(z) \setminus T_i'(z)$.

In cases (b), (c), *and* (d), *we also have $T_i'(z) \subseteq T_i(z) \cup T_i(u)$. Case* (a) *happens only if either $\{u, v\} \subseteq T_i'(z)$ or $\{u, v\} \cap T_i'(z) = \emptyset$.*

The different cases are illustrated in Figure 2. The proof is given in the Appendix; it is a simple case distinction based on whether u and v are contained in $T'_i(z)$. Let us now derive some useful implications from this lemma. We say that the push along the i-arc (u,v) is *saturating* if the pushed flow amount is $\varepsilon = c_i(u,v)$, and it is *non-saturating* otherwise. Let $T'_i(z)$ denote the sets after pushing on the i-arc (u,v). By the definition of $c_i(u,v)$ (1), we see that $T'_i(u) = T_i(u)$ if and only if the push is non-saturating.

Lemma A.4. *Let $T_i(z)$ and $T'_i(z)$ denote the respective tight sets before and after performing a push on an arc $(u,v) \in E_i$.*

(i) If $z' \in T'_i(z) \setminus T_i(z)$, then $u \notin T_i(z)$, $v \in T_i(z)$, and $z' \in T_i(u)$.

(ii) If the push on (u,v) is non-saturating, and $v \in T_i(z)$, $u \notin T_i(z)$, then $T'_i(z) = T_i(z) \cup T_i(u)$. For all other nodes z, $T'_i(z) = T_i(z)$.

Proof: **(i)** Possible cases in such a situation are (b) or (d). In both cases, $u \notin T_i(z)$, $v \in T_i(z)$. Further, the lemma asserts $T'_i(z) \subseteq T_i(z) \cup T_i(u)$ in both cases. Since $z' \in T'_i(z)$, and we assume $z' \notin T_i(z)$, it follows that $z' \in T_i(u)$.

(ii) Assume $T'_i(z) \neq T_i(z)$. Then we must be in one of cases (b)–(d). We show that (b) is the only possibility for non-saturating pushes. This follows by showing that in in cases (c) and (d), $T'_i(u) \neq T_i(u)$ holds. Note that $u \in T'_i(z)$ and hence $T'_i(u) \subseteq T'_i(z)$ in both cases, and hence $v \notin T'_i(u)$, although $v \in T_i(u)$. We thus conclude that (b) is the only possible case, giving $v \in T_i(z)$ and $u \notin T_i(z)$. Further, $T_i(z) \subsetneq T'_i(z)$, and $u \in T'_i(z)$.

Let $S = T_i(z) \cup T_i(u)$. As the union of two (x_i, f_i)-tight sets, S is (x_i, f_i)-tight before the push; and it remains tight after the push as it contains both u and v. Hence $T'_i(z) \subseteq S$. On the other hand, since $T'_i(u) = T_i(u)$ and $u \in T'_i(z)$, we have $T_i(u) \subseteq T'_i(z)$. Together with $T_i(z) \subseteq T'_i(z)$, this guarantees $S \subseteq T_i(u)$. The claim follows. □

Using the lemma above, we can show that the distance labels remain valid.

Lemma A.5. *The property ($\star$) is maintained throughout the algorithm.*

Proof: We have ($\star$) trivially at initialization. $d(u) = 0$ for $u \in P$ is maintained since we only increase $d(u)$ for nodes in N. Moreover, when we push flow on an arc (u,v), then $x(u)$ may decrease; however, the choice of ε guarantees that $x'(u) \geq 0$ after the push. Therefore, once a node leaves P, it cannot enter again.

Let us now show that $d(z) \leq d(w) + 1$ for all $(z,w) \in E$. This is clearly maintained in all push steps. Let us consider a relabel step, when we push flow on an arc $(u,v) \in E_i$. Let E' and E'_i denote the new arc sets. The condition clearly holds for all $(z,w) \in E' \cap E$. Consider a new arc $(z,w) \in E' \setminus E$; clearly, $(z,w) \in E'_i$, since the other arc sets E_j remain unchanged. By Lemma A.4(i), we have $(z,v), (u,w) \in E_i$. Then

$$\begin{aligned} d(z) &\leq d(v) + 1 && (z,v) \in E \\ &= d(u) && (u,v) \text{ is admissible} \\ &\leq d(w) + 1 && (u,w) \in E, \end{aligned}$$

and the lemma follows. □

A.2.2 Termination

We now prove that the algorithm terminates correctly.

Lemma A.6. *The set S returned by the algorithm minimizes $f(S)$.*

Proof: If $N = \emptyset$, we have $f(S) \geq x(S) \geq 0$ for all $S \subseteq V$, and $x(\emptyset) = f(\emptyset) = 0$. Thus $f(\emptyset) = \min_{S \subseteq V} f(S)$. For the rest of the proof, we assume $N \neq \emptyset$.

Claim A.7. *We have $S \cap P = \emptyset$.*

Proof: At termination, $N \subseteq H$ and therefore $d(u) = n$ for all $u \in N$. Thus it suffices to show that there is no path from a node u with $d(u) = n$ to a node $v \in P$. This easily follows by ($\star$): if there is a path from u to v in the auxiliary graph then $d(u) \leq d(v) + n - 1$. Therefore nodes in P are not reachable from any node u with $d(u) = n$. □

Claim A.8. *S is (x_i, f_i)-tight for all $i \in [r]$, and thus $x(S) = f(S)$.*

Proof: Since no arc leaves S, it follows that $T_i(v) \subseteq S$ for all $v \in S$ and all $i \in [r]$. Therefore, $S = \bigcup_{v \in S} T_i(v)$ for all $i \in [r]$, and Lemma A.2 implies that S is (x_i, f_i)-tight. Then $x_i(S) = f_i(S)$ for all $i \in [r]$ gives $x(S) = f(S)$. □

Now we can show that S is optimal as follows. For every set $R \subseteq V$, since $S \cap P = \emptyset$, and $N \subseteq S$, we have $x(R) \geq x(S)$. Consequently, $f(R) \geq x(R) \geq x(S) = f(S)$. □

A.2.3 Subroutines for auxiliary graph updates

We now describe the subroutines TIGHT-UPDATE and CAPACITY-UPDATE, implemented using the oracles $\mathcal{O}_i$.

Observe that if the functions f_i are integer valued, then we can maintain integrality of all x_i throughout. This is because all bases in $B(f_i)$ are integer, hence the x_i's can be initialized as integers. They are updated in the flow steps; however, ε is always selected as integer, provided the previous vectors were integers.

The subroutine TIGHT-UPDATE(u, i) has to compute the minimum (f_i, x_i)-tight set $T_i(u)$ containing u. The subroutine amounts to a single call to oracle $\mathcal{O}_i$, with the weight function defined in the proof of the following lemma.

Lemma A.9. *The set $T_i(u)$ can be found by a single call to oracle $\mathcal{O}_i$.*

Proof: Let us define the vector $w \in \mathbb{Z}^V$ as $w(u) := f_i(\{u\}) + 1$, and $w(v) = x_i(v) - 1/(n+1)$ for all $v \in V_i \setminus \{u\}$. We show that $T_i(u)$ is the unique minimizer of $f_i - w$.

We start by showing that $u \in S$ for any minimizer S of $f_i(S) - w(S)$. For a contradiction, suppose that $u \notin S$. We have

$$\begin{aligned} f_i(S \cup \{u\}) - w(S \cup \{u\}) &= f_i(S \cup \{u\}) - w(u) - w(S) \\ &= f_i(S) - w(S) + f_i(S \cup \{u\}) - f_i(S) - f(\{u\}) - 1. \end{aligned}$$

By submodularity, $f_i(S \cup \{u\}) \leq f_i(S) + f_i(\{u\})$. Therefore, $f_i(S \cup \{u\}) - w(S \cup \{u\}) \leq f_i(S) - w(S) - 1$, which is a contradiction. We thus have $u \in S$. Consider now an arbitrary set $S \subseteq V$, $u \in S$.

$$f_i(S) - w(S) = f_i(S) - x_i(S) + x_i(u) - f_i(\{u\}) - 1 + |S|/(n+1) \geq x_i(u) - f_i(\{u\}) - 1 + |S|/(n+1).$$

Note that $|S|/(n+1) < 1$ always holds. Clearly, $f_i(S) - w(S) < x_i(u) - f_i(\{u\})$ can only be attained if S is (x_i, f_i)-tight, hence all minimizers must be (x_i, f_i)-tight. If S is an (x_i, f_i)-tight set, we have $f_i(S) - w(S) = x_i(u) - f_i(\{u\}) - 1 + |S|/(n+1)$. Hence there will be a unique minimizer, namely the one of the smallest cardinality, which is $T_i(u)$. □

Let us now turn to CAPACITY-UPDATE(u, v, i). That is, we want to compute the capacity of $(u, v) \in E_i$, which also means $v \in T_i(u)$. This can be also implemented by a single call to $\mathcal{O}_i$, as described in the proof of the next lemma.

Lemma A.10. *The capacity $c_i(u, v) = \min\{f_i(S) - x_i(S) \colon S \subseteq V, u \in S, v \notin S\}$ is positive, and can be computed by a single call to oracle $\mathcal{O}_i$.*

Proof: First, note that the minimum must be positive. Otherwise, there would be an (x_i, f_i)-tight set S with $u \in S$, $v \in S$, and thus $T_i(u) \not\subseteq S$. This contradicts the fact that $T_i(u)$ is the unique minimal (x_i, f_i)-tight set containing u.

Let us modify the construction of the previous proof by defining $w(u) = f_i(\{u\}) + 1$, $w(v) = f_i(V) - f_i(V \setminus \{v\}) - 1$, and $w(z) = x(z)$ for all $z \notin \{u, v\}$. Note that the term $1/(n+1)$ is omitted.

The same argument as in the previous proof shows that every minimizer S to $f(S) - w(S)$ must contain u, and an analogous argument shows that $v \notin S$. We further see that the minimizer must be (x_i, f_i)-tight. □

Remark. The argument in Lemma A.9 used that the functions f_i are integer valued; this was exploited in using the perturbation term $1/(n+1)$ in w. If we only assume that the f_i's are rational, this should be modified to $1/(Q(n+1))$, where Q is a common denominator of all function values. One can show that the x_i's remain integer multiples of $1/Q$. If we do not wish to make any assumption on the values of f_i, e.g., we want to work in the real model of computation, then finding $T_i(u)$ will require $|C_i|$ oracle calls instead of a single one. Indeed, we omit the perturbation terms; then the minimizer of $f_i(S) - w(S)$ will be an (x_i, f_i)-tight set, but not necessarily the minimal one. We can try removing nodes from S one-by-one (using a cost function as in the proof of Lemma A.10), to identify the minimal tight set $T_i(u)$. On the other hand, Lemma A.10 does not require any integrality assumption, and remains true even in the real model of computation.

A.2.4 The running time analysis

We now give the proof of Theorem A.1. We need to bound the number of push and relabel operations.

Relabel operations. Since the algorithm never relabels a node with $d(v) \geq n$, the total number of relabel operations is at most n^2.

Push operations. It is convenient to index the iterations. If t is a push iteration, we let $d^{(t)}$, $x^{(t)}$, $G^{(t)}$, $E_i^{(t)}$, $T_i^{(t)}(v)$, $N^{(t)}$, $P^{(t)}$ denote the quantities d, x, G, E_i, $T_i(v)$, N, P right before the push operation. Recall the notions of saturating and non-saturating pushes from Section A.2.1: a push on the i-arc (u, v) is saturating if $\varepsilon = c_i(u, v)$, and non-saturating if $\varepsilon = -x(u) < c_i(u, v)$. Note that for non-saturating pushes, $x^{(t+1)}(u) = 0$ holds; that is, $u \notin N^{(t+1)}$. We first upper bound the total number of saturating pushes.

Lemma A.11. *The total number of saturating pushes on i-arcs is $O(n|C_i|^2)$. Thus the total number of saturating pushes is $O(n \sum_{i=1}^{r} |C_i|^2)$.*

Lemma A.11 is a consequence of the following lemma showing that, in between two saturating pushes on the same arc (z, w), the distance label $d(z)$ increases by at least one. Thus, for a fixed arc (z, w), there are at most n saturating pushes. Since the total number of possible i-arcs is bounded by $|C_i|^2$, Lemma A.11 follows.

Lemma A.12. *Suppose two iterations $t < t'$ push flow along the same i-arc (z, w). If both pushes are saturating, $d^{(t')}(z) \geq d^{(t)}(z) + 1$.*

Proof: The main strategy behind the proof is to trace out the evolution of the tight set $T_i(z)$ between time t and t'. The relevant events for us will be the time steps τ where the tight set acquires new nodes, i.e., $T_i^{(\tau+1)}(z) \setminus T_i^{(\tau)}(z)$ is non-empty. For such an iteration τ, Lemma A.4(i) shows that we push flow on an arc (u, v) with $v \in T_i^{(\tau)}(z)$ and $u \notin T_i^{(\tau)}(z)$; we refer to these pushes as *critical pushes*. Since the push on (z, w) at time t is saturating, $w \notin T_i^{(t+1)}(z)$ (thus the arc (z, w) disappears from the auxiliary graph). Since at time $t' > t$ we push on (z, w) again, there must be a sequence of critical pushes that led to $T_i(z)$ reacquiring w (and thus the reappearance of the arc (z, w) in the auxiliary graph). The main strategy is to trace back this sequence of critical pushes, starting from the first time $T_i(z)$ gained back w and going backward in time until we reach time t.

More precisely, we define a sequence $\{(t_j, u_j, v_j) \colon 0 \leq j < k\}$, where $t_j \in [t, t']$ is a time step, and (u_j, v_j) is an arc, on which we push at time t_j. The sequence goes backwards in time: $t' > t_0 > t_1 > \ldots > t_k = t - 1$.

Let t_0 be the first time in the range $[t + 1, t' - 1]$ such that $w \in T_i^{(t_0+1)}(z)$. Note that there exists such a time t_0. As noted above, $w \notin T_i^{(t+1)}(z)$. On the other hand, we push on (z, w) at time t', so $w \in T_i^{(t')}(z)$. Since $T_i^{(t_0+1)}(z) \setminus T_i^{(t_0)}(z)$ is non-empty, iteration t_0 must perform a push on some i-arc (u_0, v_0) (by Lemma A.4(i)). Let $t_1 \in [t - 1, t_0)$ be the first time such that $v_0 \in T_i^{(t_1+1)}(z)$. Either $t_1 = t - 1$ or, at time t_1, the algorithm performs a push on some i-arc (u_1, v_1) such that $v_1 \in T_i^{(t_1)}(z)$ and $u_1 \notin T_i^{(t_1)}(z)$. We continue defining (t_j, u_j, v_j) based on $(t_{j-1}, u_{j-1}, v_{j-1})$, until we have $t_k = t - 1$. Note that u_k, v_k are not defined. Let us highlight the key properties:

$$\begin{cases} v_j \in T_i^{(j)}(z), \qquad u_j \notin T_i^{(j)}(z), \quad \forall 0 \leq j \leq k-1 \\ v_j \in T_i^{(t_{j+1}+1)}(z) \setminus T_i^{(t_{j+1})}(z) \quad \forall 0 \leq j \leq k-2. \end{cases} \tag{2}$$

We need to show $d^{(t')}(z) > d^{(t)}(z)$. For a contradiction, assume $d^{(t')}(z) = d^{(t)}(z)$; let $d + 1$ be this common value. Since the arc (z, w) is admissible both at time t and at t', we see that $d^{(t')}(w) = d^{(t)}(w) = d$. Since labels may only increase, $d^{(\tau)}(z) = d + 1$ and $d^{(\tau)}(w) = d$ for every

τ for $t \leq \tau \leq t'$. We observe the following properties of the sequence $\{(t_j, u_j, v_j)\colon 0 \leq j < k\}$ defined above.

Claim: For all $0 \leq j < k$, we have $d^{(t_j)}(u_j) = d^{(t_j)}(v_j) + 1 = d + 1$.

Proof: Since (u_j, v_j) is admissible in iteration t_j, we immediately have $d^{(t_j)}(u_j) = d^{(t_j)}(v_j) + 1$. Thus it suffices to show that $d^{(t_j)}(u_j) = d + 1$. We can show that $d^{(t_j)}(u_j) \geq d + 1$ as follows. By the first part of (2), $v_j \in T_i^{(t_j)}(z)$, which gives $d^{(t_j)}(v_j) \geq d^{(t_j)}(z) - 1 = d$. Consequently, $d^{(t_j)}(u_j) = d^{(t_j)}(v_j) + 1 \geq d + 1$.

We prove the reverse inequality $d^{(t_j)}(u_j) \leq d + 1$ by induction. We start with the base case $j = 0$. Since $w \in T_i^{(t_0+1)} \setminus T_i^{(t_0)}$, Lemma A.4(i) is applicable with $z' = w$, showing that $(u_0, w) \in E_i^{(t_0)}$. Thus, $d^{(t_0)}(u_0) \leq d^{(t_0)}(w) + 1 = d + 1$.

Assuming the claim is true for j, we now show it holds for $j + 1$. Using the third part of (2), Lemma A.4(i) is applicable with $z' = v_j$, giving $(u_{j+1}, v_j) \in E_i^{(t_{j+1})}$, and thus

$$d^{(t_{j+1})}(u_{j+1}) \leq d^{(t_{j+1})}(v_j) + 1 \leq d^{(t_j)}(v_j) + 1 = d^{(t_j)}(u_j) \leq d + 1.$$

The second inequality used that the distance labels are non-decreasing. This completes the proof. □

The last step of the proof leverages the tie-breaking rule. Recall that if for a node u, there are more than one admissible arcs (u, v), then we select v as the $\prec$-minimal one for a fixed linear ordering.

Claim: We have $w \preceq v_0$.

Proof: We first show that $w \preceq v_{k-1}$, and then, for every $j = k-2, k-3, \ldots, 0$, we show that $v_{j+1} \preceq v_j$. These together imply the claim.

Let us start by showing $w \preceq v_{k-1}$. Because $v_{k-1} \in T_i^{(t_k+1)}(z) = T_i^{(t)}(z)$, we have $(z, v_{k-1}) \in E_i^{(t)}$. Further, (z, v_{k-1}) is admissible. This follows since $d^{(t)}(z) = d + 1$ and $d^{(t)}(v_{k-1}) \leq d^{(t_{k-1})}(v_{k-1}) = d$ by the previous claim. Thus, $w \preceq v_{k-1}$ because of the tie-breaking rule.

Let us now show $v_{j+1} \prec v_j$. Using (2) and Lemma A.4(i), we have $(u_{j+1}, v_j) \in E_i^{(t_{j+1})}$. Again, (u_j, v_{j+1}) is admissible, since $d^{(t_{j+1})}(u_{j+1}) = d + 1$ and $d^{(t_{j+1})}(v_j) \leq d^{(t_j)}(v_j) = d$, by the previous claim. Since (u_{j+1}, v_{j+1}) is pushed, we get $w \preceq v_{j+1} \prec v_j$ by the induction hypothesis and the tie-breaking rule. □

Using that $w \in T_i^{(t_0+1)} \setminus T_i^{(t_0)}$, Lemma A.4(i) gives $(u_0, w) \in E_i^{(t_0)}$ and $w \neq v_0$. Again, we see that (u_0, w) is admissible, since $d^{t_0}(u_0) = d + 1$, and $d^{(t_0)}(w) = d$. However, we pushed on (u_0, v_0), in contradiction to $w \preceq v_0$ shown in the previous claim. We thus derived a contradiction from the assumption $d^{(t')}(z) = d^{(t)}$. The proof of the lemma is complete. □

Now we upper bound the number of non-saturating pushes.

Lemma A.13. *The total number of non-saturating pushes is $O(n^2 \sum_{i=1}^r |C_i|^2)$.*

Proof: We say that u is active at time t if $u \in N^{(t)}$. To analyze the number of non-saturating pushes, we use the standard potential function $\Phi(t) = \sum_{v \in N^{(t)}} d^{(t)}(v)$. We have $\Phi(0) = 0$, and $\Phi(t) \leq n^2$ at every time t. The relabel operations altogether increase $\Phi(t)$ by at most n^2. Each saturating push on (u, v) can activate v, increasing $\Phi(t)$ by at most n in the process. By Lemma A.11, the total number of saturating pushes is $O(n \sum_{i=1}^r |C_i|^2)$, and thus the saturating pushes altogether increase Φ by $O(n^2 \sum_{i=1}^r |C_i|^2)$. Each non-saturating push on (u, v) deactivates u but could activate v; since (u, v) is admissible, $d(u) = d(v) + 1$ and thus the push decreases $\Phi(t)$ by at least 1. Therefore the number of non-saturating pushes is at most $O(n^2 \sum_{i=1}^r |C_i|^2)$. □

Finally, let us call a non-saturating push on an i-arc (u, v) an *expanding push*, if $T_i(v) \subsetneq T_i(u)$ before the push. A *non-expanding* push is a non-saturating push, where $T_i(v) = T_i(u)$ before the push.

Lemma A.14. *Non-expanding pushes do not change any of the sets $T_i(z)$. There can be at most $|C_i|$ expanding pushes on i-arcs between two saturating pushes on i-arcs. Further, non-saturating pushes do not change the values* $\min\{d(w) : w \in T_i(z)\}$ *for any $z \in V_i$.*

Proof: Consider a non-saturating push on an i-arc (u, v). By Lemma A.4(ii), we see that if $T'_i(z) \neq T_i(z)$ for some $z \in V_i$, then $T_i(v) \subseteq T_i(z) \cap T_i(u) \subsetneq T_i(u)$. Consequently, the push is expanding. Let us define the potential

$$\Psi_i := |\{T_i(z) : z \in V_i\}|,$$

i.e. the different number of sets among the $T_i(z)$'s. The same Lemma A.4(ii) implies that $T'_i(v) = T'_i(u)$ after the push; further, if $T_i(z) = T_i(w)$ before the push, then $T'_i(z) = T'_i(w)$ after the push. Hence the value of Ψ_i decreases by at least one. Clearly, the non-expanding pushes do not affect Ψ_i. Since $\Psi_i \leq |C_i|$, we see the bound on the number of non-expanding pushes.

Let us now turn to the final claim. There is nothing to prove for non-expanding pushes, as they do not affect any $T_i(z)$'s. If $T_i(z)$ is modified during the expanding push on the i-arc (u, v), it is updated to $T_i(z) \cup T_i(u)$, with $v \in T_i(z) \cap T_i(u)$. We have $d(u) = d(v) + 1$, and $d(u) \leq d(w) + 1$ for any $v' \in T_i(u)$, therefore $d(v) = \min\{d(w) : v \in T_i(u)\}$. The claim follows since $v \in T_i(z)$ and therefore $\min\{d(w) : w \in T_i(z)\} \leq d(v)$. □

Proof of Theorem A.1: At initialization, we compute all sets $T_i(z)$, in running time $\sum_{i \in r} |C_i|\Theta_i$. We note that for $z \notin V_i$, we have $T_i(z) = \{z\}$ throughout and hence these sets do not need to be maintained. The initial solutions x_i can be obtained in time $O(\sum_{i \in r} |C_i|)$ using the greedy algorithm for each f_i (recall that a value oracle call is assumed to take $O(1)$ time).

At every push operation, we need to call CAPACITY-UPDATE. Lemmas A.11 and A.13 give a bound $O(n^2 \Theta_{\max} \sum_{i=1}^{r} |C_i|^2)$. We use $\Theta_{\max}$, since we do not have individual bounds for non-saturating pushes on i-arcs, just on their total number.

The operations described in Algorithm 1 correctly update the sets $T_i(u)$. After every saturating push on an i-arc (u, v), we call TIGHT-UPDATE(z, i) for every arc $(z, v) \in E_i$. By Lemma A.4(i), all other sets $T_i(z)$ are unchanged. Also, we see that for non-saturating pushes, Lemma A.4(ii) implies that setting $T'_i(z) = T_i(z) \cup T_i(u)$ whenever $v \in T_i(z)$, and $u \notin T_i(z)$ correctly updates the sets.

In order to achieve a better complexity bound, we now describe an alternative variant. Instead of maintaining the graph $G = (V, E)$ explicitly, we just keep track of a node $v \in T_i(u)$ with $d(v)$ minimal, for every $i \in [r]$ and every $u \in V_i$. Let

$$\beta_i(u) := \operatorname{argmin}\{d(v) \colon v \in T_i(u)\}$$

denote this node. We further maintain $\beta^*(u) := \operatorname{argmin}\{\beta_i(u) \colon u \in V_i\}$ as an out-neighbour of u of minimum label. Given these values, identifying admissible arcs and computing the new label values are trivial.

Let us now describe how the $\beta_i(u)$ and $\beta^*(u)$ values can be maintained. Lemma A.14 shows that no changes are required for non-saturating pushes. Whenever we perform a saturating push on an i-arc (u, v), let us call TIGHT-UPDATE(z, i) for every $z \in V_i$, to recompute all sets $T_i(z)$.[7] We then find the new $\beta_i(z)$'s, and also update $\beta^*(z)$ if needed. These operations take $|C_i|(\Theta_i + |C_i|) \leq 2|C_i|\Theta_i$ time. According to Lemma A.11, their total complexity is bounded by $O(n \sum_{i=1}^{r} |C_i|^3 \Theta_i)$ throughout the algorithm.

Whenever we increase the label of a node u, we also recompute all the $T_i(z)$'s for every i with $u \in V_i$, and every $z \in V_i$. One such update requires $\sum_{i: u \in V_i} |C_i|(\Theta_i + |C_i|) \leq 2 \sum_{i: u \in V_i} |C_i|\Theta_i$ time. We recall that every node label can increase at most n times. Hence the total complexity of these updates is at most

$$2n \sum_{i \in V} \sum_{i: u \in V_i} |C_i|\Theta_i = 2n \sum_{i=1}^{r} \sum_{u \in V_i} |C_i|\Theta_i = 2n \sum_{i=1}^{r} |C_i|^2 \Theta_i.$$

We see that the running time of the capacity updates dominates the overall running time. □

A.3 Preflow-push algorithm with excess scaling

In the previous section, we analyzed the basic version of preflow-push. Our framework allows for adaptation of other more advanced versions and in this section, as an example, we consider the scaling version of preflow-push. This is the adaptation of the classical excess scaling algorithm for maximum flows, as described in [1, Section 7.9].

[7] Note that we do not only call this for z with $v \in T_i(z)$. The reason is that we do not maintain the $T_i(z)$'s, and hence cannot decide whether $v \in T_i(z)$ holds.

The algorithm runs in scaling phases, governed by the scaling factor Δ. This is initialized as $\Delta = 2^k$ for the smallest value of k such that $\Delta \geq \max_{u\in N} -x(u)$ of an initial solution $x = \sum_{i=1}^r x_i$, x_i is a vertex of $B(f_i)$. Clearly, we can select $k \leq \log_2 F_{\max} + 1$.

Throughout the Δ-phase, we maintain the invariant that

$$x(u) \geq -\Delta \qquad \forall u \in V. \tag{3}$$

In each iteration in the Δ-phase, the algorithm finds the indices u with $x(u) < -\Delta/2$ and among those, finds the one with minimum label $d(u)$. If there are admissible arcs from u, the algorithm pushes on the admissible i-arc (u, v) with the minimum v. Otherwise, the algorithm updates $d(u)$ by the usual relabel operation. The amount to push on an i-arc (u, v) is

$$\min\{-x(u), c_i(u, v), \Delta + x(v)\}.$$

The last term is needed, so that the resulting $x(v)$ after the push is at least $-\Delta$. The Δ-phase terminates, once $x(u) > -\Delta/2$ holds for all $u \in N \setminus H$. We select the next scaling factor as $\Delta \leftarrow \Delta/2$.

We selected the initial value of Δ as a power of 2. We terminate at the end of the 1-phase. Hence if the f_i's are integer, then we can maintain integrality throughout. The condition that $x(u) > -1/2$ for all $u \in N \setminus H$ means that $x(u) = 0$ for all such notes. The termination subroutine of the preflow-push algorithm is applicable to find a minimizer of $f(S)$.

Lemma A.15. *The scaling algorithm performs at most $O(n^2 \log F_{\max})$ non-saturating pushes.*

Proof: Consider the potential function

$$\Phi := -\sum_{u\in N} \frac{x(u)d(u)}{\Delta}.$$

Relabelling u increases Φ by $0 \leq -x(u)/\Delta \leq 1$. Thus, all relabel operations together increase Φ by $O(n^2)$. Due to (3), $0 \leq \Phi \leq n^2$ throughout. The rescaling at the end of a phase doubles Φ, but because of this upper bound, these increases amount to a total $O(n^2 \log F_{\max})$, noting that the number of phases is bounded by $\log_2 F_{\max} + 1$.

Each push on any i-arc (u, v) decreases Φ, because $d(u) > d(v)$. Recall that selection rule of pushing from a node u with minimal label $d(u)$. Therefore, for every push on (u, v), it must be the case that $x(v) \geq -\Delta/2$, i.e., $\Delta + x(v) \geq \Delta/2$. Thus, a non-saturating push always pushes at least $\min\{-x(u), \Delta + x(v)\} \geq \Delta/2$ unit of flow. Therefore, a non-saturating push decreases Φ by at least $1/2$. We conclude that the number of non-saturating pushes is $O(n^2 \log F_{\max})$. $\square$

Theorem A.16. *The scaling algorithm runs in time $O(n^2\Theta_{\max} \log F_{\max} + n \sum_{i=1}^r |C_i|^3\Theta_i)$.*

Proof: The proof correctness and the bound on the number of saturating pushes, as well as the bounds on the time complexity of the update operations are the same as in Section A.2 for the preflow-push algorithm. We use the bound on non-saturating pushes from the previous lemma. $\square$

B Rates of convergence of the continuous DSFM algorithms

The following theorems state the running time dependence of the continuous algorithms given in Section 3, as a function of the parameters defined in Definition 3.1 and Definition 3.2.

Theorem B.1 ([25]). *Let $(a^{(0)}, x^{(0)} = \operatorname{argmin}_{x\in\mathcal{P}} \|a^{(0)} - x\|_2)$ be the initial solution and let (a^*, x^*) be an optimal solution to (Best-Approx). The alternating projection algorithm produces in*

$$k = \Theta\left(\kappa_*^2 \ln\left(\frac{\|x^{(0)} - x^*\|_2}{\epsilon}\right)\right)$$

iterations a pair of points $a^{(k)} \in \mathcal{A}$ and $x^{(k)} \in \mathcal{P}$ that is ϵ-optimal, i.e.,

$$\|a^{(k)} - x^{(k)}\|_2^2 \leq \|a^* - x^*\|_2^2 + \varepsilon.$$

Theorem B.2 ([6]). *Let $x^{(0)} \in \mathcal{P}$ be the initial solution and let x^* be an optimal solution to (Prox-DSFM) that minimizes $\|x^{(0)} - x^*\|_2$. The random coordinate descent algorithm produces in*

$$k = \Theta\left(\frac{r}{\ell_*} \ln\left(\frac{\|x^{(0)} - x^*\|_2}{\epsilon}\right)\right)$$

iterations a solution $x^{(k)}$ that is ϵ-optimal in expectation, i.e., $\mathbb{E}\left[\frac{1}{2}\|Ax^{(k)}\|_2^2\right] \leq \frac{1}{2}\|Ax^*\|_2^2 + \epsilon$. The accelerated coordinate descent algorithm produces in

$$k = \Theta\left(r\sqrt{\frac{1}{\ell_*}}\ln\left(\frac{\|x^{(0)} - x^*\|_2}{\epsilon}\right)\right)$$

iterations (specifically, $\Theta\left(\ln\left(\frac{\|x^{(0)}-x^*\|_2}{\epsilon}\right)\right)$ epochs with $\Theta\left(r\sqrt{\frac{1}{\ell_*}}\right)$ iterations in each epoch) a solution $x^{(k)}$ that is ϵ-optimal in expectation, i.e., $\mathbb{E}\left[\frac{1}{2}\|Ax^{(k)}\|_2^2\right] \leq \frac{1}{2}\|Ax^*\|_2^2 + \epsilon$.

C Proofs omitted from Section 5

Lemma C.1. *The capacity $c(u,v) := \min\{f_i(S) - x_i(S) \colon S \subseteq C_i, u \in S, v \notin S\}$ can be computed as the minimum value of $\min_{S \subseteq C_i} f_i(S) + w(S)$ for an appropriately chosen vector $w \in \mathbb{R}^n$, supp$(w) \subseteq C_i$.*

Proof: We define a weight vector $w \in \mathbb{R}^n$ as follows: $w(u) = -(f_i(\{u\}) + 1)$; $w(v) = -(f_i(C_i) - f_i(C_i \setminus \{v\}) - 1)$; $w(a) = -x(a)$ for all $a \in C_i \setminus \{u,v\}$, and $w(a) = 0$ for all $a \notin C_i$. Let $A \subseteq C_i$ be a minimizer of $\min_{S \subseteq C_i} f_i(S) + w(S)$. It suffices to show that $u \in A$ and $v \notin A$. Note that $f_i(\{u\}) = f_i(\{u\}) - f(\emptyset)$ is the maximum marginal value of u, i.e., $\max_S(f_i(S \cup \{u\}) - f_i(S))$. Moreover, $f_i(C_i) - f_i(C_i \setminus \{v\})$ is the minimum marginal value of v. To show $u \in A$, let us assume for a contradiction that $u \notin A$.

$$\begin{aligned}
&f_i(A \cup \{u\}) + w(A \cup \{u\}) \\
&= (f_i(A) + w(A)) + (f_i(A \cup \{u\}) - f_i(A)) + w(u) \\
&= (f_i(A) + w(A)) + (f_i(A \cup \{u\}) - f_i(A)) \\
&\quad - f_i(\{u\}) + 1 \\
&\leq f_i(A) + w(A) - 1.
\end{aligned}$$

Similarly, to show that $v \notin A$, suppose for a contradiction that $v \in A$, and consider the set $A \setminus \{v\}$. Since $f_i(C_i) - f_i(C_i \setminus \{v\}) \leq f_i(A) - f_i(A \setminus \{v\})$, we have

$$\begin{aligned}
&f_i(A \setminus \{v\}) + w(A \setminus \{v\}) \\
&= (f_i(A) + w(A)) - (f_i(A) - f_i(A \setminus \{v\})) - w(v) \\
&= (f_i(A) + w(A)) - (f_i(A) - f_i(A \setminus \{v\})) \\
&\quad + (f_i(C_i) - f_i(C_i \setminus \{v\})) - 1 \\
&\leq f_i(A) + w(A) - 1.
\end{aligned}$$

Therefore $u \in A$ and $v \notin A$, and hence $A \in \operatorname{argmin}\{f_i(S) - x_i(S) \colon u \in S, v \notin S\}$. □

D Full experimental results

We evaluate the algorithms on energy minimization problems that arise in image segmentation problems. We follow the standard approach and model the image segmentation task of segmenting an object from the background as finding a minimum cost $0/1$ labeling of the pixels. The total labeling cost is the sum of labeling costs corresponding to *cliques*, where a clique is a set of pixels. We refer to the labeling cost functions as clique potentials.

The main focus of our experimental analysis is to compare the running times of the decomposable submodular minimization algorithms. Therefore we have chosen to use the simple hand-tuned potentials that were used in previous work [30, 2, 31]: the *edge-based costs* defined by [2] and the *count-based costs* defined by [31]. Specifically, we used the following clique potentials in our experiments, all of which are submodular:

- **Unary potentials** for each pixel. The unary potentials are derived from Gaussian Mixture Models of color features [27].
- **Pairwise potentials** for each edge of the 8-neighbor grid graph. Each graph edge (i,j) between pixels i and j is assigned a weight that is a function of $\exp(-\|v_i - v_j\|^2)$, where v_i is the RGB color vector of pixel i. The clique potential for the edge is the cut function of the edge: the cost of a labeling is equal to zero if the two pixels have the same label and it is equal to the weight of the edge otherwise.

- **Square potentials** for each 2×2 square of pixels. We view a 2×2 square as a graph on 4 nodes connected with 4 edges (two horizontal and two vertical edges). The cost of a labeling is the square root of the number of edges of the square that have different labels. This is the basic edge-based potential defined by [2].
- **Region potentials** for a set of regions of the image. We compute a set of regions of the image using the region growing algorithm suggested by [31]. For each region C_i, we define a count-based clique potential as in [31, 30]: for each set $S \subseteq C_i$ of pixels, $f_i(S) = |S||C_i \setminus S|$.

We used five image segmentation instances to evaluate the algorithms[8]. Table 1 in the supplement provides the sizes of the resulting instances. The experiments were carried out on a single computer with a 3.3 GHz Intel Core i5 processor and 8 GB of memory. The reported times are averaged over 10 trials.

We have run the coordinate descent algorithms for $1000r$ iterations, where r is the number of functions in the decomposition. Our choice is based on the empirical results of Jegelka *et al.* [19] that showed that this number of iterations suffices to obtain good results. The running time per iteration of ACDM is higher than RCDM, but ACDM converges faster both theoretically and empirically [6].

We performed several experiments with various combinations of potentials and parameters.

Minimum cut experiments. We evaluated the algorithms on instances containing only the unary potentials and the pairwise potentials.

Small cliques experiments. We evaluated the algorithms on instances containing the unary potentials, the pairwise potentials, and the square potentials.

Large cliques experiments. We evaluated the algorithms on instances containing all of the potentials: the unary potentials, the pairwise potentials, the square potentials, and the region potentials. For the region potentials, we used a potential-specific level-0 algorithm that performs quadratic minimization over the base polytope in time $O(|C_i| \log(|C_i|) + |C_i|\mathrm{EO}_i)$. Additionally, due to the slow running time of IBFS, we used smaller regions: 50 regions with an average size between 45 and 50.

Large cliques experiments with Fujishige-Wolfe algorithm. We also ran a version of the large cliques experiments with the Fujishige-Wolfe algorithm as the level-0 algorithm for the region potentials. The Fujishige-Wolfe algorithm was significantly slower than the potential-specific quadratic minimization algorithm and in our experiments it was prohibitive to run the Fujishige-Wolfe algorithm to near-convergence. Since the IBFS algorithm requires almost exact quadratic minimization in order to compute exchange capacities, it was prohibitive to run the IBFS algorithm with the Fujishige-Wolfe algorithm. In contrast, the coordinate descent methods can potentially make progress even if the level-0 solution is far from being converged.

In order to empirically evaluate this hypothesis, we made a simple but crucial change to the Fujishige-Wolfe algorithm: we *warm-started* the algorithm with the current solution. Recall that the coordinate descent algorithms maintain a solution $x_i \in B(f_i)$ for each function f_i in the decomposition. We warm-started the Fujishige-Wolfe algorithm with the current solution x_i, and we ran the algorithm for a small number of iterations. In our experiments, we ran the Fujishige-Wolfe algorithm for 10 iterations. These changes made the level-0 running time considerably smaller, which made it possible to run the level-1 coordinate descent algorithms for as many as $1000r$ iterations. At the same time, performing 10 iterations starting from the current solution seemed enough to provide an improvement over the current solution.

[8]The data is available at `http://melodi.ee.washington.edu/~jegelka/cc/index.html` and `http://research.microsoft.com/en-us/um/cambridge/projects/visionimagevideoediting/segmentation/grabcut.htm`

Table 1: Instance sizes

image	# pixels	# edges	# squares	# regions	min, max, and average region size		
bee	273280	1089921	68160	50	298	299	298.02
octopus	273280	1089921	68160	49	7	299	237.31
penguin	154401	615200	38400	50	5	299	279.02
plant	273280	1089921	68160	50	8	298	275.22
plane	154401	615200	38400	50	10	299	291.48

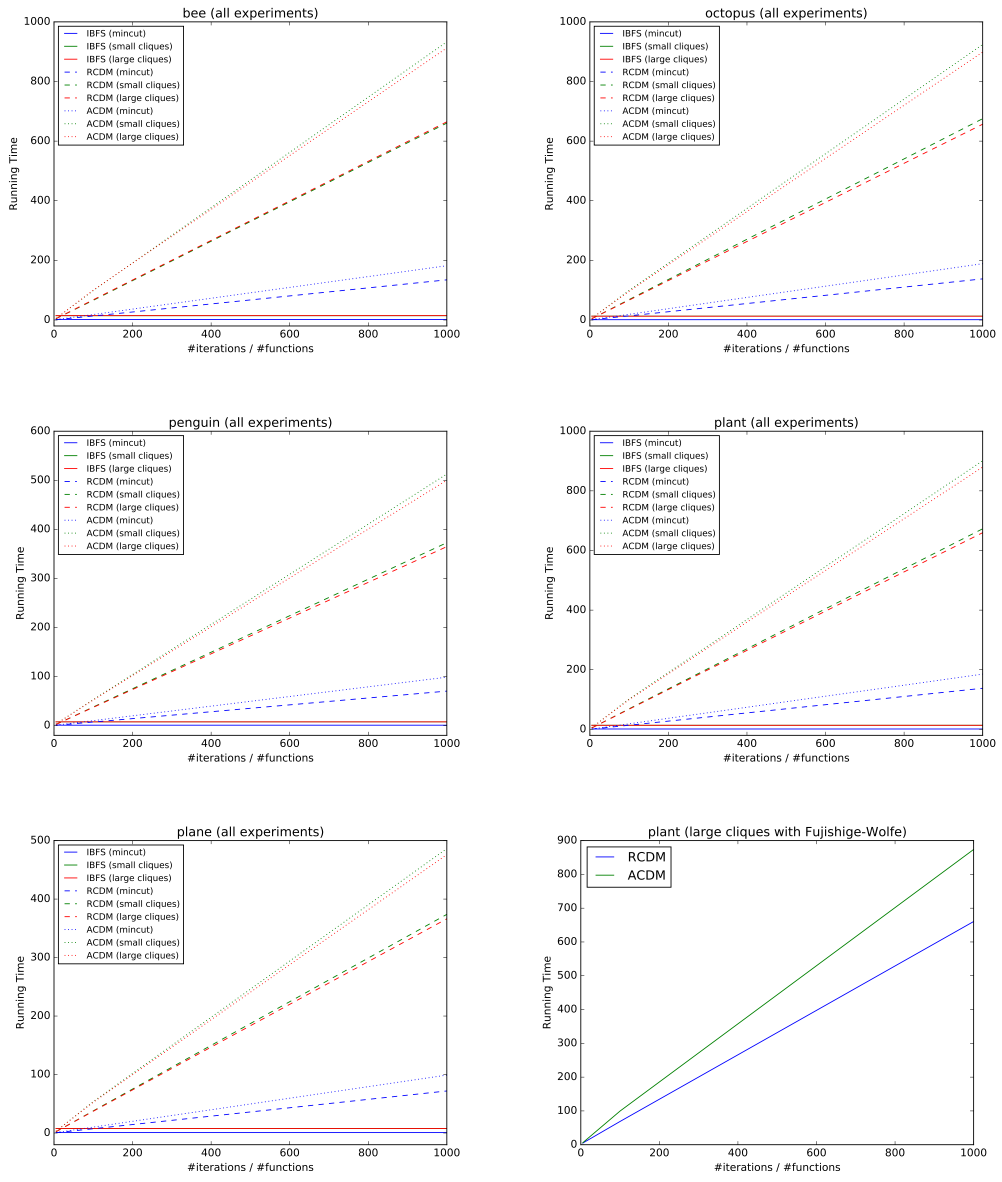

Figure 3: Running times (in seconds). The x-axis shows the number of iterations for the continuous algorithms. The IBFS algorithm is exact, and we display its running time as a flat line.

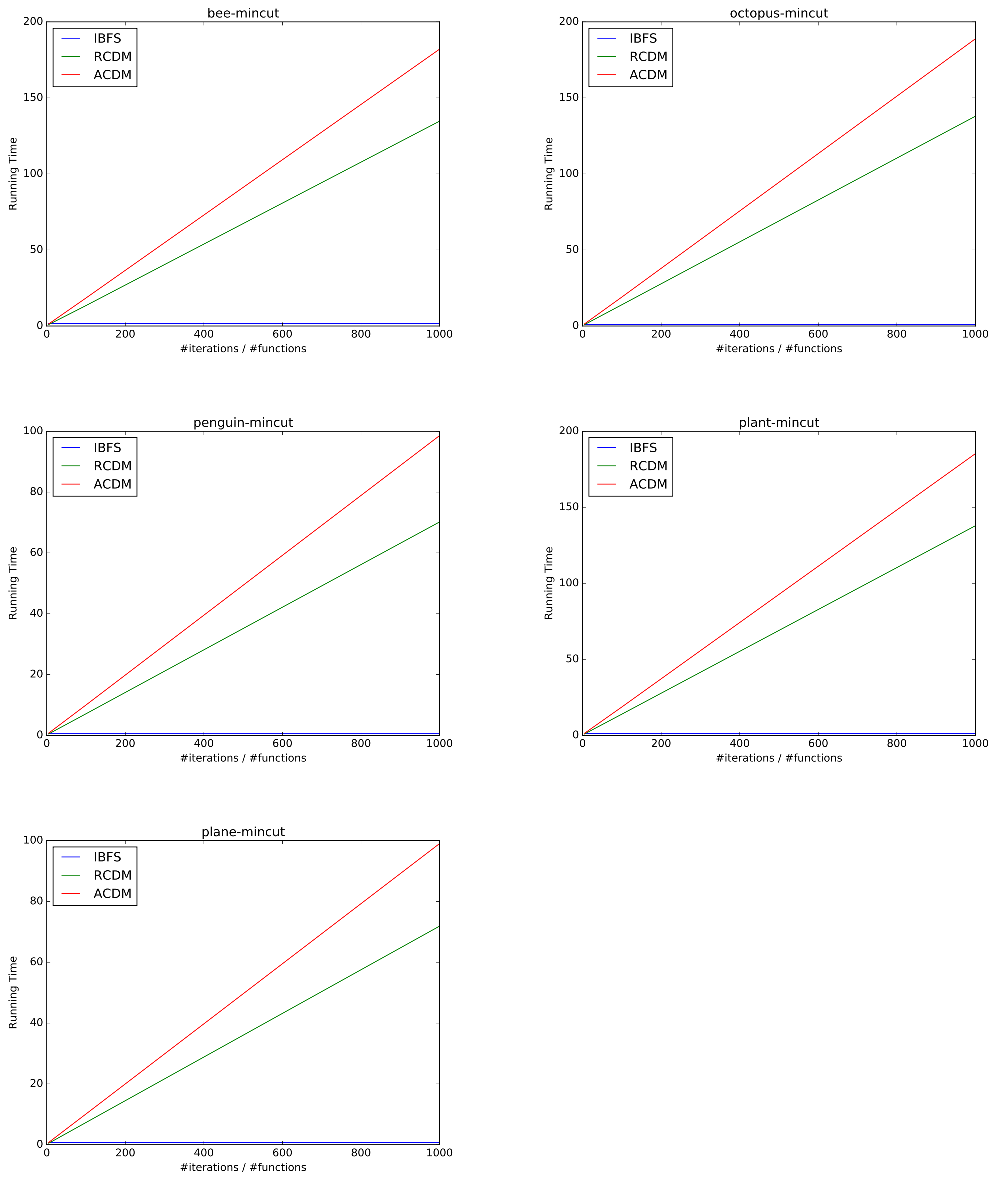

Figure 4: Running times (in seconds) for the minimum cut experiments. The x-axis shows the number of iterations for the continuous algorithms. The IBFS algorithm is exact, and we display its running time as a flat line.

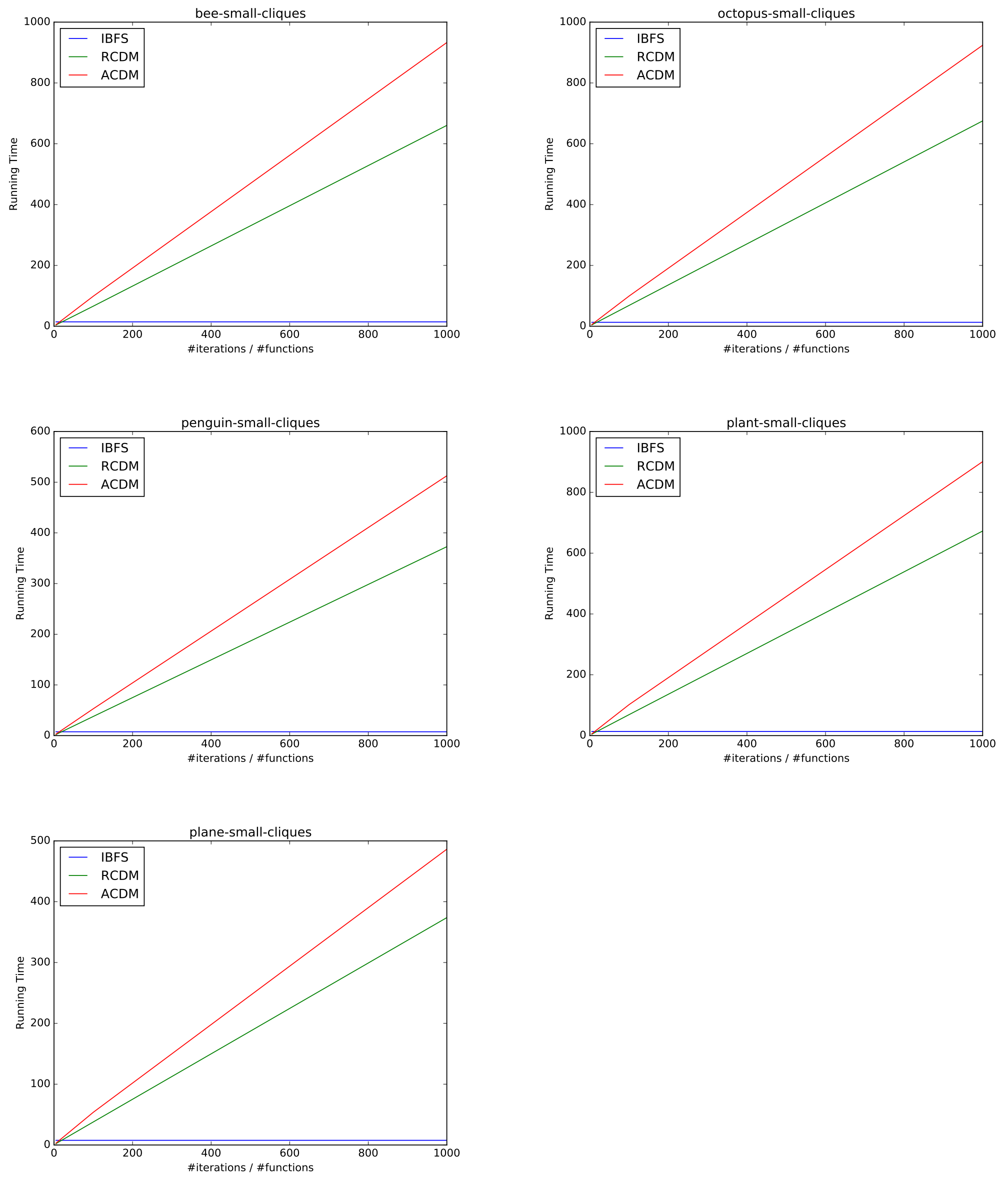

Figure 5: Running times (in seconds) for the small cliques experiments. The x-axis shows the number of iterations for the continuous algorithms. The IBFS algorithm is exact, and we display its running time as a flat line.

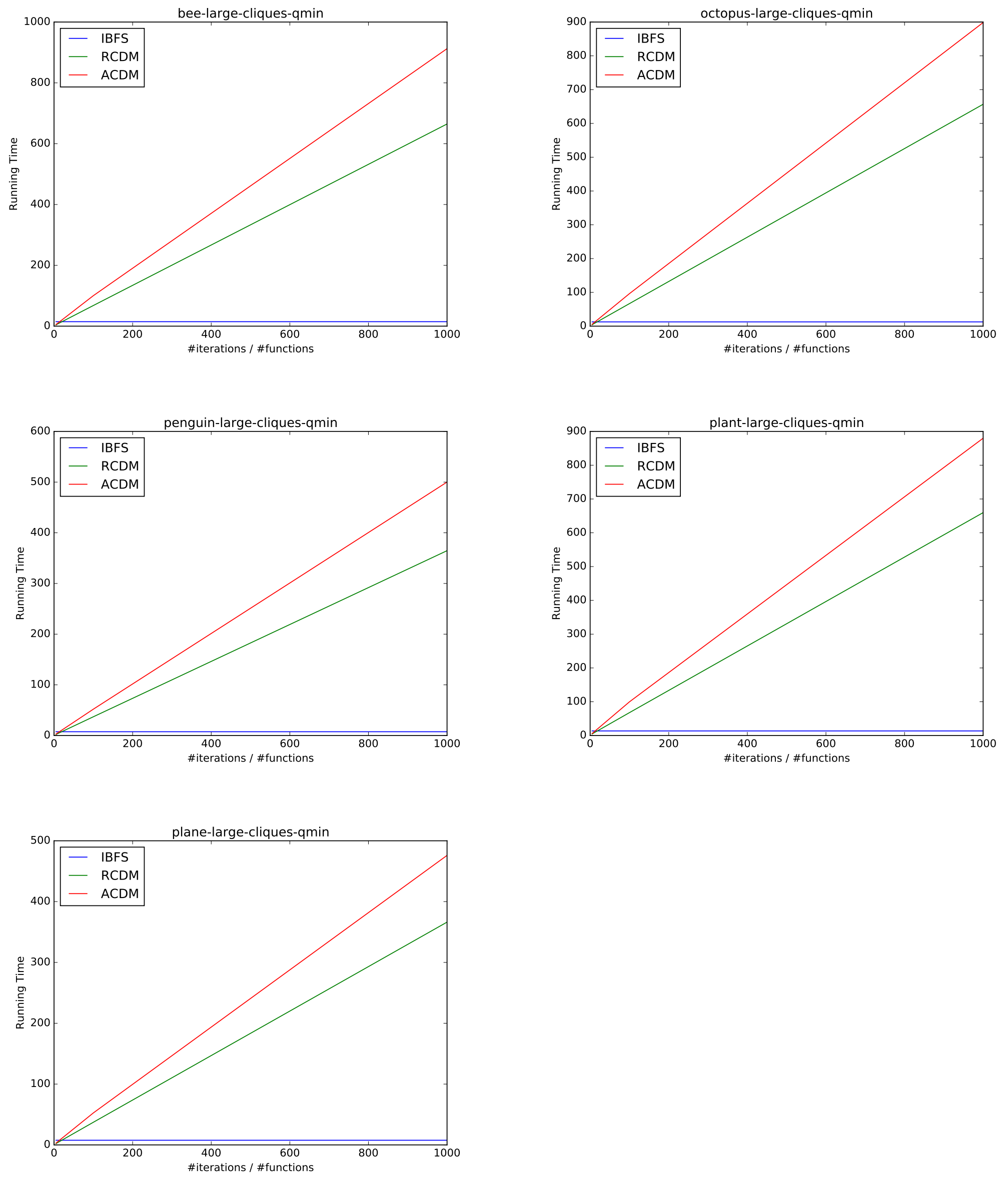

Figure 6: Running times (in seconds) for the large cliques experiments with potential specific quadratic minimization for the region potentials. In order to be able to run IBFS, we used smaller regions: 50 regions with an average size between 45 and 50. The x-axis shows the number of iterations for the continuous algorithms. The IBFS algorithm is exact, and we display its running time as a flat line.

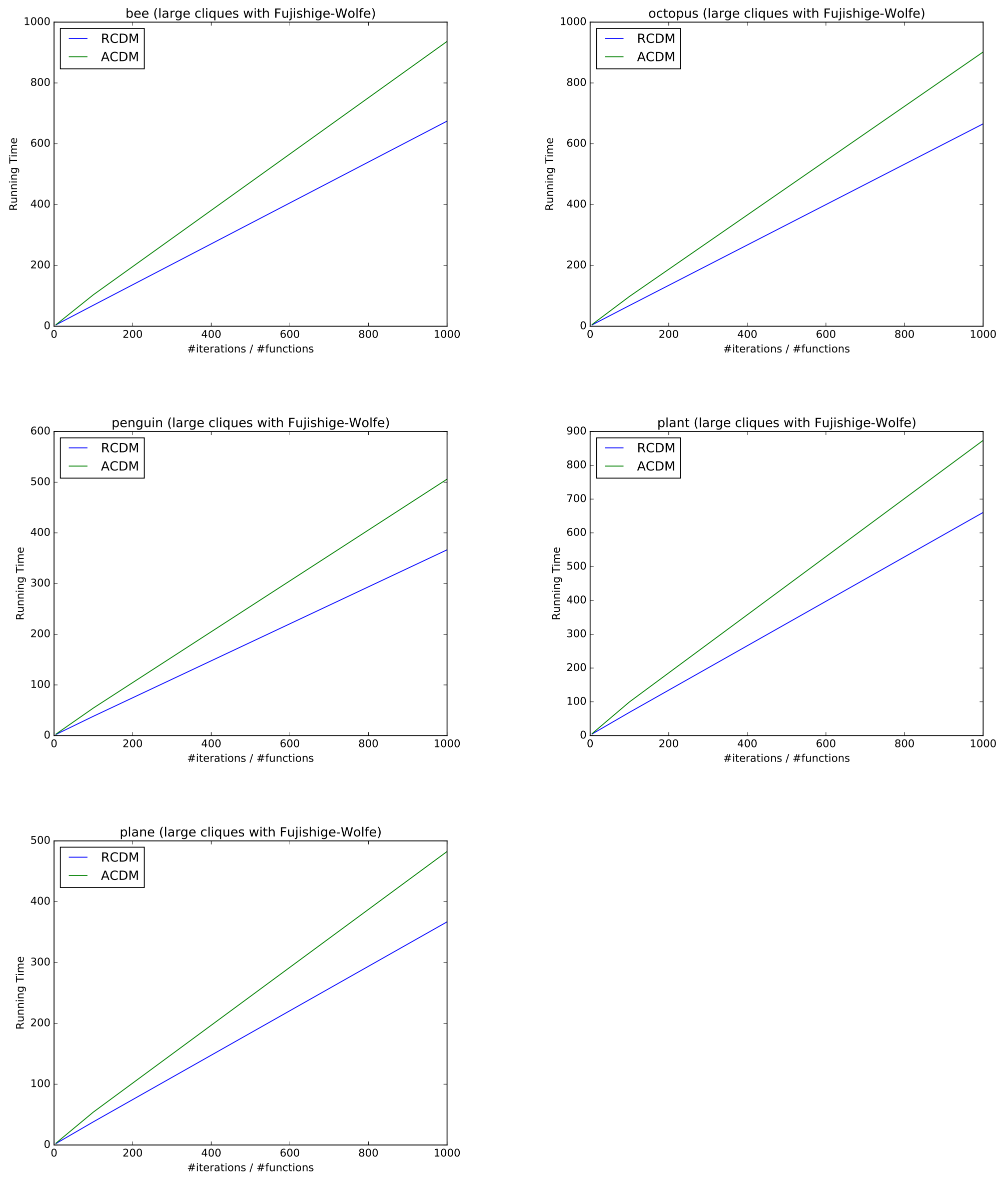

Figure 7: Running times (in seconds) for the large cliques experiments with the Fujishige-Wolfe algorithm. The x-axis shows the number of iterations for the continuous algorithms. We could not run the IBFS algorithm on these instances.