Solving relaxations of MAP-MRF problems: Combinatorial in-face Frank-Wolfe directions

Vladimir Kolmogorov
Institute of Science and Technology Austria (ISTA)
Am Campus 1, Klosterneuburg 3400, Austria
vnk@ist.ac.at

Abstract

We consider the problem of solving LP relaxations of MAP-MRF inference problems, and in particular the method proposed recently in [16, 35]. As a key computational subroutine, it uses a variant of the Frank-Wolfe (FW) method to minimize a smooth convex function over a combinatorial polytope. We propose an efficient implementation of this subroutine based on in-face Frank-Wolfe directions, *introduced in [4] in a different context. More generally, we define an abstract data structure for a combinatorial subproblem that enables in-face FW directions, and describe its specialization for tree-structured MAP-MRF inference subproblems. Experimental results indicate that the resulting method is the current state-of-art LP solver for some classes of problems. Our code is available at* pub.ist.ac.at/~vnk/papers/IN-FACE-FW.html.

1. Introduction

The main focus of this paper is on the problem of minimizing a function of discrete variables $z = (z_1, \dots, z_n)$ with unary and pairwise terms:

$$\min_{z \in D_1 \times \dots \times D_n} \sum_{v \in [n]} f_v(z_v) + \sum_{uv \in \mathcal{E}} f_{uv}(z_u, z_v) \quad (1)$$

Here $\mathcal{G} = ([n], \mathcal{E})$ is an undirected graph and $D_1, \dots, D_n$ are finite sets. This problem is often referred to as *MAP-MRF inference* (*maximum a posteriori inference in a Markov Random Field*).

A prominent approach to tackle this NP-hard problem in practice is to solve its natural LP relaxation (see e.g. [39]), also called *Basic LP relaxation* [17]:

$$\min_{\xi \geq \mathbf{0}} \sum_{\substack{v \in [n] \\ a \in D_v}} f_v(a) \xi_{va} + \sum_{\substack{uv \in \mathcal{E} \\ (a,b) \in D_u \times D_v}} f_{uv}(a,b) \xi_{ua;vb} \quad (2a)$$

$$\sum_{b' \in D_v} \xi_{ua;vb'} = \xi_{ua}, \ \sum_{a' \in D_u} \xi_{ua';vb} = \xi_{vb} \quad \forall uv, a, b \quad (2b)$$

$$\sum_{a \in D_v} \xi_{ua} = 1 \quad \forall v \quad (2c)$$

Designing algorithms to (approximately) solve this relaxation for large-scale problems has been a very active area of research. A popular approach is to use *message passing* techniques, which perform a block-coordinate ascent on the dual objective [5, 11, 13, 36, 38, 40] This strategy is very effective for some problems, but for other problems it may get stuck in a suboptimal point. Many techniques have been developed that are guaranteed to converge to the optimal solution of the LP relaxation [8, 9, 16, 18, 20, 21, 25, 26, 28–32, 34, 35].

In this paper we revisit the approach in [16, 35]. Its key computational subroutine is to minimize a quadratic convex function over combinatorial polytope, which is done by invoking a variant of the Frank-Wolfe (FW) algorithm [3]. We study efficient implementations of the latter in the context of MAP-MRF inference. Our main contribution is incorporating *in-face FW directions* introduced in [4]. The idea is to speed-up computations by running FW algorithm on a smaller "contracted" subproblem obtained by taking a face of the polytope containing the current point. It has been used for applications such as low-rank matrix completion [4], cluster detection in networks [1], and training sparse neural networks with ℓ_1 regularization [6]. We investigate the use of in-face FW directions for general combinatorial polytopes, and describe an abstract data structure that enables such directions. We then specialize it to subproblems corresponding to tree-structured MAP-MRF inference problems. Our approach has the following features:

- It may happen that the contracted subproblem splits into independent subproblems. These subproblems are handled by a block-coordinate version of FW.
- We store a cache of "atoms" for each contracted subproblem. We describe how to efficiently transform these atoms when the current face is recomputed.
- For an edge $uv \in \mathcal{E}$ and fixed fractional unary vectors for u, v we can compute an optimal fractional pairwise vector for edge uv by solving a small-scale optimal transportation (OT) problems. Such computations were used in [27] for computing primal feasible solutions of relax-

ation (2). We show how to use them for improving the performance of in-face FW directions.

To our knowledge, the issues above have not been discussed in the literature so far.

We remark that in-face FW directions effectively implement the following rather natural idea: the optimization should be performed only over "active" pairs (v, a) that are likely to be present in the support of an optimal solution; the other pairs should be fixed. A related idea appeared in the context of message passing algorithms in [37], where messages are updated only in a subgraph in which the current best labels keep changing. The method in [37] works only with dual variables, and uses heuristic criteria for choosing the subgraph. We believe that in-face FW directions allow a more principled criterion for deciding which variables to fix and for how long. Note that Freund et al. [4] proved that their criterion retains the convergence rate of the basic FW algorithm. We use a different criterion that also takes into account the ratio of runtimes on the original and on contracted subproblems.

In Section 4 we test the algorithms on benchmark problems in the evaluation [10], and compare them with LP solvers used in [10]. Results suggest that the method in [16, 35] with in-face FW directions is the current state-of-the-art LP solver for certain classes of problems.

2. Background

We will consider a more general problem of minimizing a function represented as a sum of *tractable subproblems*:

$$\min_{X \in \mathbb{R}^d} f(X), \quad f(X) := \sum_{t \in T} f_t(X_{A_t}) \tag{3}$$

Here term $t \in T$ is specified by a subset of variables $A_t \subseteq [d]$ and a function $f_t : \mathbb{R}^{A_t} \to \mathbb{R} \cup \{+\infty\}$ of $|A_t|$ variables. Vector $X_{A_t} \in \mathbb{R}^{A_t}$ is the restriction of vector $X \in \mathbb{R}^d$ to A_t. We assume that the effective domain $\operatorname{dom} f_t = \{X \in \mathbb{R}^{A_t} \mid f_t(X) < +\infty\}$ is a finite non-empty set (implying that (3) is a *discrete* optimization problem); usually one has $\operatorname{dom} f_t \subseteq \{0,1\}^{A_t}$. The arity $|A_t|$ of function f_t can be arbitrarily large, however we assume the existence of an efficient *min-oracle* that for a given vector $y \in \mathbb{R}^{A_t}$ computes $X \in \operatorname*{arg\,min}_{X \in \operatorname{dom} f_t} [f_t(X) + \langle X, y \rangle]$ together with the cost $f_t(X)$.

Note, in the case of problem (1) vector X has the form $X = (X_{va} : v \in [n], a \in D_v)$, where X_{va} is the indicator variable of the event $[z_v = a]$. Thus, we have $X \in \{0,1\}^d$ where $d = \sum_{v=1}^n |D_v|$. Each term $f_t(\cdot)$ is a MAP-MRF problem on a tree-structured graph; they are chosen in such a way that their sum equals the objective function in (1).

Relaxation of (3) We form a relaxation of the problem as in [35]. First, define subsets $\dot{\mathcal{X}}_t, \mathcal{X}_t \subseteq \mathbb{R}^{A_t} \times \mathbb{R}$ via

$$\dot{\mathcal{X}}_t = \{[X \; f_t(X)] : X \in \operatorname{dom} f_t\} \qquad \mathcal{X}_t = \operatorname{conv}(\dot{\mathcal{X}}_t)$$

Throughout the paper we will refer to elements of $\operatorname{dom} f_t$ as "atoms" and to elements as $\dot{\mathcal{X}}_t$ as "extended atoms". For a vector $z \in \mathbb{R}^{A_t} \times \mathbb{R}$ let $z_\star \in \mathbb{R}^{A_t}$ be its first $|A_t|$ components and $z_\circ \in \mathbb{R}$ be its last component (so that $z = [z_\star \; z_\circ]$). Also denote $\dot{\mathcal{X}} = \bigotimes_{t \in T} \dot{\mathcal{X}}_t$ and $\mathcal{X} = \bigotimes_{t \in T} \mathcal{X}_t = \operatorname{conv}(\dot{\mathcal{X}})$. The t-th component of vector $x \in \mathcal{X}$ will be denoted as $x^t \in \mathcal{X}_t$. Problem (3) can now be equivalently written as

$$\min_{\substack{x \in \mathcal{X} \,,\, X \in \mathbb{R}^d \\ x^t_\star = X_{A_t} \in \operatorname{dom} f_t \;\; \forall t \in T}} \sum_{t \in T} x^t_\circ \tag{4}$$

The relaxation is formed by removing non-convex constraints $X_{A_t} \in \operatorname{dom} f_t$. Dualizing constraints $x^t_\star = X_{A_t}$ with Lagrange multipliers $y = (y^t \in \mathbb{R}^{A_t} : t \in T)$ and eliminating variables X yields Lagrangian

$$\begin{aligned} \mathcal{L}(x; y) &= \sum_{t \in T} (x^t_\circ + \langle x^t_\star, y^t \rangle) \\ &= \sum_{t \in T} \langle x^t, [y^t \; 1] \rangle \qquad\qquad (x, y) \in \mathcal{X} \times \mathcal{Y} \end{aligned}$$

where we denoted

$$\mathcal{Y} = \left\{ y : \sum_{t \in T_i} y^t_i = 0 \quad \forall i \in [d] \right\}, \qquad T_i = \{t \in T : i \in A_t\}$$

Note that $\mathcal{L}(x; y)$ is convex in x and concave in y. Define primal and dual objectives via

$$F(x) = \max_{y \in \mathcal{Y}} \mathcal{L}(x; y) \qquad H(y) = \min_{x \in \mathcal{X}} \mathcal{L}(x; y) \tag{5}$$

Assuming strong duality, the optimal value of the relaxation of (4) is then equals

$$\min_{x \in \mathcal{X}} F(x) = \max_{y \in \mathcal{Y}} H(y) \tag{6}$$

The problem becomes to compute saddle point (x, y) that attains the optimum values in (6).

Proximal point algorithm Next, we review a method for solving (6) proposed in [16, 35]. At each iteration it considers a smoothed version of the problem with the following Lagrangian and primal and dual objectives:

$$\begin{aligned} \mathcal{L}_{\gamma,\bar{y}}(x; y) &= \mathcal{L}(x; y) - \tfrac{||y - \bar{y}||^2}{2\gamma} \\ F_{\gamma,\bar{y}}(x) &= \max_{y \in \mathcal{Y}} \mathcal{L}_{\gamma,\bar{y}}(x; y) \\ H_{\gamma,\bar{y}}(y) &= \min_{x \in \mathcal{X}} \mathcal{L}_{\gamma,\bar{y}}(x; y) = H(y) - \tfrac{||y - \bar{y}||^2}{2\gamma} \end{aligned}$$

Here $\gamma > 0$ is a fixed regularization parameter and $\bar{y} \in \mathcal{Y}$ is the current proximal center. The algorithm first selects

arbitrary vector $y_0 \in \mathcal{Y}$, sets $\bar{y}_0 = y_0$, and then computes iterates $(x_n, y_n, \bar{y}_n)$ for $n = 1, 2, \ldots$ using the following equations:

$$x_n \approx \arg\min_{x \in \mathcal{X}} F_{\gamma, \bar{y}_{n-1}}(x) \quad (7a)$$

$$y_n = \arg\max_{y \in \mathcal{Y}} \mathcal{L}_{\gamma; \bar{y}_{n-1}}(x; y) \quad (7b)$$

$$\bar{y}_n = y_n + \alpha_n (y_n - y_{n-1}) \quad (7c)$$

[35] used a simple proximal point method (PPA) in which $\alpha_n = 0$ for all n. [16] used an accelerated version (APPA) where α_n is given by $\alpha_n = \frac{t_n - 1}{t_{n+1}}$ and $t_1, t_2, \ldots$ is a sequence satisfying $t_1 = 1$ and $t_{n-1}^2 - t_n^2 + t_n > 0$ for all $n \geq 2$, e.g. given by the Nesterov's choice $t_{n+1} = (1 + \sqrt{1 + 4t_n^2})/2$.

The main computational bottleneck of the algorithm is the approximate minimization problem in (7a). It is solved using several steps of the Frank-Wolfe (FW) algorithm (or one of its variants) until the FW gap becomes smaller than some threshold ε_n. (The background on FW is given below). As shown in [16], this guarantees that point y_n satisfies $H_{\gamma, \bar{y}_{n-1}}(y_n) \geq \max_{y \in \mathcal{Y}} H_{\gamma, \bar{y}_{n-1}}(y) - \varepsilon_n$, and yields accuracy $O(1/n^2)$ after n iterations[1] assuming that $\varepsilon_n = O(1/n^{4+\delta})$ for some $\delta > 0$.

Frank-Wolfe algorithms For brevity, let us denote $\bar{y} = \bar{y}_{n-1}$ and $\tilde{F}(x) \stackrel{\text{def}}{=} F_{\gamma, \bar{y}}(x)$. Expanding this, we obtain

$$\tilde{F}(x) = \sum_{t \in T} \left(\tfrac{\gamma}{2} ||x_\star^t||^2 + \langle x^t, [\bar{y}^t \ 1] \rangle \right) - \sum_{i=1}^{d} \tfrac{|T_i|}{2\gamma} \nu_i^2,$$

$$\nu_i = \nu_i(x) = \tfrac{1}{|T_i|} \sum_{t \in T_i} (\gamma \cdot x_i^t + \bar{y}_i^t)$$

This is a convex differentiable function over polytope $\mathcal{X}$. Its gradient is given by

$$\nabla_t \tilde{F}(x) = [y^t \ 1], \qquad y^t = y^t(x) = \gamma \cdot \bar{x}_\star^t + \bar{y}^t - \nu_{A_t}$$

Note that $y = y(x) = (y^t(x))_{t \in T}$ is the vector that maximizes $\mathcal{L}_{\gamma; \bar{y}_{n-1}}(x; y)$ in eq. (7b). The basic Frank-Wolfe algorithm with line search [3] minimizes $\tilde{F}$ by iteratively repeating the following steps:
(1) compute $\nabla \tilde{F}(x)$ at current point x;
(2) compute $s \in \arg\min_{s \in \mathcal{X}} \langle \nabla \tilde{F}(x), s \rangle$;
(3) define $x^\gamma = x + \gamma(s - x)$, compute $\gamma = \arg\min_{\gamma \in [0,1]} \tilde{F}(x^\gamma)$;
(4) update $x := x^\gamma$.
Note that step (2) requires calling min-oracles for each subproblem $t \in T$. The quantity $\texttt{gap}^{\texttt{FW}}(x; \tilde{F}) = \langle \nabla \tilde{F}(x), x - s \rangle = \max_{s \in \mathcal{X}} \langle \nabla \tilde{F}(x), x - s \rangle$ is called the *Frank-Wolfe gap* at x. It upper-bounds the suboptimality gap: $\texttt{gap}^{\texttt{FW}}(x; \tilde{F}) \geq \tilde{F}(x) - \min_{x' \in \mathcal{X}} \tilde{F}(x')$.

[1] The bound $O(1/n^2)$ holds for the dual, primal and infeasibility gaps, see [16] for details.

Several techniques have been proposed in the literature for speeding-up the basic algorithm. We will make use of the following ideas.

- Block-Coordinate Frank-Wolfe (BCFW) [19]. At each step it updates only variables x^t for some fixed $t \in T$ while keeping all other components fixed. More precisely, it computes partial derivatives $\nabla_t \tilde{F}(x)$, then computes $s^t \in \arg\min_{s^t \in \mathcal{X}_t} \langle \nabla_t \tilde{F}(x), s^t \rangle$ and defines x^γ via $(x^\gamma)_t = x^t + \gamma(s^t - x^t)$ and $(x^\gamma)_{t'} = x^{t'}$ for $t' \neq t$. The rest is as above.
- Caching atoms [7, 23, 33]. The idea is store "atoms" $s^t \in \dot{\mathcal{X}}_t$ returned by the t-th min-oracle in cache $\mathcal{W}_t \subset \dot{\mathcal{X}}_t$. Iterations are then divided into "exact" iterations which call the (expensive) min-oracle that optimize over $\mathcal{X}_t$, and "approximate" iterations which optimize over $\mathcal{W}_t$.
- Optimizing the objective over extended atoms in $\mathcal{W}_t$ (i.e. subject to the constraint $x^t \in \texttt{conv}(\mathcal{W}_t)$, assuming that current x satisfies this constraint). This amounts to minimizing a smooth convex function (quadratic in our case) over a simplex. An example is the BCG method [2].
- In-face Frank-Wolfe directions [4]. In addition to "regular" FW steps, this method performs the following operations: find a face $\mathcal{X}' \subseteq \mathcal{X}$ of polytope $\mathcal{X}$ containing current point x (e.g. the minimal such face), and run several FW steps to minimize function $\tilde{F}$ over $\mathcal{X}'$. This may lead to a speed-up if oracle calls over $\mathcal{X}'$ are faster than over $\mathcal{X}$. We will say that $\mathcal{X}'$ is obtained from $\mathcal{X}$ via "contraction", and refer to FW steps over $\mathcal{X}'$ as "contracted steps". The same terminology will be used for subproblems $\mathcal{X}_t$.

3. Our implementation

Following [7, 23, 33], for each subproblem t we maintain cache $\mathcal{W}_t \subseteq \dot{\mathcal{X}}_t$ of extended atoms. We implemented two versions:
(1) x^t is represented as a convex combination of elements in $\mathcal{W}_t$. Then we store coefficients of this combination. Extended atoms with zero weight are immediately removed.
(2) Cache size is limited by a constant (10 in our implementation). As in previous works, we maintain a timestamp for each atom, updating it whenever the atom is "accessed" (i.e. returned as optimal in one of the operations). If the cache becomes too big, we remove an atom with the oldest timestamp.
Let "conv" be the flag that specifies the version: $\texttt{conv} = \texttt{true}$ corresponds to option (1) and $\texttt{conv} = \texttt{false}$ corresponds to option (2).

In-face FW directions As stated in the introduction, the main motivation of this paper is to incorporate in-face FW directions for combinatorial subproblems into the framework of [16, 35]. We focus on faces of polytopes $\mathcal{X}'_t \subseteq \mathcal{X}_t$

that are specified by constraints of the form $x_i^t = \texttt{const}_i$ for some $i \in A_t$. [2] The following issues should be taken into account:

- Fixing some variables may split subproblems into several independent subproblems (that we call "contracted subproblems"), and so variables of x^t decouple into independent blocks. Optimizing over $x^t \in \mathcal{X}_t'$ should be done via the block-coordinate version of FW.
- Atoms in the cache $\mathcal{W}_t$ for the original subproblem t should be transformed to atoms of contracted subproblems, and vice versa. Ideally, the time for processing each atom should depend on the size of the contracted subproblems (in the case when $|\mathcal{W}_t| \gg 1$).

To address these issues, we will describe an abstract data structure called `Subproblem`. We will later show to implement it for subproblems t corresponding to the MAP-MRF inference problems on tree-structured graphs.

We distinguish between "parent" and "child" subproblems. The former are added to the solver during initialization. The user should specify mapping A_t when adding parent subproblem (which is an array size $|A_t|$). Now suppose that the algorithm decides to contract parent subproblem t. Given current vector x, the user should first partition set A_t as $A_t = A_{t0} \cup A_{t1} \cup \ldots \cup A_{tk}$ for some $k \geq 1$ where A_{t0} is the set of components that will be fixed to their current values, and $A_{t1}, \ldots, A_{tk}$ correspond to the independent subproblems. Let us write vectors $z \in \mathbb{R}^{A_t}$ as $(z_{A_{t0}}, z_{A_{t1}}, \ldots, z_{A_{tk}})$. We require the contraction operation to satisfy the following conditions:
(1) There should exist functions $f_{ti} : \{0,1\}^{A_{ti}} \to \mathbb{R} \cup \{+\infty\}$ so that

$$f_t(x_{A_{t0}}, X^1, \ldots, X^k) = f_{t1}(X^1) + \ldots + f_{tk}(X^k)$$
$$\forall (X^1, \ldots, X^k) \in \operatorname{dom} f_{t1} \times \ldots \times \operatorname{dom} f_{tk}$$

(2) For each $i \in [k]$ there should exist vector $\bar{x}^i \in \mathcal{X}_{ti}$ with $\bar{x}^i_\star = x^t_{A_{ti}}$ so that $\bar{x}^1_\circ + \ldots + \bar{x}^k_\circ = x_\circ$, where we denoted $\dot{\mathcal{X}}_{ti} = \{[X \ f_{ti}(X)] : X \in \operatorname{dom} f_{ti}\}$ and $\mathcal{X}_{ti} = \operatorname{conv}(\dot{\mathcal{X}}_{ti})$.
(3) Extended atom $s^t \in \arg\min_{s^t \in \dot{\mathcal{X}}_t} \langle \nabla_t \tilde{F}(x), s^t \rangle$ should satisfy $s^t_{A_{ti}} \in \operatorname{dom} f_{ti}$ for each $i \in [k]$.

Accordingly, each parent subproblem should implement function $\texttt{Contract}(x^t, s^t)$. This function should find a decomposition into k child subproblems as above, and return these subproblems together with mappings $A_{t1}, \ldots, A_{tk}$ and values $\bar{x}^1_\circ, \ldots, \bar{x}^k_\circ$. Note that the latter values are actually not used if $\texttt{conv} = \texttt{true}$, since in this case the solver has enough information to recompute them. However, if $\texttt{conv} = \texttt{false}$ then the solver does need values $\bar{x}^1_\circ, \ldots, \bar{x}^k_\circ$.

Below we will use index symbol t to denote both parent and child subproblems (together with notation A_t, x^t, f_t, etc). Note, for child subproblems we have $t = t'i$ for some parent subproblem t'.

Compact representation of atoms In many combinatorial problems, atoms (i.e. elements of $\operatorname{dom} f_t$) can be described in a compact way. Accordingly, for each subproblem we introduce type `Atom` whose implementation should be specified by the user. The user should implement the following basic functions:

- $\texttt{AtomToVector}(a)$: compute extended atom $[a \ f(a)] \in \dot{\mathcal{X}}_t$ corresponding to atom a.
- $\texttt{DotProduct}(a, g)$: compute inner product $\langle a, g \rangle$ of atom a and vector $g \in \mathbb{R}^{A_t}$.
- $\texttt{WeightedDotProduct}(a, b)$: compute $\sum_{i \in A_t} (1 - \frac{1}{|T_i|}) a_i b_i$ for atoms a, b.
- $\texttt{MinOracle}(g)$: return atom $a \in \arg\min_a (\langle a, g \rangle + f_t(a))$.

Next, we discuss how to transform atoms during contractions. Each parent subproblem t must store a current atom denoted as a^t, and implement functions $t::\texttt{SetAtom}(a)$ and $t :: \texttt{GetAtom}()$ that respectively set and return a^t. Child subproblems ti must also implement these two functions, but they are now defined as follows. $ti :: \texttt{SetAtom}(a)$ should set components of the parent atom a^t to a, i.e. update $a^t_{A_{ti}} := a$. Similarly, $ti :: \texttt{GetAtom}()$ should return atom $a^t_{A_{ti}}$ of the child subproblem (or `NULL`, if $a^t_{A_{ti}} \notin \operatorname{dom} f_{ti}$).

We now describe how these functions are used. Consider subproblem t which has been contracted to child subproblems $t1, \ldots, tk$ via the call $\texttt{Contract}(x, s)$ (for brevity, we write x, s instead of x^t, s^t). By construction, atom a^t will satisfy $a^t_{A_{t0}} = s_{A_{t0}}$ where A_{t0} are the variables that have been fixed. Now suppose that the solver decides to contract t again (i.e. recompute child subproblems). The solver first computes atom $s' \in \arg\min_{s' \in \dot{\mathcal{X}}_t} \langle \nabla_t \tilde{F}(x'), s' \rangle$ where $x' \in \mathcal{X}_t$ is the current vector for t. Then it calls $\texttt{Contract}(x', s')$ which returns new child subproblems $t1', \ldots, tk'$. If $\texttt{conv} = \texttt{false}$ then caches $\mathcal{W}_{t1'}, \ldots, \mathcal{W}_{tk'}$ are set by repeating the following steps:
(1) for each $i \in [k]$ pick atom $a^i \in \mathcal{W}_{ti}$ (in a round-robin fashion), call $ti::\texttt{SetAtom}(a^i)$;
(2) for each $i' \in [k']$ call $a^{i'} \leftarrow ti' :: \texttt{GetAtom}()$, add $[a^{i'} \ f_{ti'}(a^{i'})]$ to $\mathcal{W}_{ti'}$ (if $a^{i'} \neq \texttt{NULL}$).
If $\texttt{conv} = \texttt{true}$ then we use $ti :: \texttt{SetAtom}(\cdot)$ and $ti' :: \texttt{GetAtom}()$ in a similar way to restore the desired invariant, i.e. make sure that $x^{ti'}$ is a convex combination of current atoms for each $i' \in [k']$; details are omitted. In the end of the contraction operation we call $t::\texttt{SetAtom}(s')$.

3.1. MAP-MRF tree subproblems

We now detail the implementation of `Subproblem` in the case when $f_t(\cdot)$ corresponds to the MAP-MRF inference problem on a tree-structured graph. In other words, we assume that $f_t(\cdot)$ corresponds to problem (1) (with labelings

[2] Our implementation also supports constraints of the form $x^t_i = x^t_j$ for some $\{i, j\} \subseteq A_t$, but we haven't used it in experiments.

z represented via vectors $X \in \{0,1\}^d$) in which the graph $([n], \mathcal{E})$ is a tree. Recall than $d = \sum_{v=1}^{n} |D_v|$.

An atom is stored in a natural way as a vector of size n (which can be much smaller than $\texttt{dim}(X) = d$). Next, we discuss the implementation of function $\texttt{Contract}(x, s)$ for vector $x \in \mathbb{R}^{d+1}$ and atom $s \in D_1 \times \ldots \times D_n$. First, we find pairs (v, a) satisfying $x_{va} = 0$ and $s_v \neq a$, and force constraint $X_{va} = 0$ for such pairs (assuming that there are least 25% of such pairs; otherwise we do not contract). Let $\mathcal{V}_{\texttt{fixed}} \subseteq [n]$ be the set of nodes v for which $|D_v| - 1$ variables X_{va} has been forced to 0, and let $\mathcal{V} = [n] - \mathcal{V}_{\texttt{fixed}}$. Clearly, the problem splits into $k \geq 0$ independent subproblems corresponding to the trees $\mathcal{G}_1, \ldots, \mathcal{G}_k$ of the induced forest $\mathcal{G}[\mathcal{V}]$. Each node of the child subproblem has a pointer to the corresponding node of the parent subproblem (and vice versa). Clearly, this allows an efficient implementation of functions $\texttt{GetAtom}()$ and $\texttt{SetAtom}(\cdot)$.

As discussed in the previous section, we need to be able to compute values $\bar{x}^k_\circ$ for each child subproblem $i \in [k]$ (in the case when $\texttt{conv} = \texttt{false}$). For that we need to know the cost of (fractional) vector x restricted to tree $\mathcal{G}_i$. This can be easily computed if we know the "fractional cost" ξ_{uv} of each edge $uv \in \mathcal{E}$ in x. This fractional cost must have the following form for some vector $(\xi_{ua;vb})_{a,b}$:

$$\xi_{uv} = \sum_{a \in D_u, b \in D_v} f_{uv}(a,b) \xi_{ua;vb} \tag{8a}$$

$$\sum_{b \in D_v} \xi_{ua;vb} = x_{ua} \qquad \forall a \in D_u \tag{8b}$$

$$\sum_{a \in D_u} \xi_{ua;vb} = x_{vb} \qquad \forall b \in D_v \tag{8c}$$

Accordingly, we additionally maintain $|\mathcal{E}|$ real numbers for the current vector x as described below.

One possibility would be to increase the dimensions of vector X, i.e. let $X = (X_{va} : v \in [n], a \in D_v) \sqcup (X_{uv} : uv \in \mathcal{E})$. If X corresponds to atom $z = (z_1, \ldots, z_n)$ then $X_{uv} = f_{uv}(z_u, z_v)$. The solver would then automatically maintain the desired fractional cost $\xi_{uv} = X_{uv}$ for each edge uv. We opted for another approach: we store costs ξ_{uv} internally at the given subproblem. Note that solver changes current x via updates of the form $x := \alpha_0 x + \sum_{i=1}^{r} \alpha_i a^i$ where $a^1, \ldots, a^r$ are extended atoms and $\alpha_0, \alpha_1, \ldots, \alpha_r$ are non-negative coefficients that sum to 1. Before applying such update, the solver calls function $\texttt{Update\_x}(\alpha_0, x, \alpha_1, a^1, \ldots, \alpha_r, a^r)$ for the the given subproblem, and this function updates ξ_{uv} for all edges uv. This allows the solver to manipulate with vectors of smaller sizes.

Optimal transportation problem Note that minimizing the value of ξ_{uv} over $(\xi_{ua;vb})_{a,b}$ in (8) (for fixed vectors $(x_{ua})_a$ and $(x_{vb})_b$) is a classical optimal transportation (OT) problem. It is usually small-scale; we solve it via a successive shortest path algorithm. We use in two ways.

First, we observed that sometimes the FW approach with option $\texttt{conv} = \texttt{false}$ makes some components of vector x extremely small, e.g. 10^{-20} or 10^{-30}. This can be explained as follows. Suppose that the value x_{va} becomes positive in the early stage of the algorithm, but later on all atoms z satisfy $z_v \neq a$. Furthermore, suppose that optimizing the objective over $\texttt{conv}(x^t \cup \mathcal{W}_t)$ always assigns a positive weight to x^t, and this weight is smaller than some constant $c < 1$ sufficiently often. Then after n such steps x_{va} becomes smaller than c^n.

To address this issue, we do the following from time to time (if $\texttt{conv} = \texttt{false}$): we go through all subproblems t and update x^t by calling $x^t \leftarrow t :: \texttt{Adjust\_x}(x^t)$. The latter function is implemented as follows for tree-structured MAP-MRF subproblems: we change all fractional components smaller than 10^{-8} to zero, renormalize unary fractional components for each node so that they sum to 1, and then compute the fractional cost ξ_{uv} for each edge uv and the total cost by solving OT problems. The frequency of such updates is set so that their time is at most 20% of the total time. More precisely, we run the $\texttt{Adjust\_x}$ loop after each update of proximal center $\bar{y}$ assuming that $\tau > 5\tau_0$ where τ_0 is the time spent in the previous $\texttt{Adjust\_x}$ loop and τ is the time that elapsed after the last $\texttt{Adjust\_x}$ loop. This scheme improved the performance of in-face FW directions for some instances, since it led to smaller contracted subproblems. For the algorithm without in-face FW directions this scheme affected the performance only marginally.

Following [27], we also use the OT procedure for computing feasible primal solutions of relaxation (2). Note that vectors $x \in \mathcal{X}$ do not directly give such a solution, since for each node $v \in [n]$ and label $a \in D_v$ variables x^t_{va} will in general be different for different subproblems t containing v. To circumvent this problem, we compute the average of values x^t_{va} over t containing v; this gives a feasible fractional solution for nodes. The corresponding fractional solution for edges (and the LP cost of this solution) is then computed by solving OT problems.

3.2. Perfect Matching subproblems

We also experimented with subproblems corresponding to functions of the following form: $f_t(X) = \langle C, X \rangle$ if $X \in \{0,1\}^{\mathcal{E}}$ is a perfect matching in an undirected graph $(\mathcal{V}, \mathcal{E})$, and $f_t(X) = +\infty$ otherwise. We obtained such subproblems by considering quadratic pseudo-boolean optimization problems (minimize $f(z) = \sum_v c_v z_v + \sum_{uv} c_{uv} z_u z_v$ over $z \in \{0,1\}^n$), and tightening it with planar subproblems [41]; the latter are solved by a reduction to the perfect matching problem. A face of the perfect matching polytope can be described by constraints of the form

$$X(\mathcal{E}_0) = \mathbf{0} \tag{9a}$$

$$X(\delta S) = 1 \qquad \forall S \in \mathcal{S} \tag{9b}$$

where $\mathcal{E}_0 \subseteq \mathcal{E}$, $\mathcal{S}$ is a laminar family of odd-cardinality subsets of $\mathcal{V}$, and δS is the set of edges of the cut $(S, \mathcal{V} - S)$. Given a fractional point X in the polytope, computing the minimal face containing X boils down to computing a Gomory-Hu tree in graph $(\mathcal{V}, \mathcal{E})$ with appropriate weights [24].

In our informal experiments (not reported here) we were not able to obtain a speed-up with in-face FW directions. First, we observed that set $\mathcal{E}_0$ is usually empty (or very small). This could be due to the way that our planar subproblems are formed (we add to planar subproblems odd cycles that are currently violated). The runtime of Gomory-Hu computations was negligible using the recent code of [14, 15]. However, adding constraints (9b) did not not make the Blossom V code [12] faster. Note that we incorporated constraints (9b) by adding a large constant to edge weights of edges in sets δS for $S \in \mathcal{S}$.

We conjecture that in-face FW directions could still be beneficial either (i) in applications where set $\mathcal{E}_0$ is large, or (ii) if the perfect matching code is modified so that constraints (9b) are incorporated more directly. We leave this as a future work.

3.3. Frank-Wolfe algorithm

In this section we describe the FW version that we have implemented. One basic operation is to minimize $\tilde{F}(\dots, x^t, \dots)$ over $x^t \in \texttt{conv}(Z)$ for some finite set $Z = \{z^1, \dots, z^k\}$; all other components except for x^t are fixed.[3] Denote $x^t = \sum_{i=1}^k \alpha_i z^i$, then equivalently we need to minimize quadratic function $f(\alpha) = \frac{1}{2}\alpha^T A \alpha + b^T \alpha$ over simplex $\alpha \in \{\alpha \in \mathbb{R}^k_{\geq 0} : \sum_{i=1}^k \alpha_i = 1\}$. Define vector $\lambda \in \mathbb{R}^{A_t}$ via $\lambda_i = \sum_{t \in T_i} \frac{\gamma x_i^t}{|T_i|} - \nu_i(x)$ where x is the current vector, then matrix $A \in \mathbb{R}^{k \times k}$ and vector $b \in \mathbb{R}^k$ are given by $A_{ij} = \gamma \cdot \texttt{WeightedDotProduct}(z^i, z^j)$ and $b_i = \langle z^i, [\bar{y}^t \ \ 1] \rangle + \langle z^i_*, \lambda \rangle$. Since values $\texttt{WeightedDotProduct}(a, b)$ for $a, b \in \mathcal{W}_t$ are needed multiple times, we store them for each cache $\mathcal{W}_t$ in a matrix of size $|\mathcal{W}_t| \times |\mathcal{W}_t|$.

To minimize $f(\cdot)$, we implemented a projected conjugate gradient descent (PCG) method with the following modification. Suppose that a step takes the procedure outside the simplex, i.e. the weight α_i of point z^i becomes negative. We then stop at the boundary (making $\alpha_i = 0$), remove z^i from Z, and restart PCG. Thus, each step either makes Z smaller (which can happen at most $|Z| - 1$ times) or makes a sufficient progress on the objective (as guaranteed by PCG). We stop when the FW gap is reduced by a factor of 10. This can be viewed as a particular implementation of the *Simplex Descent Oracle* in the BCG method [2].

We use three types of operations: `CacheLoop`, `OracleLoop` and `ContractLoop`. In the first two we cycle through child subproblems in a random order, and in the third one we cycle through parent subproblems in a random order. For each subproblem t we do the following.

- `CacheLoop`: run Simplex Descent Oracle (`SiDO`) described above (for $Z = \mathcal{W}_t$ or $Z = \mathcal{W}_t \cup \{x^t\}$, depending on flag `conv`).
- `OracleLoop`: run `SiDO`; compute $\nabla_t \tilde{F}(x)$; call t-th oracle: $s^t \leftarrow \arg\min_{s^t \in \mathcal{X}_t} \langle \nabla_t \tilde{F}(x), s^t \rangle$; add s^t to $\mathcal{W}_t$; run `SiDO` again. Note that `SiDO` is usually much faster than the min-oracle (especially if $|\mathcal{W}_t|$ is small), so using it before the oracle call is a cheap way to get a better gradient $\nabla_t \tilde{F}(x)$.
- `ContractLoop`: compute $\nabla_t \tilde{F}(x)$; call t-th oracle: $s^t \leftarrow \arg\min_{s^t \in \mathcal{X}_t} \langle \nabla_t \tilde{F}(x), s^t \rangle$; call $\texttt{Contract}(x^t, s^t)$; update child subproblems and transform atoms as described in the previous section.

An important question is how often each operation should be run. In general, `CacheLoop` makes a smaller progress compared to `OracleLoop` (i.e. improves the objective by a smaller amount), and similarly in-face FW steps can be expected to make a smaller progress compared to regular steps. However, the former operations can be much faster. We thus argue that the criterion for choosing the operation should be based on the actual runtimes. To choose between `CacheLoop` and `OracleLoop`, we use the same method as in [33]. Namely, we define procedure `InnerIteration` as follows. It calls `OracleLoop` once and then `CacheLoop` one or more time. Let Δ_i be the improvement in the objective after the i-th call, and τ_i be the time after the i-th call (both relative to the beginning of `InnerIteration`). We terminate the iteration if after the i-th call we have $\frac{\Delta_i}{\tau_i} \leq \frac{\Delta_{i-1}}{\tau_{i-1}}$ (for $i \geq 2$). We use a similar technique to choose between `InnerIteration` and `ContractLoop`: we define procedure `OuterIteration` that calls `ContractLoop` once and then `InnerIteration` one or more time, using a similar termination criterion.

Proximal Point Algorithm Following [16], we use inexact Accelerated Proximal Point Algorithm (APPA). The n-subproblem is solved to accuracy $\varepsilon_n = \varepsilon_0 / n^2$ where ε_0 is the initial FW gap. Unlike [16], we restart the APPA method if the the primal objective $\tilde{F}(x)$ becomes worse; this is a well-known strategy in the case of (exact) accelerated algorithms [22].

Numerical stability As we described in Section 3.1, if $\texttt{conv} = \texttt{false}$ then some components of vector x may become extremely small. A similar issue may happen with $\texttt{conv} = \texttt{true}$: weights of some atoms may become small, e.g. $1e^{-20}$. As a result, the algorithm may get stuck, or $|\mathcal{W}_t|$ may grow very large. To avoid this issue, we use a similar technique as in Section 3.1: we simply remove atoms whose weight is smaller than $1e^{-8}$.

[3] If $\texttt{conv} = \texttt{true}$ then $Z = \mathcal{W}_t$, and if $\texttt{conv} = \texttt{false}$ then $Z = \mathcal{W}_t \cup \{x^t\}$ where x^t is the current vector.

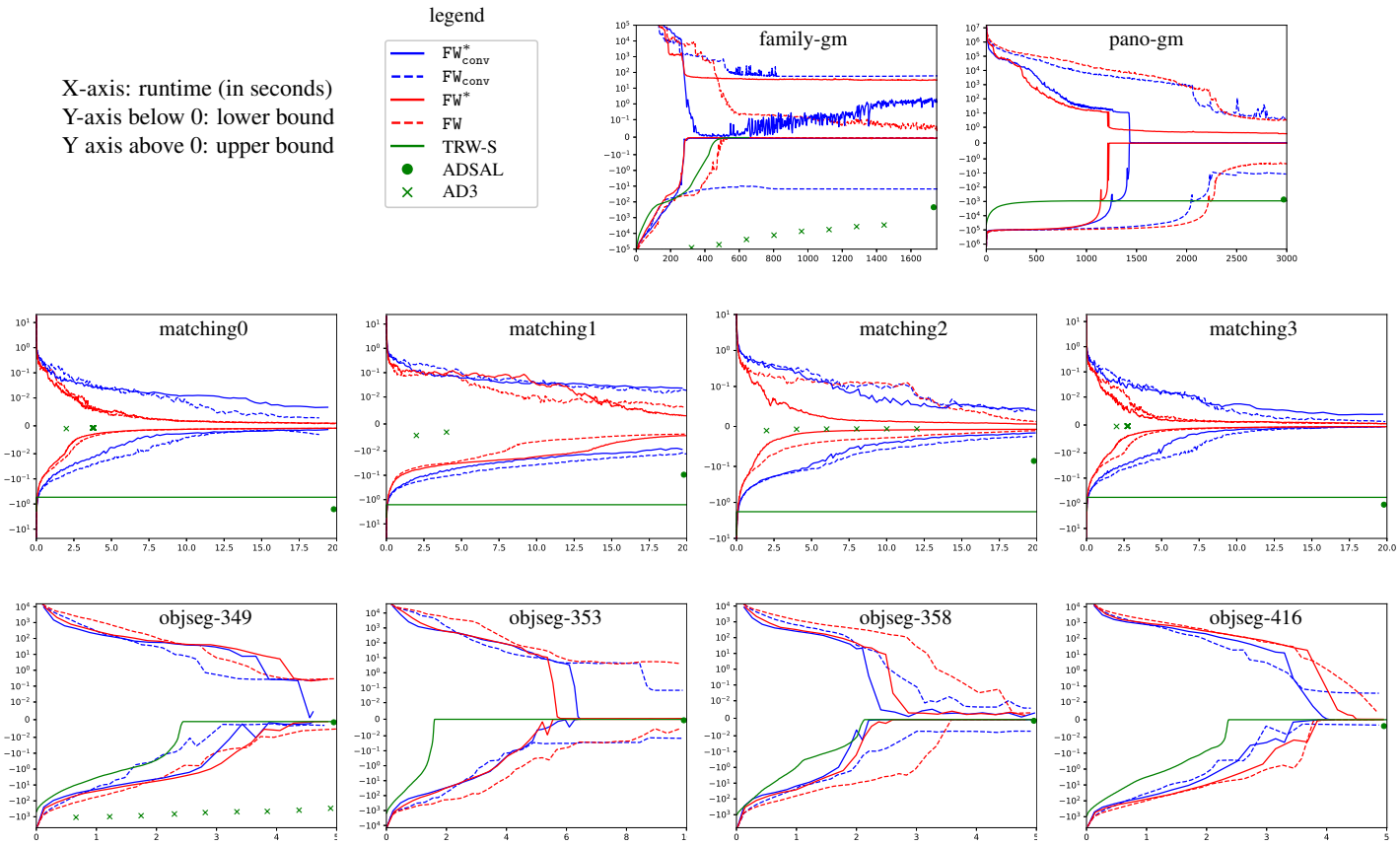

Figure 1. Top row: 'mrf-photomontage' ($\gamma = 10$). Middle row: 'matching' ($\gamma = 1$). Bottom row: 'object-seg' ($\gamma = 1$).

4. Experimental results

In this section we test different algorithms on (pairwise) MAP-MRF inference problems from the OPENGM2 benchmark study [10]. Unlike [10], we do not aim to obtain the best possible solution of (1). Instead, we investigate what is the best algorithm for solving its BLP relaxation (2). Note that this is a well-defined subproblem, and used as a subroutine for other solvers in [10] (branch-and-bound techniques, techniques combining LP solvers with persistency criteria, etc).

Accordingly, we included TRW-S method [11] and ADSAL method [28] that seem to dominate other LP solvers according to the results of [10]. On a subset of instances we also ran the Gurobi solver (ver. 10.0.0) and the ADMM-based "AD3" method [21].[4] Gurobi runtimes were as follows (in seconds, using 4 physical cores; for the first instance Gurobi aborted, probably because of requesting too much memory: we used a machine with 16Gb RAM, out of which 12Gb was free):

family-gm	matching0	objseg-349	1CKK
-	1.36	24.05	3795

We tested 4 versions of FW algorithms that we term as $\texttt{FW}, \texttt{FW}_{\texttt{conv}}, \texttt{FW}^*, \texttt{FW}^*_{\texttt{conv}}$. Subscript $\texttt{conv}$ indicates option $\texttt{conv} = \texttt{true}$, while "*" means that in-face FW directions are used. The FW method depends on parameter γ; values that are too small or too large result in slow convergence (see [35]). For each family of problems we used the same γ of the form $\gamma = 10^k$, $k \in \mathbb{Z}$ with the best performance on one of the instances. Since the ratios between allowed γ's are rather large, we believe that it should be feasible to learn such γ (or even fine-tune it) for a given application using parameters such as the number of subproblems, etc.

In Fig. 1 and 2 we plot lower and upper bounds as functions of time. We added a constant to all values so that the average of the best known lower and upper bounds is zero, and then used symmetric log scaling ('symlog' in python).

[4] The code of https://github.com/andre-martins/AD3 includes parameter η and also a possibility to adapt this parameter during optimization. It appears that [10] used the default value of η, namely $\eta = 0.1$. We noticed that η was changed very rarely during optimization (usually halved once). We turned off adapt_eta flag and hand-picked η which gave the best result.

Note, objseg-349 uses Potts interaction potentials. This is exploited in our implementation (via distance transforms) but not in AD3 and not in the reduction to Gurobi.

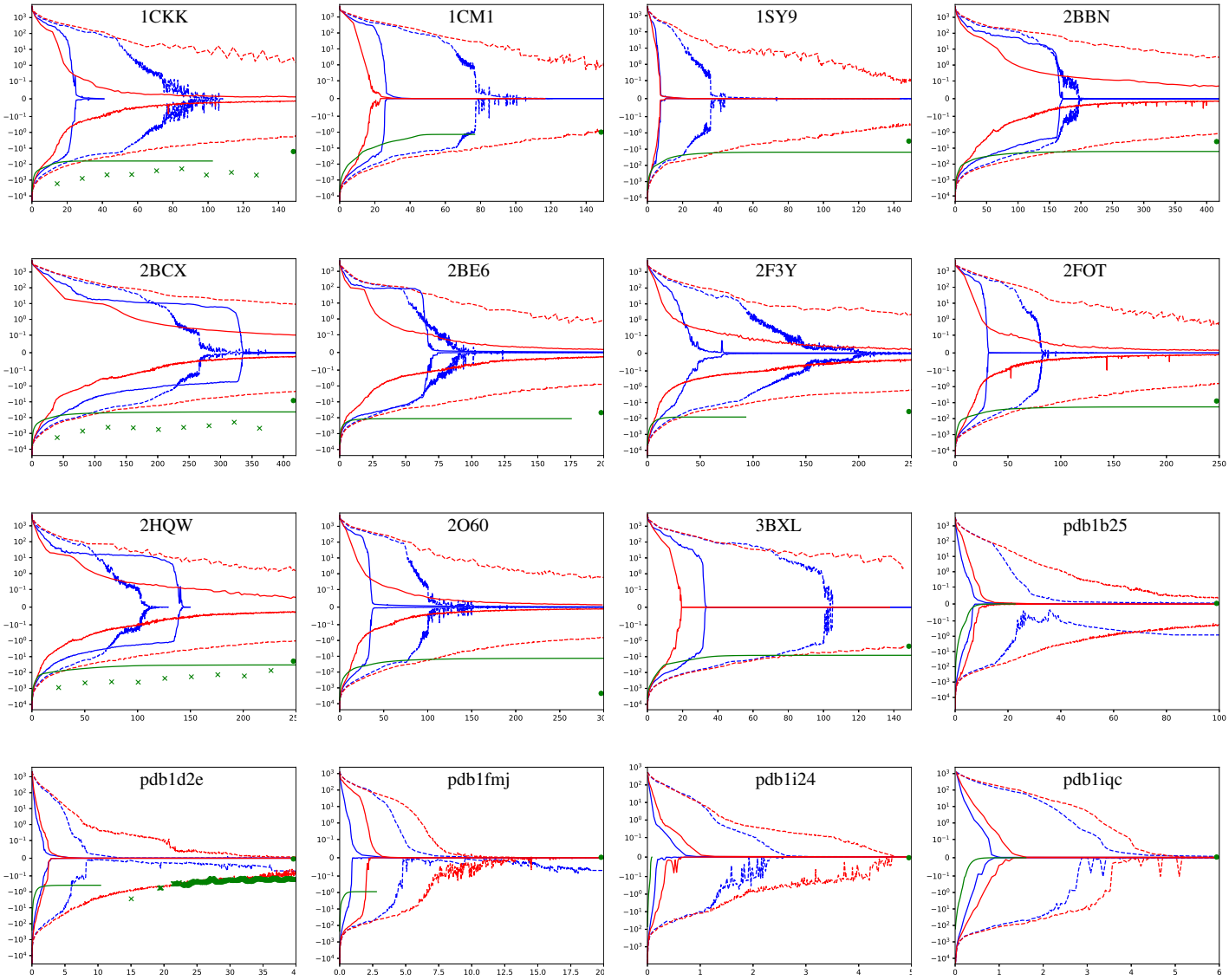

Figure 2. Results on 'protein-folding' ($\gamma = 100$).

Note that the time for computing the upper bound (via OT) was not counted. The result of ADSAL is a single point that we copied from [10]. The runtime of ADSAL was always larger than the maximum X-range in the plots (sometimes significantly), even though we used a slower machine: Intel Core i5-10210U CPU @ 1.60GHz and 16Gb RAM vs. Intel Core i5-4570 CPU @ 3.20GHz and 32GB RAM used in [10].

Discussion We can see from the plots that $\texttt{FW}^*_\sigma$ significantly outperforms $\texttt{FW}_\sigma$ on 'mrf-photomontage' and 'protein-folding' (both for $\sigma = \texttt{conv}$ and empty σ), as well as TRW-S and ADSAL. [5] This suggests that FW with interface FW directions is the current state-of-the-art LP solver for these applications. On two other applications, however ('matching' and 'object-seg') $\texttt{FW}^*_\sigma$ and $\texttt{FW}_\sigma$ are roughly similar and in general outperformed by other techniques, e.g. AD3 & Gurobi on 'matching' and TRW-S on 'object-seg'.

On most plots upper and lower bounds seem to be converging to each other. We conclude that the FW approach can be used for finding both primal and dual solutions of relaxation (2).

[5] As a single exception, FW starts giving better upper bound than FW* once the lower bound reaches the optimal value (after roughly 500 seconds). We do not have an explanation for this behavior. Perhaps, an alternative method for extracting a primal solution should be used in this regime.

References

[1] Immanuel M. Bomze, Francesco Rinaldi, and Damiano Zeffiro. Fast cluster detection in networks by first order optimization. *SIAM J. MATH. DATA SCI.*, 4(1):285–305, 2022. 1

[2] G. Braun, S. Pokutta, D. Tu, and S. Wright. Blended conditional gradients: the unconditioning of conditional gradients. In *International Conference on Machine Learning (ICML)*, 2019. 3, 6

[3] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:149–154, 1956. 1, 3

[4] Robert M. Freund, Paul Grigas, and Rahul Mazumder. An extended Frank-Wolfe method with "in-face" directions, and its application to low-rank matrix completion. *SIAM J. Optimization*, 27(1):319–346, 2017. 1, 2, 3

[5] A. Globerson and T. S. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Conference on Neural Information Processing Systems (NIPS)*, 2007. 1

[6] Paul Grigas, Alfonso Lobos, and Nathan Vermeersch. Stochastic in-face Frank-Wolfe methods for non-convex optimization and sparse neural network training. *CoRR*, arXiv:1906.03580, 2019. 1

[7] T. Joachims, T. Finley, and C. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 1:27–59, 2009. 3

[8] J. Johnson, D. M. Malioutov, and A. S. Willsky. Lagrangian relaxation for map estimation in graphical models. In *45th Annual Allerton Conference on Communication, Control and Computing*, 2007. 1

[9] V. Jojic, S. Gould, and D. Koller. Accelerated dual decomposition for MAP inference. In *International Conference on Machine Learning (ICML)*, 2010. 1

[10] J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, T. Kröger, J. Lellmann, N. Komodakis, B. Savchynskyy, and C. Rother. A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, 115(2):155–184, 2015. 2, 7, 8

[11] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1568–1583, Oct. 2006. 1, 7

[12] V. Kolmogorov. Blossom V: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, July 2009. 6

[13] V. Kolmogorov. A new look at reweighted message passing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(5):919–930, 2015. 1

[14] Vladimir Kolmogorov. A computational study of Gomory-Hu tree construction algorithms. *arXiv:2204.10169v3*, Aug. 2022. 6

[15] Vladimir Kolmogorov. OrderedCuts: A new approach for computing Gomory-Hu tree. *arXiv*, Aug. 2022. 6

[16] V. Kolmogorov and T. Pock. One-sided Frank-Wolfe algorithms for saddle problems. In *International Conference on Machine Learning (ICML)*, 2021. 1, 2, 3, 6

[17] Vladimir Kolmogorov, Johan Thapper, and Stanislav Živný. The power of linear programming for general-valued CSPs. *SIAM Journal on Computing*, 44(1):1—-36, 2015. 1

[18] N. Komodakis, N. Paragios, and G. Tziritas. MRF energy minimization and beyond via dual decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):531–552, March 2011. 1

[19] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *International Conference on Machine Learning (ICML)*, 2013. 3

[20] D. V. N. Luong, P. Parpas, D. Rueckert, and B. Rustem. Solving MRF minimization by mirror descent. In *Advances in Visual Computing - 8th International Symposium, ISVC 2012, Rethymnon, Crete, Greece, July 16-18, 2012, Revised Selected Papers, Part I*, pages 587–598, 2012. 1

[21] A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing. An augmented Lagrangian approach to constrained MAP inference. In *International Conference on Machine Learning (ICML)*, 2011. Code: https://github.com/andre-martins/AD3. 1, 7

[22] B. O'Donoghue, , and E. Candes. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732, 2015. 6

[23] A. Osokin, J.-B. Alayrac, I. Lukasewitz, P. K. Dokania, and S. Lacoste-Julien. Minding the gaps for block Frank-Wolfe optimization of structured SVMs. In *International Conference on Machine Learning (ICML)*, 2016. 3

[24] M.W. Padberg and M.R. Rao. Odd minimum cut-sets and b-matchings. *Math. Oper. Res.*, 7:67–80, 1982. 6

[25] P. Ravikumar, A. Agarwal, and M. J. Wainwright. Message-passing for graph-structured linear programs: Proximal methods and rounding schemes. *Journal of Machine Learning Research*, 11:1043–1080, 2010. 1

[26] B. Savchynskyy, J. Kappes, S. Schmidt, and C. Schnörr. A study of nesterov's scheme for lagrangian decomposition and map labeling. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1817–1823. IEEE, 2011. 1

[27] Bogdan Savchynskyy and Stefan Schmidt. Getting feasible variable estimates from infeasible ones: MRF local polytope study. In *ICCV Workshops*, 2013. 1, 5

[28] B. Savchynskyy, S. Schmidt, J. H. Kappes, and C. Schnörr. Efficient MRF energy minimization via adaptive diminishing smoothing. In *Uncertainty in Artificial Intelligence (UAI)*, 2012. 1, 7

[29] M. I. Schlesinger and V. V. Giginyak. Solution to structural recognition (MAX,+)-problems by their equivalent transformations. (2):3–18, 2007. 1

[30] S. Schmidt, B. Savchynskyy, J. H. Kappes, and C. Schnörr. Evaluation of a first-order primal-dual algorithm for mrf energy minimization. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 89–103. Springer, 2011. 1

[31] Alexander Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Globally convergent parallel MAP LP re-

laxation solver using the Frank-Wolfe algorithm. In *International Conference on Machine Learning (ICML)*, 2014. 1

[32] Alexander G. Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Globally convergent dual MAP LP relaxation solvers using Fenchel-Young margins. In *NIPS*, 2012. 1

[33] N. Shah, V. Kolmogorov, and C. H. Lampert. A multi-plane block-coordinate Frank-Wolfe algorithm for training structural SVMs with a costly max-oracle. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3, 6

[34] G. Storvik and G. Dahl. Lagrangian-based methods for finding MAP. *IEEE Trans. on Image Processing*, 9(3):469–479, march 2000. 1

[35] Paul Swoboda and Vladimir Kolmogorov. MAP inference via block-coordinate Frank-Wolfe algorithm. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 2, 3, 7

[36] P. Swoboda, J. Kuske, and B. Savchynskyy. A dual ascent framework for Lagrangean decomposition of combinatorial problems. In *CVPR*, 2017. 1

[37] Daniel Tarlow, Dhruv Batra, Pushmeet Kohli, and Vladimir Kolmogorov. Dynamic tree block coordinate ascent. In *International Conference on Machine Learning (ICML)*, 2011. 2

[38] Siddharth Tourani, Alexander Shekhovtsov, Carsten Rother, and Bogdan Savchynskyy. Taxonomy of dual block-coordinate ascent methods for discrete energy minimization. In *Conference on Uncertainty in Artificial Intelligence (AISTATS)*, 2020. 1

[39] Tomas Werner. A linear programming approach to max-sum problem: A review. *IEEE transactions on pattern analysis and machine intelligence*, 29(7):1165–1179, 2007. 1

[40] T. Werner. A linear programming approach to max-sum problem: A review. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, 2007. 1

[41] J. Yarkony, R. Morshed, A. Ihler, and C. Fowlkes. Tightening MRF relaxations with planar subproblems. In *Uncertainty in Artificial Intelligence (UAI)*, 2011. 5