# Approximate Decomposable Submodular Function Minimization for Cardinality-Based Components

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Minimizing a sum of simple submodular functions of limited support is a special case of general submodular function minimization that has seen numerous applications in machine learning. We develop faster techniques for instances where components in the sum are cardinality-based, meaning they depend only on the size of the input set. This variant is one of the most widely applied in practice, encompassing, e.g., common energy functions arising in image segmentation and recent generalized hypergraph cut functions. We develop the first approximation algorithms for this problem, where the approximations can be quickly computed via reduction to a sparse graph cut problem, with graph sparsity controlled by the desired approximation factor. Our method relies on a new connection between sparse graph reduction techniques and piecewise linear approximations to concave functions. Our sparse reduction technique leads to significant improvements in theoretical runtimes, as well as substantial practical gains in problems ranging from benchmark image segmentation tasks to hypergraph clustering problems.

## 1 Introduction

Given a ground set $V$, a function $f\colon 2^V \to \mathbb{R}$ is submodular if for every $A, B \subseteq V$ it satisfies $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$. Submodular functions are ubiquitous in combinatorial optimization and machine learning, arising in the context of image segmentation [22], hypergraph clustering [29], data subset selection [42], and document summarization [30], among many other applications. Algorithms for minimizing submodular functions have been particularly well-studied. Although strongly-polynomial time algorithms for general submodular function minimization exist [35, 17, 18], runtimes for these methods are impractical in most cases. The past decade has witnessed a number of key advances in faster algorithms for *decomposable submodular function minimization* (DSFM), i.e., minimizing a sum of simpler submodular functions [9, 24, 38, 27, 34, 10, 21, 19]. This problem is defined by identifying a set $E \subseteq 2^V$ of subsets of the ground set. The goal is then to solve

$$\text{minimize}_{S \subseteq V} \; f(S) = \textstyle\sum_{e \in E} f_e(S \cap e), \tag{1}$$

where for each $e \in E$, $f_e$ is a submodular function supported on a subset $e \subseteq V$. Functions of this form often arise as energy functions in computer vision [22, 23, 13], and generalized cut functions for hypergraph clustering and learning [12, 26, 29, 31, 40, 41], among other applications.

In this paper we focus on *cardinality-based* DSFM (Card-DSFM), where every component in the sum is a concave cardinality function, i.e., for each $e \in E$ we are given a concave function $g_e$ such that $f_e(A) = g_e(|A|)$. Although a single concave cardinality function is effectively a function of one variable and is trivial to minimize, set functions obtained via *sums* of these functions are much more complex and have broad modeling power, making this one of the most widely studied and applied variants since the earliest work on DSFM [24, 38]. In terms of theory, previous research has

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

addressed specialized runtimes and solution techniques [24, 38, 22, 40]. In practice, cardinality-based decomposable submodular functions frequently arise as higher-order energy functions in computer vision [22], as set cover functions [38], and as generalized hypergraph cut functions [29, 31, 40, 41]. Even previous research on algorithms for the more general DSFM problem tends to focus on cardinality-based examples in experimental results [9, 38, 19, 28].

We present a new algorithm for finding fast approximate solutions for Card-DSFM that is purely combinatorial and relies on new methods for approximately reducing (1) to a *sparse* graph cut problem. The fact that concave cardinality functions are representable by graph cuts has previously been noted in different contexts [22, 38, 40]—this can be accomplished by combining simple graph *gadgets*, whose cut properties together model the function. However, previous work focuses on exact reduction techniques, does not consider the density of the resulting graph, and does not address any questions regarding the optimality of such a reduction. In constrast, we develop new *approximate* reduction methods leading to a *sparse* graph, which we show is *optimally* sparse in terms of the number of graph gadgets required. To do so, we show that representing a concave cardinality function with a sparse graph is equivalent to approximating a concave function with a piecewise linear curve, where the number of linear pieces determines the sparsity of the reduced graph. We then develop a new algorithm for the resulting piecewise linear approximation problem, and prove new bounds on the number of linear pieces needed for different types of concave functions that arise in practice.

Combining our reduction strategy with state of the art algorithms for maximum flow [15, 25, 39, 14] improves theoretical runtime guarantees for many problem regimes, including regimes where we are able to find *optimal* solutions faster than existing approaches. Most importantly, we obtain much faster guarantees for solving problems when an approximate solutions suffices. Our algorithm is also very easy to implement and lead to substantial practical improvements on benchmark image segmentation experiments and flow-based algorithms for hypergraph clustering.

## 2 Background on Graph Reductions

A nonnegative set function $f\colon 2^V \to \mathbb{R}^+$ on a ground set $V$ is *graph reducible* if there exists a directed graph $G = (V_f, E_f)$ on a larger node set $V_f = V \cup \mathcal{A} \cup \{s, t\}$, which includes an auxiliary node set $\mathcal{A}$ as well as source and sink nodes $s$ and $t$, whose $s$-$t$ cut structure models $f$. More precisely, $f$ is graph reducible if for every $S \subseteq V$ we have

$$f(S) = \underset{T \subseteq \mathcal{A}}{\text{minimize}} \ \ \mathbf{cut}_G(\{s\} \cup S \cup T), \tag{2}$$

where $\mathbf{cut}_G$ is the cut function on the graph $G$. In words, this says that evaluating $f(S)$ is equal to finding the minimum $s$-$t$ cut penalty that could be obtained by placing $S$ on the $s$-side of the cut and $V \backslash S$ on the $t$-side. Given such a graph, $f$ can be minimized by finding the minimum $s$-$t$ cut set $U^* \subseteq V \cup \mathcal{A}$ in $G$ and taking $S^* = V \cap U^*$. When (2) holds, we say that $G$ *models* $f$.

**Graph reduction strategy for Card-DFSM.** In general, the function $f(S) = \sum_{e \in E} f_e(S \cap e)$ is *not* a concave cardinality function, even if individual components $f_e$ in the sum are. However, in order to show $f$ is graph reducible, it suffices to find a small reduced graph $G_e$ for each $e \in E$ that models $f_e$ in the sense of (2), and then combine these together into a larger graph. We briefly review the reduction strategy of Veldt et al. [40], which is closely related to other approaches [38, 22]. Modeling $f_e$ can be accomplished by combining a set of *cardinality-based* (CB) gadgets [40]. For a set $e \subseteq V$ of size $k = |e|$, a CB-gadget is a small graph parameterized by positive weights $(a, b)$. Each $v \in e$ defines a node, and the gadget also involves a single auxiliary node $v_e$. Then for each $v \in e$, there is a directed edge from $v$ to $v_e$ with weight $a \cdot (k - b)$, and a directed edge from $v_e$ to $v$ with weight $a \cdot b$. The resulting graph gadget models the following concave cardinality function in the sense of (2):

$$f_{a,b}(S) = a \cdot \min\{|S| \cdot (k - b), (k - |S|) \cdot b\}. \tag{3}$$

To see why, consider where we must place the auxiliary node $v_e$ when solving a minimum $s$-$t$ cut problem involving this small graph gadget. If we place $i = |S|$ nodes on the $s$-side, then placing $v_e$ on the $s$-side has a cut penalty of $ab(k - i)$, whereas placing $v_e$ on the $t$-side gives a penalty of $ai(k - b)$. To minimize the cut as in (2) for a fixed $S \subseteq e$, we choose the smaller of the two cut penalties. Veldt et al. [40] showed that any cardinality-based submodular function $f_e$ on a ground set $e$ can be modeled by combining $|e| - 1$ CB-gadgets. Analogously, Stobbe and Krause [38] showed that a

concave cardinality function on a ground set $e$ can be decomposed into a sum of $|e| - 1$ *threshold potentials* [38] and can be represented by graph cuts, though they provided no explicit reduction strategy. Priori to this, Kohli et al. [22] highlighted a similar type of gadget that can be used when $f_e$ is symmetric. Using any of these techniques, we see that $f_e$ in the sense of (2) using a graph $G$ with $O(|e|)$ nodes and $O(|e|^2)$ edges.

# 3 Sparse Reductions via Piecewise Linear Approximation

We now turn to a new and refined problem of finding *sparse* graph reduction strategies for Card-DFSM. Using the fact that a set function $f$ defined by $f(S) = g(|S|)$ is submodular if $g$ is concave, we will prove a precise way in which finding a sparse approximate graph reduction for $f$ is equivalent to finding a piecewise linear approximation to $g$. All proofs are given in the supplementary material.

## 3.1 The sparse approximate reduction problem

We first introduce a special parameterized function with broad modeling power.

**Definition 1** *A $k$-node combined gadget function (CGF) of order $J$ is a function $\ell\colon [0,k] \to \mathbb{R}^+$ parameterized by $z_0 \geq 0$, $z_k \geq 0$, and $a_j > 0, b_j > 0$ for $j \in \{1, 2, \ldots, J\}$, with the form*

$$\ell(x) = z_0 \cdot (k - x) + z_k \cdot x + \sum_{j=1}^{J} a_j \min\{x \cdot (k - b_j), (k - x) \cdot b_j\}. \tag{4}$$

*where the parameters additionally satisfy $b_j < b_{j+1}$ for $j \in \{1, 2, \ldots, J-1\}$ and $b_J < k$.*

If we define a set function on a ground set $e$ of size $k$ by $f_e(S) = \ell(|S|)$, then $f_e$ is exactly the function that is modeled by combining $J$ CB-gadgets with parameters $(a_j, b_j)_{j=1}^{J}$, and additionally placing a directed edge from a source node $s$ to each $v \in e$ of weight $z_0$, and an edge from $v \in e$ to a sink node $t$ with weight $z_k$. Previous reduction techniques amount to proving that any concave function $g$ on an interval $[0, k]$ (representing a submodular function $f_e(S) = g(|S|)$) matches some $k$-node CGF of order $k - 1$ at integer inputs [40, 38]. We focus on a new sparse reduction problem.

**Definition 2** *Let $g\colon [0,k] \to \mathbb{R}^+$ be a nonnegative concave function and fix $\varepsilon \geq 0$. The sparsest approximate reduction (SpAR) problem seeks a $k$-node CGF $\ell$ with minimum order $J$ so that*

$$g(i) \leq \ell(i) \leq (1 + \varepsilon) g(i) \text{ for all } i \in \{0, 1, 2, \ldots k\}. \tag{5}$$

This problem is equivalent to finding the minimum number of CB-gadgets needed to approximately model a concave cardinality function. Thus, solving this problem provides the sparsest reduction in terms of a standard reduction strategy. Without loss of generality, Definition 2 is restricted to considering functions that upper bound $g$. Any other approximating function can be scaled to produce an upper bound without affecting the approximation guarantee we would eventually obtain for Card-DSFM. Importantly, we care about approximating the concave function $g$ only at integer inputs, since we are ultimately concerned with modeling the set function $f_e(S) = g(|S|)$. Satisfying $g(x) \leq \ell(x) \leq (1 + \varepsilon) g(x)$ for all $x \in [0, k]$ is a much stronger requirement and may require more CB-gadgets than is actually necessary to approximate $f_e$.

**Connection to piecewise linear approximation** The following new result establishes a one-to-one correspondence between a certain class of piecewise linear functions and combined gadget functions.

**Lemma 1** *The $k$-node CGF in* (4) *is nonnegative, piecewise linear, concave, and has exactly $J + 1$ linear pieces. Conversely, let $\ell'\colon [0,k] \to \mathbb{R}^+$ be concave and piecewise linear with $J + 1$ linear pieces, and let $m_i$ be the slope of the $i$th linear piece and $B_i$ be the $i$th breakpoint. Then $\ell'$ is uniquely characterized as the $k$-node CGF parameterized by $a_i = \frac{1}{k}(m_i - m_{i+1})$ and $b_i = B_i$ for $i \in \{1, 2, \ldots J\}$, $z_0 = g(0)/k$, and $z_k = g(k)/k$.*

This result tells us that solving SpAR (Def 2) for $g$ is equivalent to finding a concave piecewise linear function with the smallest number of linear pieces approximating $g$ at integer points. Although some techniques for approximating a concave function with a piecewise linear curve already exist [32, 33]

**Algorithm 1** GREEDYPLCOVER$(g, \varepsilon)$

**Input**: $\varepsilon \geq 0$, concave function $g$
**Output**: piecewise linear $\ell$ with fewest linear pieces such that $g(i) \leq \ell(i) \leq (1+\varepsilon)g(i)$
$\mathcal{L} \leftarrow \emptyset$
$u \leftarrow 0$ `// smallest integer not covered by approximating line`
**while** $u \leq k$ **do**
  $u', L \leftarrow$ NEXTLINE$(g, u, \varepsilon)$ `//line covering widest range [u, u'-1]`
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{L\}$
  $u \leftarrow u'$ `//new smallest integer not covered by linear piece`
**end while**
Return $\ell(x) = \min_{L \in \mathcal{L}} L(x)$

and could be used as heuristics for our problem, these are ultimately unable to find optimally solutions for SpAR. First of all, these methods focus on approximating concave functions over continuous intervals rather than just at integer inputs. More importantly, they do not provide instance optimal approximations, but only give upper bounds on the number of linear pieces needed. We therefore turn to new methods that will allow us to exactly solve our sparse approximation problem.

### 3.2 Optimal sparse approximate reduction

For a concave function $g$, our goal is to find a piecewise linear function $\ell$ with a minimum number of linear pieces satisfying condition (5). This is equivalent to finding a minimum-sized collection $\mathcal{L}$ of linear functions that "cover" points $\{i, g(i)\}$ for integers $i$, in the sense that (i) each linear function $L \in \mathcal{L}$ satisfies $g(i) \leq L(i)$ for all $i \in \{0, 1, \ldots, k\}$, and (ii) for every $i \in \{0, 1, \ldots, k\}$ there exists some $L \in \mathcal{L}$ satisfying $L(i) \leq (1+\varepsilon)g(i)$. Taking the lower envelop of all of all linear pieces in $\mathcal{L}$ yields the desired piecewise linear curve: $\ell(x) = \min_{L \in \mathcal{L}} L(x)$.

We develop a simple greedy algorithm to grow a collection $\mathcal{L}$ of linear functions satisfying these conditions. We let $\mathcal{L}$ only include linear pieces that are tangent to at least one point $\{i, g(i)\}$. If any linear piece were not tangent, we could replace it with a parallel line that is tangent to one of the points and provides a better approximation everywhere. Pseudocode for our greedy method is given in Algorithm 1. At each iteration, the algorithm identifies the integer $u$ that is not currently covered by a $(1+\varepsilon)$-approximating line. It then uses a subroutine NEXTLINE to find a line $L$ that covers $u$ with a $(1+\varepsilon)$ approximation, and additionally satisfies $g(t) \leq L(t) \leq (1+\varepsilon)g(t)$ for the largest possible integer $t$. This can be accomplished by taking the line through the point $\{u, (1+\varepsilon)g(u)\}$ that has the minimum possible slope while still upper bounding every point $\{i, g(i)\}$. Pseudocode for this subroutine, and a proof of the following result, are given in the supplementary material.

**Lemma 2** *Algorithm 1 solves the sparsest approximate reduction problem in $O(k)$ operations.*

The greedy strategy provides an optimal solution for a simple intuitive reason. We know that $\mathcal{L}$ must contain *some* line that provides a $(1+\varepsilon)$-approximation at $\{0, g(0)\}$. We may as well choose a line that provides this guarantee while also providing the same approximation for the widest possible range of integer values $\{0, 1, \ldots, t\}$. After including such a line, the same logic applies to finding linear pieces to approximate the function at remaining integer inputs.

### 3.3 Bounds on optimal reduction size

For a concave function $g$, an existing method of Magnanti and Stratila [32, 33] can find a piecewise linear function that approximates $g$ over a continuous interval $[a, b]$ with $O(\frac{1}{\varepsilon} \log \frac{b}{a})$ linear pieces. Although this method does not optimally solving SpAR, it can be used to show the following worst-case upper bound on the number of linear pieces found by our instance-optimal method.

**Theorem 3** *Let $g\colon [0, k] \to \mathbb{R}^+$ be concave and let $\varepsilon \geq 0$. Algorithm 1 will return a piecewise linear function $\ell$ with at most $O(\min\{k, \frac{1}{\varepsilon} \log k\})$ linear pieces satisfying* (5).

Magnanti and Stratila [33] also consider lower bounds on the number of linear pieces needed to approximate certain concave functions, but these do not directly apply to our setting as they focus

on approximating functions over a continuous interval rather than just at integer inputs. Since it is always possible to cover $k$ points on a concave function using $k$ linear pieces, identifying meaningful lower bounds for our problem is much more challenging. Nevertheless, using the lower-bound results of Magnanti and Stratila [33] as a black-box, we are able to prove that the upper bound in Theorem 3 is nearly asymptotically tight for our problem.

**Theorem 4** *Let $g(x) = \sqrt{x}$ and $\varepsilon \geq k^{-\delta}$ for some constant $\delta \in (0, 2)$. Every piecewise linear $\ell$ satisfying $g(i) \leq \ell(i) \leq (1+\varepsilon)g(i)$ for $i \in \{0, 1, 2, \ldots k\}$ contains $\Omega(\varepsilon^{-1/2} \log k)$ linear pieces.*

Despite these worst case results, Algorithm 1 finds an instance optimal approximation for any concave function $g$, and in many cases the number of linear pieces it returns will be far smaller than the bounds above and the number of pieces returned by previous approaches [33]. A key contribution of our work is the following guarantee for a concave function that arises extensively in applications.

**Theorem 5** *Let $g(x) = x \cdot (k - x)$ for a positive integer $k$. For $\varepsilon > 0$, the approximating function $\ell$ returned by Algorithm 1 will have $O(\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ linear pieces.*

This result is significant in a number of ways. First of all, and somewhat surprisingly, the number of linear pieces needed to approximate $g$ is independent of $k$. Our instance optimal algorithm therefore leads to a significant improvement over the $O(\frac{1}{\varepsilon} \log k)$ upper bound in Theorem 3 and the piecewise linear approximation that could be obtained using existing techniques [33].

Even more importantly, Theorem 5 has significant implications for approximately modeling the concave cardinality function $f_e(S) = c_e|S||e\backslash S|$ (where $c_e$ can be any positive constant) using graph cuts. Previous graph reduction techniques [22, 40] require $O(|e|)$ nodes and $O(|e|^2)$ directed edges to exactly model this function. Our result shows that $O(|e|\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ edges suffice to approximately model it.[1] This savings is particularly significant given how extensively this function appears in practice. This is a commonly used *region potential* function in image segmentation, and shows up frequently in work on DSFM [9, 19, 38, 28]. It also appears often in hypergraph clustering applications, where it results from applying a (weighted or unweighted) *clique expansion* to a hypergraph [2, 1, 7, 16, 43], which replaces each hyperedge $e$ with a clique on nodes in $e$. This results in a graph whose cut function is exactly a sum of submodular functions of the form $f_e(S) = c_e|S||e\backslash S|$. A similar decomposable submodular function also arises as the cut function of a graph obtained from co-occurrence relationships, or a graph obtained from a projection of a bipartite graph [8]. Our results provide new efficient sparsifiers for all of these types of graphs.

As a side note, this theorem has interesting implications for sparsifying a complete graph, a problem that has been extensively considered in the theoretical computer science literature. Previous work has shown how to model the cut properties of a complete graph on $n$ nodes with $O(n/\varepsilon^2)$ edges. This is tight for spectral sparsifiers [6], as well as for degree-regular cut sparsifiers with uniform edge weights [3]. Our result implies that if we are willing to include a small number of additional nodes and use directed edges, the cut properties of the complete graph can be modeled using only $O(n)$ nodes and $O(n\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ edges.

## 4 Theoretical Runtime Analysis and Comparison

The ability to approximately model a single concave cardinality function makes it possible to quickly obtain an arbitrarily good approximate solution to an instance of Card-DSFM by reducing it to a minimum $s$-$t$ cut problem on a sparse graph. Define a function $f$ on a ground set $V$ by

$$f(S) = \sum_{e \in E} f_e(S \cap e) = \sum_{e \in E} g_e(|S \cap e|), \tag{6}$$

where each $g_e$ is concave. We assume each $f_e$ is nonnegative; if not we can adjust the objective by a constant without affecting optimal solutions. Algorithm 2 gives pseudocode for our new method SPARSECARD for minimizing (6). The method finds a sparse approximate graph reduction for each concave function $g_e$ using Algorithm 1, combines these into a larger graph whose cut properties

[1] Recall that if we need $t$ linear pieces to approximate a concave function $g$, this means we need $t$ CB-gadgets (see (3)) to model $f_e(S) = g(|S|)$, each of which has $O(|e|)$ edges, for a total of $O(t|e|)$ edges.

**Algorithm 2** SPARSECARD$(f, \varepsilon)$

**Input**: $\varepsilon \geq 0$, function $f(S) = \sum_{e \in E} f_e(S \cap e) = \sum_{e \in E} g_e(|S \cap e|)$ on ground set $V$
**Output**: Set $S' \subseteq V$ satisfying $f(S') \leq (1+\varepsilon) \min_{S \subseteq V} f(S)$.
$\mathcal{A} \leftarrow \emptyset, \mathcal{E} \leftarrow \emptyset$ `//initialize auxiliary node and edge set for reduced graph`
**for** $e \in E$ **do**
  `//Step 1: solve piecewise linear function approximation problem`
  $\ell_e \leftarrow$ GREEDYPLCOVER$(g_e, \varepsilon)$
  `//Step 2: Construct graph gadget for e with auxiliary nodes` $\mathcal{A}_e$
  $G_e = (e \cup \mathcal{A}_e, \mathcal{E}_e) \leftarrow$ CGFTOGADGET$(\ell_e)$
  `//Step 3: Add new auxiliary nodes and edges for building graph G`
  $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}_e, \mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}_e$
**end for**
$G = (V \cup \mathcal{A} \cup \{s, t\}, \mathcal{E})$ `//Build graph G modeling f`
$T =$ MINSTCUT$(G)$ `//Find minimum s-t cut`
Return $S' = T \cap V$ `//Ignore auxiliary nodes`

approximate $f$, and then applies a minimum $s$-$t$ cut solver to that graph. Finding sparse reductions is fast, so the asymptotic runtime guarantee is just the time it takes to solve the cut problem, which can be accomplished using any algorithm for solving the dual maximum $s$-$t$ flow problem [15, 14, 25, 39].

**Theorem 6** *Let $n = |V|$ and $R = |E|$. When $\varepsilon = 0$, the graph constructed by* SPARSECARD *will have $O(\sum_{e \in E} |e|)$ nodes and $O(\sum_{e \in E} |e|^2)$ edges. When $\varepsilon > 0$, the graph will have $O(n + \sum_{e \in E} \varepsilon^{-1} \log |e|)$ nodes and $O(\varepsilon^{-1} \sum_{e \in E} |e| \log |e|)$ edges. In either case, and the method will return a set $T$ satisfying $f(T) \leq (1+\varepsilon) \min_{S \subseteq V} f(S)$.*

The graph returned by SPARSECARD can also be asymptotically sparser for specialized concave functions, such as the popular clique expansion spliting function in Theorem 5. In the remainder of the section, we provide a careful runtime comparison between SPARSECARD and competing runtimes for Card-DSFM. We focus on each runtime's dependence on the ground set size $n$, number of functions $R$, and support sizes $|e|$, and use $\tilde{O}$ notation to hide logarithmic factors of $n$, $R$, and $\frac{1}{\varepsilon}$. To easily compare weakly polynomial runtimes, we assume that each $f_e$ has integer outputs, and assume that $\log(\max_S f(S))$ is small enough that it can also be absorbed by $\tilde{O}$ notation. Our primary goal is to highlight the runtime improvements that are possible when an approximate solution suffices. Among algorithms for DSFM, SPARSECARD is unique in its ability to quickly find solutions with a priori multiplicative approximation guarantees. Previous approaches for DSFM focus on either obtaining exact solutions, or finding a solution to within an *additive* approximation error $\epsilon > 0$ [4, 9, 27, 19]. In the latter case, setting $\epsilon$ small enough will guarantee an optimal solution in the case of integer output functions. However, these results provide no a priori multiplicative approximation guarantee, which is the traditional focus of approximation algorithms. Furthermore, using a larger value of $\epsilon$ for these additive approximations only improves runtimes in logarithmic terms. For our runtime comparison we will therefore assume that these methods are run until optimality. In constrast, setting $\varepsilon > 0$ will often lead to substantial runtime decreases for SPARSECARD.

**Competing runtime guarantees.** Table 1 lists runtimes for existing methods for DSFM. We also list asymptotic runtime for SPARSECARD when applying the recent maximum flow algorithm of van den Brand et al. [39]. While this leads to the best theoretical guarantees for our method, asymptotic runtime improvements over competing methods can also be shown using alternative fast algorithms for maximum flow [14, 25, 15]. For the submodular flow algorithm of Kolmogorov [24], we have reported the runtime guarantee provided specifically for Card-DSFM. While other approaches have frequently been applied to Card-DSFM [9, 38, 27], runtimes guarantees for this special case have not been presented explicitly and are more challenging to exactly pinpoint. Runtimes for these algorithms depend on certain oracles for solving smaller minimization problems at functions $f_e$ in an inner loop. For $e \in E$, let $\mathcal{O}_e$ be a *quadratic minimization oracle*, which for an arbitrary vector $w$ solves $\min_{y \in B(f_e)} \|y + w\|$ where $B(f_e)$ is the base polytope of the submodular function $f_e$ (see [9, 5, 19] for details). Let $\theta_e$ be the time it take to evaluate the oracle at $e \in E$, and define $\theta_{max} = \max_{e \in E} \theta_e$ and $\theta_{avg} = \frac{1}{R} \sum_{e \in E} \theta_e$. Although these oracles admit faster implementations in the case of concave cardinality functions, it is not immediately clear from previous work what is the best possible runtime.

Table 1: Runtimes for Card-DFSM for various methods,where $R = |E|$, $n = |V|$, $\mu = \sum_e |e|$, and $\mu_2 = \sum_e |e|^2$. Oracle runtimes satisfy $\theta_{max} = \Omega(\max |e|)$, and $\theta_{avg} = \Omega(\frac{1}{R}\sum_{e\in E}|e|)$. $T_{mf}(N, M)$ is the time to solve a max-flow problem with $N$ nodes and $M$ edges.

| Method | Discrete/Cont | Runtime |
|---|---|---|
| Kolmogorov SF [24] | Discrete | $\tilde{O}(\mu^2)$ |
| IBFS Strong [11, 9] | Discrete | $O(n^2\theta_{max}\mu_2)$ |
| IBFS Weak [11, 9] | Discrete | $\tilde{O}(n^2\theta_{max} + n\sum_e |e|^4)$ |
| AP [34, 9, 28] | Continuous | $\tilde{O}(nR\theta_{avg}\mu)$ |
| RCDM [10, 9] | Continuous | $\tilde{O}(n^2R\theta_{avg})$ |
| ACDM [10, 9] | Continuous | $\tilde{O}(nR\theta_{avg})$ |
| Axiotis et al. [4] | Discrete | $\tilde{O}(\max_e |e|^2 \cdot \left(\sum_e |e|^2\theta_e + T_{mf}\left(n, n+\mu_2\right)\right)$ |
| SparseCard $\varepsilon = 0$ | Discrete | $\tilde{O}(T_{mf}(\mu, \mu_2)) = \tilde{O}\left(\mu_2 + \mu^{3/2}\right)$ |
| SparseCard $\varepsilon > 0$ | Discrete | $\tilde{O}\left(T_{mf}(n + \frac{R}{\varepsilon}, \frac{1}{\varepsilon}\mu)\right) = \tilde{O}\left(\frac{\mu}{\varepsilon} + (n + \frac{R}{\varepsilon})^{3/2}\right)$ |

When $w = 0$, solving $\min_{y\in B(f_e)} \|y + w\|$ takes $O(|e| \log |e|)$ time [19], so this serves as a best case runtime we can expect for the more general oracle $\mathcal{O}_e$ based on previous results. We note also that in the case of a region potential function of the form $f_e(S) = |S||e\backslash S|$, Ene et al. [9] highlight that an $O(|e| \log |e| + |e|\tau_e)$ algorithm can be used, where $\tau_e$ denote the time it take to evaluate $f_e(S \cap e)$ for any $S \subseteq e$. In our runtime comparisons will use the bound $\theta_e = \Omega(|e|)$, as it is reasonable to expect that any meaningful submodular functions we consider should take a least linear time to minimize.

**Fast approximate solutions ($\varepsilon > 0$).** Barring extreme cases where support sizes $|e|$ are all very small, the accelerated coordinate descent method (ACDM) of Ene et al. [10] provides the fastest previous runtime guarantee. For a simple parameterized runtime analysis, consider a DSFM problem where the average support size is $(1/R)\sum_e |e| = \Theta(n^\alpha)$ for $\alpha \in [0, 1]$, and $R = \Theta(n^\beta)$, where $\beta \geq 1 - \alpha$ must hold if we assume each $v \in V$ is in the support for at least one function $f_e$. An exact runtime comparison between SparseCard and ACDM depends on the best runtime for the oracle $\mathcal{O}_e$ for concave cardinality functions. If an $O(|e| \log |e|)$ oracle is possible, the overall runtime guarantee for ACDM would be $\tilde{O}(n^{1+\alpha+\beta})$. Meanwhile, for a small constant $\varepsilon > 0$, SparseCard provides a $(1 + \varepsilon)$-approximate solution in time $\tilde{O}(n^{\alpha+\beta} + \max\{n^{3/2}, n^{3\beta/2}\})$, which will faster by at least a factor $\tilde{O}(\sqrt{n})$ whenever $\beta \leq 1$. When $\beta \geq 1$, finding an approximation with SparseCard is guaranteed to be faster whenever $R = o(n^{2+2\alpha})$. If the best case oracle $\mathcal{O}_e$ for concave cardinality functions is $\omega(|e| \log |e|)$, the runtime improvement of our method is even more significant.

**Guarantees for exact solutions ($\varepsilon = 0$).** As an added bonus, running SparseCard with $\varepsilon = 0$ leads to the fastest runtime for finding *exact* solutions in many regimes. Using similar reasoning as above, we can show that finding exact solutions with SparseCard will be faster than ACDM when the average support size is $\Theta(n^\alpha)$ and $R = o(n^{2-\alpha})$. Compared with other discrete methods, SparseCard has a faster runtime guarantee than the submodular flow algorithm of Kolmogorov [24] in all parameter regimes, and has a better runtime guarantee than the strongly-polynomial IBFS algorithm [11] when $\sum_e |e| = o(n^4)$, which includes all cases where $R = o(n^3)$. Both variants of IBFS as well as the recent method of Axiotis et al. [4] become impractical if even a *single* function $f_e$ has support size $|e| = O(n)$ (even when the average support size is much smaller). In this case, runtimes for these methods are $\Omega(n^5)$. Even using the very loose bounds $\sum_e |e|^2 \leq Rn^2$ and $\sum_e |e| \leq nR$, our method is guaranteed to find exact solutions faster as long as $R = o(n^{7/3})$, with significant additional improvements when approximate solutions suffice.

We note finally that the asymptotic guarantees for SparseCard when $\varepsilon = 0$ could also be obtained by combining existing exact reduction techniques [22, 38, 40] with fast maximum flow algorithms. However, our method has the practical advantage of finding the *sparsest* exact reduction in terms of CB-gadgets. This results in a reduced graph with roughly half the number of edges used by the reduction of Veldt et al. [40]. Analogously, Stobbe and Krause showed that a concave cardinality function can be decomposed as a sum of modular functions plus a combination of $|e| - 1$ threshold potentials (Appendix C in [38]). Our best approximation technique finds a linear combination with $\lfloor |e|/2 \rfloor$ threshold potentials. This amounts to the observation that any $k + 1$ points $\{i, g(i)\}$ can be joined by $\lfloor k/2 \rfloor + 1$ lines instead of using $k$. Overall though, the most significant advantage of

SPARSECARD over existing reduction methods is its ability to find fast approximate solutions with sparse approximate reductions.

# 5 Experiments

In addition to its improved theoretical runtime, SPARSECARD is easy to implement and leads to substantial improvements in benchmark image segmentation problems and hypergraph clustering tasks. We focus on solving DSFM problems that simultaneously include component functions of large and small support, which is common in many computer vision and hypergraph clustering applications [37, 9, 40, 31, 36]. We primarily compare against state of the art continuous optimization techniques for DSFM. Although these do not provide a priori approximation guarantees, truncating the number of iterations for these methods will in practice provide approximate solutions that we can use to compare against approximate solutions returned by SPARSECARD. Note that we do not compare against other discrete methods for DSFM [11, 4, 24] as these are designed for finding exact solutions for problems where functions have small support, unlike the experiments we consider here. Furthermore, to our knowledge, no implementations for the methods of Kolmogorov [24] or Axiotis et al. [4] exist, as these were primarily presented as theoretical contributions. For comparisons between IBFS [9] and continuous optimization methods, we refer to the work of Ene et al. [9], who confirmed that continuous approaches are more successful in the case of functions with large support. Experiments were performed on a laptop with 8GB of RAM and 2.2 GHz Intel Core i7 processor. Anonymized code for our methods and experiments, along with further details regarding the experimental setup, are provided in the supplement.

**Benchmark Image Segmentation Tasks.** Our sparse approximation techniques provide noticeably faster approximate solutions to standard image segmentation tasks previously used as benchmarks for DSFM [19, 28, 9]. We specifically consider the *smallplant* and *octopus* segmentation tasks [2] from Jegelka et al. [20, 19]. Each segmentation task amounts to minimizing a decomposable submodular function on a ground set of size $N = |V| = 427 \cdot 640 = 273280$, where each object in $V$ is a pixel from a $427 \times 640$ pixel image. There are three types of submodular functions in the sum, corresponding to potential functions for the segmentation problem. The first type are unary potentials for each pixel/node, i.e., functions of size 1 representing each node's bias to be in the output set. The second type are pairwise potentials from a 4-neighbor grid graph; pixels $i$ and $j$ share an edge if they are directly adjacent vertically or horizontally. The third type are region potentials of the form $f_e(A) = |e \backslash A||A|$ for $A \subseteq e$, where $e$ represents a superpixel region. We use edge weights, unary potentials, and the set of 500 superpixels, that come with the dataset. The problem can be solved via maximum flow even without sophisticated reduction techniques for cardinality functions. Unary potentials correspond to edges adjacent to source and sink nodes, pairwise potential are interior edges, and the region potentials can be collapsed into pairwise edges by adding an appropriately weighted clique of edges between all nodes in a region $e$. Using such a reduction, we compute the optimal solution, which allows us to compute exact approximation ratios achieved by other methods.

Compared with the exact reduction method, running SPARSECARD with $\varepsilon > 0$ leads to much sparser graphs, much faster runtimes, and a posteriori approximation factors that are significantly better than $(1 + \varepsilon)$. For example, on the *smallplant* dataset, when $\varepsilon = 1.0$, the returned solution is within a factor $1.004$ of optimality, even though the a priori guarantee is a 2-approximation. The sparse reduced graph has only $0.013$ times the number of edges in the exact reduced graph, and solving the sparse flow problem is over two orders of magnitude faster than solving the problem on the exact reduced graph (which takes roughly 20 minutes on a laptop). In Table 2 we list the sparsity, runtime, and a posteriori guarantee obtained for a range of other $\varepsilon$ values.

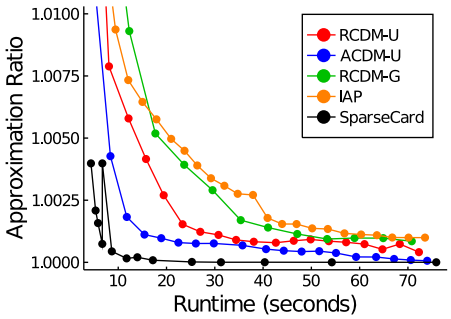
We also compare against recent C++ implementations[3] of ACDM, RCDM, and AP provided by Li and Milenkovic [28]. These use the divide-and-conquer method of Jegelka et al. [19] to solve the quadratic minimization oracle $\mathcal{O}_e$ for region potential functions. We specifically use the improved Incidence Relation AP method (IAP) provided by these authors. Further details about these algorithms and parameter settings are shown in the supplement. As a general rule, in order to ensure a fair comparison we use parameter settings leading to the best possible runtimes. In order to compare against these methods we compute a posteriori approximations obtained using intermediate solutions

[2] These dataset are available at `http://people.csail.mit.edu/stefje/code.html`.

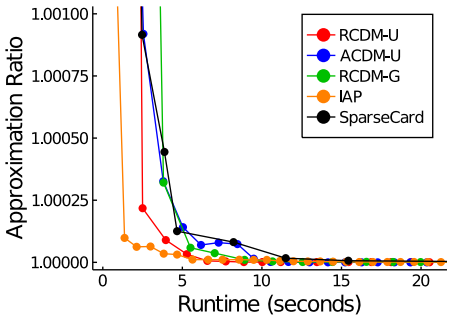
[3] Code (MIT license): `https://github.com/lipan00123/DSFM-with-incidence-relations` .

Table 2: Results from SPARSECARD for a range of $\varepsilon$ values on the *smallplant* image segmentation problem. The second row gives the approximation factor relative to the optimal solution. Sparsity is given as the ratio between the number of edges in the approximate reduced graph and the dense graph resulting from the exact reduction. Finding the exact solution using the same maxflow routine on the dense exact reduced graph takes roughly 20 minutes.

| $\varepsilon$ | 1.0 | 0.2683 | 0.072 | 0.0193 | 0.0052 | 0.0014 | 0.0004 | 0.0001 |
|---|---|---|---|---|---|---|---|---|
| Approx. | 1.004 | 1.002 | 1.0007 | 1.0002 | 1.00009 | 1.000004 | 1.000002 | 1.000001 |
| Sparsity | 0.01 | 0.02 | 0.02 | 0.03 | 0.05 | 0.08 | 0.14 | 0.24 |
| Runtime | 6.7 | 5.3 | 6.7 | 11.7 | 17.2 | 31.3 | 54.2 | 96.3 |




(a) *smallplant*




(b) *octopus*

Figure 1: Approximation factor (relative to the optimal solution) vs. runtime for approximate solutions returned by SPARSECARD and competing continuous optimization methods for DSFM on the *smallplant* dataset. Approximate solutions for continuous methods are obtained by outputting intermediate solutions.

returned by the method after every few iterations. In more detail, we extract the best level set $S_\lambda = \{i \colon x_i > \lambda\}$ from the vector of dual variables $\mathbf{x} = (x_i)$ at various steps, and compute the approximation ratio $\operatorname{argmin}_\lambda f(S_\lambda)/f(S^*)$ where $S^*$ is the optimal solution determined via the exact max-flow solution. In Figure 1a we plot approximation ratios achieved by these intermediate solutions on the *small-plant* dataset, along with runtimes and approximations achieved by running SPARSECARD for 15 logarithmically spaced values of $\varepsilon$ between $10^{-4}$ and 1. Our method obtains much better approximations in a shorter runtime. We ran similar experiments on the *octopus* dataset provided by Jegelka et al. [19], and observed similar improvements when using SPARSECARD with $\varepsilon > 0$ vs. using an exact and dense graph reduction. However, this instance appears significantly easier than the *smallplant* dataset, and we find that the difference between SPARSECARD and competing continuous methods is negligible, with all methods finding a nearly optimal solution within a few seconds.

**Conductance Improvement in Clique Expanded Hypergraphs.**

## References


[1] Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher order learning with graphs. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 17–24, New York, NY, USA, 2006. ACM.

[2] Sameer Agarwal, Jongwoo Lim, Lihi Zelnik-Manor, Pietro Perona, David Kriegman, and Serge Belongie. Beyond pairwise clustering. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '05, pages 838–845, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Noga Alon. On the edge-expansion of graphs. *Comb. Probab. Comput.*, 6(2):145–152, June 1997.

[4] Kyriakos Axiotis, Adam Karczmarz, Anish Mukherjee, Piotr Sankowski, and Adrian Vladu. Decomposable submodular function minimization via maximum flow. *arXiv:2103.03868*, 2021.

[5] Francis Bach. *Learning with Submodular Functions: A Convex Optimization Perspective*. Now Publishers Inc., Hanover, MA, USA, 2013.

[6] Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Review*, 56(2):315–334, 2014.

[7] Austin R. Benson, David F. Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

[8] Austin R. Benson, Paul Liu, and Hao Yin. A simple bipartite graph projection model for clustering in networks. *arXiv preprint: https://arxiv.org/abs/2007.00761*, 2020.

[9] Alina Ene, Huy Nguyen, and László A Végh. Decomposable submodular function minimization: discrete and continuous. In *Advances in Neural Information Processing Systems*, NeurIPS '17, pages 2870–2880, 2017.

[10] Alina Ene and Huy L. Nguyen. Random coordinate descent methods for minimizing decomposable submodular functions. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 787–795. JMLR.org, 2015.

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

[12] Kimon Fountoulakis, Pan Li, and Shenghao Yang. Local hyper-flow diffusion. *arXiv/2102.07945*, 2021.

[13] D. Freedman and P. Drineas. Energy minimization via graph cuts: settling what is possible. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2 of *CVPR '05*, pages 939–946 vol. 2, 2005.

[14] Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. *arXiv:2101.07233*, 2021.

[15] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, September 1998.

[16] Scott W. Hadley. Approximation techniques for hypergraph partitioning problems. *Discrete Applied Mathematics*, 59(2):115 – 127, 1995.

[17] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, July 2001.

[18] Satoru Iwata and James B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 1230–1237, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[19] Stefanie Jegelka, Francis Bach, and Suvrit Sra. Reflection methods for user-friendly submodular optimization. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NeurIPS '13, pages 1313–1321, 2013.

[20] Stefanie Jegelka and Jeff Bilmes. Submodularity beyond submodular energies: Coupling edges in graph cuts. In *CVPR 2011*, pages 1897–1904, 2011.

[21] Stefanie Jegelka, Hui Lin, and Jeff A Bilmes. On fast approximate submodular minimization. In *Advances in Neural Information Processing Systems*, NeurIPS '11, pages 460–468, 2011.

[22] Pushmeet Kohli, Philip HS Torr, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 82(3):302–324, 2009.

[23] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, Feb 2004.

[24] Vladimir Kolmogorov. Minimizing a sum of submodular functions. *Discrete Appl. Math.*, 160(15):2246–2258, October 2012.

[25] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

[26] Pan Li and Olgica Milenkovic. Inhomogeneous hypergraph clustering with applications. In *Advances in Neural Information Processing Systems 30*, NeurIPS '17, pages 2308–2318, 2017.

[27] Pan Li and Olgica Milenkovic. Revisiting decomposable submodular function minimization with incidence relations. In *Advances in Neural Information Processing Systems 31*, NeurIPS '18, pages 2237–2247, 2018.

[28] Pan Li and Olgica Milenkovic. Revisiting decomposable submodular function minimization with incidence relations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NeurIPS'18, page 2242–2252, 2018.

[29] Pan Li and Olgica Milenkovic. Submodular hypergraphs: p-laplacians, Cheeger inequalities and spectral clustering. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *ICML '18*, pages 3014–3023. PMLR, 2018.

[30] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 510–520, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[31] Meng Liu, Nate Veldt, Haoyu Song, Pan Li, and David F. Gleich. Strongly local hypergraph diffusions for clustering and semi-supervised learning. In *TheWebConf 2021*, volume cs.SI, 2021.

[32] Thomas L. Magnanti and Dan Stratila. Separable concave optimization approximately equals piecewise linear optimization. In *IPCO 2004*, pages 234–243, 2004.

[33] Thomas L Magnanti and Dan Stratila. Separable concave optimization approximately equals piecewise-linear optimization. *arXiv preprint arXiv:1201.3148*, 2012.

[34] Robert Nishihara, Stefanie Jegelka, and Michael I. Jordan. On the convergence rate of decomposable submodular function minimization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NeurIPS '14, pages 640–648, 2014.

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

[36] Pulak Purkait, Tat-Jun Chin, Alireza Sadri, and David Suter. Clustering with hypergraphs: the case for large hyperedges. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1697–1711, 2016.

[37] I. Shanu, C. Arora, and P. Singla. Min norm point algorithm for higher order mrf-map inference. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, pages 5365–5374, 2016.

[38] Peter Stobbe and Andreas Krause. Efficient minimization of decomposable submodular functions. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems*, NeurIPS '10, pages 2208–2216, 2010.

[39] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and $\ell_1$-regression in nearly linear time for dense instances. *arxiv:2101.05719*, 2021.

[40] Nate Veldt, Austin R. Benson, and Jon Kleinberg. Hypergraph cuts with general splitting functions. *arXiv preprint: 2001.02817*, 2020.

[41] Nate Veldt, Austin R. Benson, and Jon Kleinberg. Minimizing localized ratio cut objectives in hypergraphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (to appear)*, KDD '20, 2020.

[42] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning*, pages 1954–1963. PMLR, 2015.

[43] J. Y. Zien, M. D. F. Schlag, and P. K. Chan. Multilevel spectral hypergraph partitioning with arbitrary vertex sizes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1389–1399, Sep. 1999.

# Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Yes, we begin a discussion of limitations in the conclusions section and expound in the supplementary material.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Yes, we include a section on possible negative societal impacts in the supplement.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Yes, all assumptions are included in theorem statements.
   (b) Did you include complete proofs of all theoretical results? [Yes] Yes, detailed proofs are included in the supplement, with intuition and proof sketches in the main text.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] All code and data is included as a part of the supplement.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section **??**. Additional details for parameter settings are included in the supplement as well.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See section **??**. For randomized methods we compare against, error regions are shown. Our method is deterministic and outputs the same solutions when the same parameters are used, so no error bars are needed.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section **??** for details.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See section **??** and in particular footnotes and references.
   (b) Did you mention the license of the assets? [Yes] Yes, whenever one was available. In other cases, no license was given by creators of the asset, but we have carefully followed creator guidelines for how to cite the material.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The implementations for code and experimental results are provided.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? **[TODO]**
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? **[TODO]**
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]