Near Optimal algorithms for constrained submodular programs with discounted cooperative costs

Rishabh Iyer
Department of Electrical Engineering
University of Washington
rkiyer@u.washington.edu

Jeff Bilmes
Department of Electrical Engineering
University of Washington
bilmes@u.washington.edu

Abstract

In this paper, we investigate two problems, namely (a) minimizing a submodular cost function under combinatorial constraints, which include cuts, matchings, paths, etc., and (b) optimizing a submodular function under submodular cover and submodular knapsack constraints. While both problems have hardness factors of $\Omega(\sqrt{n})$ for general submodular cost functions, we show how we can achieve constant approximation factors (in certain cases, even fully polynomial time approximation schemes), when we restrict the cost functions to low rank sums of concave over modular functions. A wide variety of machine learning applications are very naturally modeled via this subclass of submodular functions. Our work therefore provides a tighter connection between theory and practice by enabling theoretically satisfying guarantees for a rich class of expressible, natural, and useful submodular cost models. We empirically demonstrate the utility of our models on a real world problem of cooperative image matching.

1 Introduction

Submodular functions provide a rich class of expressible models for a variety of machine learning problems. Submodular functions occur naturally in two flavors. In minimization problems, they model notions of cooperation, attractive potentials, and economies of scale, while in maximization problems, they model aspects of coverage, diversity, and information. A set function $f : 2^V \to \mathbb{R}$ over a finite set $V = \{1, 2, \ldots, n\}$ is *submodular* [3] if for all subsets $S, T \subseteq V$, it holds that $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$. Given a set $S \subseteq V$, we define the *gain* of an element $j \notin S$ in the context S as $f(j|S) \triangleq f(S \cup j) - f(S)$. An equivalent characterization is the *diminishing marginal returns*, namely $f(j|S) \geq f(j|T)$ for all $S \subseteq T, j \notin T$.

In this paper, we address the following three problems. The first one is constrained submodular minimization [4, 10, 11, 12, 21], while the other two problems are submodular optimization problems subject to submodular constraints [9]:

$$\text{P1: } \min\{f(X)|X \in \mathcal{C}\}, \ \text{P2: } \min\{f(X) \,|\, g(X) \geq c\}, \quad \text{and} \quad \text{P3: } \max\{g(X) \,|\, f(X) \leq b\},$$

where the functions f and g are *monotone submodular*, and $\mathcal{C}$ is a *combinatorial constraint*, which could represent a cardinality lower bound constraint, or more complicated ones like cuts, matchings, trees, or paths in a graph.

While submodularity, like convexity, occurs naturally in a wide variety of problems, recent studies have shown that in the general case, many submodular problems of interest are very hard. This includes, for example, the three problems we consider in this paper. While unconstrained submodular minimization is polynomial time, even simple constraints in the form of cardinality lower bounds (i.e $\mathcal{C} = \{X \subseteq V : |X| \leq k\}$ make P1 very hard [21], with a hardness factor of $\Omega(\sqrt{n})$. Other constraints like cuts, matchings, spanning trees and paths have similar hardness factors [21, 4, 12]. Similarly, P2 and P3 in the worst case, also admit hardness factors of $\Omega(\sqrt{n})$ [9].

On the other hand, these problems come up as models in many machine learning applications. These lower bounds are specific to rather contrived classes of functions, whereas much better results can be achieved for many practically relevant cases. The pessimistic worst case results are somewhat discouraging, thus begging the need to quantify sub-classes of submodular functions that are more amenable to these optimization problems. An important observation is that the polynomial hardness of P1, P2 and P3, comes up mainly due to the submodular cost function f – they do not depend as much on the constraints $\mathcal{C}$ or the submodular function g [11, 9]. In this paper, we focus on a tractable yet expressive subclass of submodular cost functions f, namely low rank sums of concave over modular functions. Low rank sums of concave over modular functions are the class of functions representable as $f(X) = \sum_{i=1}^{k} \psi_i(w_i(X))$, where ψ_i's are monotone concave, and k is constant or $O(\log n)$. *Low Rank* in this context means that the number of componets in the sum is small (i.e k is small). Our use of the term "low rank" is identical to that used in, e.g., [6]. We argue how this subclass naturally models many interesting applications of problems 1, 2 and 3 in machine learning. We do not need to consider the entire class of submodular functions (which include rather contrived instances), but only this subclass. This observation helps us in providing better connections between theory and practice. The main specialty of this subclass is that these functions effectively model cooperation between objects via discounts provided by concave functions. Moreover, we show that this subclass admits fully polynomial time approximation schemes for P1, and constant factor approximation guarantees for P2 and P3, a significant improvement over [11, 9].

Low rank sums of concave over modular functions in P1, P2 and P3, fit as natural models in several machine learning problems. Below, we summarize some of these.

Image segmentation (Cooperative Cuts): Markov random fields with pairwise attractive potentials occur naturally in modeling image segmentation and related applications [1]. While models are tractably solved using graph-cuts, they suffer from the shrinking bias problem, and images with elongated edges are not segmented properly. When modeled via a submodular function, however, the cost of a cut is not just the sum of the edge weights, but a richer function that allows cooperation between edges, and yields superior results on many challenging tasks (see, for example, the results of the image segmentations in [13]). This was achieved in [13] by partitioning the set of edges $\mathcal{E}$ of the grid graph into groups of similar edges (or types) $\mathcal{E}_1, \cdots, \mathcal{E}_k$, and defining a function $f(S) = \sum_{i=1}^{k} \psi_i(w(S \cap \mathcal{E}_i)), S \subseteq \mathcal{E}$, where ψ_is are concave functions and w encodes the edge potentials. This ensures that we offer a *discount* to edges of the same type. Moreover, the number of types of edges are typically much smaller than the number of pixels, so this is a low-rank sum of concave functions.

Image Correspondence (Cooperative Matchings): The simplest model for matching key-points in pairs of images (which is also called the correspondence problem) can be posed as a bipartite matching. These models, however, do not capture interaction between the pixels. One kind of desirable interaction is that similar or neighboring pixels be matched together. We can acheive this via a minimum submodular matching (in fact, using low rank sums of concave over modular functions). We illustrate this application in the experiments section, at the end of this paper.

Sensor Placement or Feature Selection: Often, the problem of choosing sensor locations A from a given set of possible locations V can be modeled [17, 7] by maximizing the mutual information between the chosen variables A and the unchosen set $V \backslash A$ (i.e., $g(A) = I(X_A; X_{V \backslash A})$). Alternatively, we may wish to maximize the mutual information between a set of chosen sensors X_A and a quantity of interest C (i.e., $g(A) = I(X_A; C)$) assuming that the set of features X_A are conditionally independent given C [17, 7]. Both these functions are submodular. Since there are costs involved, we want to simultaneously minimize the cost $f(A)$. Often this cost is submodular [17, 7], since there is typically a discount when purchasing sensors in bulk (or computing features), and we can express this via P2 and P3, in fact using low rank sums of concave over modular functions [7].

2 Background and Existing Algorithms

The basic idea for most combinatorial algorithms solving P1, P2 and P3, are based on approximating the cost function f with a tractable surrogate function $\hat{f}$ [4, 5, 12, 10, 9, 11]. Moreover, as we shall see, all three problems have analogous guarantees. We characterize the quality of the solution via the notion of approximation factors. In particular, we say that an algorithm achieves an approximation factor of $\alpha \geq 1$ for Problem 1, if we can obtain a set $\hat{X}$ such that $f(\hat{X}) \leq \alpha f(X^*)$, where X^* is the optimizer of P1. For P2 and P3, we use the notion of bicriterion approximation factors. An algorithm is a $[\sigma, \rho]$ bi-criterion algorithm for P2 if it is guaranteed to obtain a set $\hat{X}$ such that $f(\hat{X}) \leq \sigma f(X^*)$ (approximate optimality) and $g(\hat{X}) \geq \rho c$ (approximate feasibility), where X^* is an optimizer of P2. Typically, $\sigma \geq 1$ and $\rho \leq 1$. Similarly, an algorithm is a $[\rho, \sigma]$ bi-criterion algorithm for Problem 3 if it is guaranteed to obtain a set $\hat{X}$ such that $g(\hat{X}) \geq \rho g(X^*)$ and $f(\hat{X}) \leq \sigma b$, where X^* is the optimizer of Problem 3. Moreover, P2 and P3 are very closely related [9], in that an approximation algorithm for one problem can be used to obtain guarantees for the other problem. The two problems also have matching hardness factors.

Supergradient based Algorithm (SGA): One such method uses the supergradients of a submodular function [11, 10, 4, 13, 8] to obtain modular upper bounds in an iterative manner. In particular, define a modular upper bound: $m^f_X(Y) \triangleq f(X) - \sum_{j \in X \setminus Y} f(j|V \setminus j) + \sum_{j \in Y \setminus X} f(j|X) \geq f(Y)$. The algorithm starts with the $X^0 = \emptyset$ and sequentially sets X^{i+1} as the solution of the corresponding problem (1, 2 or 3) with a surrogate function as $\hat{f}(X) = m^f_{X^i}(X)$ [11, 12, 9]. In each case, this subproblem is much easier. For example, in the case of P1, the subproblem becomes, $X^{i+1} = \min\{m^f_{X^i}(X) | X \in \mathcal{C}\}$, which is a linear cost problem, poly-time solvable for many constraints, like cardinality, cuts, matchings, paths etc. In the case of P2 and P3, these subproblems are $X^{i+1} = \min\{m^f_{X^i}(X) | g(X) \geq c\}$ and $X^{i+1} = \max\{g(X) | m^f_{X^i}(X) \leq b\}$, which are the submodular set cover and the submodular knapsack problems respectively [19, 9], and are constant factor approximable to a factor of $1 - 1/e$.

The supergradient based iterative algorithm (SGA) achieves an approximation factor of $\alpha = \frac{|X^*|}{1+(|X^*|-1)(1-\hat{\kappa_f}(X^*)} \leq \min\{|X^*|, \frac{1}{1-\hat{\kappa_f}(X^*)}\}$ for P1, and bicriteria factors satisfying $\sigma = \frac{|X^*|}{1+(|X^*|-1)(1-\hat{\kappa_f}(X^*)} \leq \min\{|X^*|, \frac{1}{1-\hat{\kappa_f}(X^*)}\}$ and $\rho = 1 - 1/e$ for P2 and P3, where $\hat{\kappa_f}(X) = 1 - \frac{\sum_{j \in X} f(j|X \setminus j)}{\sum_{j \in X} f(j)}$ represents the average curvature of the function f. These factors follow easily from the results in [11, 10, 9]. We can also achieve a non-bicriteria approximation factor for P2, which is worse than the bicriteria factor by a $\log$ factor [9]. A key quantity which defines the approximation factor above is the average curvature $\hat{\kappa_f}(X^*)$, which in turn depends on the concave functions. If the concave function $\psi_i(x) = x^a, a \in (0, 1)$, SGA admits approximation factors of $O(|X^*|^{1-a})$ [10]. On the other hand, if the concave function is $\psi(x) = \log(1 + x)$, the guarantees are $O(|X^*|)$, which is much poorer.

The supergradient based algorithm is easy to implement, and also works well in practice [11, 13]. For the general class of submodular functions, these results are close to the optimal bounds, and are, in fact, tight for some constraints. Nevertheless, the worst case guarantees seem discouraging, particularly for the class of low rank sums of concave over modular functions that we consider here, and that, as mentioned above, are natural for many applications.

Ellipsoidal Approximation based Algorithm (EA): Another generic approximation of a submodular function, introduced by Goemans et. al [5], is based on approximating the submodular polyhedron by an ellipsoid. The main result states that any polymatroid (monotone submodular) function f, can be approximated by a function of the form $\sqrt{w^f(X)}$ for a certain modular weight vector $w^f \in \mathbb{R}^V$, such that $\sqrt{w^f(X)} \leq f(X) \leq O(\sqrt{n}\log n)\sqrt{w^f(X)}, \forall X \subseteq V$. A simple trick then provides a curvature-dependent approximation [10] — we define the κ_f*-curve-normalized* version of f as follows: $f^\kappa(X) \triangleq \big[f(X) - (1-\kappa_f)\sum_{j \in X} f(j)\big]/\kappa_f$. Then, the submodular function $f^{\text{ea}}(X) = \kappa_f \sqrt{w^{f^\kappa}(X)} + (1-\kappa_f)\sum_{j \in X} f(j)$ satisfies [10]: $f^{\text{ea}}(X) \leq f(X) \leq O\left(\frac{\sqrt{n}\log n}{1+(\sqrt{n}\log n - 1)(1-\kappa_f)}\right) f^{\text{ea}}(X), \forall X \subseteq V$.

Similar to SGA, the Ellipsoidal Approximation (EA) based algorithm solves P1, P2 and P3 using the surrogate function $\hat{f}(X) = f^{\text{ea}}(X)$. This subproblem can be done efficiently for all linearly solvable constraints in P1, and approximately for P2 and P3 [10, 9]. The main result shows the approximation factor: The Ellipsoidal Approximation based algorithm (EA) achieves an approximation factor of $\alpha = O(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f})$ for P1, and bicriteria factors satisfying $\sigma = O(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f})$ and $\rho = 1 - 1/e$ for P2 and P3, where $\kappa_f = 1 - \frac{\min_{j \in X} f(j|V \setminus j)}{\min_{j \in X} f(j)}$ represents the worst case curvature of the function f. The Ellipsoidal Approximation often obtains the tightest bounds for many instances of P1, P2 and P3 [9, 10, 5, 4, 12]. This is again for the general class of submodular functions and the worst case factor of $O(\sqrt{n})$ is quite discouraging. This algorithm, however is very expensive computationally, and is not practical for solving machine learning applications [11].

3 Improved Algorithms for Low-rank sums of concave-modular functions

In this paper, our main new results are that we can achieve a fully polynomial time approximation scheme for P1, and constant factor approximation guarantees for P2 and P3, when the cost function f is a low rank sum of concave over modular functions (Theorem 3.2). We only give the results here, and defer the proofs and details to the extended version. Our techniques build on recent methods used for minimizing quasi-concave functions over solvable polytopes [20, 18, 6, 15].

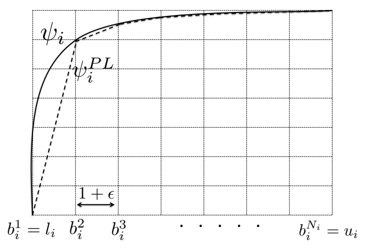

Figure 1: Visualizing ψ_i^{PL} and ψ_i.

Assume the concave functions ψ_i's are monotone functions, i.e $\psi_i(y) \leq \psi_i(y'), \forall y \leq y'$. We also assume that for all i, $\psi_i(ky) \leq k^c y$ for $k \geq 1, y \geq 0$ and some constant c. The second assumption holds for a number of concave functions, including $\psi_i(x) = x^a, a \in (0,1)$, $\psi_i(x) = \log(1+x)$ and $\psi_i(x) = \min(x, a)$.

The main idea of this approach is to replace the concave functions ψ_i's by piece-wise linear approximations $\psi_i^{PL}(x)$. We define an approximation of $f(X)$ as $f^{PL}(X)$ defined as $f^{PL}(X) = \sum_{i=1}^k \psi_i^{PL}(w_i(X))$. We then optimize this piece-wise linear approximation function, and the approximation factor comes based on the tightness of this piece-wise linear approximation. We call this procedure the *piece-wise linear approximation based algorithm* (PLA).

We compute this approximation as follows. In the case of P1, compute $l_i = \min\{w_i(X) | X \in \mathcal{C}\}$ and $u_i = \max\{w_i(X) | X \in \mathcal{C}\}$ for each $i = 1, 2, \cdots, k$. Both these computations are linear cost problems and are polynomial time for most constraints. In case these are NP hard for P1, or in the case of P2 and P3, we set $l_i = \min\{w_i(j), j \in V : w_i(j) > 0\}$ and $u_i = w_i(V)$. Then divide the range $[l_i, u_i]$ into pieces with breakpoints $b_i^1, b_i^2, \cdots, b_i^{N_i}$ such that $b_i^1 = l_i$, $b_i^2 = l_i(1+\epsilon)$, $b_i^3 = l_i(1+\epsilon)^2$ and so on, for any $\epsilon > 0$. It is easy to see that $N_i = \log_{1+\epsilon} u_i/l_i = \log(u_i/l_i)/\epsilon$. The precision ϵ defines the fineness of the points, and the quality of the approximation.

For all $i = 1, 2, \cdots, k$, define the piece-wise linear function ψ_i^{PL}, via the breakpoints $b_i^1, b_i^2, \cdots, b_i^{N_i}$. A visualization of this is shown in Figure 1, where the dotted lines are the piece-wise approximation, while the solid curve is the concave function ψ_i. We first show that the function f^{PL} approximates the function f within a factor of $1+\epsilon$.The proof of this is in the extended version.

Lemma 3.1. *The piece-wise linear function f^{PL} defined with a precision ϵ' satisfies, $f^{PL}(X) \leq f(X) \leq (1+\epsilon')^c f^{PL}(X) = (1+\epsilon) f^{PL}(X)$ where c is a constant such that $\psi_i(ky) \leq k^c y$ for $k \geq 1, y \geq 0$ for all $i = 1, 2, \cdots, k$.*

We now show how we can exactly solve P1, P2 and P3 using the cost function f^{PL}. Let s_i^j denote the slopes of the piece-wise linear functions – in other words, $s_i^j = [\psi_i(b_i^{j+1}) - \psi_i(b_i^j)]/[b_i^{j+1} - b_i^j]$. Also, we denote c_i^j as the corresponding intercepts. The functions ψ_i^{PL} are characterized by the pairs $\{(s_i^1, c_i^1), (s_i^2, c_i^2), \cdots, s_i^{N_i}, c_i^{N_i})\}$, and $\psi_i^{PL}(y) = s_i^j.y + c_i^j, \forall y \in [b_i^j, b_i^{j+1}]$. We then consider the $\prod_{i=1}^k N_i$ different possibilities of the cross-terms. Define $J = [j_1, j_2, \cdots, j_k]$ as a vector such that $J \in [1, N_1] \times [1, N_2] \times \cdots \times [1, N_k]$.

In the case of P1, PLA solves a set of optimization problems, $\hat{X}_J = \text{argmin}\{\sum_{i=1}^k s_i^{j_i} w_i(X) | X \in \mathcal{C}\}, \forall J \in [1, N_1] \times [1, N_2] \times \cdots \times [1, N_k]$. The final solution $\hat{X}$ is the minimum amongst the ones above. For problem 2, we consider the set of problems, $\hat{X}_J = \min\{\sum_{i=1}^k s_i^{j_i} w_i(X) | g(X) \geq c\}, \forall J \in [1, N_1] \times [1, N_2] \times \cdots \times [1, N_k]$, and again set $\hat{X}$ is the minimum amongst the $\hat{X}_J$'s above. Similarly, for Problem 3, we solve, $\hat{X}_J = \max\{g(X) | \sum_{i=1}^k s_i^{j_i} w_i(X) + c_i^{j_i} \leq b\}. \forall J \in [1, N_1] \times [1, N_2] \times \cdots \times [1, N_k]$. We set $\hat{X}$ corresponding to the set with the largest value of $g(\hat{X}_J)$. Our main result is that these simple procedures provide improved guarantees for all three problems.

Theorem 3.2. *PLA achieved an approximation factor of $1+\epsilon$ for Problem 1 as long as a linear function can be exactly minimized under $\mathcal{C}$. PLA also achieves a bicriterion approximation factor satisfying $\sigma = 1+\epsilon$ and $\rho = 1 - 1/e$ for Problems 2 and 3. PLA also achieves a non bicriterion approximation factor of $(1+\epsilon)\log g(V)$ for Problem 2. The worst case complexity of PLA is $\prod_{i=1}^k \log(u_i/l_i)(\frac{1}{\epsilon})^k T = O((\frac{1}{\epsilon})^k T)$, where T is the complexity of Problems 1, 2 and 3, with a linear cost function f.*

Results similar to Theorem 3.2 have been shown for a generalization of Problem 1, which asks for constrained optimization of low rank functions [20, 18, 6, 15, 16]. This problem in general is not

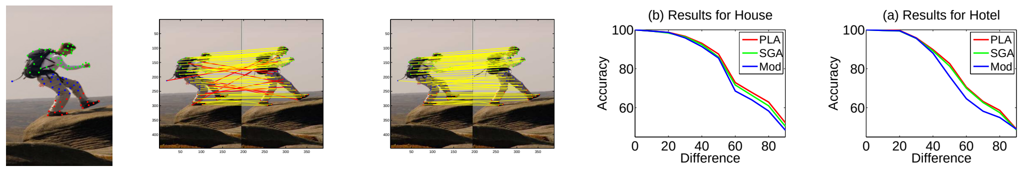

Figure 2: (a) (far left) shows the clustering used in cooperative matching, (b) and (c) show the results with cooperative matching and bipartite matching respectively, and (d) and (e) give the results on the House and Hotel dataset, showing PLA (this paper) achieves slightly better than SGA [11, 9], and both submodular methods performing better than standard matching (Mod).

a combinatorial optimization problem. However, when the functions are quasi-concave, the optimum lies on an extreme point, and hence, can be posed as a combinatorial optimization problem. Problem 1 asks for optimizing a specific subclass of concave (and hence quasi-concave) functions. [20, 6, 15] focus on the class of low rank quasi-concave functions, while [18] consider the general class of low rank functions. While their algorithms apply to our class of functions as well, their approach while being more general, is also more complicated and involved. [16] also consider a special case of Problem 1, with $\mathcal{C}$ being the family of cuts (i.e the cooperative cut problem). Interestingly, they suggest an algorithm that is identical to PLA when f is a (low rank) sum of truncations (i.e., $\psi_i(x) = \min(x, a)$). For general sums of low-rank concave functions, they resort to the algorithms of [18, 6]. We provide a generic algorithm (which relies on approximating the function directly), which not only works for a much large class of constraints and functions, but also extends to the Problems 2 and 3. Moreover, it is easy to see that our algorithms would also work for the more general problem of minimizing low rank sums of concave functions, over a solvable polytope.

Note that the complexity of PLA is polynomial in $\frac{1}{\epsilon}$, but exponential in k. Hence this makes sense only if k is a constant or is $O(\log n)$. If k is a constant (with respect to n), PLA is a fully polynomial time approximation scheme (FPTAS). If $k = O(\log n)$, then PLA is a polynomial-time approximation scheme (PTAS). This assumption is reasonable for many of the applications of Problems 1, 2 and 3 (see details of this in the experiments section). Moreover, there a number of ways one can speed up PLA. A very simple observation is that PLA is amenable to a distributive implementation via Map-Reduce. In particular, let $N = O(\frac{1}{\epsilon}^k)$ denote the total number of computations of PLA (i.e., this is the number of times one performs an instance of Problems 1, 2 and 3 with a modular function). All these can be performed in parallel on m processing systems. We output the best from each system to a central processor, which finds the optimal amongst these. The complexity of this distributive procedure is $O(NT/m + m)$, (where T is the complexity of using a modular function in the place of f in Problems 1, 2 and 3), which improves the overall complexity by a factor of m.

In addition, we can also provide early stopping criterion and heuristics for speeding up PLA. One strategy of implementing PLA, is to start with $j_i = 1, \forall i = 1, 2, \cdots, k$, and incrementally increase j_i in a coordinate ascent fashion. The following lemma gives a sufficient condition for stopping PLA.

Lemma 3.3. *Let $J = [j_1, j_2, \cdots, j_k]$ be such that the corresponding solution $\hat{X}_J$ satisfies $w_i(X_J) \in [b_i^{j_i}, b_i^{j_i+1}], \forall i$. Then $\hat{X}_J$ is the (near) optimal solution for Problems 1, 2 and 3.*

The values of $w_i(X_J)$ also suggest the direction of the co-ordinate wise algorithm. For example, if $w_i(X_J) < b^i_{j_i}$, it suggests that the value of j_i be decreased. Similarly, if $w_i(X_J) > b^i_{j_i+1}$, its a sign that j_i be decreased. In this manner, one can define a greedy like heuristic to implement PLA [16], which picks for every coordinate, the slope which increases the objective value the most. Many of these heuristics have been considered in [16] in the case of cuts, and when the function class is low rank sums of truncations. These heuristics are all polynomial in k, but are not guaranteed to obtain the optimal solutions. Moreover, in certain cases (for example, the case of cuts), one can do parametric versions, thereby solving a set of related problems simultaneously [16].

4 Experiments

In this section, we experimentally evaluate the performance of our algorithm. The utility of the constrained minimization algorithms for cooperative cuts have been investigated in [16]. In this paper, we consider the application of cooperative matching.

The problem of matching key-points in images, also called image correspondence, is a very important problem in computer vision. The simplest model for this problem is to do matching with linear scores, i.e., bipartite matching, called a *linear assignment*. This model does not represent interaction between the pixels. For example, we see many obviously spurious matches in figure 2b. Many models try to capture this, via, for example via quadratic assignments [2]. Instead of just looking at the best linear assignment, the quadratic models try to incorporate pairwise constraints. This is also called graph matching. We introduce a new and different model here. First, we cluster key-points, separately in each of the two images, into k clusters. Figure 2a shows a particular clustering of an image into $k = 3$ groups (these images were taken from a dataset used in [14]). The clustering can be performed based on the pixel color map, or simply the distance of the key-points. That is, each image has k clusters. Let $\{V_i^{(1)}\}_{i=1}^k$ and $\{V_i^{(2)}\}_{i=1}^k$ be the two sets of clusters. We then compute the linear assignment problem, letting $\mathcal{M} \subseteq \mathcal{E}$ be the resulting maximum matching. We then partition the edge set $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \mathcal{E}_k \cup \mathcal{E}'$ where $\mathcal{E}_i = \mathcal{M} \cap (V_\ell^{(1)} \times V_s^{(2)})$ for $\ell, s \in \{1, 2, \ldots, k\}$ corresponding to the i'th largest intersection, and $\mathcal{E}' = \{\mathcal{E} \setminus \cup_{i=1}^k \mathcal{E}_i\}$ are the remaining edges either that were not matched or that did not lie within a frequently associated pair of image key-point clusters. We then define a submodular function as follows: $f(S) = \sum_{i=1}^k \psi_i(w(S \cap \mathcal{E}_i)) + w(S \cap \mathcal{E}')$, which provides an additional discount to the edges $\{\mathcal{E}_i\}_{i=1}^k$ corresponding to key-points that were frequently associated in the initial pass. Figures 2b and 2c shows how the submodular matchings gains viz-a-viz the simple bipartite matching, with $k = 3$. The minimum matching approach obtains many spurious matches between clusters (shown in red), while the cooperation described above reduces these spurious matches. The cooperative matching improves the performance over the modular method on these images, by about 20%.

We also test the performance of our algorithms on the CMU House and Hotel dataset [2]. The house dataset has 111 images, while the hotel dataset has 101 images. We consider all possible pairs of images, with differences between the two images ranging from $0 : 10 : 90$ in both cases. We consider three algorithms: PLA, SGA (on the submodular function above) and the simple modular bipartite matching as a baseline (Mod). Again, we set $k = 3$. The results are shown in Figure 2(d-e) where we observe that PLA and SGA beat Mod by about $3 - 5\%$ on average. Moreover, we also see that PLA, in general, outperforms SGA, thus showing how superior theoretical guarantees translate into better empirical performance. In PLA, we chose ϵ such that each concave function ψ_i has four break points. We observed, moreover, that setting lower values of ϵ does not improve the objective value in this application. We observe, moreover, that PLA also beats SGA in terms of objective value. We do not compare the ellipsoidal approximation algorithm (EA) [5], mainly because it is too slow to run on real world problems. Moreover, this algorithm has been observed to perform comparably to the much simpler SGA [11]. While we considered the simple linear assignment as a baseline for the cooperative matching, it seems possible to embed this cooperation on more involved graph matching models as well.

References

[1] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *TPAMI*, 26(9):1124–1137, 2004.

[2] T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, and A. J. Smola. Learning graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(6):1048–1058, 2009.

[3] S. Fujishige. *Submodular functions and optimization*, volume 58. Elsevier Science, 2005.

[4] G. Goel, P. Tripathi, and L. Wang. Combinatorial problems with discounted price functions in multi-agent systems. In *FSTTCS*, 2010.

[5] M. Goemans, N. Harvey, S. Iwata, and V. Mirrokni. Approximating submodular functions everywhere. In *SODA*, pages 535–544, 2009.

[6] V. Goyal and R. Ravi. An fptas for minimizing a class of low-rank quasi-concave functions over a convex set. *Operations Research Letters*, 41(2):191–196, 2013.

[7] R. Iyer and J. Bilmes. Algorithms for approximate minimization of the difference between submodular functions, with applications. *In UAI*, 2012.

[8] R. Iyer and J. Bilmes. The submodular Bregman and Lovász-Bregman divergences with applications. In *NIPS*, 2012.

[9] R. Iyer and J. Bilmes. Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints. In *NIPS*, 2013.

[10] R. Iyer, S. Jegelka, and J. Bilmes. Curvature and Optimal Algorithms for Learning and Minimizing Submodular Functions . In *NIPS*, 2013.

[11] R. Iyer, S. Jegelka, and J. Bilmes. Fast Semidifferential based Submodular function optimization. In *ICML*, 2013.

[12] S. Jegelka and J. A. Bilmes. Approximation bounds for inference using cooperative cuts. In *ICML*, 2011.

[13] S. Jegelka and J. A. Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *CVPR*, 2011.

[14] S. Jegelka, A. Kapoor, and E. Horvitz. An interactive approach to solving correspondence problems. *International Journal of Computer Vision*, pages 1–10, 2013.

[15] J. A. Kelner and E. Nikolova. On the hardness and smoothed complexity of quasi-concave minimization. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 472–482. IEEE, 2007.

[16] P. Kohli, A. Osokin, and S. Jegelka. A principled deep random field for image segmentation. In *CVPR*, 2013.

[17] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 9:235–284, 2008.

[18] S. Mittal and A. S. Schulz. An fptas for optimizing a class of low-rank functions over a polytope. *Mathematical Programming*, 141(1-2):103–120, 2013.

[19] G. Nemhauser and L. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[20] E. Nikolova. Approximation algorithms for offline risk-averse combinatorial optimization, 2010.

[21] Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. In *FOCS*, pages 697–706, 2008.