Max–Planck–Institut für biologische Kybernetik
Max Planck Institute for Biological Cybernetics

Technical Report No. TR-189

Cooperative Cuts: Graph Cuts with Submodular Edge Weights

Stefanie Jegelka,[1] Jeff Bilmes,[2]

March 2010

[1] Department Schölkopf, email: stefanie.jegelka@tuebingen.mpg.de, [2] University of Washington, email: bilmes@u.washington.edu

Cooperative Cuts: Graph Cuts with Submodular Edge Weights

Stefanie Jegelka, Jeff Bilmes

Abstract.
We introduce a problem we call *Cooperative cut*, where the goal is to find a minimum-cost graph cut but where a submodular function is used to define the cost of a subsets of edges. That means, the cost of an edge that is added to the current cut set C depends on the edges in C. This generalization of the cost in the standard min-cut problem to a submodular cost function immediately makes the problem harder. Not only do we prove NP hardness even for nonnegative submodular costs, but also show a lower bound of $\Omega(|V|^{1/3})$ on the approximation factor for the problem. On the positive side, we propose and compare four approximation algorithms with an overall approximation factor of $\min\big\{|V|/2, |C^*|, O(\sqrt{|E|}\log|V|), |P_{\max}|\big\}$, where C^* is the optimal solution, and $P_{\max}$ is the longest s,t path across the cut between given s,t. We also introduce additional heuristics for the problem which have attractive properties from the perspective of practical applications and implementations in that existing fast min-cut libraries may be used as subroutines. Both our approximation algorithms, and our heuristics, appear to do well in practice.

1 Introduction

The standard minimum cut (min-cut) problem asks to find a minimum-cost *cut* in a graph $G = (V, E)$. This is defined as a set $C \subseteq E$ of edges whose removal cuts the graph into two separate components with nodes $X \subseteq V$ and $V \setminus X$. A cut is *minimal* if no subset of it is still a cut; equivalently, it is the *edge boundary*

$$\delta X = \{(v_i, v_j) \in E \mid v_i \in X, v_j \in V \setminus X\} \subseteq E.$$

of $X \subseteq V$ and partitions the graph into two connected components. The cost $f(C)$ of the cut C is traditionally measured as the number of edges in C, or as a sum of edge weights $w(e)$: $f_{\text{trad}}(C) = \sum_{e\in C} w(e)$. We extend the class of cost functions from such modular (or sometimes called linear) functions to the broad class of *submodular* set functions. Submodular functions allow a form of "cooperation" to exist between subsets of edges — this cooperation may reduce the cost of the entire subset compared to the (modular) summed cost of the single edges.

A function $f : 2^E \to \mathbb{R}$ on subsets of edges is *submodular* if for all sets A, $B \subseteq E$, it holds that

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B).$$

A *modular* function, such as f_{trad}, satisfies this equation with equality. An alternative definition of submodularity refers to *diminishing marginal returns*: adding an element e to $A \subset E$ increases the cost no more than adding e to a smaller subset $B \subseteq A$, i.e.,

$$f(A \cup \{e\}) - f(A) \leq f(B \cup \{e\}) - f(B).$$

A set function is *monotone* if $f(B) \leq f(A)$ for all $B \subseteq A \subseteq E$. It is *normalized* if $f(\emptyset) = 0$. Here, we always assume the cost function to be normalized. Unlike arbitrary functions, submodular functions are attractive in that they are quite general and widely applicable, yet there are often efficient algorithms for their either exact or approximate optimization. For more details about submodular functions and their optimization, the reader may refer to the surveys by Lovász [33], Fujishige [13], Narayanan [34].

Bestowed with these definitions, we can formally state the problems we address.

Problem 1 (Cooperative cut (CoopCut)). *Find a partition $(X, V \setminus X)$ of the nodes that minimizes the cost $f(\delta X)$ measured by a submodular function $f : 2^E \to \mathbb{R}$ defined on subsets of E.*

The (s, t) cut version of CoopCut seeks for a min-cost cut that separates two distinct nodes s and t.

Problem 2 (Cooperative (s,t) cut (Coop-(s,t) Cut))**.** *Given $s, t \in V$, find a partition $(X, V \setminus X)$ of the nodes with $s \in X$, $t \notin X$ that minimizes the cost $f(\delta X)$ measured by a submodular function $f : 2^E \to \mathbb{R}$.*

Submodular functions occur in various contexts, and submodularity has been an important concept in combinatorics, economics, operations research, and game theory. Recently, it has also enjoyed increased attention in Machine Learning [29, 30, 31]. One prominent example of a submodular function is entropy. Furthermore, "discounted price functions"[1] of the form $f(A) = h\big(\sum_{e \in A} w(e)\big)$ for a concave, increasing function h and non-negative weights $w(e)$ [33], occur in economics and illustrate diminishing returns. Other examples include *matroid rank functions* and label cost functions, where we count the number of different labels in a set of labeled elements; this sum can also be weighted. In general, one may also think of submodular costs as "cooperative" costs: the cost of a set of elements can be significantly smaller than the sum of the single-element costs. The additional cost contributed by a newly chosen element e strongly depends on the elements that have been chosen so far and may be far less than the cost $f(e)$ of the single element.[2] We will refer to this cost reduction as *cooperation* between e and the relevant edges in the chosen set. As an example, cooperation for entropy corresponds to statistical dependence. For the label cost, two elements cooperate if they have the same label, because that label is only counted once. Often, cooperation can be related to similarity, but it can also be global and general as in the case of discounted price functions. In any case, the interaction between elements, rather than being arbitrary, is limited to that which is expressible by a submodular function f.

We wish to stress that the cost function is a submodular function on subsets of *edges*[3]. In contrast, the standard (edge-modular cost) graph cut problem can be viewed as the minimization of a submodular function defined on subsets of *nodes*. While modular edge costs lead to submodular node costs, submodular edge costs do not necessarily lead to submodular node costs.

CoopCut also differs from submodular flows [13], (solvable in polynomial time), with a polynomial number of calls to a submodular function minimization (SFM) oracle), where submodularity defines feasible flows but the cost of an edge set is still modular.

CoopCut can be treated in either of two ways: 1) as a constrained minimization of a submodular function on *edges*, where the constraint is that the solution must be a cut; or 2) as the unconstrained problem of finding a "good" set of *nodes* $\emptyset \neq X \subset V$. For this latter case, the cost function is $g(X) \triangleq f(\delta X)$. As mentioned above, however, this node-based function is in general not submodular. More interestingly, Section 3.1 illustrates that the unconstrained problem with cost function g does not provide enough information about the structure of the problem to yield any form of approximation guarantees without an exponential number of queries. Therefore, we consider the first edge-based approach to be the fruitful path to approximation algorithms and heuristics. In contrast, the standard minimum cut problem can be solved exactly by minimizing the corresponding cost function on subsets of nodes [40]: modular costs apparently retain enough structural information when seen through the lens $g(X) \triangleq f(\delta X)$ to avoid loosing polynomiality, while submodular edge costs do not.

1.1 Contributions

The main contributions of this work are hardness results for CoopCut and a theoretical and empirical comparison of approximation algorithms. We prove the NP hardness of CoopCut by a reduction from the graph bisection problem, and also prove a lower bound of $|V|^{1/3-\epsilon}/(1+\delta)$ on its approximation factor for monotone cost functions, for any constant $\epsilon, \delta > 0$; for non-monotone cost functions, we show an inapproximability result (Section 2). On the positive side, we propose four approximation algorithms (which we refer to as PMF, MBI, EA, CR) with different upper bounds, as well as a heuristic greedy algorithm and a general heuristic improvement step (Section 3).

Table 1 summarizes the theoretical results of this paper. The approximation algorithms are based on two basic strategies: first, an approximation of the cost function that is easier to optimize, and second, a relaxation with a convex cost function. The first strategy includes three approaches: a local restriction of the submodularity that yields a tractable dual problem (PMF), a modular approximation with an extended search space (MBI), and finally, a geometric approximation of the cost function (EA). The three function approximations together with the convex

[1]The definition of "discounted price functions" is as used in [17].

[2]Subadditive functions also have this property, but, in contrast to submodular functions, general subadditive functions cannot be minimized in polynomial time even without constraints. For the example in Section 3.1, no approximation factor can be guaranteed in general, and neither for a minimization with cut constraints. Monotone submodular functions allow for linear approximation guarantees (in n), and non-monotone submodular functions too if we remove the minimality constraint.

[3]For this reason, we referred to CoopCut as "edge-submodular cut" in a contribution to the NIPS 2009 workshop on Discrete Optimization in Machine Learning.

Table 1: Summary of our results for monotone submodular costs (all normalized and non-negative). In this paper, $C^* \subseteq E$ is the optimal cut, $\Delta_s(C^*)$ the nodes on the s side of C^* that are incident to a cut edge, and $\Delta_t(C^*)$ the correspondent on the t side. The longest path (by number of edges) between s and t is $P_{\max}$. For the min-cut, the s and t sides of the cut are assigned arbitrarily. The better upper bound for Algorithm III holds only for matroid rank functions or, with a constant factor, for bounded integer-valued polymatroid rank functions.

Problem	Lower Bound	Upper Bound			
		Alg. PMF	Alg. MBI	Alg. EA	Alg. CR
(s,t) cut monot.	$\Omega(n^{1/3})$ (rank fcts.)	$\min\{\Delta_s(C^*), \Delta_t(C^*)\} \leq n/2$	$\lvert C^*\rvert$	$\sqrt{m+1}$ or $O(\sqrt{m}\log n)$	$\lvert P_{\max}\rvert \leq n-1$
min-cut monot.		$\min\{\Delta_s(C^*), \Delta_t(C^*)\} \leq n/2$	$\lvert C^*\rvert$	$\sqrt{m+1}$ or $O(\sqrt{m}\log n)$	$\min_{s,t \text{ separ. by } C^*} \lvert P_{\max}\rvert \leq n-1$

relaxation (CR) make up the four approximation algorithms. The last, heuristic algorithm is a greedy method. This algorithm as well as a generic improvement step only considers submodular interactions with a reference set.

In a range of experiments (Section 4), we show that some of the algorithms already perform well on many instances. To push specific algorithms to their theoretical limits, we present some worst-case instances of CoopCut and empirical results for them.

1.2 Notation and minimality

We denote by $G = (V, E)$ an undirected graph with $n = |V|$ nodes and $m = |E|$ edges. The cost function $f : 2^E \to \mathbb{R}$ is monotone submodular and normalized, unless stated otherwise. Sets of edges are denoted by capital letters $A, B, C, Q, R \subseteq E$; sets of nodes by $X, Y \subseteq V$. The letters s, t, x, y stand for nodes, the letter e for an edge. Edge-wise weights are denoted by $w(e)$, and the respective (modular) cost of a set A of edges is $w(A) = \sum_{e\in A} w(e)$. The set $\mathcal{C}$ denotes the set of all minimal cuts in a given graph: in short, a CoopCut finds an element in $\operatorname{argmin}_{C\in\mathcal{C}} f(C)$.

The indicator function $\mathbf{1}[\cdot]$ is one if the argument is true, and zero otherwise.

1.2.1 Minimality

Problems 1 and 2 ask for a cut that is the boundary of a set of nodes and partitions the graph into two connected components. For a normalized, monotone cost function (which is always nonnegative), there is always a min-cut C^* that is such a *minimal* cut, since the cost of any cut $B \subset A$ that is a subset of C^* cannot be higher than $f(C^*)$. Thus, there is always a min-cost cut that is an edge boundary δX for some connected $X \subset V$. For a non-monotone cost function, this minimality does not necessarily hold, since non-monotonicity allows that $f(B) < f(\delta X)$ for a superset $B \supset \delta X$ of a cut. If the solution only has to partition the graph, but may be a superset of a minimal cut, then the non-monotone problem can be approximated to within a factor of $n-1$ by our Algorithm CR.

The same issue arises if negative edge weights are allowed in the standard min-cut problem and make f_{trad} non-monotone. For the resulting cost, the solution has commonly been constrained to be a (minimal) multi-cut, that is, it must lie in the cut polytope [11], as for example asked for in [6, 10, 21, 37]. As opposed to the standard min-cut, the (multi-cut) version with arbitrary edge weights is NP hard [21]. We can adapt our problems to the multicut constraint as well, but we do not provide further details in this paper.

Our algorithms and experiments mostly focus on monotone submodular cost functions, such as rank functions, entropy and nondecreasing discounted price functions.

1.3 Motivation and applications

Our initial motivation for CoopCut comes from the problem of finding good separators in probabilistic graphical models, for which there are several applications (e.g. [4]).

A completely different application comes from the analysis of attack graphs in computer security (see [48] for references). The problem here is an (s,t) cut problem in a directed graph, where the nodes s and t represent the initial and the success state of an intruder, respectively. The edges are state transitions, labeled by the respective action (or “atomic attack”) of the attacker. The protector can hinder each type of atomic attack for a specific cost, the cost of that particular label. An (s-t) min-cut gives the cheapest set of actions whose blockage will prevent an intrusion and thereby protect the system.

Several other applications, for instance in image processing, arise from targeting the cooperation to subsets of specific elements.

1.4 Graph cut problems with submodular costs: an overview

Not only the min-cut problem, but many standard cut problems can be generalized to have submodular costs. Before we start with details about edge-submodular min-cuts, we introduce generalized versions of several standard graph cut problems, where we replace the linear cost function on edges by a submodular cost function on subsets of either edges or nodes. Only some of the node-based extensions have been analyzed in [44]. Here we only list the problems as an overview, and in the sequel analyze two particular cases, namely edge-submodular min-cuts and (s,t)-cuts.

The first type of generalization replaces the modular cost function on subsets of edges, that is, the sum of edge weights, by a submodular cost function on subsets of edges. This is the direction we take in this paper. Table 2 lists the edge-based extension for a number of well-known problems. It shows both the general min-cut version and the (s,t) cut version. The latter has the additional constraint that specified nodes s, t must be separated by the cut. The "(s,t)" version of a multicut problem (often called a "multiway cut" [45]) demands that each of the k partitions created by the cut must contain exactly one of k specified terminal nodes.

The second type of generalization refers to the modular edge cost function as a particular submodular function on subsets of nodes and replaces this particular function by an arbitrary submodular function on subsets of nodes. The resulting problems are listed in Table 3. The node-based extension is not a cut in the strict sense, since it might not involve the structure of a graph at all (if the common cost is replaced by something independent of the edges). Still, Svitkina and Fleischer [44] call their extended problems "cuts". We retain this name, but, as opposed to them, we restrict "cut" problems to have a symmetric cost function, that is $g(X) = g(V \setminus X)$ for all $X \subseteq V$. Table 3 shows a collection of generalizations for the node-based cost. The algorithms in [44] for Sparsest Cut and Min-Quotient Cut can be extended to (s,t) cuts in a straightforward way, as we show in Appendix A.

For node-based submodular generalizations, the (simple) (s,t) cut problem may be much harder than its min-cut version. The latter, which is an unconstrained minimization of a symmetric submodular function, can be solved in $O(n^3T)$ time [40], where T is the time to evaluate f on any set[4]. The (s,t) cut, on the other hand, is as hard as general submodular function minimization [40], for which the best known algorithm has complexity $O(n^5T + n^6)$ [38]. With the traditional cost f_{trad}, the difference in complexity is not as pronounced. The edge-based submodular cost functions might again show a tendency similar to that of the node-based generalization: our lower bound is stronger for the (s,t) cut problem than for the general min-cut. In the (s,t) case it holds even for matroid rank functions, the simplest case of submodular functions. The observation about (s,t) cuts being harder goes along with the fact that min-cut can be reduced to (s,t) cut via at most $O(n)$ (s,t) cuts, whereas the other direction is not as straightforward.

[4]In the *value-oracle model*, T is treated as constant.

Name	Cost	Constraints	(s,t) cut LB	(s,t) cut UB	Min-cut LB	Min-cut UB
Standard Min-cut	$f(\delta X) = \sum_{e \in \delta X} w(e)$ modular	–	1	1	1	1
CoopCut	$f(\delta X)$ matroid rank	–	$\frac{n^{1/3-\epsilon}}{(1+\delta)}$ [H]	$\min\{n/2, \lvert C^*\rvert, O(\sqrt{m}), \lvert P_{\max}\rvert\}$ [H]	?	$\min\{n/2, \lvert C^*\rvert, O(\sqrt{m}), \min_{s \in X, t \notin X} \lvert P_{\max}\rvert\}$ [H]
CoopCut	$f(\delta X)$ nonneg., monotone submodular	–	$\frac{n^{1/3-\epsilon}}{(1+\delta)}$ [H]	$\min\{n/2, \lvert C^*\rvert, O(\sqrt{m}\log n), \lvert P_{\max}\rvert\}$ [H]	?	$\min\{n/2, \lvert C^*\rvert, O(\sqrt{m}), \min_{s \in X, t \notin X} \lvert P_{\max}\rvert\}$ [H]
Coop Min-quotient Cut	$\frac{f(\delta X)}{\min\{\lvert X\rvert, n-\lvert X\rvert\}}$	–	NP hard		NP hard	
Coop Ratio/ Normalized Cut	$\frac{f(\delta X)}{w(X)w(\overline{X})}$; $w(X) = \lvert X\rvert$ or $w(X) = \deg(X)$	–	NP hard		NP hard	
Coop Sparsest Cut	$\frac{f(\delta X)}{\sum_{i:\lvert X \cap \{s_i, t_i\}\rvert = 1} d_i}$	–	NP hard		NP hard	
CoopCut, modular normalization	$\frac{f(\delta X)}{h(X)}$, h modular		NP hard		NP hard	
CoopCut, submodular normalization	$\frac{f(\delta X)}{h(X)}$, h submodular		NP hard		NP hard	
Coop b-balanced Cut	$f(\delta X)$	$\min\{w(X), w(\overline{X})\} \geq bn$	NP hard		NP hard	
Coop load balancing	$\max_i f_i(\delta X_i)$		?		NP hard	
Coop Multicut	$f(\bigcup_i \delta X_i)$		NP hard [9]		NP hard for arbitrary k, open for fixed k	

Table 2: List of submodular generalized cut problems where the cost is a function defined on subsets of edges, [H] means that the result is proved in this report. Here, $\delta X \subseteq E$ denotes the set of edges between X and $V \setminus X$, and $f(\delta X)$ the cost function. In addition, there is the multi-agent version of each of these problems, analogous to the versions in [16]. If there is no specification, f is a general submodular function. An asterisk means that the algorithm is randomized. Here, $n = \lvert V\rvert$, $m = \lvert E\rvert$, $P_{\max}$ is the length of the longest path between s and t. The problems are NP hard if their modular correspondent is NP hard, since submodular functions include modular ones. Multicut for fixed k is in P with traditional costs f_{trad} [19], thus the generalized version cannot "inherit" NP hardness. The Coop b-balanced min-cut is in P for at least one specific submodular function: $f(A) = \max_{e \in A} w(A)$ [23], however, the general problem is still NP hard since it is so for a modular f.

Name	Cost	Constraints	(s,t) cut LB	(s,t) cut UB	Min-cut LB	Min-cut UB
Standard min-cut	g graph cut function	–	1	1	1	1
NCoop Min-cut	$g(X)$, g symmetric submodular	–	like general non-symmetric SFM [40, Sec. 4] 1	1	1	1
NCoop Min-quotient Cut	$\frac{g(X)}{\min\{\lvert X\rvert, n-\lvert X\rvert\}}$	–	NP hard	$O(\sqrt{\frac{n}{\ln n}})^*$ [44] (see text)	NP hard	$O(\sqrt{\frac{n}{\ln n}})^*$ [44]
NCoop Ratio/ Normalized Cut (sub-case of Sp. Cut)	$\frac{g(X)}{w(X)w(\overline{X})}$; $w(X) = \lvert X\rvert$ or $w(X) = \deg(X)$	–	NP hard	$O(\sqrt{\frac{n}{\ln n}})^*$ [44] (see text)	NP hard	$O(\sqrt{\frac{n}{\ln n}})^*$ [44]
NCoop Sparsest Cut (NCSC)	$\frac{g(X)}{\sum_{i: \lvert X \cap \{s_i, t_i\}\rvert = 1} d_i}$	–	NP hard	$O(\sqrt{\frac{n}{\ln n}})^*$ [44] (see text)	NP hard	$O(\sqrt{\frac{n}{\ln n}})^*$ [44]
NCoop cut, modular normalization	$\frac{g(X)}{h(X)}$, h modular	–	NP hard	$O(\sqrt{\frac{n}{\ln n}})^*$ [44] (NSSC)	NP hard	$O(\sqrt{\frac{n}{\ln n}})^*$ (NSSC) [44]
NCoop cut, submodular normalization	$\frac{g(X)}{h(X)}$, h submodular		NP hard		NP hard	
NCoop b-balanced Cut	$g(X)$	$\min\{w(X), w(\overline{X})\} \geq bn$	NP hard		NP hard	b'-balanced cut, with cost within $O\big(\frac{\sqrt{n}}{\sqrt{\ln n}(b-b')}\big)^*$ of any b-balanced cut, $b' < b \leq 1/2$; $b' \leq 1/3$ [44]
NCoop load balancing	$\max_i g_i(X_i)$	–	NP hard			$O(\sqrt{\frac{n}{\ln n}})^*$ [44]
NCoop Multicut	$\sum_i g(X_i)$	–	NP hard	$2-2/k$ (monot. or symm. g [41, 49]), $k-1$ ($g \geq 0$) [49]	NP hard?	$2-2/k$ (monot. or symm. g [41, 49]), $k-1$ ($g \geq 0$) [49]

Table 3: List of submodular generalized cut problems where the cost is defined on subsets of nodes with the help of a symmetric submodular function $g(X)$ – we refer to them as "node-cooperative" (NCoop). An asterisk means that the algorithm is randomized. Here, $n = \lvert V\rvert$ and $m = \lvert E\rvert$, and "SFM" is submodular function minimization. For references for unconstrained SFM, see [13]. Note: Svitkina and Fleischer [44] solve the problems for a more general, non-symmetric nonnegative g, but symmetric nonnegative g are included in the upper bounds. Their lower bounds hold for non-symmetric g, and not trivially for a symmetric g.

1.5 Other related work

A variety of previous work relates to our problem. The traditional min-cut and (s, t) cut problems are special cases of CoopCut, since modular functions are also submodular. A lot of work has been devoted to these problems and lead to efficient polynomial-time algorithms (see e.g. [1, 27]). Both the traditional (s, t) cut and its dual, the max-flow problem, have been generalized in certain directions, but different from ours. Apart from traditional cuts, we are only aware of one special case of CoopCut in the literature. We summarize next the related generalizations of the primal and dual cut problem.

First, Hassin et al. [22] extend the cost function in the min-cut problem to non-submodular set functions of the form $f(A) = \max_i f_m(A \cap P_i)$ for a known partition $\{P_1, \dots P_k\}$ of the edge set E and a modular function f_m. If each P_i is interpreted as a color, then a min-cut with respect to f minimizes the maximum number of edges of the same color in the cut. A submodular cost could, on the other hand, minimize the total number of colors or labels in the cut and thus aim for a "uniform" cut. This latter, possibly with weights for labels, is the cost of *Label Cuts*, a special case of CoopCut. Zhang et al. [48] prove the NP hardness of such label (s, t) cuts and a lower bound of $2^{(\log n)^{1-(\log \log n)^{-c}}}$, which is weaker than our bound of $\Omega(n^{1/3})$ for general nonnegative submodular functions. Regarding upper bounds, their approximation guarantee of $O(\sqrt{m})$ is achieved by our (general) Algorithm EA if the cost is integral.

In general, a recent interest has arisen to replace modular functions with submodular ones in standard combinatorial problems. For example, Svitkina and Fleischer [44] consider submodular load balancing, sparsest cut and balanced cut. Submodular vertex cover, spanning trees, shortest paths, and matchings are addressed by Goel et al. [16, 17]. These works contain many examples how submodular costs render the combinatorial problem harder, even if the traditional modular-cost version is already NP hard. The function approximation in [18] makes algorithms for linear costs amenable to approximately solve problems with submodular costs, as exploited in our Algorithm EA. Further recent work considers submodular minimization with set cover constraints [25], uncapacitated facility location with submodular constraints [8], or submodular maximization with matroid constraints [46].

Second, several extensions have addressed the dual problem of the traditional min-cut, the max-flow problem. In the submodular flow problem (see references in [13]), a submodular function generalizes the "Kirchhoff laws" and defines feasible flows by restricting the net outflow out of a set of nodes. The cost, however, is still modular, and the problem solvable in polynomial time. Polymatroidal max-flow (PF) [32] comes closer to CoopCut, since it generalizes capacity constraints to be determined by locally submodular functions: two submodular capacity functions at each node restrict the in- and outflow of that node, respectively. The inflow (resp. outflow) must lie within the submodular polyhedron associated with the corresponding inflow (resp. outflow) capacity. We use PF for a polynomial-time approximation to CoopCut (Algorithm PMF).

1.6 More preliminaries about submodular functions

Next, we briefly provide a few additional definitions and details that will be used in the sequel.

First, our algorithms work for both directed and undirected graphs. Algorithms PMF, CR and the greedy algorithm are formulated for directed graphs. We transform an undirected graph into a directed one by replacing each edge $\{v_i, v_j\}$ by two opposing edges (v_i, v_j) and (v_j, v_i) with equivalent (parallel) cost.

A useful object is the *characteristic vector* $\chi_A \in \{0, 1\}^E$ of a set $A \subseteq E$. It has entries $\chi_A(e) = 1$ if $e \in A$ and $\chi_A(e) = 0$ otherwise.

The most basic class of submodular functions are *matroid rank functions*, which are normalized, monotone, integral and satisfy $f(e) \in \{0, 1\}$ for all $e \in E$. We mostly consider *polymatroid rank functions* that are normalized and monotone but may be nonintegral and may have $f(e) > 1$.

The class of *subadditive* functions is a superset of the class of submodular functions. A function is subadditive if for all $A, B \subseteq E$, it holds that $f(A) + f(B) \geq f(A \cup B)$.

An important concept in combinatorics is the *submodular polyhedron*

$$P_f = \{x \in \mathbb{R}^E \mid x(A) \leq f(A) \text{ for all } A \subseteq E\}.$$

For any submodular f, it holds that $f(A) = \max_{y \in P_f} y \cdot \chi_A$. The *Lovász extension* $\tilde{f}$ of f is the convex extension $\tilde{f} : (\mathbb{R}_0^+)^E \to \mathbb{R}$ with $\tilde{f}(x) = \max_{y \in P_f} y \cdot x$, so $\tilde{f}(\chi_A) = f(A)$ for all $A \subseteq E$ [33]. This definition shows that $g = \mathrm{argmax}_{g \in P_f}\, g \cdot x$ is a subgradient of $\tilde{f}$ in x [13, Lemma 6.19], because it implies that $g \cdot x' \leq \tilde{f}(x')$ for all $x' \in \mathbb{R}_0^+$, and, in consequence, $\tilde{f}(x') - \tilde{f}(x) \geq g \cdot (x' - x)$. The vector g can be found via the greedy algorithm

[12, 33]: sort the elements in x so that $x(\pi(1)) \geq x(\pi(2)) \geq \ldots \geq x(\pi(m))$, then $g(e_i) = f(\{e_{\pi(1)}, \ldots, e_{\pi(i)}\}) - f(\{e_{\pi(1)}, \ldots, e_{\pi(i-1)}\})$.

2 Hardness

We start by proving that submodular edge weights render CoopCut NP hard, as opposed to the standard modular-cost min-cut problem. We then prove that no polynomial-time algorithm can guarantee an approximation factor better than $\Omega(n^{1/3})$ for Coop-(s,t) cut with a monotone cost, and that essentially none can guarantee any approximation factor for non-monotone cost functions. All hardness results hold for Coop-(s,t) cut with nonnegative, monotone costs. If we relax the monotonicity condition, then the lower bound also holds for Coop min-cut by adding an edge (s,t) like in the proof of NP hardness.

The proofs here work for both directed and undirected graphs.

2.1 CoopCut is NP hard

It is well known that the common min-cut problem with nonnegative, modular edge weights becomes hard if edge weights can also be negative, or if size constraints are added on the partitions [47].

If we allow an arbitrary submodular function f to determine the edge costs, then it is immediately clear that the CoopCut problem becomes NP hard. For example, a simple modular edge-cost function with both positive and negative weights is submodular. Another example is correlation clustering [3] (CC), where a graph's edges are marked with either $+$ or $-$ corresponding to the case where the adjacent nodes are "similar" or "different", and the goal is to cluster the nodes to minimize the number of differences within the clusters and the similarities across clusters. A version of CC corresponds to CoopCut with a modular f that takes positive and negative values (see Appendix B). CC for a fixed number of partitions is NP hard [42] but does have a PTAS on complete graphs, and an $O(\sqrt{\log n})$ approximation algorithm for a 2-partition [15]. With strictly negative modular f, CoopCut becomes the max-cut problem, also NP hard but with a constant-factor approximation [27].

In the sequel, we prove that even with a nonnegative f, CoopCut is hard, by using a reduction from graph bisection[5].

Theorem 1 (NP hardness for nonnegative costs)**.** *CoopCut is NP hard for nonnegative submodular costs. Coop-(s,t) cut is NP hard even for monotone, nonnegative, integer-valued submodular costs (that is, integer-valued polymatroid rank functions).*

Definition 1 (Graph Bisection (GB))**.** *Given an undirected graph $G = (V,E)$ with weights $w : E \to \mathbb{R}_0^+$, find a partition $V_1, V_2 \subset V$ of the nodes, such that $|V_1| = |V_2| = |V|/2$ and $\sum_{e \in E \cap (V_1 \times V_2)} w(e)$ is minimal.*

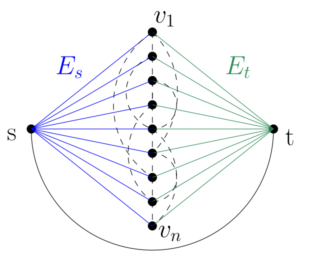


Figure 1: Reduction graph. The vertical subgraph with dashed edges is G_B.

GB is NP hard [14] and does not have a PTAS [26]. Let $G_B = (V_B, E_B)$ be an instance of GB with n nodes. We create an auxiliary graph G and submodular function f whose minimum CoopCut C^* bisects G_B optimally. That is, the restriction of C^* to E_B is the optimal solution to the GB of G_B. G has two additional nodes s, t and $2n+1$ additional edges. To form G, retain G_B with the costs on E_B and connect the additional nodes s and t to every vertex in G_B with corresponding new edge sets E_s and E_t. Also connect s with t. Thus, $G = (V_B \cup \{s,t\}, E_B \cup E_s \cup E_t \cup \{(s,t)\})$. The minimum CoopCut will (i) separate s and t, (ii) separate the nodes in V_B into two equal-sized partitions by assigning them to either s or t, that is, cut $n/2$ edges of each E_s and E_t, and (iii), have minimum cost with respect to the edges E_B. We will enforce the structural constraints (i) and (ii) with submodular penalty functions f_1 and f_2, respectively, and then add the original cost $f_3(A) = \sum_{e \in A \cap E_B} w(e)$. The penalty functions will be defined below. The overall cost is

$$f(A) = \alpha_1 f_1(A) + \alpha_2 f_2(A) + \alpha_3 f_3(A),$$

defined on $E(G)$ with $\alpha_i > 0$ to be specified later. First, let

$$f_1(A) = \begin{cases} 0 & \text{if } (s,t) \in A \\ |A| & \text{if } (s,t) \notin A. \end{cases}$$

This function is submodular and favors (i.e., is zero for) the inclusion of edge (s,t). Together with a large weight α_1, f_1 ensures that any good cut must separate s and t. This separation implies that at least n edges in $E_s \cup E_t$ must be cut as well.

[5]Only after completing our proofs we came to know about [48]. They address a sub-case of CoopCut, hence their proof of NP hardness of (s,t) label cuts implies the NP hardness of ES (s,t) cuts. We still include our proof since it is different and illustrates the expressive power of submodular costs. The lower bound for the special case of label cuts in [48] is weaker than the one we prove here.

Constraint (ii) addresses the equipartition. First, any (s,t) cut must include at least n edges from $E_s \cup E_t$. A balanced cut of V_B assigns $n/2$ nodes to s, cutting their edges to t, and the other $n/2$ nodes to t, cutting their edges to s. Hence, our penalty function should reach its minimum when the cut includes as many edges from E_s as from E_t, namely $n/2 = |E_s|/2 = |E_t|/2$, and not both edges (s,v) and (v,t) for a node v. The straightforward solution would be a function $f_2(A) = g(|A|)$, where g is a convex function that reaches its minimum at $n/2$. However, for f_2 to be submodular, g must be *concave* [33] and thus cannot reach its minimum at an arbitrary value. We could define two functions, each acting on half of E_s, that reach their minimum at their boundary. But this strategy will favor a particular subset of size $n/2$, and we want only the size to be relevant. The rescue comes in combining structure and cost, and in randomizing over functions that favor a specific configuration. "Randomization" means to use the expectation over drawing one such function uniformly at random. This expectation is proportional to the sum over the values of all possible such functions. We will derive the following solution in the next section:

$$f_2(A) = (|A_s| + |A_t|)D(n) - (|A_s||A_t| - |A_{s\cap t}|)D'(n-1),$$

where $D(n)$ and $D'(n-1)$ are suitable constants depending on n. The first term (involving $|A_s| + |A_t|$) when minimal ensures that we cut no more than n edges in $E_s \cup E_t$, which is the minimum required (the negative term cannot change this fact: below, we argue that f_2 is monotone). Therefore, for the right $D(n)$, we may assume that $|A_s| + |A_t| = n$ is constant. We also need to ensure that the sizes are appropriate (each $n/2$) which is where the rest of the derivation comes in. The term $|A_s||A_t|$ is maximal if $|A_s| = |A_t| = (|A_s| + |A_t|)/2$. Finally, the penalty for $A_{s\cap t}$ disfavors the "overlap" of A_s and A_t, that is, cutting off both (v_i, s) and (v_i, t) for a node v_i. Thanks to the choice of $D(n)$ and $D'(n-1)$, the function f_2 is nonnegative and monotone and minimal for any cut that cuts exactly half of the edges to s and to t, with no "overlap".

If $D(n)$ is the number of derangements of n elements, and $D'(n)$ is the number of "derangements" where one specific element is allowed all mappings (including $\sigma(i) = i$), then f_2 can be derived as a sum of $D(n)$ rank functions (see next subsection), and is thus nonnegative, submodular and monotone. The monotonicity implies that a cut with more than n edges in $E_s \cup E_t$ will have a higher cost than the optimal bipartition that cuts n edges in $E_s \cup E_t$. In this case, the constants are $D(n) = n! \sum_{k=0}^{n} (-1)^k/k!$ [43], and $D'(n-1) = \sum_{k=0}^{n-1} (n-2)!(n-1-k)!(-1)^k$ (see Appendix C). Both are computable in polynomial time, which is necessary for a polynomial-time reduction.

Lastly, we choose $\alpha_3 = 1$, $\alpha_2 = 10 \sum_{e \in E_B} w(e)$ and $\alpha_1 = 5\alpha_2 n^2 D(n)$.

2.1.1 Derivation of f_2

For the interested reader, we derive the function f_2 as a sum of matroid rank functions. Since rank functions are monotone submodular, f_2 must be so as well.

Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be two sets of n linearly independent vectors, depicted as nodes in Figure 2(a). Each x_i has a linearly dependent correspondent in Y, illustrated by a connecting line. The connections form a mapping $X \to Y$ – actually a permutation $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$, $x_i \mapsto y_{\sigma(i)}$. Each edge $(v_i, s) \in E_s$ corresponds to an element $\phi((v_i, s)) = x_i$ in X, and equally, each edge in E_t to an element in Y via $\phi((v_i, t)) = y_i$. For each permutation σ, we get a rank function

$$r_\sigma(A) = \hat{r}_\sigma(\phi(A_s) \cup \phi(A_t)).$$

The rank $\hat{r}_\sigma$ measures the rank of the set of vectors in $X \cup Y$ chosen by A, based on the dependencies determined by σ. This rank is the total number of vectors, $|A_s| + |A_t|$, minus the number of coincidences (vector pairs across $\phi(A_s)$ and $\phi(A_t)$ that are dependent) under the current permutation; see also Figures 2(b), 2(c). That is,

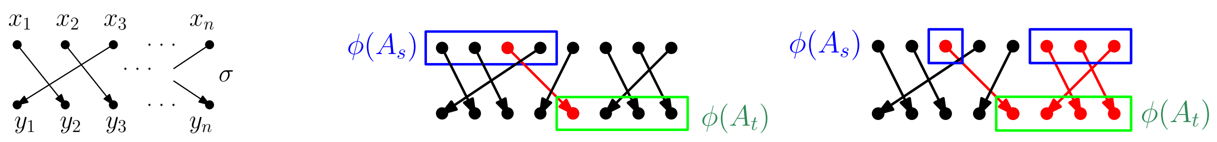
$$\hat{r}_\sigma(\phi(A_s) \cup \phi(A_t)) = |A_s| + |A_t| - \underbrace{\left|\{(x_i, y_{\sigma(i)})\}_{i=1}^{n} \;\cap\; (\phi(A_s) \times \phi(A_t))\right|}_{\text{coincidences}}. \tag{1}$$

It can also be seen as the number of connected components in the subgraph defined by $\phi(A_s \cup A_t)$.

Consider now a fixed permutation σ. For a fixed size of $A_s \cup A_t$, the rank $\hat{r}_\sigma(\phi(A_s) \cup \phi(A_t))$ is minimal if the number of coincidences is maximal, that is, $\phi(A_t) = \sigma(\phi(A_s))$, which implies $|A_s| = |A_t|$.

However, a single permutation restricts the minimizing A_t to satisfy $\phi(A_t) = \sigma(\phi(A_s))$. For more freedom, we consider a large set of permutations: the set $\mathfrak{S}$ of all derangements, that is, all permutations with $\sigma(i) \neq i$ for all i. Then f_2 is

$$f_2(A) = \sum_{\sigma \in \mathfrak{S}} r_\sigma(A).$$


x_1 x_2 x_3 $\cdots$ x_n
σ
y_1 y_2 y_3 $\cdots$ y_n
$\phi(A_s)$
$\phi(A_t)$
$\phi(A_s)$
$\phi(A_t)$

(a) Mappings via σ. (b) One coincidence. (c) Many coincidences.

Figure 2: Mappings and coincidences. (a) Mappings between elements $x_i = \phi(v_i, s)$ and $y_i = \phi(v_i, t)$ via the derangement σ: the vectors x_i and $y_{\sigma(i)}$ are linearly dependent. (b,c) Illustration of coincidences: The rank function $\hat{r}_\sigma$ for the permutation depicted here is $4 + 4 - 1 = 7$ in (b), and $4 + 4 - 4 = 4$ in (c). The coincidences, that is, linearly dependent vectors across $\phi(A_s)$ and $\phi(A_t)$, are marked in red. They can only contribute one to the rank, but are counted once in $|A_s|$ and once in $|A_t|$, thus the negative correction is needed.

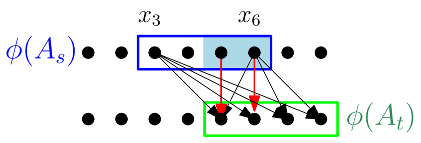


Figure 3: Counts of coincidences and forbidden mappings. The set $\phi(A_{s\cap t})$ is shaded in light blue. The vector x_3 can have coincidences $\sigma(3)$ with all $y_j \in \phi(A_t)$, indicated by black arrows; vector $x_6 \in \phi(A_{s\cap t})$ is never mapped to $y_6 \in \phi(A_t)$. The forbidden connections are red, whereas the possible ones for x_3 and x_6 are black.

Normalized by $|\mathfrak{S}|$, this quantity is the expected rank if a derangement σ is chosen uniformly at random from $\mathfrak{S}$. Why derangements? Derangements are exactly those permutations for which $\sigma(\phi(e)) \neq \phi(e)$, that is, including (s, v) never favors the inclusion of (v, t). This restriction from general permutations will contribute a penalty for any cut that separates a node from both s and t.

As mentioned above, $f_2(A)$ is submodular since it is a sum of rank functions. From Equation (1), this sum of ranks can be seen to consist of two terms:

$$\begin{aligned}\sum_{\sigma\in\mathfrak{S}} r_\sigma(A) &= |\mathfrak{S}|\big(|A_s| + |A_t|\big) \; - \; \sum_{\sigma\in\mathfrak{S}} \big|\big\{(x_i, y_{\sigma(i)})\big\}_{i=1}^n \; \cap \; (\phi(A_s) \times \phi(A_t))\big| \\ &= |\mathfrak{S}|\big(|A_s| + |A_t|\big) \; - \; \sum_{x_i\in\phi(A_s)} \sum_{\sigma\in\mathfrak{S}} \big|(x_i, y_{\sigma(i)}) \cap (\{x_i\} \times \phi(A_t))\big| \qquad (2)\end{aligned}$$

The first term is the total number of derangements, $|\mathfrak{S}| = D(n)$, times the total number of vectors in $\phi(A_s) \cup \phi(A_t)$. This quantity would be the sum of ranks if all vectors were linearly independent in all derangements. To correct for the dependencies, we subtract the total number of coincidences in all derangements as above.

We will count the total number of coincidences as the sum of the number of coincidences for each x_i in $\phi(A_s)$, as shown in (2). Consider a given $x_i \in \phi(A_s)$. How many derangements map it to an element in $\phi(A_t)$ to yield a coincidence? We know that any $x_i \in \phi(A_s)$ cannot be mapped to its correspondent $y_i \in \phi(A_t)$ by any σ. To account for this restriction, partition A_s into a set $A_{s\cap t}$ and $A_s \setminus A_{s\cap t}$ as in Figure 3. Each element in the projection $\phi(A_s \setminus A_{s\cap t})$ of the latter set can be mapped to any element $y_k \in \phi(A_t)$. For each such (fixed) pairing (x_i, y_k), any of the remaining $n - 1$ elements x_j can be mapped to any y_ℓ with $j \neq \ell$. In fact, the element x_k can be mapped to any remaining target in Y, since its counterpart y_k is already "taken" by x_i. Let $D'(n - 1)$ denote the number of such permutations of $n - 1$ elements (pair (x_i, y_k), i.e., $\sigma(i) = k$, is fixed), where one specific element x_k can be mapped to any other of the $n - 1$ elements, and the remaining elements must not be mapped to their counterparts ($\sigma(j) \neq j$). Then there are $D'(n - 1)$ derangements σ realizing $\sigma(i) = k$, for each $y_k \in \phi(A_t)$, yielding $|A_t|D'(n - 1)$ coincidences for each x_i in $\phi(A_s \setminus A_{s\cap t})$, so $|A_s \setminus A_{s\cap t}| \cdot |A_t| \cdot D'(n - 1)$ in total.

Each element x_i in the remaining $\phi(A_{s\cap t})$ can be mapped to $|A_t| - 1$ elements in $\phi(A_t)$, since its counterpart y_i is in $\phi(A_t)$. This leads to another $|A_{s\cap t}| \cdot (|A_t| - 1) \cdot D'(n - 1)$ coincidences. Hence, in total we get

$$\begin{aligned}f_2(A) &= (|A_s| + |A_t|)D(n) - \sum_{x_i\in A_s\setminus A_{s\cap t}} \sum_{y_k\in A_t} D'(n - 1) - \sum_{x_i\in A_{s\cap t}} \sum_{y_k\in A_t, k\neq i} D'(n - 1) \\ &= (|A_s| + |A_t|)D(n) - \big(|A_s| - |A_{s\cap t}|\big)|A_t|D'(n - 1) - |A_{s\cap t}|(|A_t| - 1)D'(n - 1) \\ &= (|A_s| + |A_t|)D(n) - (|A_s||A_t| - |A_{s\cap t}|)D'(n - 1),\end{aligned}$$

with $D(n) = n! \sum_{k=0}^{n}(-1)^k/k!$ [43], and $D'(n-1) = \sum_{k=0}^{n-1}(n-2)!(n-1-k)!(-1)^k$, as derived in Appendix C.

A similar construction with entropies yields the same cost function. Let the nodes x_i, y_i be random variables in a graphical model, each with an entropy $H(x_i) = H(y_j) = c$ for all i, j. All variables in X are mutually independent, and so are the variables in Y. The linear dependencies of the vector above are now statistical dependencies with $H(x_i, y_{\sigma(i)}) = H(x_i) = c$ and $H(x_i, y_j) = 2c$ if $j \neq \sigma(i)$. The rank r_σ corresponds to the joint entropy of the variables $\phi(A_s \cup A_t)$, and the rest is the same as above.

2.2 Lower bounds for Coop-(s, t) cut

In this section, we show lower bounds on the approximation factor of Coop-(s, t) cut.

Theorem 2 (Lower bound for Coop-(s, t) cut with nonnegative, monotone costs)**.** *For any fixed $\epsilon > 0$, $\delta > 0$, any (randomized) approximation algorithm for Coop-(s, t) cut with monotone cost needs exponentially many queries for an approximation factor of or better than $n^{1/3-\epsilon}/(1 + \delta)$.*

2.2.1 Proof of Theorem 2

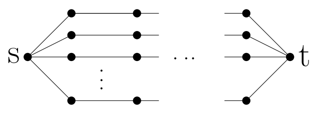


Figure 4: Ladder graph

We prove Theorem 2 with the technique of [18] that was also used in [16, 25, 44]. The proof shows a type of input where for a polynomial number of evaluations, it is very unlikely that we can distinguish between two cost functions f, h that may appear as input. Their optima differ by a large factor, say α, and any solution for f that is within a factor of α of the optimum would be enough evidence to discriminate f and h. Thus, no polynomial-time algorithm can guarantee an approximation ratio better than α, since it would have to distinguish between the two functions. To achieve a low probability of discrimination, we randomly pick a cut $R \subset E$ and design f so that for a query $Q \subseteq E$, $f(Q) \neq h(Q)$ only if $|Q \cap R|$ is large, an event of exponentially small probability. By a union bound argument, the probability of having $f(Q) \neq h(Q)$ for any query in a set of polynomially many queries is still very small — too small for an approximation guarantee better than α.

Consider the graph in Figure 4. It has k columns of edges, ℓ parallel paths from s to t, $m = k\ell$ edges and $n = m - \ell + 2$ nodes. Any (s, t) separator cuts each path at least once. Thus, there are k^ℓ minimal (s, t) cuts. To sample a random cut $R \subset E$ with $|R| = \ell$, we choose one edge from each path uniformly at random with probability $1/k$. Let $\beta = (1 + \delta)\ell/k < \ell$ and

$$h(Q) = \min\{|Q|, \ell\}; \qquad\qquad f(Q) = \min\{|Q \cap \overline{R}| + \min\{|Q \cap R|,\ \beta\},\ \ell\}. \tag{3}$$

The functions f and h are equal on most queries, and differ only if $f(Q) < h(Q)$. This is only the case if Q overlaps with R in more than β edges, and with $\overline{R}$ in not too many edges (i.e., less than $\ell - \beta$). In particular on all other cuts, $f(Q) = h(Q) = \ell$. We choose $k = m^{1/3-\epsilon}$ and $\ell = m^{2/3+\epsilon}$, so the ratio of the optima of h and f is $\ell/\beta = m^{1/3-\epsilon}/(1 + \delta)$. Finding the optimum for f means to be able to distinguish h and f.

We will make these ideas more formal and compute the probability $P(f(Q) \neq h(Q)) = P(f(Q) < h(Q))$ for a given $Q \subseteq E$. If $|Q| \leq \ell$, then $f(Q) < h(Q)$ only if $\beta < |Q \cap R|$, and the probability

$$P(f(Q) < h(Q)) = P(|Q \cap R| > \beta)$$

increases as Q grows. If, on the other hand, $|Q| \geq \ell$, then the probability

$$P(f(Q) < h(Q)) = P(|Q \cap \overline{R}| + \min\{|Q \cap R|,\ \beta\} < \ell)$$

decreases as Q grows. Hence, the probability of difference is largest when $|Q| = \ell$.

So let $|Q| = \ell$. Then we can distribute Q over at most $d = \ell$ and at least $d > \beta$ paths to make $P(|Q \cap R| > \beta)$ nonzero. If Q covers $b \leq k$ edges of a path, then the probability that Q includes the edge in this path that is in R is b/k. The expected overlap is $\mathbb{E}[\ |Q \cap R|\ ] = |Q|/k = \ell/k$. Since the edges in R were sampled independently from identical distributions, we can bound the probability of a large intersection via Hoeffding's bound [24]:

$$P\big(\ |Q \cap R| \geq (1 + \delta)\ell/k\ \big)\ \leq\ \exp(-2\delta^2\ell^2/(dk^2))\ \leq\ \exp(-2\delta^2\ell/k^2)\ =\ \exp(-2m^{3\epsilon}\delta^2).$$

Since the probability of $f(Q) < h(Q)$ is exponentially small in $m = n + \ell - 2$, the theorem holds for the bound $m^{1/3-\epsilon}/(1 + \delta)$ and thus also for $n^{1/3-\epsilon}/(1 + \delta) < m^{1/3-\epsilon}/(1 + \delta)$.

Note that the proof only relies on nonnegative monotone submodular functions, in fact, truncated matroid rank functions [18]. Rounding ℓ and β to the closest integer will make h and f true (integer-valued) rank functions, so a rounded bound holds even for matroid rank functions.

2.3 Worst-case versus special case

We showed that going from modular to submodular cost functions makes the min-cut and (s,t) cut problems much harder. However, it is not always the case that a submodular cost function makes the problem harder than the modular-cost version. As an example, the "bottleneck" cost $f(A) = \max_{e \in A} w(e)$ is submodular but leads to min-cut problems that are easy to optimize by a greedy algorithm: initially, each node is a cluster and then greedily we merge the two clusters that are connected by the heaviest edge (see also Section 3.1). Even stronger, the modular-cost graph bisection problem is NP hard [14], whereas graph bisection with bottleneck cost is in P [23].

Nevertheless, for the general class of (monotone) submodular functions, our hardness results hold.

3 Approximation Algorithms

After showing the hardness of CoopCut, we will analyze four approximation algorithms and compare them to a number of heuristics. All of them solve the (s,t) cut problem, and Algorithm MBI and EA also directly the min-cut problem. The other algorithms solve the general min-cut by repeating the (s,t) cut for a fixed s with all possible t and selecting the best of those cuts[6].

Before delving into the details of the algorithms, some general thoughts will help to get a better picture of the problem. First, we illustrate that an approximation algorithm must use the graph structure. Second, we briefly introduce the general principles we use, and finally we devote a subsection to each algorithm and its guarantees.

In the sequel, $C^* \subseteq E$ denotes the optimal cooperative cut, and $n = |V|$ the number of nodes. Unless stated otherwise, in this section we assume f to be nonnegative and monotone.

3.1 Optimizing a mere node function is hard

A preliminary thought about algorithms addresses the need to use the graph structure. Queyranne's algorithm for minimizing symmetric submodular functions [40] finds the standard (modular) min-cut without explicitly using the graph structure, which is only implicit in defining the node-based cost $g(X) = f(\delta X)$. Instead of minimizing an edge-based submodular function with difficult cut constraints, can we in general solve the unconstrained problem $\min_{X \subset V} g(X)$ for a nontrivial $X \neq \emptyset, V$?

If f is nonnegative, normalized, and monotone, then g is subadditive, i.e., g satisfies $g(X) + g(Y) \geq g(X \cup Y)$ for all $X, Y \subseteq V$. This has no general benefit though: The following example shows that not fully exploiting the graph structure makes the minimization problem inapproximable at any arbitrary factor $b > 0$. Let $R \subseteq V$ be an arbitrary set of nodes and $b > 1$ a large number, and define a subadditive function $g : 2^V \to \mathbb{R}$ as

$$g(X) = \begin{cases} 1 & \text{if } X = R \text{ or } X = V \setminus R \\ 0 & \text{if } X = \emptyset \text{ or } X = V \\ b & \text{otherwise.} \end{cases}$$

If R is unknown and the trivial solutions $\emptyset$, V forbidden, then only exponentially many evaluations of g can guarantee a solution with a cost lower than b times the minimum. This difficult function is the node-based cost $g(X) = f(\delta X)$ of CoopCut with edge costs $f(A) = \max_{e \in A} w(e)$ and

$$w(e) = \begin{cases} 1 & \text{if } e \in \delta R \\ b & \text{otherwise.} \end{cases}$$

Knowing the graph structure (thereby breaking apart g), however, we can find the optimum of this particular example in polynomial time by greedily merging node pairs that are connected by "heavy" edges of weight b.

[6]If the adjacency matrix of the graph is not symmetric, that is, the cut direction matters, then we do $2(n-1)$ (s,t) cuts by also swapping s and t each time.

3.2 Techniques

The difficulty of CoopCut lies in the non-locality of the edge cooperations with respect to the graph structure. That is, the joint cost of two disjoint edge sets A, B can be much smaller than the sum of their costs: $f(A \cup B) \ll f(A) + f(B)$. The cooperation may only become evident for large sets, as in the proof of the lower bounds – and there are exponentially many such sets, of which only a few might enjoy cooperation. Exactly this reduction in cost for specific edges, however, can determine the minimum cut, again as in the proof for the lower bound (Section 2.2). The minimum cut can have a lot of edges ($n^2/4$ in the examples in Section 4.2) that have a low joint cost.

If submodular cooperations are restricted to the sets of edges that share an adjacent node, and the cost function is modular on anything coarser, then the problem can be exactly solved in polynomial time (Section 3.5 and [32]). Even simpler, the common min-cut problem with a modular cost completely lacks edge cooperations; they can be viewed as limited to single edges. Two of our approximation algorithms, Algorithm PMF and MBI, rely on a local approximation of the submodularity, that is, we split the set E into small local sets E_i (single edges or neighborhoods). The new cost function may be submodular within a set, but behaves in a modular way across sets, i.e., $\hat{f}(A) = \sum_{i=1}^{k} f_i(E_i \cap A)$. If the E_i are a particular improved version of edge neighborhoods δv of single nodes v, then minimizing $\hat{f}$ corresponds to the dual problem of polymatroidal network flows [32], a strategy used by Algorithm PMF. Algorithm MBI reduces E_i to single edges, that is, a modular approximation $\hat{f}$, and then tests a set of candidate cuts in the min-cost cut basis of the modular approximation. Another strategy, applied by Algorithm EA, is to first approximate the cost function by a submodular function that is amenable to efficient optimization [18]. The efficiency of the optimization again relies on algorithms for the modular case; in the end, a modular function is minimized. Finally, f can be seen as a function on indicator vectors $\{0,1\}^E$. The *Lovász extension* [33] extends this function to a convex function on $(\mathbb{R}_0^+)^E$. Algorithm CR solves this convex relaxation of CoopCut, retaining the cut constraints.

3.3 A useful approximation Lemma

For Algorithms PMF and EA that rely on a simplifying approximation of the cost function, the following Lemma will be useful.

Lemma 1. *Let $\hat{C} = \arg\min_{C \in \mathcal{C}} \hat{f}(C)$ for a global approximation $\hat{f}$ with $f(A) \leq \hat{f}(A) \leq \alpha f(A)$ for all $A \subseteq E$, and $C^* = \operatorname{argmin}_{C \in \mathcal{C}} f(C)$. Then*

$$f(\hat{C}) \leq \alpha f(C^*).$$

In particular, it is enough if $f(C^*) \leq \hat{f}(C^*) \leq \alpha f(C^*)$ holds for C^*.

Proof. Since $\hat{f}(\hat{C}) \leq \hat{f}(C^*)$, it is $f(\hat{C}) \leq \hat{f}(\hat{C}) \leq \hat{f}(C^*) \leq \alpha f(C^*)$. □

3.4 A reference-based improvement step

Any solution $C \subseteq E$ returned by any of the four algorithms might be improved upon by a post-processing step that finds a cut minimizing the bounds [35]

$$f(B) \leq h_1(B, C) \triangleq f(C) - \sum_{e \in C \setminus B} \rho_e(E \setminus \{e\}) + \sum_{e \in B \setminus C} \rho_e(C) \tag{4}$$

$$f(B) \leq h_2(B, C) \triangleq f(C) - \sum_{e \in C \setminus B} \rho_e(C \setminus \{e\}) + \sum_{e \in B \setminus C} \rho_e(\emptyset), \tag{5}$$

where the gain is defined as $\rho_A(D) \triangleq f(A \cup D) - f(D)$. In particular bound h_1 includes the cooperations of any edge in $E \setminus C$ with the reference set C, and thus goes beyond the local restriction of submodularity in some approximations.

The minimizer of h_ℓ can be found via a modular min-cut with modified edge weights. Set

$$w_{1,C}(e) = \begin{cases} \rho_e(E \setminus \{e\}) & \text{if } e \in C \\ \rho_e(C) & \text{otherwise;} \end{cases} \qquad w_{2,C}(e) = \begin{cases} \rho_e(C \setminus \{e\}) & \text{if } e \in C \\ \rho_e(\emptyset) & \text{otherwise.} \end{cases}$$

With these weights, the modular weight of a cut B is

$$\sum_{e \in B} w_{1,C}(e) = \sum_{e \in C \cap B} \rho_e(E \setminus \{e\}) + \sum_{e \in B \setminus C} \rho_e(C) = h_1(B) - \underbrace{f(C) + \sum_{e \in C} \rho_e(E \setminus \{e\})}_{\text{constant w.r.t. } B},$$

and analogously for $w_{2,C}$. If the optimizer of h_j has a lower f-cost than the initial C, we can take it as the next comparison set C and iterate. The pseudocode for this iterative bound minimization is shown as Algorithm 1. To improve on one solution C, we call Algorithm 1 with $\mathcal{I} = \{C\}$ and use the returned solution if it is better than C. In Section 3.6, we will use a larger set $\mathcal{I}$.

Algorithm 1: Iterative bound minimization

```
Input: $G = (V, E)$; nonnegative monotone cost function $f$: $2^E \to \mathbb{R}_0^+$; reference initialization set $\mathcal{I} = \{I_1, \ldots, I_k\}$, $I_j \subseteq E$; [source / sink nodes $s$, $t$]
Output: cut $B \subseteq E$
for $j = 1$ to $k$ do
    set weights $w_{\ell,I_j}$ for $\ell = 1,2.$;
    find [$(s,t)$-]min-cut $C_\ell$ for edge weights $w_{\ell,I_j}(\cdot, I_j)$;
    set $C = \text{argmin}_{C_\ell} f(C)$;
    repeat
        $B_j = C$;
        set weights $w_{\ell,B_j}$ for $\ell = 1,2.$;
        find [$(s,t)$-]min-cut $C_\ell$ for edge weights $w_{\ell,B_j}$;
        $C = \text{argmin}_{C_\ell} f(C)$;
    until $f(C) > f(B_j)$;
end
return $B = \arg\min_{B_1,\ldots,B_k} f(B_j)$;
```

The reference-based improvement helps most if $\rho_e(E \setminus \{e\})$ is larger than zero for most edges (this does not hold, for instance, for truncated functions), and if the low cost of C^* with respect to f is based on cooperations that can be identified from small sets of edges. If the edges in $C \cap C^*$ suffice to reduce the new weight $\rho_e(C)$ of any $e \in C^* \setminus C$ enough compared to the original weight $f(e)$, then the minimizer of h_1 will be close to C^*.

A tighter bound can be optimized by the algorithm for polymatroidal network flows, via the same construction as the approximation for Algorithm PMF. Let $C \subseteq E$ again be the comparison set, and $\{P_i\}_i$ a partition of $E \setminus C$, in conformity with the neighborhood sets $\delta v \subset E$, that means, each $P_i \subset \delta v$ for some $v \in V$. The tighter bound is

$$f(B) \;\leq\; h_1'(B) \triangleq f(C) - \sum_{e \in C \setminus B} \rho_e(E \setminus \{e\}) + \sum_i \rho_{B \cap P_i}(C).$$

It is an upper bound since, by diminishing returns and subadditivity of f, it holds for any partition that

$$f(B) + \sum_{e \in C \setminus B} \rho_e(E \setminus \{e\}) \;\leq\; f(B \cup C) \;\leq\; f(C) + \rho_{B \setminus C}(C) \;\leq\; f(C) + \sum_i \rho_{B \cap P_i}(C).$$

The cost function we minimize for this bound is

$$f_C(B) = \sum_i \rho_{B \cap P_i}(C) + \sum_{e \in B \cap C} \rho_e(E) = h_1'(B) - f(C) + \sum_{e \in S} \rho_e(E \setminus \{e\}).$$

Since $\rho_{B \cap P_i}(C)$ is submodular for a fixed C, and the submodular interactions are restricted to local neighborhoods, a tightened version of $f_C(B)$ can be optimized as the dual of a polymatroidal flow problem (see Section 3.5 for details). The tightened version automatically chooses the partition that gives the lowest function value, i.e., the best bound of the neighborhood type. If we prefer to use a fixed partition, we can use the trick outlined in Section 3.5.1.

3.5 Algorithm I: Approximation via "polymatroidal network flows" (PMF)

As mentioned above, the intractability of CoopCut relies on the global, unknown interaction of edge sets. Here, we relax this difficulty by restricting the submodular behavior to known, limited sets of edges. To do so, we partition E into disjoint sets E_i and then use the approximation $\hat{f}(A) = \sum_i f(A \cap E_i)$. For tractability, we make the partition "local". Let $\Pi(A) = \{A_1, \ldots, A_n\}$ be a partition of an edge set $A \subseteq E$, where A_i only contains edges incident to node $v_i \in V$. With $\mathcal{P}_A$ denoting the set of all such partitions, let

$$\hat{f}_{\text{PMF}}(A) = \min_{\Pi(A) \in \mathcal{P}_A} \sum_i f(A_i), \tag{6}$$

that is, each edge is assigned to one of its incident nodes, and this is what defines the variants in the set of partitions $\mathcal{P}_A$. Figure 5 illustrates an example of such a partition, where all the green edges are assigned to their tail node, the blue edge is assigned to its tail node, and the red edges are assigned to their head node.

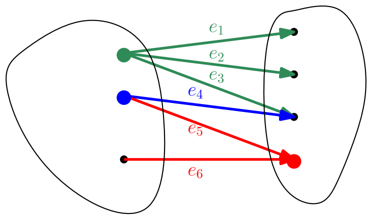


Figure 5: Partition of the cut edges; edges of the same color are in the same set A_i. The approximation here is $\hat{f}_{\text{PMF}}(e_1, e_2, e_3, e_4, e_5, e_6) = f(e_1, e_2, e_3) + f(e_4) + f(e_5, e_6)$.

Similar to the traditional max-flow min-cut duality, the cut problem with cost $\hat{f}_{\text{PMF}}$ corresponds to the dual of a generalized, polymatroidal max-flow problem [32] with the capacity function f at each node.

Polymatroidal network flows [32] generalize the capacity function of the traditional max-flow problem as follows. At each node v_i in the directed graph, a submodular function f_i^{in} defines the capacities of the incoming edges $\delta^- v_i$, and a submodular function f_i^{out} defines the capacities of the outgoing edges $\delta^+ v_i$. The flow $\varphi(\delta^- v_i)$ into v_i must be in the submodular polyhedron of f_i^{in}, that is, satisfy $\varphi(A) \leq f_i^{\text{in}}(A)$ for all edge sets $A \subseteq \delta^- v_i$. Equivalent constraints hold for the in- and out-flow of all nodes. An augmenting paths algorithm solves the maximum (s,t) flow for these capacities exactly in $O(|E|^5 d)$ time, where d is the time to solve problems of the form $\min_{A \subseteq (\delta^- v) \setminus e} f_i^{\text{in}}(A) - \varphi(A)$. Let f^{in} be the direct sum of the f_i^{in}, and f^{out} the direct sum of the f_i^{out}, then the dual of the polymatroidal max-flow problem (PF) is a min-cut for the convolution cost $(f^{\text{in}} * f^{\text{out}})(A) = \min_{B \subseteq A}(f^{\text{in}}(B) + f^{\text{out}}(A \setminus B))$ [33]. The convolution of two submodular functions is not in general submodular, and may thus be hard to optimize. The specific function here, however, can be optimized exactly, thanks to local restriction of submodularity and the combinatorial structure. The PF framework works for any submodular function.

To see how $\hat{f}_{\text{PMF}}$ corresponds to the convolution $(f^{\text{in}} * f^{\text{out}})$, let us look at a minimal cut C in a directed graph. In $\hat{f}_{\text{PMF}}$, each edge $e = (v_i, v_j)$ will be assigned either to its head node v_j or tail node v_i. Let, for a partition $\Pi(C)$, the set C_i^{in} be the set of incoming edges that are assigned to v_i, and C_i^{out} be the set of outgoing edges that are assigned to v_i. Since C is a minimal cut, at least one of C_i^{in} and C_i^{out} must be empty. This sparseness follows because C only includes edges that are directed from s to t (since, analogous to the modular-cost mincut-maxflow duality, the "back-edges" are void in the flow problem). Then

$$\begin{aligned}
\hat{f}_{\text{PMF}}(C) &= \min_{\Pi(C)} \sum_i f(C_i^{\text{in}} \cup C_i^{\text{out}}) \\
&= \min_{C_i^{\text{in}}, C_i^{\text{out}}} \sum_i f(C_i^{\text{in}}) + f(C_i^{\text{out}}) \\
&= \min_{C^{\text{in}}} f^{\text{in}}(C^{\text{in}}) + f^{\text{out}}(C \setminus C^{\text{in}}) \\
&= (f^{\text{in}} * f^{\text{out}})(C).
\end{aligned}$$

Since we only care about the approximation at cuts, we can use $\hat{f}_{\text{PMF}}(A) = (f^{\text{in}} * f^{\text{out}})(A)$.

All capacity functions f_i^{in}, f_i^{out} are set to f restricted to the particular domain $\delta^- v_i$, $\delta^+ v_i$, respectively. The corresponding f^{in}, f^{out} are modular across neighborhood sets, and the convolution automatically assigns edges to their head or tail node to minimize the resulting cost, that is, to get the tightest approximation.

To solve the (s,t)-cut for $\hat{f}_{\text{PMF}}$ as a PF, we transform the undirected graph into a directed one by replacing each undirected edge e by two opposing directed edges e^+, e^- that are "parallel" with respect to the cost f (the signs here are assigned arbitrarily); with a little abuse of notation we use the same f for the corresponding cost of the directed edges. Edges e^+, e^- are parallel if

$$f(A \cup \{e^+\}) = f(A \cup \{e^-\})$$

for all $A \subseteq E$; we also set $f(A \cup \{e^+, e^-\}) = f(A \cup \{e^+\})$. The cost on the directed edges is equivalent to that on the undirected edges: parallelism yields that a set of directed edges has the same cost as the union of the undirected counterparts, regardless of whether both e^+ and e^- or only one of them is in the directed set.

The parallelism in f is lost in $\hat{f}_{\text{PMF}}$ if e^+ and e^- are assigned to different neighborhood sets. In that case, both edges are counted separately because $\hat{f}_{\text{PMF}}$ is modular across neighborhood sets, and the underlying undirected edge contributes doubly to the cost. Does this affect the approximation? At least the cost of any cut remains unaffected, because for any (s,t) cut in the dual, the back-edges (direction t to s) across the cut must be void in the primal flow solution [32], as we mentioned above. Thus, only one of e^+, e^- belongs to a blocking (tight) set, and only those tight edges are counted in the cut. Therefore, the cost of the directed edges is still equivalent to the cost of undirected edges in Equation (6), at least on all cuts.

For fixed s, t, let the set C^* be the optimal directed (s,t) cut. Let further, for any cut C, $\Delta_s(C) \subset V$ be the set of nodes adjacent to C on the s side, and $\Delta_t(C) \subset V$ its analogue on the t side.

Lemma 2. *Let C_{PMF} be the cut returned by Algorithm PMF. Then*

$$f(C_{PMF}) \leq \min\left\{\, |\Delta_s(C^*)|,\ |\Delta_t(C^*)| \,\right\} f(C^*).$$

Proof. We will use Lemma 1. By subadditivity and nonnegativity of f, we know that $\sum_i f(A_i) \geq f(\bigcup_i A_i)$ for any collection of disjoint sets $\{A_i\}_i$, and thus $f(A) \leq \hat{f}_{\text{PMF}}(A)$ for any $A \subseteq E$. Let δv denote the set of edges adjacent to node v. To bound $\hat{f}_{\text{PMF}}(C^*)$, we use the convolution:

$$\begin{aligned}
\hat{f}_{\text{PMF}}(C^*) &= (f^{\text{in}} * f^{\text{out}})(C^*) \\
&\leq \min\{f^{\text{in}}(C^*), f^{\text{out}}(C^*)\} \qquad (7)\\
&\leq \min\Big\{ \sum_{v \in \Delta_s(C^*)} f(C^* \cap \delta v),\ \sum_{v \in \Delta_t(C^*)} f(C^* \cap \delta v) \Big\} \\
&\leq \min\Big\{ |\Delta_s(C^*)| \max_{v \in \Delta_s(C^*)} f(C^* \cap \delta v),\ |\Delta_t(C^*)| \max_{v \in \Delta_t(C^*)} f(C^* \cap \delta v) \Big\} \\
&\leq \min\left\{\, |\Delta_s(C^*)|,\ |\Delta_t(C^*)| \,\right\} f(C^*). \qquad (8)
\end{aligned}$$

Relation (7) follows from the definition of the convolution, and (8) from monotonicity of f. For more generality, we can bound $\min\{|\Delta_s|, |\Delta_t|\} \leq n/2$. □

If we know that $f(C_{\text{PMF}})/\hat{f}((C_{\text{PMF}}) = n^{-\beta}$, then we get a more specific ratio $f(C_{\text{PMF}})/f(C^*) \leq n^{1-\beta}/2$. On dense graphs where $m^{1/2} \log m > n$, the approximation factor for Algorithm PMF is better than the one for Algorithm EA.

3.5.1 Aside: enforcing a particular edge assignment $\Pi(E)$ in PMF

If, for some reason, we do not want to use the convolution in $\hat{f}_{\text{PMF}}$, but one specific partition $\Pi(E)$ of the edge set, then we can still solve the cut problem as a polymatroidal network flow. To do so, we use that a high modular function $h(A) = \beta|A| > mf(A)$, for all $A \subseteq E$, does not affect a convolution: $(h * f)(A) = f(A)$ for all A. Here, we must fix the assignment of all edges beforehand. Let E_i^{in} be the incoming edges assigned to v_i, and E_i^{out} the outgoing edges assigned to v_i. These sets form a partition of E. Then we set $f_i^{\text{in}}(A) = f(A \cap E_i^{\text{in}}) + h((A \cap \delta v_i) \setminus E_i^{\text{in}})$, and analogously for f_i^{out}. Then $f^{\text{in}}(A) = \sum_i f(A \cap E_i^{\text{in}}) + h((A \cap \delta v_i) \setminus E_i^{\text{in}})$, and equivalently for f^{out}, that is, in $f^{\text{in}} * f^{\text{out}}$ each edge can either count in h or in f, and by the definition of h, it is always better to assign an edge to its set under $\Pi(E)$:

$$\begin{aligned}
(f^{\text{in}} * f^{\text{out}})(A) &= \min_{A^{\text{in}} \subseteq A, A^{\text{out}} = A \setminus A^{\text{in}}} \sum_i f(A^{\text{in}} \cap E_i^{\text{in}}) + h((A^{\text{in}} \cap \delta v_i) \setminus E_i^{\text{in}}) \\
&\qquad\qquad + \sum_i f(A^{\text{out}} \cap E_i^{\text{out}}) + h((A^{\text{out}} \cap \delta v_i) \setminus E_i^{\text{out}}) \\
&= \min_{A^{\text{in}} \subseteq A, A^{\text{out}} = A \setminus A^{\text{in}}} \Big(\sum_i f(A^{\text{in}} \cap E_i^{\text{in}}) + f(A^{\text{out}} \cap E_i^{\text{out}}) \Big) + h(A^{\text{in}} \setminus E^{\text{in}}) + h(A^{\text{out}} \setminus E^{\text{out}}) \\
&= \sum_i f(A \cap E_i^{\text{in}}) + f(A \cap E_i^{\text{out}}).
\end{aligned}$$

The last equality follows from h being much larger than f on any set, so the minimum is achieved for $A_i^{\text{in}} = A \cap E_i^{\text{in}}$ and $A_i^{\text{out}} = A \cap E_i^{\text{out}}$. To unify A_i^{in} and A_i^{out} for cuts, we use the previous argument that a cut only contains outgoing or incoming edges for a particular node, but never both.

3.6 Algorithm II: Modular minimum cut basis with reference-based improvements (MBI)

In this section, we use the modular approximation $\hat{f}_{\text{MBI}}(A) = \sum_{e \in A} f(e) \geq f(A)$. The minimum cut for $\hat{f}_{\text{MBI}}$ is simply the common modular cost min-cut (MC baseline in experiments). For a wider range of candidate solutions that is still good with respect to $\hat{f}_{\text{MBI}}$, we construct the minimum cut basis for the graph with weights $w(e) = \hat{f}_{\text{MBI}}(e) = f(e)$. The cuts of a graph form a vector space over $\mathbb{F}_2$, and the minimum weight basis for this space can be found by a minimum cut tree [7]. This Gomory-Hu tree is computable by solving $O(n)$ min-cut problems [20]. The corresponding cut basis contains a minimum cut with respect to $\hat{f}_{\text{MBI}}$ for any pair of vertices in the graph. Of the $n-1$ basis cuts, we pick the one with the minimum submodular f-cost (MB baseline in experiments).

Among the basis cuts is the minimum cut C_{M} with respect to $\hat{f}_{\text{MBI}}$ [7, 20] with the following guarantee, using, as above, $\rho_e(A) = f(A \cup \{e\}) - f(A)$.

Lemma 3. *Let* $e' = \operatorname{argmax}_{e \in C^*} f(e)$. *Then*

$$f(C_M) \leq \frac{\sum_{e \in C^*} f(e)}{f(e') + \sum_{e \in C^* \setminus e'} \rho_e(C^* \setminus e)} f(C^*) \leq \frac{|C^*|}{1 + (|C^*| - 1)\gamma(C^*)} f(C^*)$$

for $\gamma(C^*) = \min_{e \in C^*} \rho_e(C^* \setminus e)/f(e')$.

The first bound shows how the quality of C_{M} depends on the degree of subadditivity of f, that is, how much $\rho_e(A)$ differs from $f(e)$ for any A that does not contain e. A modular function always satisfies $\rho_e(A \setminus e) = f(e)$, leading to an approximation factor of one. If $\rho_e(C^* \setminus e)$ is zero for many edges e, that is, $\gamma(C^*) = 0$, then the denominator is much smaller than the numerator. Still, the fraction is never larger than $|C^*|$.

Proof. Thanks to the subadditivity of f and the optimality of C_{M} for $\hat{f}_{\text{MBI}}$, it holds that

$$f(C_{\text{M}}) \;\leq\; \sum_{e \in C_{\text{M}}} f(e) = \hat{f}_{\text{MBI}}(C_{\text{M}}) \;\leq\; \hat{f}_{\text{MBI}}(C^*) \;\leq\; |C^*| f(e'). \tag{9}$$

The last relation again follows from the subadditivity of f. To reach at an approximation factor, we lower bound $f(C^*)$:

$$\begin{aligned} f(C^*) &\geq f(e') + \sum_{e \in C^* \setminus \{e'\}} \rho_e(C^* \setminus \{e\}) && (10) \\ &\geq f(e') + (|C^*| - 1) \min_{e \in C^* \setminus \{e'\}} \rho_e(C^* \setminus \{e\}) \\ &\geq f(e') + (|C^*| - 1) \min_{e \in C^*} \rho_e(C^* \setminus \{e\}). && (11) \end{aligned}$$

The first bound in Lemma 3 follows from dividing $\hat{f}_{\text{MBI}}(C^*)$ from (9) by (10). Dividing (9) by (11) (and dividing both by $f(e')$) yields the looser bound. □

To improve on the set of basis cuts for $\hat{f}_{\text{MBI}}$, we use each basis cut C_j as the reference for the bounds $h_1(A, C_j)$ and $h_2(A, C_j)$ defined in Equations (4) and (5), respectively. That is, we call the iterative bound minimization, Algorithm 1 in Section 3.4 for $\mathcal{I} = \{\emptyset, C_1, \ldots, C_n\}$ (including $\emptyset$ ensures that C_{M} is included in the search, it will be the first cut found. The better one of the minimizers of $h_1(A, C_j)$ and $h_2(A, C_j)$ is the next reference set until there are no more improvements. The algorithm usually stops after few steps. Both computing the basis and minimizing the bounds h_ℓ only means that we solve standard modular min-cut problems for which very efficient algorithms (and implementations) exist.

If we want an (s, t) cut, we remove s and t from the graph and compute a cut basis $\{C_1, \ldots, C_{n-2}\}$ for the remaining graph. Finally, we set $\mathcal{I} = \{\emptyset, C_1, \ldots, C_{n-2}\}$ for the full graph and use the iterative bound minimization.

A bit of reflection can explain how the basis and bound heuristics can work. Any edge in the graph is contained in at least one cut in the minimum cut basis. Hence, any edge, in particular any edge in the optimal C^*, occurs in at least one reference set C_j. Furthermore, if $|C^*| \gg n$, then there are basis cuts that include more than one edge from C^*, and potentially many. The more edges $|C^* \cap C_j|$ contains, the more likely is $h_1(B, C_j)$ to reveal critical cooperations of edges within C^* that identify the complete set C^*. The experiments in Section 4 show that this approximation works well in practice, but can reach the upper bound in Lemma 3 for the reasons mentioned in Section 3.4. In general though, using an entire basis instead of one min-cut, and minimizing the bounds h_ℓ in addition, does improve the solution, as the experiments show.

3.7 Algorithm III: Ellipsoid-based approximation of the cost function (EA)

Goemans et al. [18] present an approximation $\hat{f}_{\text{EA}}$ of a submodular function f by using the relation $f(A) = \max_{y \in P_f} y \cdot \chi_A$ (χ_A is the characteristic vector of A, and P_f the submodular polyhedron of f). They approximate P_f by an ellipsoid $\mathcal{E}$ and set $\hat{f}_{\text{EA}}(A) = \max_{y' \in \mathcal{E}} y' \cdot \chi_A$. As a result, the approximating function $\hat{f}_{\text{EA}}$ is the square root of a modular function, i.e., of the form $\hat{f}_{\text{EA}}(A) = \sqrt{\sum_{e \in A} w(e)}$. Since the minimizer of $\hat{f}_{\text{EA}}$ is the same as that of $\hat{f}^2_{\text{EA}}$, we set the weight of each edge to $w(e)$ and then solve a traditional min-cut (or (s,t)-cut) with edge cost function $\hat{f}^2_{\text{EA}}(A) = \sum_{e \in A} w(e)$. This problem can be efficiently and exactly solved.

Computing $\mathcal{E}$ is easier for matroid rank functions than for general monotone submodular functions, which require an additional approximation. For general non-monotone submodular functions, [18] only show a lower bound. In essence, [18] give an approximation guarantee for their functions of $f(A) \leq \hat{f}_{\text{EA}}(A) \leq \alpha f(A)$[7], with $\alpha = \sqrt{m+1}$ for a matroid rank function and $O(\sqrt{m}\log m)$ for a general polymatroid rank function. We add that for an integer-valued polymatroid rank function whose maximum cost of a single element is bounded (i.e., $\max_{e \in E} f(e) \leq c < \infty$), we can replace the logarithmic factor by a constant: $\alpha = O(\sqrt{cm})$ instead of $O(\sqrt{m}\log m)$. To do so, we approximate the matroid expansion of the polymatroid (the construction of the expansion is described in [34, Section 10.3]) to achieve the bound for a matroid function.

With Lemma 1, these α immediately yield approximation factors for Algorithm EA:

Proposition 1. *Let $C_{EA} = \operatorname{argmin}_{A \in \mathcal{C}} \hat{f}_{EA}$. Then*

$$f(C_{EA}) \leq \alpha f(C^*),$$

where $\alpha = O(\sqrt{m})$ for an integer-valued polymatroid rank function, and $\alpha = O(\sqrt{m}\log m)$ for an arbitrary monotone submodular function.

For planar graphs, where the number of edges is $O(n)$, the approximation factor becomes $\alpha = O(\sqrt{n})$ or $\alpha = O(\sqrt{n}\log n)$. Note that the graph we used in the proof of Theorem 2 is planar. Therefore, for planar graphs and matroid rank functions, the above procedure achieves a lower/upper bound gap of $\Omega(n^{1/3})$ versus $O(n^{1/2})$.

3.8 Algorithm IV: Convex relaxation (CR)

A common technique to construct approximation algorithms for submodular function optimization is to view the function f as a function from the indicator vectors $\{0,1\}^E$ to $\mathbb{R}$, and then extend it to a convex function $\tilde{f}$: $[0,1]^E \to \mathbb{R}$, the *Lovász extension* [33]. This extension leads to a convex relaxation of the edge-submodular (s,t) cut problem as the following constrained convex optimization problem.

$$\text{(P1)} \qquad \begin{aligned} \min \;& \tilde{f}(x) && (12)\\ \text{s. t.} \;& \pi(v_i) - \pi(v_j) + x(e) \geq 0 \quad \text{for all } e = (v_i, v_j) \in E && (13)\\ & \pi(s) = 0 \\ & \pi(t) = 1 \\ & \pi \in [0,1]^n, \quad x \in [0,1]^E \end{aligned}$$

Problem (P1) is a variation of the Linear Program (LP) for the standard, modular-cost (s,t) cut (e.g., [39, Section 6]). The graph partition is defined by node labels $\pi \in \{0,1\}^V$ indicating the two resulting parts and the cut indicator vector $x \in \{0,1\}^E$. Constraint (13) ensures that $x(e) = 1$ whenever the two incident nodes have increasing labels, i.e., $\pi(v_j) > \pi(v_i)$, and can be expressed via the adjacency matrix $\mathbf{A} \in \{-1,0,1\}^{|E| \times |V|}$. To make the problem easier, the integrality constraints on π and x were relaxed in (P1). For the standard (s,t) cut LP with cost f_{trad}, unimodularity of $\mathbf{A}$ always guarantees an integral optimal solution. This guarantee, however, is lost for problem (P1), thanks to the nonlinearity of $\tilde{f}$. As a result, relaxing the integrality constraints does usually lead to a non-integral solution.

[7]They use $\hat{f} \leq f \leq \alpha' \hat{f}$, so simply divide their $\hat{f}$ by α'.

For an integral approximate solution $\hat{x}^*$, we choose a threshold θ and round x^* to one if $x^*(e) \geq \theta^{-1}$, and to zero otherwise. We select θ^{-1} to be the largest value such that $\hat{x}^*$ is the indicator vector of a cut – the approximate solution.

Any algorithm for non-smooth constrained optimization problems solves Problem (P1). Since subgradients for $\tilde{f}$ are known [13, Lemma 6.19], a subgradient method is applicable, too. On a different problem involving submodular costs, Chudak and Nagano [8], for instance, use [36] for an approximate solution with adaptable precision.

Owing to the similarity with the standard cut LP, the dual of (P1) is a modified flow problem, where capacities are submodular over sets of edges. This problem, once again, is difficult because of global submodular cooperations. Restricting the cooperations leads to a problem similar to PF.

Let $\hat{C} = \{e \in E \mid x^*(e) \geq \theta^{-1}\}$. To ensure that the final solution is a *minimal* cut, we truncate $\hat{C}$ to a minimal cut with the following procedure. This procedure will return $\hat{C}$ if $\hat{C}$ is already minimal. Define edge weights $w(e) = f(e)$ if $e \in \hat{C}$, and $w(e) = \infty$ otherwise. Then find the min-cut for the modular cost w. The solution C_{CR} will be a subset of $\hat{C}$, and thus $f(C_{\text{CR}}) \leq f(\hat{C})$.

With a slight modification, Algorithm CR can also find a (non-minimal) cut that minimizes a non-monotone function. In that case, we replace the cost f in (P1) by the Lovász extension of the cost

$$f^m(A) = \min_{A \subseteq B \subseteq E} f(B),$$

which is monotone and submodular (e.g., [13, Section 3.1]). If f is already monotone, then $f^m = f$. Otherwise, the computation of f^m involves submodular function minimization, which is polynomial but still time-consuming. For the integral solution, we round x^* to $\hat{C}$ and then use $\hat{C}^m = \operatorname{argmin}_{\hat{C} \subseteq B \subseteq E} f(B)$. This is a set for which $f^m(\hat{C}) = f(\hat{C}^m)$, that is, the corresponding approximate solution for cost f. A truncation of $\hat{C}^m$ can be done by setting $w(e) = \infty$ for $e \notin \hat{C}$ and using f^m on the other edges. The resulting combined function is then used in Algorithm PMF, but we do not give any approximation guarantees here other than the fact that the truncation is a subset of $\hat{A}$.

For the truncation C_{CR} of $\hat{C}$ (minimal) for a monotone cost function, and for $C_{\text{CR}} = \hat{C}^m$ (non-minimal) for a non-monotone cost function, we can give the following approximation guarantee for an (s,t) cut. For a Coop min-cut, we use the tightest (s,t) bound across the optimal cut; $n-1$ always holds.

Lemma 4. *Let $P_{\max}$ be the longest simple (s,t) path. Then*

$$f(C_{CR}) \leq |P_{\max}| f(C^*) \leq (n-1) f(C^*).$$

Proof. To analyze the approximation factor for the rounded solution, we re-write the problem as an instance of hitting set extended to have submodular costs: a cut is a set of edges $A \subseteq E$ that hits (cuts) each (s,t) path $P \subseteq E$, that is, $|A \cap P| \geq 1$ for any (s,t) path P. The corresponding mathematical program is

$$\text{(P2)} \qquad \begin{aligned} \min\ & \tilde{f}(x) && (14)\\ \text{s. t. } & \sum_{e \in P} x(e) \geq 1 \quad \text{for all } (s,t) \text{ paths } P && (15)\\ & x \in \{0,1\}^E, \end{aligned}$$

where $x = \chi_C$ is the indicator vector of a cut C. The constraints in Problem (P1) summarize the possibly exponentially many constraints in Problem (P2) via the node labels. Thus, (P1) is the equivalent of (P2), with relaxed integrality constraints. Constraint (15) shows that in the worst case, the inverse rounding threshold θ is the length $|P_{\max}| \leq (n-1)$ of the longest path $P_{\max}$ between s and t: if the mass of solution x of the relaxed problem is distributed uniformly along the longest path, then only $\theta^{-1} \leq |P_{\max}|^{-1}$ will make the rounded solution hit $P_{\max}$. As a result, we infer $\theta \leq |P_{\max}|$. Problem (P2) is related to the set cover problems in [25], with similar approximation guarantees.

We will prove Lemma 4 for f^m, since $f^m = f$ for a monotone f. For the further analysis, it is important that $f^m(A) \leq f(A)$ for any $A \subseteq E$, and likewise for the respective Lovász extensions. This relation implies that

$$\tilde{f}^m(x^*) \ \leq\ \tilde{f}(x^*) \ \leq\ \tilde{f}(\chi_A) \ =\ f(A) \tag{16}$$

for any set $A \subseteq E$ and its characteristic vector χ_A, in particular for C^*. Thanks to the rounding procedure, the characteristic vector $\chi_{\hat{C}}$ of the rounded solution $\hat{C}$ satisfies $\chi_{\hat{C}} \leq \theta x^*$. Let furthermore $\hat{C}^m = \operatorname{argmin}_{\hat{C} \subseteq B \subseteq E} f(B)$

be a set for which $f^m(\hat{C}) = f(\hat{C}^m)$, that is, the corresponding approximate solution for cost f. For a monotone function, we simply have $\hat{C}^m = \hat{C}$. Since the Lovász extension $\tilde{f}^m$ of f^m is positively homogeneous by construction and monotone like its base f^m, it holds that

$$f(\hat{C}^m) = f^m(\hat{C}) = \tilde{f}^m(\chi_{\hat{C}}) \leq \tilde{f}^m(\theta x^*) = \theta \tilde{f}^m(x^*) \leq \theta \tilde{f}^m(x^*) \leq f(C^*).$$

For the last relation, we used Equation (16). Altogether, this derivation implies an approximation factor of $\theta \leq |P_{\max}| \leq n-1$. For a monotone function, the truncated solution C_{CR} has lower cost than its superset $\hat{C}$ because of the monotonicity. For a general cooperative min-cut solved by repeated (s,t) cuts, the approximation factor depends on the actual strategy to find the cut, but can become as low as the minimum $|P_{\max}|$ among all s, t that are separated by the optimal cut (if we test all pairs). $\square$

3.9 Algorithm V: Greedy minimization (G)

The formulation (P2) motivates an efficient, greedy variation of augmenting paths to solve the cut problem, even though without any guarantees. We maintain the current "cut" $B \subseteq E$ and, in each iteration, greedily choose an edge that cuts another as yet uncut (s,t) path. In other words, the algorithm satisfies one more violated constraint of type (15) in each iteration. Algorithm 2 shows the pseudocode. If the x-weight of the shortest path is $x(P) = \sum_{e \in P} x(e) \geq 1$, then at least one edge of P must be in B, that means all paths are cut (hit). Otherwise, P corresponds to a violated constraint in (P2): it is not fully cut. In that case, the algorithm greedily chooses the edge in P that keeps the cost as low as possible, to satisfy the violated constraint at low cost.

This algorithm is obviously polynomial, since C_{V} grows in each iteration, and it can include at most m edges. An efficient implementation could maintain distance labels to efficiently find the shortest augmenting path by the number of edges, treating already selected edges (with weight one) as inadmissible, blocked or very costly, similar to the shortest augmenting paths algorithm for maximum flow (see e.g. [1, Section 7]).

Several greedy algorithms for set cover-type problems optimize the ratio between the cost and the elements covered. Here, however, counting the cut (covered) elements, that is, all paths through an edge, it is too time-consuming. In addition, Iwata and Nagano [25] show that greedily minimizing this ratio does not help for a submodular set cover, and set cover is very similar to the hitting set problem.

Algorithm 2: Greedy Augmenting Paths

```
Input: $G = (V, E)$; nonnegative monotone cost function $f\colon 2^E \to \mathbb{R}_0^+$; source / sink nodes $s, t$
Output: cut $C_{\text{V}} \subseteq E$
Initialize $C_{\text{V}} = \emptyset$, edge weights $\chi_{C_{\text{V}}} = \mathbf{0}$, $\epsilon \leq (2n)^{-1}$;
Find shortest $(s,t)$ path $P \subseteq E$ with respect to cost $w(P) \triangleq \sum_{e \in P} \chi_{C_{\text{V}}}(e) + \epsilon|P|$;
while $\chi_{C_V}(P) < 1$ do
    Let $e^* = \text{argmin}_{e \in P} f(C_{\text{V}} \cup \{e\})$;
    Set $C_{\text{V}} = C_{\text{V}} \cup \{e^*\}$;
    Find shortest $(s,t)$ path $P \subseteq E$ with respect to cost $w(P)$;
end
If $C_{\text{V}}$ is not minimal, reduce it to a minimal cut with Algorithm PMF or MBI.
```

The final reduction step is the same as for Algorithm CR. Overall, this algorithm is very fast. It considers submodularity in the greedy choice of the edge to join C_{V}, the edge with the lowest cost with respect to the current reference set C_{V}.

4 Experiments

After having introduced a number of algorithms and analyzed their theoretical properties, we compare them empirically on a range of cost functions. In addition, we illustrate the algorithms' behavior on difficult examples that are specifically designed to test the limits of algorithms based on modular approximations and of an entirely node-based algorithm that ignores the graph structure.

Overall, Algorithms MBI, CR and the greedy algorithm perform well on the synthetic problems in Section 4.1. Which algorithm is best depends on the cost function and graph at hand, though. In general, the empirical approximation factors are far below the theoretical factors in our experiments – except for the worst-case limits in Section 4.2.

Algorithms I-V	
PMF	Algorithm I, approximation via **poly****m**atroidal network **f**lows
MBI	Algorithm II, **m**inimum cut **b**asis for a modular approximation with reference-based **i**mprovement step
EA	Algorithm III, **e**llipsoid-based function **a**pproximation
CRI	Algorithm IV, **c**onvex **r**elaxation with post-processing by Algorithm PMF
CRII	Algorithm IV, **c**onvex **r**elaxation with post-processing by Algorithm MBI
GI	Algorithm V, **g**reedy algorithm with post-processing by Algorithm PMF
GII	Algorithm V, **g**reedy algorithm with post-processing by Algorithm MBI
comparison methods	
QU	**Qu**eyranne's algorithm for minimizing symmetric submodular functions
MC	**M**in-**c**ut for a modular approximation
MB	**M**inimum cut **b**asis for a modular approximation
RBI	**R**andom cut **b**asis (random spanning tree) with reference-based **i**mprovement step

Table 4: Acronyms for the algorithms used in the experiments.

In addition to Algorithms I to V, we run four baseline methods. First, Queyranne's algorithm (QU) [40] minimizes symmetric submodular functions without constraints in $O(n^3)$ time. We use it to approximately solve the unconstrained problem $\min_{X \subseteq V} g(X)$ for $g(X) = f(\delta X)$. Since $f(\delta X)$ as a function on subsets of nodes is not submodular and at most subadditive, this algorithm cannot give any guarantees (as we demonstrate in Section 4.2). It completely ignores the structure of the graph. Nevertheless, it often does find a good solution.

The remaining baseline methods provide a comparison to investigate the effect of the ingredients of Algorithm MBI:

1. using the extended search space of a cut basis
2. using the *minimum* cut basis
3. using the iterative bound minimization.

Baseline 2 (MC) addresses Ingredient 1 and 3: it computes a single min-cut solution for the modular approximation $\hat{f}_{\text{MBI}}(A) = \sum_{e \in A} f(e)$ used in Algorithm MBI. Baseline 3 (MB) includes the special extended search space (Ingredient 2), but spares Ingredient 3. For Baselines 2 and 3, the upper bound of Lemma 3 still holds.

Finally, Baseline 4 (RBI) shows the effect of sparing Ingredient 2: it replaces the minimum cut basis in Algorithm MBI by a random one and otherwise proceeds identically to Algorithm MBI. A random cut basis can be derived from a spanning tree: each basis cut corresponds to the partition induced by cutting one edge in the tree. RBI does not necessarily include the modular min-cut, and hence does not come with any of the above guarantees.

Reducing the cut to a minimal cut in Algorithm CR and the greedy method can be done either by polymatroidal flows (PMF) or the modular cut basis approach (MBI). We include both variations in the experiments, but the results do not differ significantly. In most cases, very little reduction was necessary.

For reference, all algorithms and their acronyms are listed in Table 4.

4.1 Synthetic graphs and a range of cost function types

First, we test a range of cost functions on two types of synthetic graphs.

Grid graphs. The grid graphs are regular graphs with node degree four or six. Type I is a plane grid with horizontal and vertical edges displayed as solid edges in Figure 6. Type II is similar, but has additional diagonal edges (dashed in Figure 6). Type III is a cube with plane square grids on four faces (sparing the top and bottom faces). Different from Type I, the nodes in the top row are connected to their counterparts on the opposite side of the cube. The connections of the bottom nodes are analogous.

Clustered graphs. The clustered graphs consist of a number of cliques that are connected to each other by few edges, as depicted in Figure 6 on the right.

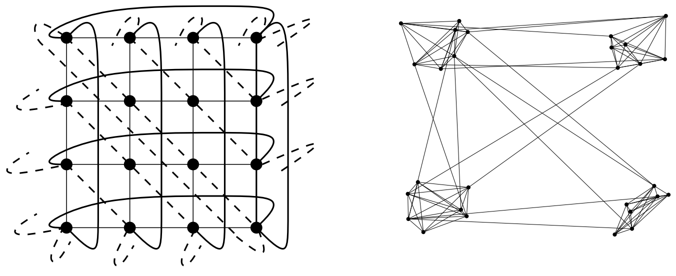


Figure 6: Examples of our test graphs. The grid (left) was used with and without diagonal edges, as indicated by dashed lines, and also with a variation of the connections in the first and last row. The clustered graphs were similar to the example shown here (right).

We used the three grid types (25-32 nodes) and five clustered graphs (30 nodes, 90 edges). For each graph, we generated five to ten instances of each cost function. The cost functions are listed in Table 5. To estimate the approximation factor on one problem instance (one graph and one cost function), we divide by the cost of the best solution found by any of the eleven algorithms, unless the optimal solution is known (bestcut I and II).

All algorithms were implemented in Matlab, with the help of a graph cut toolbox [2, 5], and a toolbox for submodular function optimization [28].

Figure 7 shows the empirical approximation factors for the cost functions in Table 5; Table 4 lists the acronyms for the algorithms. None of the algorithms actually reaches its theoretical upper bound. Neither is there a clear winner: no algorithm performs significantly better than all others on all cost functions and graphs.

Despite its variable worst-case bound, Algorithm MBI provides good solutions throughout all cost functions, followed by the convex relaxation (Algorithm CR), which is only worse than the others for the discounted price functions. The reason for this weakness is that the latter functions implement a *global* interaction between edges, so it is more likely that several edges along one path cooperate and the algorithm distributes the x-weight between all those edges, and the rounding is not very selective. For more restricted cooperation within limited groups like in the other cost functions, it is more likely that x^* is closer to being integral.

On most functions, MBI profits from the range of candidates in the minimum cut basis together with the iterative bound minimization and is better than MC or MB – even though the added heuristics cannot change the theoretical worst-case bound, as we will demonstrate in Section 4.2. The iterative bound minimization helps a lot on the "bestcut" functions, where $|C^*|$ is rather large and its optimality heavily depends on edge cooperations (so that the modular min-cut MC is not a very good solution). In addition, the cooperations are such that having one edge of C^* in the cut basis is enough to identify the reduction in cost for the edges in C^*, and therewith find C^* in the iterative step. The minimality of the cut basis, on the contrary, affects the solution much less than the iterative minimizations: RBI and MBI achieve almost the same quality of solutions; only for rank-like functions, the quality of the RBI solution varies more.

The greedy heuristic (Algorithm V) performs surprisingly well, with the exception of the rank-like functions and the difficult truncated rank functions. On the "bestcut" functions, it profits in a similar way as MBI: C^* can be identified by a small intersection of the current (partial) cut B with C^*. For the "truncated rank" functions, however, the cost-reducing cooperation of edges only becomes obvious if more than four to five edges of R are in the reference set – much less likely for the greedy choice in Algorithm V, where the reference set is the current "cut" set, than for a principled covering of all edges by a cut basis as in Algorithm MBI. In summary, if edge interactions can be identified from small sets, then the greedy approach has good chances to find a good solution; and it is simple and fast. The type of post-processing for truncation in the greedy algorithm and CR does not matter in these experiments, probably because the rounding or greedy selection mostly lead to an almost minimal or scattered solution already. In consequence, PMF might have received an almost modular cost function and thus returned the same result as MBI.

Despite its lack of guarantees, Queyranne's algorithm does find good solutions for most instances. It has the greatest variance for the discounted price function II on the clustered graphs.

The ellipsoid-based approximation of Algorithm EA works best for the discounted price functions, probably because its approximation models a *global* cooperation, but not specific local ones. For the second of the discounted functions, this result is expected, since the approximation has exactly the form of the cost function. On

name	description
matrix rank I	$f_{\text{mrI}}(A) = \text{rank}(\mathbf{X}_A)$. Each element $e \in E$ is an index to a column in matrix $\mathbf{X}$. The cost is the rank of the sub-matrix $\mathbf{X}_A$ of the columns indexed by the $e \in A$. The matrix $\mathbf{X}$ is of the form $[\ \mathbf{I}'\ \ \mathbf{R}\ ]$, where $\mathbf{R} \in \{0,1\}^{d \times (m-d)}$ is a random binary matrix with $d = 0.9\sqrt{m}$, and $\mathbf{I}'$ is a column-wise permutation of the identity matrix.
matrix rank II	$f_{\text{mrII}}(A) = 0.33 \sum_{i=1}^{3} f_{\text{mrI}}^{(i)}(A)$ is the sum of three functions $f_{\text{mrI}}^{(i)}$ of type "matrix rank I" with different random $\mathbf{X}$.
labels I	$f_{\ell\text{I}}(A) = \lvert \bigcup_{e \in A} \ell(e) \rvert$. Each element e is assigned a random label $\ell(e)$ from $0.8\sqrt{m}$ possible ones. The cost counts the number of labels in A.
labels II	$f_{\ell\text{II}}(A) = 0.33 \sum_{i=1}^{3} f_{\ell\text{I}}^{(i)}(A)$ is the sum of three functions of type "labels I" with different random labels.
bestcut I	$f_{\text{bcI}}(A) = \mathbf{1}[\lvert A \cap \delta X^* \rvert \geq 1] + \sum_{e \in A \setminus \delta X} w(e)$. Here, we randomly pick a cut and make it the optimal one. This cut is usually very different from the cut with fewest edges. For the cut, randomly pick a connected subset $X^* \subseteq V$ of size $0.4n$. Set $f_1(A) = 1$ for all $A \subseteq \delta X^*$. This will be the best cut. The cost of the other edges ($B \cap \delta X^* = \emptyset$) is $f_2(B) = \sum_{e \in B} w(e)$ for random weights $w(e)$ between 1.5 and 2. The cost is the direct sum of f_1 and f_2. If there exists a different cut $C \neq \delta X^*$ with cost one or lower, correct w by increasing the weight of one $e \in C$ to two.
bestcut II	Similar to bestcut I, but with submodularity on all edges. Partition E into three sets, $E = (\delta X^*) \dot\cup B \dot\cup C$. Then $f_{\text{bcII}}(A) = \mathbf{1}[\lvert A \cap \delta X^* \rvert \geq 1] + \sum_{e \in A \cap (B \cup C)} w(e) + \max_{e \in A \cap B} w(e) + \max_{e \in A \cap C} w(e)$. The weights of two edges in B and two edges in C are set to larger than two (2.1,2.2). The optimum is again δX^*.
discounted price function I	$f_{\text{dpI}}(A) = \log \sum_{e \in A} w(e)$, where weights $w(e)$ are chosen randomly as follows. Sample an $X \subset V$ with $\lvert X \rvert = 0.4n$, and set $w(e) = 1.001$ for all $e \in \delta X$. Then randomly assign some "heavy" weights in $[n/2, n^2/4]$ to some edges not in δX, so that each node is incident to one or two heavy edges. The remaining edges get random (mostly integer) weights between 1.001 and $n^2/4 - n + 1$.
discounted price function II	$f_{\text{dpII}}(A) = \sqrt{\sum_{e \in A} w(e)}$ with weights assigned as for "discounted price function I".
truncated rank	This function is similar to the truncated rank in the proof of the lower bound. Sample a connected $X \subseteq V$ with $\lvert X \rvert = 0.3\lvert V \rvert$ and set $R = \delta X$. The cost is $f_{\text{tr}}(A) = \min\{\lvert A \cap \overline{R} \rvert + \min\{\lvert A \cap R \rvert, \beta\},\ \lambda\}$ for $\beta = \sqrt{\lvert R \rvert}$ and $\lambda = 2\lvert R \rvert$. Here, R is not necessarily the optimal cut.

Table 5: Cost functions for the experiments in Section 4.1. "Matrix rank I,II" and "labels I,II" are summarized as "rank-like" costs in the results. The indicator function is denoted by $\mathbf{1}[\cdot]$.

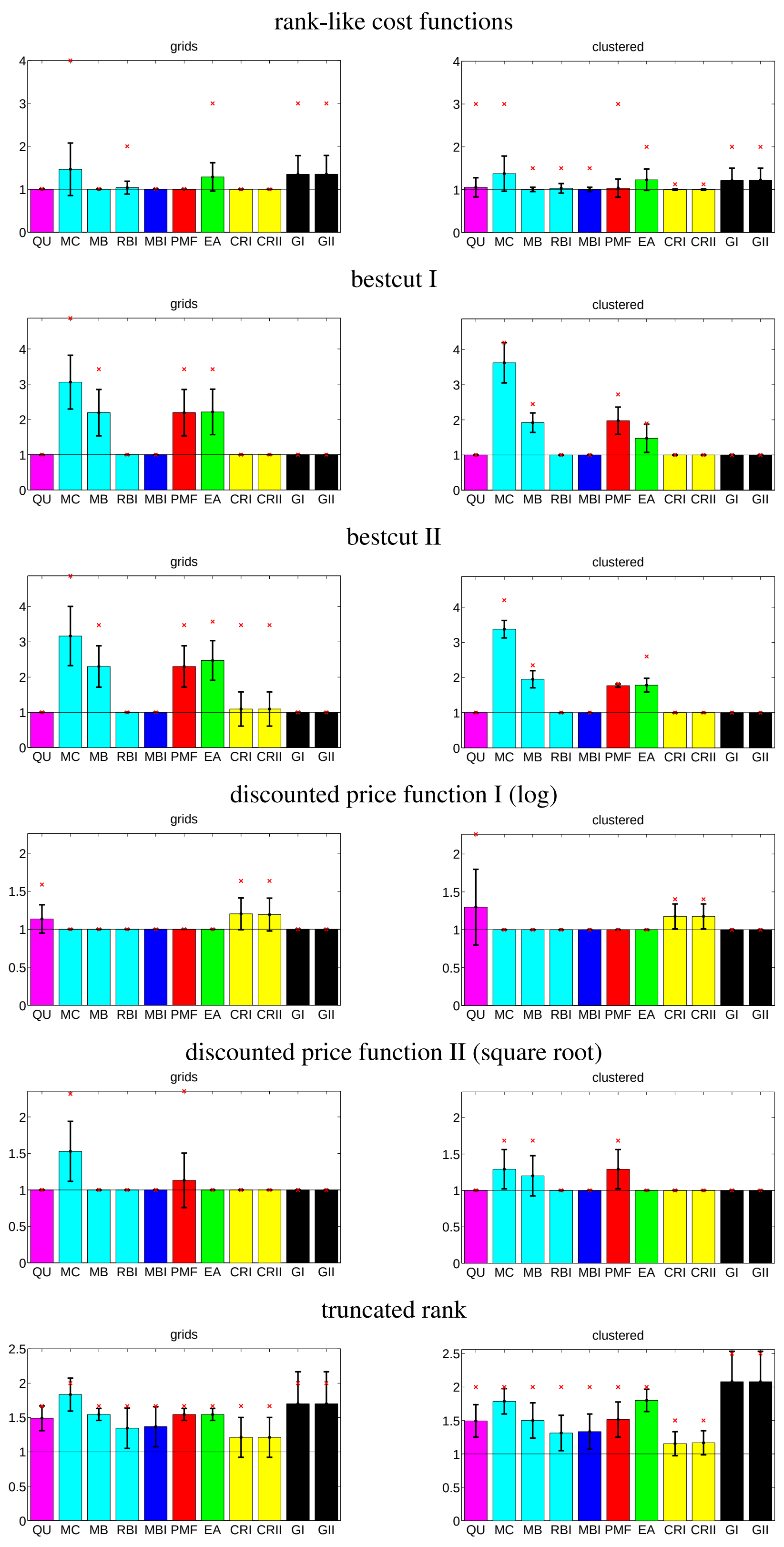


Figure 7: Results for the experiments of Section 4.1. The bars show the mean empirical approximation factors, the red crosses mark the maximum empirical approximation factor. Plots in the right column are for the grid graphs, plots in the left column for the clustered graphs.

the downside, this algorithm can take long to converge, thanks to the procedure to compute the ellipsoid. Then the greedy method or MBI are much more efficient.

Overall, the algorithms' performance on these synthetic examples is encouraging. Note that the truncated rank function is a difficult example and similar to the function in the proof of the lower bound. The experiments also demonstrate that knowing properties of the cost function at hand helps to choose an appropriate algorithm. After the positive examples in this section, the next section demonstrates the limits of the algorithms MC, MB, RBI, MBI and QU.

4.2 Worst-case examples

In this section, we explore problem instances that are specifically crafted to mislead a particular algorithm. As usual, n denotes the number of nodes. The graphs here are undirected.

4.2.1 Type I: difficult for purely modular approximations

The first example exploits the weakness of the modular approximation $\hat{f}_{\text{MBI}}(A) = \sum_{e \in A} f(e)$ to ignore the interaction of edges, that is, the reduction of an edge's cost if certain other edges are in A: $f(A) - f(A \setminus \{e\}) \ll f(e)$. In version I(a), the modular cost of the true optimum C^* is $n^2/4$ times higher than its submodular cost, that is, $\hat{f}_{\text{MBI}}(C^*) = f(C^*)n^2/4$. On the contrary, the cost of the min-cut C_{M} with respect to $\hat{f}_{\text{MBI}}$ is not much lower for f: $f(C_{\text{M}}) = \hat{f}_{\text{MBI}}(C_{\text{M}}) - n/2 + 1$. As a result, $\hat{f}_{\text{MBI}}$ is not a good estimate for the relative costs of C_{M} and C^* when measured with f. The graph, shown in Figure 8, is a clique with three types of edges, marked by different colors. Let E_k, E_b, and E_r be the set of black, blue and red edges, respectively. The cost f_{Ia} is the direct sum of the cost functions f_k, f_b, f_r on these sets, with

$$\begin{aligned} f_k(A) &= 1 && \forall\, A \subseteq E_k; \\ f_b(A) &= |A|n/2 && \forall\, A \subseteq E_b; \\ f_r(A) &= |A| \left(\frac{n}{2} - \frac{\epsilon}{n/2 - 1} \right) && \forall\, A \subseteq E_r \end{aligned}$$

for a small $\epsilon > 0$. The optimal cut is $C^* = E_k$, and relies on the only but strongly submodular part of the cost function, f_k. The optimal cut for $\hat{f}_{\text{MBI}}$ is to separate out node $v_{n/2+1}$, cutting all red edges and the black edges adjacent to $v_{n/2+1}$, with cost $f_{Ia}(\delta v_{n/2+1}) = n^2/4 - n/2 + 1 - \epsilon$. Thus, the approximation factor for this cut grows as $n^2/4$, which is the order of the theoretical worst-case bound.

Both the min-cut with weights $\hat{f}_{\text{MBI}}$ (MC) and the minimum cut basis (MB) return the cut $\delta v_{n/2+1}$. All other algorithms that take into account submodularity, Algorithm MBI via the iterative bound minimization, find the optimal solution.

Version (b) of the problem instance, with the same graph structure but a modified cost function, renders the iterative bound minimization ineffective via a truncation. The modified cost function is

$$f_{Ib}(A) = \min\{f_{Ia}(A) + \epsilon' |A \cap E_k|,\ \lambda\}$$

for a truncation threshold $\lambda = f_r(E_r) + f_b(E_b) - (n/2 - \epsilon(n/2-1)^{-1}) + 1$ and a small $\epsilon' > 0$. The truncation makes the weight $w_{1,C}(e)$ zero for all edges in the current cut C in the iterative minimization, but nonzero for all other edges, so C is the optimal cut as measured by h_1, and we never move away from C. Thus, Algorithm MBI will only find the optimal solution if it is in the cut basis. Since C^* has the maximum possible number of edges and weights are counted in a modular way for the basis, it is not in the minimum cut basis. The result is obvious in Figure 8(b): the advantage of the iterative minimization is gone and all algorithms using $\hat{f}_{\text{MBI}}$ find the quadratically worse second-best cut. In comparison, in Version (a), the weights $w_{1,C}(e)$ are zero for the black edges for any cut, since every cut must contain a black edge.

We remark that Instance I(b) is theoretically useful, but not realistic.

4.2.2 Type II: difficult for Queyranne's algorithm

The second group of examples, II(a)-(c), again misleads variations of Algorithm MBI, but II(d) finally demonstrates the benefit of theoretical approximation guarantees: there is no upper bound on the solution quality for Queyranne's algorithm, and QU can indeed perform arbitrarily bad, whereas the other algorithms are saved by their approximation factors.

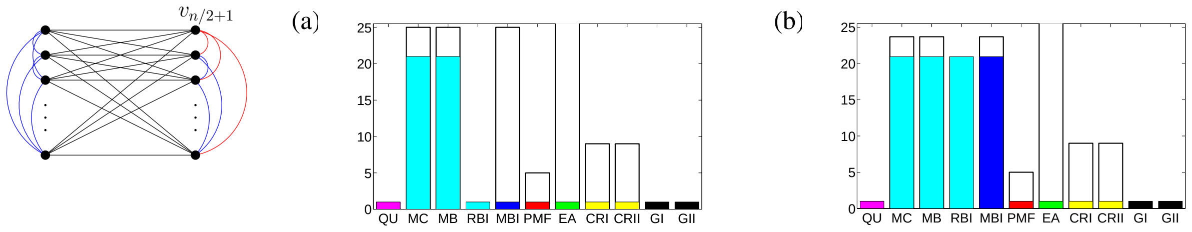


Figure 8: Graph I and approximation factors with $n = 10$ nodes, so $n^2/4 - n/2 + 1 = 21$. The white bars illustrate the theoretical approximation bound, where applicable.

The graph is again a clique, but its edges are partitioned into $n/2$ sets, as indicated by colors in Figure 9. The black set E_k is as in Graph I. The remaining sets are constructed node-wise as

$$E_i = \{(v_i, v_j) \in E \mid i < j \leq n/2\} \cup \{(v_{n/2+i}, v_j) \in E \mid n/2 + i < j \leq n\}$$

for each $1 \leq i < n/2$. In Figure 9, set E_1 is red, E_2 is blue, and so on. The cost function adds cost b for any set E_i intersecting the cut, and cost 1 if any black edge is in the cut:

$$f_{IIa}(A) = \mathbf{1}[|A \cap E_k| \geq 1] + \sum_{i=1}^{n/2-1} b \cdot \mathbf{1}[|A \cap E_i| \geq 1],$$

with $b = n/2$ for Versions (a) to (c). As before, $\mathbf{1}[\cdot]$ denotes the indicator function. The optimal solution is again $C^* = E_k$ with $f_{IIa}(C^*) = 1$. The results for the different algorithms are illustrated in Figure 9.

In Version (a), the iterative bound minimization comes to the rescue of the algorithms that use the modular approximation $\hat{f}_{\mathrm{II}}$. In Versions (b) and (c), this benefit vanishes thanks to two modifications: a truncation in (c) and the addition of a tiny modular cost in (d) render the iterative minimization in MBI and RBI inefficient. The cost function for II(b) is

$$f_{IIb}(A) = \min\{f_{IIa}(A),\ n\}.$$

and for II(c)

$$f_{IIc}(A) = f_{IIa}(A) + \epsilon |A \cap E_k|.$$

Finally, Version (d) tests the approximation factors. It uses f_{IIa}, but with a higher b. For any $b > n/2$, any solution other than C^* is more than $n^2/4 = |C^*| > n$ times worse than the optimal solution and the approximation guarantees come into play: all algorithms except for QU find the optimal solution. The result of the latter depends on how it chooses the minimizer of $f(B \cup \{e\}) - f(\{e\})$ in the search for a pendent pair; this quantity often has several minimizers here. Some of those will lead to a good solution and some to a bad one. Versions (a) to (c) show lucky cases. Version (d) is like (a), but uses a different sequence, that is, permuted node labels, and $b = n^2 = 100$. For the permutation in (d), QU will always return the same solution δv_1 with cost $b + 1$, no matter how large b is.

Algorithms PMF, EA, CR and the greedy algorithm perform well on the examples of this section. Yet, for fairness, one should keep in mind that the examples here were designed to be difficult for QU and the algorithms based on a purely modular approximation of the cost function (Algorithms MBI, MC, MB and RBI).

5 Conclusion and open problems

We have introduced *Cooperative cut*, an extension to the min-cut problem where the cost is measured by a submodular set function on subsets of edges. The richness of submodular functions covers a much wider range of applications than the standard modular cost, but at the same time makes the problem much harder to solve. In addition to lower bounds on the approximation factor, we present four approximation algorithms and some heuristics that rely on different techniques. We compare those methods theoretically and empirically.

An open question remains how to close the gap between the upper and lower bounds, ideally with particular attention given to the usefulness the algorithms in practice.

The work presented here points to general open questions:

- Many traditional cut problems can be generalized to submodular costs, in particular the ones listed in Tables 2 and 3. How hard are these problems, and are there efficient (approximation) algorithms?

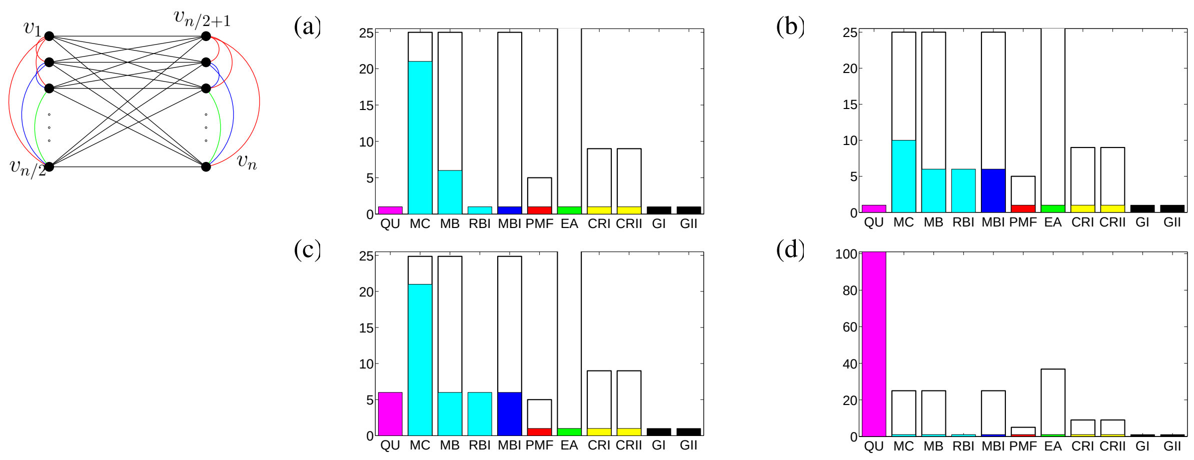


Figure 9: Graph II and empirical approximation factors with $n = 10$ nodes. White bars illustrate theoretical approximation bounds where applicable. In (a) and (b), cutting off v_1 costs $f(\delta v_1) = n/2 + 1 = 6$ and is the second-best cut. Cutting off v_n costs $f(\delta v_n) = n/2(n/2 - 1) + 1 = 21$, the worst cut that cuts off a single node. For (b), the maximum cost of a cut is $n = 10$. In (d), the second-best cut δv_1 has cost $b = 101 \gg \max\{|C^*|, n, \sqrt{m}\log m\}$.

- We show that the (s, t) cut problem becomes NP hard with submodular costs, with a lower bound of $\Omega(n^{1/3})$ for monotone, normalized submodular costs.The modular-cost minimum cut problem becomes NP hard if we allow negative weights, but can still be approximated within factors better than our lower bounds [15]. The "bottleneck labeled" min-cuts in [22] have, like CoopCut, a non-modular, non-separable cost function (which is though not submodular). The authors show (weak) NP hardness results and a lower bound of $\Omega(2)$ for two labels and the min-max version, and mention an inapproximability result for the max-min version with an arbitrary number of labels. These results lead to the question whether there is a more general relation between the cost function and the hardness of the cut problem.

  Our lower bound and those in [16, 22, 25] show how combinatorial problems become harder with non-modular, non-separable costs. An interesting open question is, in general, how does non-modularity affect the complexity of a general combinatorial problem with monotone costs — is there, for example, a general, widely-applicable quantifiable relation that characterizes the decisive properties of such a substitution? We leave this to future work.

Acknowledgments: We wish to thank Jens Vygen for the example of an intractable subadditive function, Richard Karp for the name "cooperative cut", and Andrew Guillory for his Gomory-Hu tree code.

Note: Removed previous Theorem 3 in July 2010.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows.* Prentice Hall, 1993.

[2] S. Bagon. Matlab wrapper for graph cut, December 2006. `http://www.wisdom.weizmann.ac.il/~bagon`.

[3] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1):89–113, 2004.

[4] J. Bilmes and C. Bartels. On triangulating dynamic graphical models. In *Uncertainty in Artificial Intelligence (UAI)*, pages 47–56, Acapulco, Mexico, 2003. Morgan Kaufmann Publishers.

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

[6] U. Brandes, D. Delling, M. Gartler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. *IEEE Trans. on Knowledge and Data Eng.*, 20(2):172–188, 2008.

[7] F. Bunke, H. W. Hamacher, F. Maffioli, and A. Schwahn. Minimum cut bases in undirected networks. *Discrete Applied Mathematics*, In Press, 2009.

[8] F. A. Chudak and K. Nagano. Efficient solutions to relaxations of combinatorial problems with submodular penalties via the lovász extension and non-smooth convex optimization. In *SODA*, pages 79–88, 2007.

[9] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiway cuts. In *STOC*, pages 241–251, 1992.

[10] E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2–3):172–187, 2006. Special Issue on Approximation and Online Algorithms.

[11] M. Deza and M. Laurent. *Geometry of Cuts and Metrics*, volume 15 of *Algorithms and Combinatorics*. Springer, 1997.

[12] J. Edmonds. *Combinatorial Structures and their Applications*, chapter Submodular functions, matroids and certain polyhedra, pages 69–87. Gordon and Breach, 1970.

[13] S. Fujishige. *Submodular Functions and Optimization*. Number 58 in Annals of Discrete Mathematics. Elsevier Science, 2nd edition, 2005.

[14] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[15] I. Giotis and V. Guruswami. Correlation clustering with a fixed number of clusters. *Theory of Computing*, 2:249–266, 2006.

[16] G. Goel, C. Karande, P. Tripati, and L. Wang. Approximability of combinatorial problems with multi-agent submodular cost functions. In *FOCS*, 2009.

[17] G. Goel, P. Tripathi, and L. Wang. Optimal approximation algorithms for multi-agent combinatorial problems with discounted price functions. *arXiv*, 2009.

[18] M. X. Goemans, N. J. A. Harvey, A. Iwata, and V. S. Mirrokni. Approximating submodular functions everywhere. In *SODA*, 2009.

[19] O. Goldschmidt and D. S. Hochbaum. Polynomial algorithm for the k-cut problem. In *FOCS*, pages 444–451, 1988.

[20] R. E. Gomory and T. Hu. Multi-terminal network flows. *Journal of the SIAM*, 9(4), 1961.

[21] M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Math. Programming*, 45:59–96, 1989.

[22] R. Hassin, J. Monnot, and D. Segev. The complexity of bottleneck labeled graph problems. *Algorithmica*, 2008.

[23] D. S. Hochbaum and A. Pathria. The bottleneck graph partitioning problem. *Networks*, 28(4):221–225, 1996.

[24] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[25] S. Iwata and K. Nagano. Submodular function minimization under covering constraints. In *FOCS*, 2009.

[26] S. Khot. Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. In *Proceedings of the 45th Annual IEEE Symposium on FOCS*, pages 136–145, 2004.

[27] B. Korte and J. Vygen. *Combinatorial Optimization - Theory and Algorithms*. Springer, 2008.

[28] A. Krause. Matlab toolbox for submodular function optimization, 2009. `http://www.cs.caltech.edu/~krausea/sfo/`.

[29] A. Krause and C. Guestrin. ICML tutorial: Beyond convexity: Submodularity in machine learning, 2008.

[30] A. Krause and C. Guestrin. IJCAI tutorial: Intelligent information gathering and submodular function optimization, 2009.

[31] A. Krause, P. Ravikumar, and J. Bilmes. NIPS workshop on discrete optimization in machine learning: Submodularity, sparsity and polyhedra, 2009.

[32] E. L. Lawler and C. U. Martel. Computing maximal "Polymatroidal" network flows. *Mathematics of Operations Research*, 7(3):334–347, 1982.

[33] L. Lovász. *Mathematical programming – The State of the Art*, chapter Submodular Functions and Convexity, pages 235–257. Springer, 1983.

[34] H. Narayanan. *Submodular Functions and Electrical Networks*. Elsevier Science, 1997.

[35] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular functions - i. *Math. Program.*, 14:265–294, 1978.

[36] Y. Nesterov. *Introductory Lectures on convex optimization: A basic course*. Kluwer Academic Publishers, 2004.

[37] S. Nowozin and S. Jegelka. Solution stability in linear programming relaxations: Graph partitioning and unsupervised learning. In *Proc. of the 26th Int. Conf. on Machine Leanring (ICML)*, 2009.

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

[39] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Dover Publications, 1998.

[40] M. Queyranne. Minimizing symmetric submodular functions. *Mathematical Programming*, 82:3–12, 1998.

[41] M. Queyranne. On optimum size-constrained set partitions. In *AUSSOIS*, 1999.

[42] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. In *Proc. 28th Workshop on Graph Theory (WG '02)*, pages 379–390, 2002.

[43] R. P. Stanley. *Enumerative Combinatorics*, volume I of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1997.

[44] Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. In *FOCS*, 2008.

[45] V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[46] J. Vondrák. Symmetry and approximability of submodular maximization problems. In *FOCS*, 2009.

[47] D. Wagner and F. Wagner. Between min cut and graph bisection. In *FOCS*, pages 744–750, 1993.

[48] P. Zhang, Cai J.-Y, L.-Q. Tang, and W.-B. Zhao. Approximation and hardness results for label cut and related problems. *Journal of Comb. Optim.*, 2009.

[49] L. Zhao, H. Nagamochi, and T. Ibaraki. Greedy splitting algorithms for approximating multiway partition. *Mathematical Programming*, 102(1):167–183, 2004.

A The (s, t) version of node-submodular Sparsest Cut and Min-Quotient Cut

We extend the algorithms by Svitkina and Fleischer [44] for node-submodular Sparsest Cut and Min-Quotient Cut in a straightforward way to an (s, t) version, where the solution $X^* \subset V$ must separate specified nodes s and t, i.e., $s \in X^*$, $t \notin X^*$. To solve the extension, we enforce s to be included in the selected set, and exclude t from the nodes to join the selection. The algorithms in [44] repeatedly sample a set S, and then find a set T^* by minimizing the submodular term $f(T) - \alpha h(S, T)$. With high probability, one of those sets T^* satisfies the approximation factor.

To enforce the inclusion of s, let $\overline{V} = V \setminus \{s, t\}$ and $\overline{f} : 2^{\overline{V}} \to \mathbb{R}_0^+$:

$$\overline{f}(X) = f(X \cup \{s\}).$$

We essentially run a slightly modified algorithm on $\overline{V}$ with cost $\overline{f}$, automatically including s into T and S. The set to sample from is $\overline{V}$.

Uniform Sparsest Cut/Min-Quotient Cut. We can modify the sub-problem of minimizing $f(T) - \alpha|T \cap S| + \alpha|T \cap (V \setminus S)|$ to include s, that is, subtract another α since s is forcefully included in both T and S. This constant will not change the optimizing T^*, though. Let $X^* \subseteq V$ be the optimal solution of the entire cut problem, with $s \in X^*$. The algorithm on $\overline{V}$ gives a cut T^* with cost

$$\begin{aligned} f(T^* \cup \{s\}) = \overline{f}(T^*) &\leq \sqrt{\frac{n-2}{\ln(n-2)}} f(X^*) \\ &\leq \sqrt{\frac{n}{\ln(n-2)}} f(X^*) \\ &\leq \sqrt{2\frac{n}{\ln n}} f(X^*) \end{aligned}$$

(for $n \geq 4$), that is, the approximation factor of $O(\sqrt{\frac{n}{\ln n}})$ still holds. The factor for Min-Quotient Cut is always within a factor of two of that for uniform Sparsest Cut [44].

Sparsest Cut. We again retain the algorithm in [44], but replace f by $\overline{f}$ and V by $\overline{V}$. After sampling, we find the minimizer T^* of $\overline{f}(T) - \alpha \sum_{v \in T \cup \{s\}} w(v)$, where the $w(v)$ are set during the algorithm. The proof in [44] uses the fact that the expected sum of weights $w(X^*)$ of the nodes in an optimal set X^* exceeds a certain threshold – this expectation only increases with the modification. The remainder of the proof remains analogous to the original one, and the factors change at most by a constant as above.

B Correlation Clustering as CoopCut

Correlation clustering[8] (CC) is a graph-partitioning problem where a graph's edges are marked with either $+$ or $-$ corresponding to the case where the adjacent nodes are "similar" or "different". Nodes connected by a $+$ edge should be in the same cluster, while nodes connected by a $-$ edge should be in separate clusters. The goal

[8]CC has been introduced into Machine Learning by [3], but variants of the problem have been considered in other communities before, see e.g. [21] and references therein. Variants are known as aggregation (of binary relations), consensus clustering and similar terms.

of CC is to partition the nodes (i.e., a cut) so that, to the extent possible, the partition agrees with the labels, that is, similarities exist within clusters and differences exist across clusters. Quality is measured in two ways: the partitioning should either 1) minimize the number of differences within the clusters (i.e., edges that are not cut) plus the number of similarities across clusters (edges that are cut) — this case is usually referred to as "min-disagree"; or 2) maximize the number of similarities within clusters plus the number of differences between clusters — referred to as "max-agree". The two problems have the same optima, but different approximations. In standard CC, the number of clusters naturally results from optimizing the objective function [3, 10], but some work considers the case where a specified fixed number of clusters is given [15]. Also, a natural generalization of the above is when $+$ and $-$ are replaced by positive or negative weights. Then the quality measure is not the number of disagreements or agreements, but the absolute weight of the edges with disagreements or agreements, respectively.

In the traditional graph-cut problem, there is an edge-weight function f_{trad} that is modular and always non-negative. The correlation clustering objectives, by contrast, correspond to modular cost functions with both positive and negative weights. Such functions are also non-monotone submodular functions. The modular-cost formulation has been used in the LP relaxations of CC (e.g. [10], also in [21]). Shamir et al. [42] and Giotis and Guruswami [15] show that CC with the additional constraint of having exactly k parts is NP hard.

We next show how correlation clustering (restricted to the case of 2 clusters) corresponds to an instance of CoopCut. We are given a graph $G = (V, E)$. Let $E^- \subseteq E$ be the set of edges with negative weights, $E^+ = E \setminus E^-$ the set of edges with positive weights, and $\chi_C \in \{0,1\}^E$ the characteristic vector of the cut $C \subseteq E$. The objective for min-disagree can be written as:

$$\begin{aligned}
f(C) &= \sum_{e \in C \cap E^+} w(e) + \sum_{e \in (E \setminus C) \cap E^-} (-w(e)) \\
&= \sum_{e \in E^+} \chi_C(e) w(e) + \sum_{e \in E^-} \big(1 - \chi_C(e)\big)(-w(e)\big) \\
&= \sum_{e \in E^+} \chi_C(e) w(e) - \underbrace{\sum_{e \in E^-} w(e)}_{\text{constant w.r.t. } C} + \sum_{e \in E^-} \chi_C(e) w(e) \\
&= \text{const.} + \sum_{e \in E} \chi_C(e) w(e) \\
&= \text{const.} + \sum_{e \in C} w(e).
\end{aligned}$$

Thus, minimizing disagreements corresponds to minimizing a modular graph cut objective with positive and negative weights. The unweighted version corresponds simply to the case where the weights are ± 1.

Similarly, the max-agree version corresponds to maximizing a modular objective, which is equivalent to minimizing the same modular objective times minus one. The objective to maximize is

$$\begin{aligned}
f(C) &= \sum_{e \in C \cap E^-} (-w(e)) + \sum_{e \in (E \setminus C) \cap E^+} w(e) \\
&= \sum_{e \in E^-} \chi_C(e)(-w(e)) + \sum_{e \in E^+} \big(1 - \chi_C(e)\big) w(e) \\
&= -\sum_{e \in E^-} \chi_C(e) w(e) + \underbrace{\sum_{e \in E^+} w(e)}_{\text{constant w.r.t. } C} - \sum_{e \in E^+} \chi_C(e) w(e) \\
&= \text{const} - \sum_{e \in E} \chi_C(e) w(e) \\
&= -\big(- \text{const} + \sum_{e \in C} w(e)\big),
\end{aligned}$$

again an instance of CoopCut.

C Derangements with one allowable exception

In Section 2.1, we define $D'(n)$ to be the number of derangements where one fixed element can be mapped to itself. Let this specific element be n, without loss of generality, i.e., $\sigma(n) = n$ is allowed.

The derivation of $D'(n)$ follows the technique of the forbidden board [43, pp. 71-73]. Here, the forbidden board is $B = \{(1,1),(2,2),\ldots,(n-1,n-1)\}$. Let N_j be the number of permutations σ for which $\left|\{(i,\sigma(i)\}_{i=1}^{n} \cap B\right| = j$, i.e., their graph coincides with B in j positions. Furthermore, let r_k be the number of k-subsets of B such that no two elements have a coordinate in common. The polynomial

$$N_n(x) = \sum_j N_j x^j = \sum_{k=0}^{n} r_k (n-k)!(x-1)^k$$

gives the wanted solution $D'(n) = N_0 = N_n(0)$. For the board B above, $r_k = \binom{n-1}{k}$. Thus,

$$\begin{aligned} N_n(x) &= \sum_{k=0}^{n} r_k (n-k)!(x-1)^k \\ &= \sum_{k=0}^{n} \binom{n-1}{k} (n-k)!(x-1)^k \\ &= \sum_{k=0}^{n} \frac{(n-1)!}{k!(n-1-k)!}(n-k)!(x-1)^k \\ &= \sum_{k=0}^{n} \frac{(n-1)!}{k!}(n-k)(x-1)^k. \end{aligned}$$

Then $D'(n) = N_n(0) = \sum_{k=0}^{n} \frac{(n-1)!}{k!}(n-k)!(-1)^k$ and $D'(n-1) = N_{n-1}(0) = \sum_{k=0}^{n-1} \frac{(n-2)!}{k!}(n-1-k)!(-1)^k$.