HAL
open science

Convex Optimization for Parallel Energy Minimization

K. S. Sesh Kumar, Alvaro Barbero, Stefanie Jegelka, Suvrit Sra, Francis Bach

To cite this version:

K. S. Sesh Kumar, Alvaro Barbero, Stefanie Jegelka, Suvrit Sra, Francis Bach. Convex Optimization for Parallel Energy Minimization. 2015. hal-01123492

HAL Id: hal-01123492

https://inria.hal.science/hal-01123492v1

Preprint submitted on 5 Mar 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Convex Optimization for Parallel Energy Minimization

K. S. Sesh Kumar
INRIA-Sierra project-team
Département d'Informatique
de l'Ecole Normale Supérieure
Paris, France
sesh-kumar.karri@inria.fr

Álvaro Barbero
Instituto de Ingeniería del Conocimiento
and Universidad Autónoma de Madrid
Madrid, Spain
alvaro.barbero@uam.es

Stefanie Jegelka
Department of EECS
Massachusetts Institute of Technology
Cambridge, USA
stefje@csail.mit.edu

Suvrit Sra
Department of EECS
Massachusetts Institute of Technology
Cambridge, USA
suvrit@mit.edu

Francis Bach
INRIA-Sierra project-team
Département d'Informatique
de l'Ecole Normale Supérieure
Paris, France
francis.bach@inria.fr

March 5, 2015*

Abstract

Energy minimization has been an intensely studied core problem in computer vision. With growing image sizes (2D and 3D), it is now highly desirable to run energy minimization algorithms in parallel. But many existing algorithms, in particular, some efficient combinatorial algorithms, are difficult to parallelize. By exploiting results from convex and submodular theory, we reformulate the quadratic energy minimization problem as a total variation denoising problem, which, when viewed geometrically, enables the use of projection and reflection based convex methods. The resulting min-cut algorithm (and code) is conceptually very simple, and solves a sequence of TV denoising problems. We perform an extensive empirical evaluation comparing state-of-the-art combinatorial algorithms and convex optimization techniques. On small problems the iterative convex methods match the combinatorial max-flow algorithms, while on larger problems they offer other flexibility and important gains: (a) their memory footprint is small; (b) their straightforward parallelizability fits multi-core platforms; (c) they can easily be warm-started; and (d) they quickly reach approximately good solutions, thereby enabling faster "inexact" solutions. A key consequence of our approach based on submodularity and convexity is that it is allows to combine *any arbitrary combinatorial or convex methods as subroutines*, which allows one to obtain hybrid combinatorial and convex optimization algorithms that benefit from the strengths of both.

*Based on work originally submitted on 14 November, 2014.

1 Introduction

Energy minimization has become a key element in many low- to mid-level tasks in computer vision, such as segmentation or stereo correspondence (see [8] for a survey). For many, frequently occurring such minimization problems, graph cut techniques have emerged as generic, very efficient tools that provide global optima for submodular quadratic penalties, and extend to several higher-order and non-submodular potentials too. However, when applying widely used graph-cut code (e.g., [11]) to huge problems in 3D or video, running time and memory usage become problematic. Ideally, we would wish to have algorithmic flexibility to decompose the problem into (almost) arbitrary subproblems that can be solved in parallel, and that adapt to new architectures such as GPU clusters. These latter needs can be met through convex optimization.

In this paper, we explore methods that allow combining convex and combinatorial optimization, and thereby offer a way to parallelize recent successful combinatorial methods [10, 13, 17]. Our algorithms can run on large datasets while using only limited memory, and are flexible enough to be ported to different hardware architectures.

We begin by rewriting the energy minimization problem as a convex optimization problem. More precisely, we consider the equivalent reformulation of the min-cut problem as the minimization of a unary term $w^\top x$ plus the total variation $f(x)$ associated to a graph, over the vertices of the hypercube $x \in \{0,1\}^n$. The straightforward convex relaxation replaces the set of vertices $\{0,1\}^n$ of the hypercube with the full hypercube $[0,1]^n$. This relaxation is tight, a fact that has been exploited widely. Moreover, typical graph structures that occur in low-level vision problems naturally decompose as a superposition of chains, suggesting the use of iterative convex optimization methods [32, 39]. However, the lack of smoothness (in the primal and dual problems) poses some difficulties in solving the problem efficiently.

Hence, we instead consider a reformulation through total variation denoising outlined by [20, 23] for general submodular functions, which we use for cuts in this paper. The gist of this approach is to replace the non-smooth relaxation by the total variation (TV) denoising problem $\min_{x\in\mathbb{R}^n} \frac{1}{2}\|x-w\|^2 + f(x)$, from which one may obtain an optimal solution of the energy minimization problem by thresholding. The benefit of this formulation is its smooth dual problem, which has a natural geometric interpretation: it reduces to computing the distance between two convex sets. Moreover, via the equivalence between energy minimization and projections we obtain fast projection subroutines, which in turn enable the use of classical, popular projection methods [23].

While we focus on cuts in this paper, our framework extends to all higher-order submodular potentials, non-submodular quadratic potentials, and to multi-label problems (see Section 4.4 for more details). We make the following contributions:

- We study algorithms that allow to easily combine convex and combinatorial optimization (e.g., graph cut, total variation, algorithms for higher order submodular potentials), and to parallelize existing fast energy minimization methods for submodular potentials. In fact, this approach applies to arbitrary sums of submodular functions, each defined on all or a subset of the variables.
- As a case study, we specialize the optimization framework of [23] to the submodular energy functions most commonly used in computer vision, such as quadratic functions with a 2D or 3D grid structure with various connectivities (Section 3.1). The resulting min-cut algorithm (and code) is conceptually very simple and may thus be easily used and modified.
- We perform an extensive empirical evaluation of serial and parallel, combinatorial and convex min-cut codes on benchmarks [43, 25]. We observe that iterative techniques based on convex optimization match combinatorial max-flow algorithms on small problems, and on larger problems they offer complementary flexibility: (a) they have a reduced memory footprint and can tackle many problems where traditional methods run out of memory, (b) they parallelize easily on multi-core platforms, (c) they can be efficiently warm-started, (d) they quickly reach approximately good solutions, thereby enabling

faster "inexact" solutions. These observations suggest to marry convex and combinatorial methods to attain "the best of both worlds".

2 Convex optimization for graph cuts

In this paper, we focus on energy minimization problems with pairwise potentials,

$$E(x) = -\sum\nolimits_{i=1}^{n} w_i x_i + \sum\nolimits_{i,j=1}^{n} \psi_{ij}(x_i, x_j), \tag{1}$$

where the variables x_i take values in a set of discrete labels. For simplicity, we here focus on binary labels, $x_i \in \{0, 1\}$. The algorithms we discuss may be generalized to multiple labels via move-making strategies [11]. Moreover, we assume the pairwise potentials to be submodular, i.e., $\psi_{ij}(0,1) + \psi_{ij}(1,0) \geq \psi_{ij}(0,0) + \psi_{ij}(1,1)$. (One may extend to non-submodular potentials via roof duality [38]). It is well known that all such submodular energy functions may be written as graph cut functions with nonnegative "edge weights" a_{ij}, up to a constant [35]:

$$E(x) = -\sum\nolimits_{i=1}^{n} w_i x_i + \sum\nolimits_{i,j=1}^{n} a_{ij}|x_i - x_j| + \text{const}. \tag{2}$$

This function consists of two parts: (1) a sum of unary potentials $-\sum_{i=1}^{n} w_i x_i = -w^\top x$; and (2) the sum of pairwise potentials, which is equivalent to a weighted graph cut between the set of indices i in $\{1, \ldots, n\}$ for which $x_i = 1$, and its complement. For $x \in \mathbb{R}^n$, this sum is the *total variation function* $f(x) \triangleq \sum_{i,j=1}^{n} a_{ij}|x_i - x_j|$. Note that this is a case of anisotropic weighted total variation. Since the weights a_{ij} are non-negative, the function f is convex. We refer to the graph cut problem as the *discrete problem*:

$$\min_{x \in \{0,1\}^n} f(x) - w^\top x. \tag{D}$$

2.1 Convex reformulation

We obtain a relaxation to the combinatorial problem in Eq. (D) by replacing $\{0, 1\}^n$ by its convex hull $[0, 1]^n$:

$$\min\nolimits_{x \in [0,1]^n} f(x) - w^\top x. \tag{C}$$

We refer to Eq. (C) as the *continuous problem*. This relaxation is *exact*: since the continuous convex problem in (C) is a minimization problem over a larger set than the discrete problem, its minimal value has to be lower than (D). However, as a consequence of properties of the total variation and its relation to submodular graph cut functions (see, e.g., [2, Sec. 3.3] or [20, 12] for a proof dedicated to cut functions), the two optimal values are equal and a solution to (D) may be obtained from a solution $x \in [0, 1]^n$ of (C) by looking at all "level sets" of x, that is by rounding the values of x to zero or one by thresholding at a given level in $[0, 1]$ (there are at most n possible thresholds, which can be obtained by first sorting the components of x).

2.2 Convex duality

The total variation f is convex and absolutely homogeneous, that is, for any $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$, $f(\lambda x) = |\lambda| f(x)$. For all such functions, there exists a centrally symmetric convex body $K \subset \mathbb{R}^n$ such that for all $x \in \mathbb{R}^n$ [37, §13],

$$f(x) = \max\nolimits_{y \in K} \; y^\top x.$$

Note that when $f(x)$ happens to be equal to zero only for $x = 0$, then f is a norm and the set K is simply the unit ball of the dual norm.

Since f is piecewise affine, the set K is a *polytope* (i.e., the convex hull of finitely many points). The set K may be described precisely for general submodular functions and is usually referred to as the *base polytope* of the submodular function [16, 23, 2].

Using Fenchel duality, we arrive at the following dual problem to (C) [16, 2]:

$$\begin{aligned}\min_{x\in[0,1]^n} f(x) - w^\top x &= \min_{x\in[0,1]^n} \max_{y\in K} y^\top x - w^\top x \\ &= \max_{y\in K} \min_{x\in[0,1]^n} x^\top (y-w) \\ &= \max_{y\in K} \sum_{i=1}^n \min\{y_i - w_i, 0\}. \end{aligned} \tag{3}$$

This dual problem allows us to obtain *certificates of optimality*: given a pair $(x, y) \in [0, 1]^n \times K$, the quantity

$$\mathrm{gap}(x, y) := f(x) - w^\top x - \sum_{i=1}^n \min\{y_i - w_i, 0\}$$

is always non-negative, and equal to zero if and only if x is optimal for (C) and y is optimal for the dual problem Eq. (3). This duality relation is essential to certify that a given solution is optimal (and corresponds to the traditional min-cut/max-flow duality).

While the cut problem is now reformulated as a convex optimization problem, it is still hard to minimize because neither the primal nor the dual are smooth, and thus iterative methods are typically slow (see detailed comparisons in [23]). We now reformulate the problem so that the dual problem becomes smooth and potentially easier to optimize.

2.3 Equivalence to total-variation denoising

Following [16, 2, 23, 33], we consider the *total variation denoising problem*:

$$\min_{x\in\mathbb{R}^n} f(x) + \frac{1}{2}\|x - w\|^2. \tag{TV}$$

By expanding $\frac{1}{2}\|x - w\|^2$ into $\frac{1}{2}\|x\|^2 - w^\top x + \frac{1}{2}\|w\|^2$, we see that going from the continuous problem (C) to (TV) means replacing the constraint $x \in [0, 1]^n$ by the penalty $\frac{1}{2}\|x\|^2 = \frac{1}{2}\sum_{i=1}^n x_i^2$. This has a number of important consequences: (1) It makes the optimization problem strongly convex and thus the dual problem will be smooth; see Eq. (4). (2) A solution to (D) and hence (C) may be obtained by *thresholding* the unique solution x of (TV) at zero, that is, by defining $\hat{x}_i = 1$ if $x_i > 0$ and $\hat{x}_i = 0$ otherwise. This is usually not true for arbitrary convex functions f (even absolutely homogeneous) and is a direct consequence of submodularity.

Importantly, we *need not* solve the TV problem (TV) exactly to obtain a solution to (C), we only need to know which of the components are positive (resp. negative).

Analogously to Eq. (3), we may derive the dual to (TV):

$$\begin{aligned}\min_{x\in\mathbb{R}^n} f(x) + \frac{1}{2}\|w - x\|^2 &= \min_{x\in\mathbb{R}^n} \max_{y\in K} y^\top x + \frac{1}{2}\|w - x\|^2 \\ &= \max_{y\in K} \min_{x\in\mathbb{R}^n} y^\top x + \frac{1}{2}\|w - x\|^2 \\ &= \max_{y\in K} \frac{1}{2}\|w\|^2 - \frac{1}{2}\|y - w\|^2. \end{aligned} \tag{4}$$

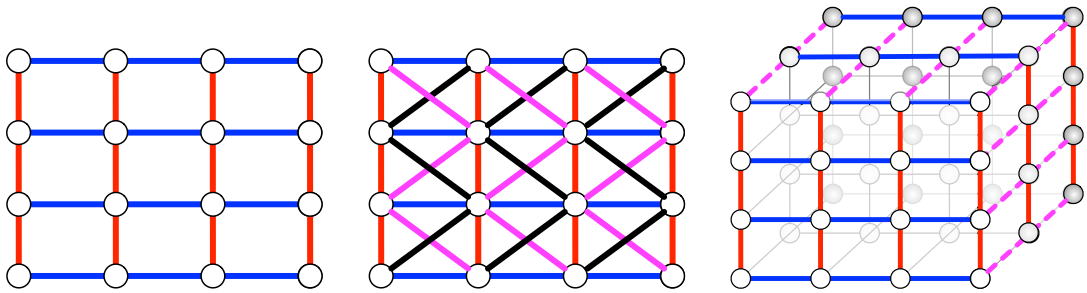


Figure 1: Grid graph structures and decompositions, indicated by different colors. Left: 2D, 4-connected, middle: 2D, 8-connected, right: 3D tensor, 6-connected.

The primal and dual solutions x and y have a simple correspondence $x = w - y$, as can be seen from the dual derivation. We have now obtained a dual problem which may be simply interpreted as the *orthogonal projection* of the vector w onto the polytope K. Importantly, a slight additional argumentation shows that the discrete energy minimization problem (D), the total variation relaxation (TV), and the projection onto K are equivalent. That means, a fast subroutine for one of the problems implies a fast subroutine for the other two.

While the reformulation is intuitive, orthogonal projections onto the polytope K are not fast to compute in general. Many special cases, however, are fast, including graphs that have fast cut subroutines (which may be combinatorial).

3 Decomposition of graphs

We assume that the total variation f (and equivalently the energy function) may be decomposed into a sum $\sum_{j=1}^{r} f_j(x)$ of r total variation functions f_j. Each function f_j can be defined on 2D grids, trees or chains. The only criterion for the decomposition is to efficiently perform orthogonal projections onto the polytope K_j corresponding to f_j. In fact, f_j does not need to have full support.

While other decompositions are possible [29, 40], in this paper, we focus on chains to illustrate our ideas; we note that 1D-TV (on chains) can be solved very efficiently–see Section 3.2. Note that the same idea works for decompositions into trees, stars, small "cubes" and 2D sheets.

3.1 2D and 3D grids

The most frequenly used 2D and 3D topologies in computer vision decompose straightforwardly in several ways [42, 32, 30, 39]. In our experiments, we split the 2D 4-connected grid into vertical and horizontal lines (Figure 1 left), and the 2D 8-connected grid additionally into zig-zagged paths (Figure 1 middle) . Diagonal lines are also possible but less efficient with respect to memory access.

Even though the set of 1D lines of equal color in the figures may be considered one f_j, the lines are independent and may be addressed in parallel. Further parallelization (across colors) results from our splitting formulation. Finally, any fast algorithm for solving a 2D or 3D TV (or graph cut) problem may be used as a subroutine when setting f_j to be an entire 2D grid sheet or 3D tensor.

3.2 Efficient 1D TV

As chains are the building blocks of our decomposition, performance of the overall method is heavily influenced by the speed at which 1D-TV can be solved. Being a classic regularizer for image denoising, literature on solvers for different variations of 1D and 2D TV abounds, though only recently fast direct methods for chains have been proposed. A notable example is the dynamic programming method of Johnson [24], which guarantees linear complexity. Another outstanding method is that of Condat [14]; it is based on a thorough analysis of the KKT conditions and manages to achieve faster running times in practice, despite a pathological quadratic cost worst-case. These TV solvers, however, only apply to chains with constant weights.

To permit varying weights we use a recent method of [3] that obtains Condat's method through a taut-string viewpoint [18], in a way that allows weights along the chain. Experiments [3] indicate that this method shares the same performance as the original procedure, therefore rapidly solving TV chains in linear time in practice.

We also point out that the choice of the 1D-TV solver is independent of the overlying topologies and optimizers. This allows us to localize complexity to highly tuned TV chain solvers for the architecture under use (multicore, GPUs, etc.), thus providing overall modularity and adaption to the underlying hardware. In this paper, we use a general implementation for CPUs.

Message passing. Alternatively to the method that we have used, we may also use message passing techniques, which could be more efficient on certain architectures. These are directly adapted to solve the min-cut problem on a chain or a tree, not the total variation problem. However, it is known that by a sequence of at most n min-cut problems, one may obtain the exact TV solution [22].

3.3 Decomposed dual problems

By their form (total variation or Lovász extensions of submodular functions), the f_i may be represented as a maximum of linear functions, that is, $f_j(x) = \max_{y_j \in K_j} \; y_j^\top x$, for K_j a certain polytope, $j \in \{1, \dots, r\}$. This form as well as the decomposability of the total variation may be used to obtain a decomposed dual problem for the continuous Problem (C). The dual splits in the same way as the primal, and admits parallel optimization algorithms [42, 32, 30, 39]. It however has a non-smooth objective function that makes optimization harder.

We hence next describe two dual problems for the (TV) problem [23].

First dual problem. We use a standard reformulation for dual decomposition: we introduce a variable $\mathbf{x} = (x_1, \dots, x_r) \in \mathbb{R}^n \times \cdots \times \mathbb{R}^n$ composed of r copies of the input variable x, with the constraint that $x_i = x$ for each $i \in \{1, \dots, r\}$ (see, e.g., [9]). We then add Lagrange multipliers $\lambda_i \in \mathbb{R}^n$ for each of these constraints. Writing f_j as a maximum of linear functions introduces a dual variable vector $y_j \in K_j$. We collect those variables in a vector $\mathbf{y} = (y_1, \dots, y_r) \in \mathbf{K} = K_1 \times \cdots \times K_r$. Overall, we obtain the following:

$$
\begin{aligned}
&\min_{x \in \mathbb{R}^n} \; \sum_{j=1}^{r} f_j(x) + \frac{1}{2}\|w - x\|^2 \\
&\quad = \max_{\boldsymbol{\lambda} \in \mathbb{R}^{n \times r},\, \mathbf{y} \in \mathbf{K}} \; \min_{x \in \mathbb{R}^n,\, \mathbf{x} \in \mathbb{R}^{n \times r}} \sum_{j=1}^{r} x_j^\top y_j + \frac{1}{2}\|w\|^2 - w^\top x + \frac{1}{2r}\sum_{j=1}^{r}\|x_j\|^2 + \sum_{j=1}^{r} \lambda_j^\top (x - x_j) \\
&\quad = \max_{\boldsymbol{\lambda} \in \mathbf{L},\, \mathbf{y} \in \mathbf{K}} \; \frac{1}{2}\|w\|_2^2 - \frac{r}{2}\sum_{j=1}^{r}\|y_j - \lambda_j\|^2
\end{aligned}
$$

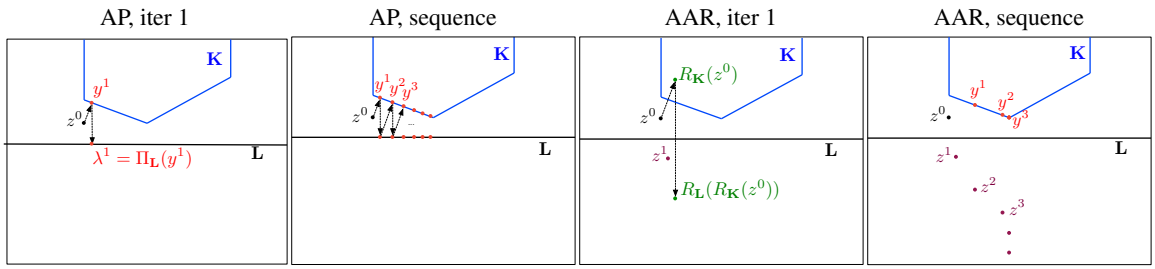


Figure 2: Illustration of alternating projections (AP) and averaged alternating reflections (AAR) for Problem (5), the problem of finding the closest points between a polytope $\mathbf{K}$ and a subspace $\mathbf{L}$. The iterations start at the black point z^0. AP alternatingly projects onto $\mathbf{K}$ and then $\mathbf{L}$. AAR reflects at $\mathbf{K}$ and then at $\mathbf{L}$; the next iterate z^{t+1} (magenta) is the midpoint between z^t and $R_{\mathbf{L}} R_{\mathbf{K}}(z^t)$. For AAR, the sequence z^t diverges, but the projected shadow sequence of $y^t = \Pi_{\mathbf{K}}(z^t)$ converges (red). Here, AAR takes larger steps than AP and hence converges more quickly.

$$= \max_{\boldsymbol{\lambda} \in \mathbf{L},\ \mathbf{y} \in \mathbf{K}} \frac{1}{2}\|w\|_2^2 - \frac{r}{2}\|\mathbf{y} - \boldsymbol{\lambda}\|^2, \tag{5}$$

where $\mathbf{L}$ denotes the set of $\boldsymbol{\lambda} \in \mathbb{R}^{n \times r}$ such that $\sum_{j=1}^{r} \lambda_j = w$. We are thus faced with the problem of finding the closest point between two convex sets, which we explore in Sections 4.2 and 4.3.

Note that if the function f_j only depends on a subset of variables of x, then we may restrict the corresponding variable λ_j to be zero on the complement of that subset in order to have faster convergence for the iterative methods presented below.

Second dual problem. Given $\mathbf{y} \in \mathbf{K}$ in (5), the optimal $\boldsymbol{\lambda} \in \mathbf{L}$ may be obtained in closed form:

$$\lambda_j = \frac{w}{r} + y_j - \frac{1}{r}\sum_{k=1}^{r} y_k. \tag{6}$$

This leads to another dual problem, where the variables $\boldsymbol{\lambda} \in \mathbf{L}$ are maximized out:

$$\max_{\mathbf{y} \in \mathbf{K}} \quad \frac{1}{2}\|w\|_2^2 - \frac{1}{2}\Big\|\sum_{j=1}^{r} y_j - w\Big\|^2. \tag{7}$$

The problem above has *separable constraints* $y_i \in K_j$, $j \in \{1, \ldots, r\}$ and a smooth objective function. We will discuss optimization procedures in Section 4.1.

Special case $r = 2$. When the function f is split into two functions, then the problem in Eq. (7) is equivalent to finding the distance between the convex set K_1 and the set $\{w - y_2 \mid y_2 \in K_2\}$, for which methods presented in Sections 4.2 and 4.3 may be used.

4 Optimization for decomposable problems

Next, we describe optimization procedures that exploit the decomposable structure of the dual problems (5) and (7), where in particular the Cartesian decomposition $\mathbf{K} = K_1 \times \cdots \times K_r$ of the constraint sets plays an important role. In particular, we exploit that the projection onto $\mathbf{K}$ consists of r independent projections onto the sets K_i. By the above derivations, each of those projections can be done quickly via a TV subroutine for each f_i.

4.1 Cyclic projections

The first method we consider for problem (7) is *block coordinate descent* (BCD), a classic method [7] that has recently witnessed a huge resurgence of interest in large-scale optimization [36, 6]. Since the cost function is a separable quadratic, BCD assumes a form that is more commonly known as *cyclic projections* (more precisely, this is so if we go through the constraint blocks in a cyclic order). Specifically, we update coordinate blocks $i = 1, \ldots, r$ as follows:

$$y_i \leftarrow \operatorname*{argmax}_{z \in K_i} -\frac{1}{2}\big\|z - w + \sum_{j \neq i} y_i\big\|^2 = \Pi_{K_i}\big(w - \sum_{j \neq i} y_i\big),$$

where Π_{K_i} denotes orthogonal projection onto set K_i. This projection is solved by solving a (fast) TV problem with f_i. Notice that the variable y_i is overwritten after the update, so that when updating y_{i+1}, the latest values $y_1, \ldots, y_i$ are used in the projection.

In addition to cyclic projections, one could solve the smooth dual (7) using a gradient-based method like FISTA [5]. Such a method is also easy to implement because the Lipschitz constant of the gradient is easily seen to be r and the required projections decompose due to the structure of $\mathbf{K}$.

4.2 Alternating projections in product space

The method of cyclic projections offers a practical choice. However, it is inherently serial. To solve the problem in parallel, the first dual formulation (5) turns out to be more suited (note that this provides a second source of parallelization, beyond the fact that each polytope K_j is itself a product of polytopes corresponding to individual lines).

The key idea is to exploit the "product space" $K_1 \times \cdots \times K_r$. Since w is constant, as previously mentioned, (5) is nothing but the problem of finding the closest point between two convex sets [23]. Applying BCD, except this time with just two coordinate blocks, we obtain the classic *alternating projections* (AP) (cast in a product space setting), which performs for $k = 0, 1, \ldots,$ the iteration:

$$\begin{aligned}
\mathbf{y}^{k+1} &\leftarrow \operatorname*{argmax}_{\mathbf{y} \in \mathbf{K}} -\frac{r}{2}\|\mathbf{y} - \boldsymbol{\lambda}^k\|^2 = \Pi_{\mathbf{K}}(\boldsymbol{\lambda}^k), \\
\boldsymbol{\lambda}^{k+1} &\leftarrow \operatorname*{argmax}_{\boldsymbol{\lambda} \in \mathbf{L}} -\frac{r}{2}\|\boldsymbol{\lambda} - \mathbf{y}^{k+1}\| = \Pi_{\mathbf{L}}(\mathbf{y}^{k+1}).
\end{aligned}$$

The key point here is that the projection $\Pi_{\mathbf{K}}$ decomposes

$$\Pi_{\mathbf{K}}(\boldsymbol{\lambda}) = (\Pi_{K_1}(\lambda_1), \ldots, \Pi_{K_r}(\lambda_r)),$$

so that each of the coordinate blocks may be computed in parallel (our implementation exploits this fact), while the projection $\Pi_{\mathbf{L}}$ is merely an averaging step detailed in (6).

4.3 Alternating reflections in product space

The recent work [23] provided strong experimental evidence that for projection problems of the form (5), AP is often outperformed by a more refined method of [4], namely, *averaged alternating reflections (AAR)*. Here, instead of alternating between the projection operations $\Pi_{\mathbf{K}}$ and $\Pi_{\mathbf{L}}$, one uses *reflection operators*

$$R_{\mathbf{K}} := 2\Pi_{\mathbf{K}} - I, \quad R_{\mathbf{L}} := 2\Pi_{\mathbf{L}} - I, \tag{8}$$

while averaging them to ensure firm nonexpansivity, a property that greatly simplifies convergence analysis [4]. To apply the AAR method, one first introduces the auxiliary vector $\mathbf{z}$, which represents $\mathbf{y} - \boldsymbol{\lambda}$. Then, AAR takes the form

$$\mathbf{z}^{k+1} = \tfrac{1}{2}(R_{\mathbf{L}} R_{\mathbf{K}} + I)\mathbf{z}^k. \tag{9}$$

However since usually $\mathbf{K} \cap \mathbf{L} = \emptyset$, the sequence $(\mathbf{z}^k)$ generated by (9) diverges to infinity! The remarkable fact is that from this diverging sequence, we can extract a solution by maintaining a "shadow sequence" $\mathbf{y}^k \equiv \Pi_{\mathbf{K}}(\mathbf{z}^k)$. See Figure 2 for an illustration, and Theorem 3.13 in [4] for a proof of convergence.

4.4 Extensions

Above, we outlined flexible, parallelizable convex optimization algorithms for energy minimization with pairwise submodular potentials. These algorithms straightforwardly generalize from binary labels to the multi-label case, to submodular higher-order potentials, and to related problems. The reasons are two-fold: (1) the above algorithms solve a minimum cut problem, and any methodological machinary that builds on graph cuts as a subroutine will work with the above algorithms too; (2) the decomposition theory and tightness of the relaxations hold generically for submodular functions, not only graph cuts.

For multi-label energy minimization, one may use move-making algorithms [11] that reduce the multi-label problem to a series of binary submodular energy minimization problems. The methods above solve those binary problems. For combinatorial algorithms, it has proved useful to reuse existing solutions and data structures [28]. "Warm-starting" is possible for the convex case too: we simply use the y_i vectors of the previous problem to initialize the new problem. If the geometry of the polytopes K has not changed too much, this can save many iterations (see Figure 3(b)).

Second, the convex approach directly generalizes to submodular potentials that involve more than two nodes at a time (following [23]); such potentials include [12, 41, 26, 27, 19]. Many of those potentials correspond to sufficiently simple submodular functions, often with small support, such that the relaxation (the equivalent to total variation for graph cuts) can be solved fast. Moreover, the same methods may even generalize to be used with roof duality [38].

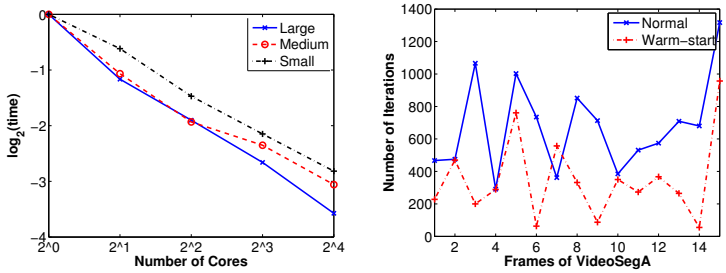
Finally, since the above methods also solve the parametric version of the discrete problem (by thresholding the solution of Eq. (TV) at different levels) as a byproduct, they are also applicable to the numerous applications of parametric graph cuts [31, 21].

5 Implementation details

The algorithms are inherently parallel by design as each projection/reflection onto a chain graph is independent of the other. Our implementation assumes decomposed functions and the decomposition depends on the problem at hand. In Section 3.1, we described some possible decompositions of grid-like graph structures on 2D and 3D graphs. However, this extends to many other decompositions. Empirically and theoretically [34], longer connected structures lead to faster convergence than decomposition e.g. into single edges.

5.1 Parallelization

We use the efficient 1D-TV implementation of [3] to solve the projection/reflection on chains in parallel. Our implementation is in C++ and uses OpenMP; it ensures that the memory access pattern across threads is streamlined, since bad memory access patterns can lead to considerable slowdowns. While the 1D-TV solver is not optimized for GPUs, as explained in Section 3.2, it can be replaced by message passing based subroutines, which are inherently parallel and also friendly to GPU architectures.



(a) (b)

Figure 3: Speedup of the AAR algorithm with parallelization and warm starts. (a) Normalized scale of performance with increasing number of cores on Bunny datasets [1] of different resolutions. (b) Number of iterations taken by each from of a video with "normal" initialization and "warm start" from the dual variables of the previous frame.

5.2 Memory footprint

In our implementation every decomposable function must maintain states of dual variables for each node in the graph. Thus, the memory requirement of our methods increases bilinearly in the number of decomposed functions and the number of nodes. Our experiments suggest that the projection-based algorithms require less memory than standard combinatorial algorithms—see Table 1. Unlike the 32 bit integers used in many other implementations, we use (64 bit) double precision numbers. Reducing those to 32 bit would reduce the memory requirements of the projection methods even further.

Algorithm	AAR	BK [10]	IBFS [17]	HPF [13]
Memory(GB)	26.82	42.83	44.16	55.46

Table 1: Memory footprint for *Abdomen* dataset ($512 \times 512 \times 551$)

5.3 Running time

With the TV subroutine we use, each projection/reflection step scales in the worst case quadratically in the length of the chain (the chain length is typically equal to $\sqrt{n}$ or $\sqrt[3]{n}$, where n is the total number of nodes in the 2D or 3D graphs), but is in practice usually linear [3]. In fact, it did not scale quadratically for any of our data. Hence, empirically, the cost of each iteration grows bilinearly with the number f of functions f_i, and with the number of nodes in the graph.

6 Experiments

Our experiments study the performance of the projection algorithms on 2D and 3D-maxflow datasets [1], exploiting in particular the parallel nature of the algorithms. We compare the algorithms to standard, popular maxflow implementations such as BK [10], IBFS [17], and HPF [13]. For other algorithms [42, 32], we have not been able to find implementations that were easily portable to 3D datasets.

Table 2 shows running time and the number of iterations for the projection algorithms and others on a multicore machine[1]. The timings are recorded using the *gettimeofday* command. All the timings are for the optimization phase only and exclude data I/O (which is common to all methods).

6.1 2D problems

As a 2D example, we use the tsukuba data [1], a multi-label task on 2D images corresponding to 4-neighborhood grids. To cope with multiple labels, we use alpha expansion [11]. Notably, the decomposition for 2D 4-connected grids uses only two functions (vertical and horizontal), and therefore corresponds to the special dual for $r = 2$ in Section 3.3. For this formulation, AAR converges remarkably faster than other iterative algorithms, and even outperforms the combinatorial methods. The time comparisons at the bottom of Table 2 show that in general, the running times of our methods are comparable to standard combinatorial algorithms on images of 384×288 size.

Warm starts. Figure 3(b) shows the number of iterations required for the algorithm to converge on each frame of size 480×360 of the VideoSegA [1] dataset. These are the consecutive frames of a video, which are 2D images with 8-neighborhood grids (see Figure 1). We use the dual variables at the convergence of the previous frame to warm-start the projection/reflection process. This makes the method converge to the optimal solution substantially quicker than with other initializations.

6.2 3D problems

On the 3D data, the running times of the algorithms differs more widely. In particular, the number of iterations for the algorithms to converge appears to depend on two important characteristics: (i) number of nodes in the graph (dimensionality) (ii) the edge weights in the graph (weights of the TV term). The latter affects the size of the polytopes K, i.e., the diameter of the domain of the dual problem, a parameter that commonly influences the convergence of convex optimization methods (see also [34]).

Effect of edge weights. Table 2 shows results for larger edge weights, as well as weights scaled by a factor of 0.1. The iterative methods become faster with smaller weights, while the combinatorial methods robustly perform well with large weights too. On many instances, AAR converges faster than the cyclic BCD, while on others BCD is faster.

Approximate solutions. Since we can obtain a feasible solution (discrete cut) from any iterate of the projection methods by thresholding the continuous vector, Table 2 also shows the time taken to obtain an approximate solution with limited error (10% and 2%, measured by Jaccard distance). The results suggest that, while a complete dual certificate of convergence for the discrete problem takes a bit longer, a reasonable approximate solution can be obtained fairly quickly.

Parallel speedup. Figure 3-(a) shows the speedup of AAR achievable with an increasing number of cores. This figure reports the running time on the Bunny dataset (3D) with different resolutions: *Large* ($401 \times 396 \times 312$), *Medium* ($202 \times 199 \times 157$), and *Small* ($102 \times 100 \times 79$). It is evident that more cores can improve the performance of the algorithm considerably. When using GPUs, it is important to consider their limited video memory, and hence the algorithms need to have a low memory footprint to perform well.

[1] 20 core, Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz with 100Gigabytes of memory. We only use up to 16 cores of the machine to ensure accurate timings.

Dataset	Time in seconds								Iterations				
	AAR ($< 10\%$)	AAR ($< 2\%$)	AAR-JD (< 0.1)	AAR-JD (< 0.02)	AAR	BK	IBFS	HPF	AAR	AAR (0.1x)	AP	BCD	FISTA
(3D) bone-100	4.2	7.4	3.9	10.9	14.4	8.5	6.3	1.0	105	73	846	146	422
(3D) bone-10	4.5	7.4	4.7	9.6	18.5	5.6	3.4	0.9	134	25	1183	206	592
(3D) bone_x-100	0.08	0.08	0.08	0.08	3.90	2.42	1.02	1.08	45	34	42	9	25
(3D) bone_x-10	0.09	0.09	0.09	0.09	3.70	1.86	0.93	0.75	45	23	44	10	26
(3D) bone_xy-100	0.01	0.01	0.02	0.02	1.25	0.79	0.68	0.40	44	36	26	6	16
(3D) bone_xy-10	0.01	0.01	0.03	0.03	1.18	0.64	0.53	0.29	36	21	27	7	17
(3D) bone_xyz-100	0.39	0.54	0.51	0.51	0.91	0.47	0.32	0.16	57	43	185	36	98
(3D) bone_xyz-10	0.34	0.57	0.46	0.48	0.82	0.36	0.33	0.11	57	21	185	37	98
(3D) bone_xyz_x-100	0.42	0.43	0.42	0.43	0.53	0.14	0.09	0.06	53	48	618	88	256
(3D) bone_xyz_x-10	0.43	0.47	0.39	0.39	0.48	0.12	0.03	0.05	50	23	615	97	290
(3D) bone_xyz_xy-c100	0.24	0.24	0.21	0.21	0.26	0.05	0.03	0.03	29	28	117	25	63
(3D) bone_xyz_xy-c10	0.18	0.22	0.18	0.22	0.24	0.04	0.02	0.02	30	23	120	25	64
(3D) babyface-100	3.6	8.2	9.2	32	33.5	25.7	12.3	9.4	509	346	873	550	1360
(3D) babyface-10	6.2	9.4	7.5	13.9	35.2	14.3	7.9	7.6	543	223	793	420	1162
(3D) bunny-lrg	16.7	28.3	1.28	1.28	186.6	9.5	6.3	41.2	145	52	796	133	406
(3D) bunny-med	1.72	2.72	0.14	0.14	7.47	1.07	1.27	2.27	52	25	94	17	52
(3D) bunny-sml	0.12	0.19	0.11	0.11	0.38	0.11	0.17	0.23	35	16	111	18	58
(3D) liver-100	10.3	15.0	4.88	4.88	38.5	7.3	4.7	4.7	654	503	1682	1444	2873
(3D) liver-10	10.4	15.8	5.06	5.06	33.1	3.4	3.2	3.3	523	407	1586	1290	2754
(3D) abdomen_long	525	701	441	1445	1445	212	110	68	468	349	2532	939	1432
(3D) abdomen_short	578	772	468	1540	1593	119	60	29	485	231	2373	953	1428
(3D) adhead-100	9.5	27.2	0.2	8.5	42.2	10.1	8.1	13.6	208	176	453	104	148
(3D) adhead-10	9.1	25.4	0.2	4.5	42.3	6.3	8.4	10.5	208	105	395	111	176
(2D) BVZ-tsukuba	0.15	0.18	0.02	0.01	0.21	0.31	0.20	0.24	30	25	110	79	50

Table 2: Performance comparison of AAR with BK [10], IBFS [17], and HPF [17] on 3D datasets with 6 connectivity. AAR($< p\%$) denotes the time taken for the algorithm to find a cut whose difference to the optimal cut is $p\%$ of the difference between the cut in the first iteration and the optimal cut. AAR-JD($< p$) denotes time taken by the algorithm to reduce Jaccard Distance to p. AAR(0.1x) is the number of iterations taken by AAR after scaling the pairwise weights by 0.1.

Memory. Apart from running time, we also investigate the memory footprint of the algorithms. Table 1 shows the memory footprint of all algorithms on the *Abdomen* data [1], which is $512 \times 512 \times 551$. AAR uses considerably less memory than the standard algorithms.

7 Conclusion and Future work

We have proposed parallel iterative algorithms for binary energy minimization problems. The algorithms rely on a fast projection subroutine. For binary submodular potentials (graph cuts), this subroutine is simply a total variation problem, which can be efficiently solved on sub-graphs with special structure. In other examples, these subroutines could be fast algorithms for solving cuts on arbitrary subgraphs, or for simpler submodular energies. Hence, while the experiments here concentrate on cuts and decompositions into line graphs, the same methods apply to decompositions into 2D sheets, 3D cubes or any other subgraphs, and to sums of simple higher-order potentials.

We observed that the iterative methods perform similarly to combinatorial methods on 2D grid graphs, and require less memory than other, popular implementations of maximum flow algorithms. The tradeoffs betweem convex and combinatorial methods illustrated here have some interesting implications, and suggest a wider study of integrating combinatorial and convex methods via different decompositions. For example, instead of TV oracles for line graphs, one may use oracles for larger specialized subgraphs. These oracles could use algorithms such as BK, HPF or IBFS, since the projection and TV oracle can be solved by parametric graph cuts. A 3D tensor is easily decomposed into two components: grids and lines. Those more complex subroutines can still be invoked in parallel using AAR. Thus, one can combine convex and combinatorial methods to greatly benefit from the strengths of both.

The convex algorithms admit stochastic variants too [15]. However, the decompositions used in the experiments here (Figure 1) only use a decomposition into 2–4 functions. Finally, given the remarkably improved behavior of iterative methods for smaller weights, it is of great empirical interest to study algorithms that can use homotopies or continuation techniques to start by solving with lower weights and use the ensuing solutions to speed up the medium and high weight regimes.

References

[1] Maxflow dataset online. `http://vision.csd.uwo.ca/maxflow-data`.

[2] F. Bach. *Learning with Submodular Functions: A Convex Optimization Perspective*, volume 6 of *Foundations and Trends in Machine Learning*. NOW, 2013.

[3] A. Barbero and S. Sra. Modular proximal optimization for multidimensional total-variation regularization. Technical Report 1411.0589, ArXiv, 2014.

[4] H. Bauschke, P. L. Combettes, and D. Luke. Finding best approximation pairs relative to two closed convex sets in Hilbert spaces. *J. Approx. Theo.*, 127(2):178–192, 2004.

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

[6] A. Beck and L. Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

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

[8] A. Blake, P. Kohli, and C. Rother, editors. *Markov Random Fields for Vision and Image Processing*. MIT Press, 2011.

[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*, volume 3 of *Foundations and Trends in Machine Learning*. NOW, 2011.

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

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

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

[13] B. G. Chandran and D. S. Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Oper. Res.*, 2009.

[14] L. Condat. A direct algorithm for 1d total variation denoising. Technical report, GREYC laboratory, CNRS-ENSICAEN-Univ. of Caen, 2012.

[15] A. Ene and H. L. Nguyen. Random Coordinate Descent Methods for Minimizing Decomposable Submodular Functions. *ArXiv e-prints*, 2015.

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

[17] A. V. Goldberg, S. Hed, H. Kaplan, R. E. Tarjan, and R. F. Werneck. Maximum flows by incremental breadth-first search. In *Proc. Europ. Conf. on Algorithms*, 2011.

[18] M. Grasmair. The equivalence of the taut string algorithm and BV-regularization. *Journal of Mathematical Imaging and Vision*, 27(1):59–66, 2007.

[19] M. Hein, S. Setzer, L. Jost, and S. Rangapuram. The total variation on hypergraphs - learning on hypergraphs revisited. In *Adv. NIPS*, 2013.

[20] D. Hochbaum. An efficient algorithm for image segmentation, markov random fields and related problems. *J. ACM*, 48(2), 2001.

[21] D. Hochbaum. A polynomial time algorithm for rayleigh ratio on discrete variables: Replacing spectral techniques for expander ratio, normalized cut, and cheeger constant. *Operations Research*, 61(1):184–198, 2013.

[22] D. Hochbaum and S.-P. Hong. About strongly polynomial time algorithms for quadratic optimization over submodular constraints. *Math. Prog.*, 69(1):269–309, 1995.

[23] S. Jegelka, F. Bach, and S. Sra. Reflection methods for user-friendly submodular optimization. In *Adv. NIPS*, 2013.

[24] N. A. Johnson. A dynamic programming algorithm for the fused Lasso and l_0-segmentation. *J. Computational and Graphical Statistics*, 2013.

[25] J. Kappes, B. Andres, C. Schnoerr, F. Hamprecht, S. Nowozin, D. Batra, J. Lellmann, N. Komodakis, S. Kim, B. Kausler, and C. Rother. A comparative study of modern inference techniques for discrete energy minimization problems. In *Proc. CVPR*, 2013.

[26] P. Kohli, M. Kumar, and P. Torr. P3 & beyond: Move making algorithms for solving higher order functions. *IEEE Trans. PAMI*, pages 1645–1656, 2009.

[27] P. Kohli, L. Ladický, and P. Torr. Robust higher order potentials for enforcing label consistency. *Int. Journal of Computer Vision*, 82(3):302–324, 2009.

[28] P. Kohli and P. Torr. Efficiently solving dynamic markov random fields using graph cuts. In *Proc. ICCV*, 2005.

[29] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. PAMI*, 28(10):1568–1583, 2006.

[30] V. Kolmogorov. Minimizing a sum of submodular functions. *Disc. Appl. Math.*, 160(15), 2012.

[31] V. Kolmogorov, Y. Boykov, and C. Rother. Applications of parametric maxflow in computer vision. In *Proc. ICCV*, 2007.

[32] N. Komodakis, N. Paragios, and G. Tziritas. MRF energy minimization and beyond via dual decomposition. *IEEE TPAMI*, 33(3):531–552, 2011.

[33] K. Nagano, Y. Kawahara, and K. Aihara. Size-constrained submodular optimization through minimum norm base. In *Int. Conf. on Machine Learning (ICML)*, 2011.

[34] R. Nishihara, S. Jegelka, and M. Jordan. On the linear convergence rate of decomposable submodular function minimization. In *Neural Information Processing Systems (NIPS)*, 2014.

[35] J. Picard and H. Ratliff. Minimum cuts and related problems. *Neworks*, 5(4), 1975.

[36] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *arXiv:1212.0873v1*, 2012.

[37] R. T. Rockafellar. *Convex Analysis*. Princeton U. P., 1997.

[38] C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer. Optimizing binary MRFs via extended roof duality. In *Proc. CVPR*, 2007.

[39] B. Savchynskyy, S. Schmidt, J. Kappes, and C. Schnörr. A study of Nesterov's scheme for Lagrangian decomposition and MAP labeling. In *Proc. CVPR*, 2011.

[40] D. Sontag and T. Jaakkola. Tree block coordinate descent for MAP in graphical models. In *Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, 2009.

[41] P. Stobbe and A. Krause. Efficient minimization of decomposable submodular functions. In *Adv. NIPS*, 2010.

[42] P. Strandmark and F. Kahl. Parallel and distributed graph cuts by dual decomposition. In *Proc. CVPR*, 2010.

[43] T. Verma and D. Batra. MaxFlow revisited: An empirical comparison of maxflow algorithms for dense vision problems. In *BMVC*, 2012.