Prismatic Algorithm for Discrete D.C. Programming Problem

Yoshinobu Kawahara* **and Takashi Washio**
The Institute of Scientific and Industrial Research (ISIR)
Osaka University
8-1 Mihogaoka, Ibaraki-shi, Osaka 567-0047 JAPAN
{kawahara,washio}@ar.sanken.osaka-u.ac.jp

Abstract

In this paper, we propose the first exact algorithm for minimizing the difference of two submodular functions (D.S.), *i.e.*, the discrete version of the D.C. programming problem. The developed algorithm is a branch-and-bound-based algorithm which responds to the structure of this problem through the relationship between submodularity and convexity. The D.S. programming problem covers a broad range of applications in machine learning. In fact, this generalizes any set-function optimization. We empirically investigate the performance of our algorithm, and illustrate the difference between exact and approximate solutions respectively obtained by the proposed and existing algorithms in feature selection and discriminative structure learning.

1 Introduction

Combinatorial optimization techniques have been actively applied to many machine learning applications, where submodularity often plays an important role to develop algorithms [10, 16, 27, 14, 15, 19, 1]. In fact, many fundamental problems in machine learning can be formulated as submoular optimization. One of the important categories would be the D.S. programming problem, *i.e.*, the problem of minimizing the difference of two submodular functions. This is a natural formulation of many machine learning problems, such as learning graph matching [3], discriminative structure learning [21], feature selection [1] and energy minimization [24].

In this paper, we propose a prismatic algorithm for the D.S. programming problem, which is a branch-and-bound-based algorithm responding to the specific structure of this problem. To the best of our knowledge, this is the first exact algorithm to the D.S. programming problem (although there exists an approximate algorithm for this problem [21]). As is well known, the branch-and-bound method is one of the most successful frameworks in mathematical programming and has been incorporated into commercial softwares such as CPLEX [13, 12]. We develop the algorithm based on the analogy with the D.C. programming problem through the continuous relaxation of solution spaces and objective functions with the help of the Lovász extension [17, 11, 18]. The algorithm is implemented as an iterative calculation of binary-integer linear programming (BILP).

Also, we discuss applications of the D.S. programming problem in machine learning and investigate empirically the performance of our method and the difference between exact and approximate solutions through feature selection and discriminative structure-learning problems.

The remainder of this paper is organized as follows. In Section 2, we give the formulation of the D.S. programming problem and then describe its applications in machine learning. In Section 3, we give an outline of the proposed algorithm for this problem. Then, in Section 4, we explain the details of its basic operations. And finally, we give several empirical examples using artificial and real-world datasets in Section 5, and conclude the paper in Section 6.

Preliminaries and Notation: A set function f is called submodular if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq N$, where $N = \{1, \cdots, n\}$ [5, 7]. Throughout this paper, we denote

---
*http://www.ar.sanken.osaka-u.ac.jp/~kawahara/

1

by $\hat{f}$ the Lovász extension of f, *i.e.*, a continuous function $\hat{f} : \mathbb{R}^n \to \mathbb{R}$ defined by

$$\hat{f}(\boldsymbol{p}) = \sum_{j=1}^{m-1} (\hat{p}_j - \hat{p}_{j+1}) f(U_j) + \hat{p}_m f(U_m),$$

where $U_j = \{i \in N : p_i \geq \hat{p}_j\}$ and $\hat{p}_1 > \cdots > \hat{p}_m$ are the m distinct elements of $\boldsymbol{p}$ [17, 18]. Also, we denote by $I_A \in \{0, 1\}^n$ the characteristic vector of a subset $A \subseteq N$, *i.e.*, $I_A = \sum_{i \in A} \boldsymbol{e}_i$ where $\boldsymbol{e}_i$ is the i-th unit vector. Note, through the definition of the characteristic vector, any subset $A \subseteq N$ has the one-to-one correspondence with the vertex of a n-dimensional cube $D := \{\boldsymbol{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1 (i = 1, \ldots, n)\}$. And, we denote by $(A, t)(T)$ all combinations of a real value plus subset whose corresponding vectors (I_A, t) are inside or on the surface of a polytope $T \in \mathbb{R}^{n+1}$.

2 The D.S. Programming Problem and its Applications

Let f and g are submodular functions. In this paper, we address an *exact* algorithm to solve the D.S. programming problem, *i.e.*, the problem of minimizing the difference of two submodular functions:

$$\min_{A \subseteq N} \quad f(A) - g(A). \tag{1}$$

As is well known, any real-valued function whose second partial derivatives are continuous everywhere can be represented as the difference of two convex functions [12]. As well, the problem (1) generalizes any set-function optimization problem. Problem (1) covers a broad range of applications in machine learning [21, 24, 3, 1]. Here, we give a few examples.

Feature selection using structured-sparsity inducing norms: Sparse methods for supervised learning, where we aim at finding good predictors from as few variables as possible, have attracted much interests from machine learning community. This combinatorial problem is known to be a submodular maximization problem with cardinality constraint for commonly used measures such as least-squared errors [4, 14]. And as is well known, if we replace the cardinality function with its convex envelope such as l_1-norm, this can be turned into a convex optimization problem. Recently, it is reported that submodular functions in place of the cardinality can give a wider family of polyhedral norms and may incorporate prior knowledge or structural constraints in sparse methods [1]. Then, the objective (that is supposed to be minimized) becomes the sum of a loss function (often, supermodular) and submodular regularization terms.

Discriminative structure learning: It is reported that discriminatively structured Bayesian classifier often outperforms generatively structured one [21, 22]. One commonly used metric for discriminative structure learning would be EAR (explaining away residual) [2]. EAR is defined as the difference of the conditional mutual information between variables by class C and non-conditional one, *i.e.*, $I(X_i; X_j|C) - I(X_i; X_j)$. In structure learning, we repeatedly try to find a subset in variables that minimize this kind of measures. Since the (symmetric) mutual information is a submodular function, obviously this problem leads the D.S. programming problem [21].

Energy minimization in computer vision: In computer vision, an image is often modeled with a Markov random field, where each node represents a pixel. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the undirected graph, where a label $x_s \in \mathcal{L}$ is assigned on each node. Then, many tasks in computer vision can be naturally formulated in terms of energy minimization where the energy function has the form: $E(\boldsymbol{x}) = \sum_{p \in \mathcal{V}} \theta_p(\boldsymbol{x}_p) + \sum_{(p,q) \in \mathcal{E}} \theta(\boldsymbol{x}_p, \boldsymbol{x}_q)$, where θ_p and $\theta_{p,q}$ are univariate and pairwise potentials. In a pairwise potential for binarized energy (*i.e.*, $\mathcal{L} = \{0, 1\}$), submodularity is defined as $\theta_{pq}(1, 1) + \theta_{pq}(0, 0) \geq \theta_{pq}(1, 0) + \theta_{pq}(0, 1)$ (see, for example, [26]). Based on this, any energy function in computer vision can be written with a submodular function $E_1(\boldsymbol{x})$ and a supermodular function $E_2(\boldsymbol{x})$ as $E(\boldsymbol{x}) = E_1(\boldsymbol{x}) + E_2(\boldsymbol{x})$ (ex. [24]). Or, in case of binarized energy, even if such explicit decomposition is not known, a non-unique decomposition to submodular and supermodular functions can be always given [25].

3 Prismatic Algorithm for the D.S. Programming Problem

By introducing an additional variable $t (\in \mathbb{R})$, Problem (1) can be converted into the equivalent problem with a supermodular objective function and a submodular feasible set, *i.e.*,

$$\min_{A \subseteq N, t \in \mathbb{R}} \quad t - g(A) \quad \text{s.t.} \quad f(A) - t \leq 0. \tag{2}$$

2

Obviously, if (A^*, t^*) is an optimal solution of Problem (2), then A^* is an optimal solution of Problem (1) and $t^* = f(A^*)$. The proposed algorithm is a realization of the branch-and-bound scheme which responds to this specific structure of the problem.

To this end, we first define a *prism* $T(S) \subset \mathbb{R}^{n+1}$ by

$$T = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{x} \in S\},$$

where S is an n-simplex. S is obtained from the n-dimensional cube D at the initial iteration (as described in Section 4.1), or by the subdivision operation described in the later part of this section (and the detail will be described in Section 4.2). The prism T has $n + 1$ edges that are vertical lines (*i.e.*, lines parallel to the t-axis) which pass through the $n + 1$ vertices of S, respectively [11].

Figure 1: Illustration of the prismatic algorithm.

Our algorithm is an iterative procedure which mainly consists of two parts; *branching* and *bounding*, as well as other branch-and-bound frameworks [13]. In *branching*, subproblems are constructed by dividing the feasible region of a parent problem. And in *bounding*, we judge whether an optimal solution exists in the region of a subproblem and its descendants by calculating an upper bound of the subproblem and comparing it with an lower bound of the original problem. Some more details for branching and bounding are described as follows.

Branching: The branching operation in our method is carried out using the property of a simplex. That is, since, in a n-simplex, any $r + 1$ vertices are not on a $r - 1$-dimensional hyperplane for $r \leq n$, any n-simplex can be divided as $S = \bigcup_{i=1}^{p} S_i$, where $p \geq 2$ and S_i are n-simplices such that each pair of simplices $S_i, S_j (i \neq j)$ intersects at most in common boundary points (the way of constructing such partition is explained in Section 4.2). Then, $T = \bigcup_{i=1}^{p} T_i$, where $T_i = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{x} \in S_i\}$, is a natural prismatic partition of T induced by the above simplical partition.

Bounding: For the bounding operation on S_k (resp., T_k) at the iteration k, we consider a polyhedral convex set P_k such that $P_k \supset \tilde{D}$, where $\tilde{D} = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{x} \in D, \hat{f}(\boldsymbol{x}) \leq t\}$ is the region corresponding to the feasible set of Problem (2). At the first iteration, such P is obtained as

$$P_0 = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{x} \in S, t \geq \tilde{t}\},$$

where $\tilde{t}$ is a real number satisfying $\tilde{t} \leq \min\{f(A) : A \subseteq N\}$. Here, $\tilde{t}$ can be determined by using some existing submodular minimization solver [23, 8]. Or, at later iterations, more refined P_k, such that $P_0 \supset P_1 \supset \cdots \supset \tilde{D}$, is constructed as described in Section 4.4.

As described in Section 4.3, a lower bound $\beta(T_k)$ of $t - g(A)$ on the current prism T_k can be calculated through the binary-integer linear programming (BILP) (or the linear programming (LP)) using P_k, obtained as described above. Let α be the lowest function value (*i.e.*, an upper bound of $t - g(A)$ on $\tilde{D}$) found so far. Then, if $\beta(T_k) \geq \alpha$, we can conclude that there is no feasible solution which gives a function value better than α and can remove T_k without loss of optimality.

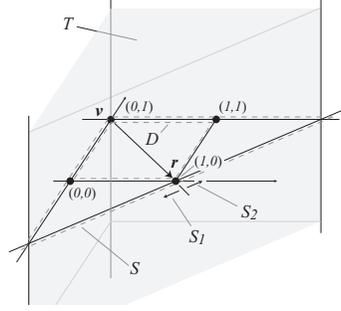
The pseudo-code of the proposed algorithm is described in Algorithm 1. In the following section, we explain the details of the operations involved in this algorithm.

4 Basic Operations

Obviously, the procedure described in Section 3 involves the following basic operations:

1. Construction of the first prism: A prism needs to be constructed from a hypercube at first,
2. Subdivision process: A prism is divided into a finite number of sub-prisms at each iteration,
3. Bound estimation: For each prism generated throughout the algorithm, a lower bound for the objective function $t - g(A)$ over the part of the feasible set contained in this prism is computed,
4. Construction of cutting planes: Throughout the algorithm, a sequence of polyhedral convex sets $P_0, P_1, \cdots$ is constructed such that $P_0 \supset P_1 \supset \cdots \supset \tilde{D}$. Each set P_j is generated by a cutting plane to cut off a part of P_{j-1}, and
5. Deletion of non-optimal prisms: At each iteration, we try to delete prisms that contain no feasible solution better than the one obtained so far.

3

---

1 Construct a simplex $S_0 \supset D$, its corresponding prism T_0 and a polyhedral convex set $P_0 \supset \tilde{D}$.

2 Let α_0 be the best objective function value known in advance. Then, solve the BILP (5) corresponding to α_0 and T_0, and let $\beta_0 = \beta(T_0, P_0, \alpha_0)$ and $(\bar{A}^0, \bar{t}_0)$ be the point satisfying $\beta_0 = \bar{t}_0 - g(\bar{A}^0)$.

3 Set $\mathcal{R}_0 \leftarrow T_0$.

4 **while** $\mathcal{R}_k \neq \emptyset$

5 $\quad$ Select a prism $T_k^* \in \mathcal{R}_k$ satisfying $\beta_k = \beta(T_k^*)$, $(\bar{v}^k, \bar{t}_k) \in T_k^*$.

6 $\quad$ **if** $(\bar{v}^k, \bar{t}_k) \in \tilde{D}$ **then**

7 $\quad\quad$ Set $P_{k+1} = P_k$.

8 $\quad$ **else**

9 $\quad\quad$ Construct $l_k(\boldsymbol{x}, t)$ according to (8), and set $P_{k+1} = \{(\boldsymbol{x}, t) \in P_k : l_k(\boldsymbol{x}, t) \leq 0\}$.

10 $\quad$ Subdivide $T_k^* = T(S_k^*)$ into a finite number of subprisms $T_{k,j}(j \in J_k)$ (cf. Section 4.2).

11 $\quad$ For each $j \in J_k$, solve the BILP (5) with respect to $T_{k,j}$, P_{k+1} and α_k.

12 $\quad$ Delete all $T_{k,j}(j \in J_k)$ satisfying (DR1) or (DR2). Let $\mathcal{M}_k$ denote the collection of remaining prisms $T_{k,j}(j \in J_k)$, and for each $T \in \mathcal{M}_k$ set
$$\beta(T) = \max\{\beta(T_k^*), \beta(T, P_{k+1}, \alpha_k)\}.$$

13 $\quad$ Let F_k be the set of new feasible points detected while solving BILP in Step 11, and set
$$\alpha_{k+1} = \min\{\alpha_k, \min\{t - g(A) : (A, t) \in F_k\}\}.$$

14 $\quad$ Delete all $T \in \mathcal{M}_k$ satisfying $\beta(T) \geq \alpha_{k+1}$ and let $\mathcal{R}_k$ be $\mathcal{R}_{k-1} \setminus T_k \in \mathcal{M}_k$.

15 $\quad$ Set $\beta_{k+1} \leftarrow \min\{\beta(T) : T \in \mathcal{M}_k\}$ and $k \leftarrow k + 1$.

Algorithm 1: Pseudo-code of the prismatic algorithm for the D.S programming problem.

4.1 Construction of the first prism

The initial simplex $S_0 \supset D$ (which yields the initial prism $T_0 \supset \tilde{D}$) can be constructed as follows. Now, let $\boldsymbol{v}$ and $A_{\boldsymbol{v}}$ be a vertex of D and its corresponding subset in N, respectively, *i.e.*, $\boldsymbol{v} = \sum_{i \in A_{\boldsymbol{v}}} \boldsymbol{e}_i$. Then, the initial simplex $S_0 \supset D$ can be constructed by

$$S_0 = \{\boldsymbol{x} \in \mathbb{R}^n : x_i \leq 1 (i \in A_{\boldsymbol{v}}), x_i \geq 0 (i \in N \setminus A_{\boldsymbol{v}}), \boldsymbol{a}^T \boldsymbol{x} \leq \gamma\},$$

where $\boldsymbol{a} = \sum_{i \in N \setminus A_{\boldsymbol{v}}} \boldsymbol{e}_i - \sum_{i \in A_{\boldsymbol{v}}} \boldsymbol{e}_i$ and $\gamma = |N \setminus A_{\boldsymbol{v}}|$. The $n + 1$ vertices of S_0 are $\boldsymbol{v}$ and the n points where the hyperplane $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}^T \boldsymbol{x} = \gamma\}$ intersects the edges of the cone $\{\boldsymbol{x} \in \mathbb{R}^n : x_i \leq 1(i \in A_{\boldsymbol{v}}), x_i \geq 0(i \in N \setminus A_{\boldsymbol{v}})\}$. Note this is just an option and any n-simplex $S \supset D$ is available.

4.2 Sub-division of a prism

Let S_k and T_k be the simplex and prism at k-th iteration in the algorithm, respectively. We denote S_k as $S_k = [\boldsymbol{v}_k^i, \ldots, \boldsymbol{v}_k^{n+1}] := \text{conv}\{\boldsymbol{v}_k^1, \ldots, \boldsymbol{v}_k^{n+1}\}$ which is defined as the convex hull of its vertices $\boldsymbol{v}_k^1, \ldots, \boldsymbol{v}_k^{n+1}$. Then, any $\boldsymbol{r} \in S_k$ can be represented as

$$\boldsymbol{r} = \sum_{i=1}^{n+1} \lambda_i \boldsymbol{v}_k^i, \ \sum_{i=1}^{n+1} \lambda_i = 1, \ \lambda_i \geq 0 \ (i = 1, \ldots, n+1).$$

Suppose that $\boldsymbol{r} \neq \boldsymbol{v}_k^i$ $(i = 1, \ldots, n+1)$. For each i satisfying $\lambda_i > 0$, let S_k^i be the subsimplex of S_k defined by

$$S_k^i = [\boldsymbol{v}_k^1, \ldots, \boldsymbol{v}_k^{i-1}, \boldsymbol{r}, \boldsymbol{v}_k^{i+1}, \ldots, \boldsymbol{v}_k^{n+1}]. \tag{3}$$

Then, the collection $\{S_k^i : \lambda_i > 0\}$ defines a partition of S_k, *i.e.*, we have

$$\bigcup_{\lambda_i > 0} S_k^i = S_k, \ \text{int } S_k^i \cap \text{int } S_k^j = \emptyset \ \text{ for } \ i \neq j \ [12].$$

In a natural way, the prisms $T(S_k^i)$ generated by the simplices S_k^i defined in Eq. (3) form a partition of T_k. This subdivision process of prisms is *exhaustive*, *i.e.*, for every nested (decreasing) sequence of prisms $\{T_q\}$ generated by this process, we have $\bigcap_{q=0}^{\infty} T_q = \tau$, where τ is a line perpendicular to $\mathbb{R}^n$ (a vertical line) [11]. Although several subdivision process can be applied, we use a classical *bisection* one, *i.e.*, each simplex is divided into subsimplices by choosing in Eq. (3) as

$$\boldsymbol{r} = (\boldsymbol{v}_k^{i_1} + \boldsymbol{v}_k^{i_2})/2,$$

where $\|\boldsymbol{v}_k^{i_1} - \boldsymbol{v}_k^{i_2}\| = \max\{\|\boldsymbol{v}_k^i - \boldsymbol{v}_k^j\| : i, j \in \{0, \ldots, n\}, i \neq j\}$ (see Figure 1).

4

4.3 Lower bounds

Again, let S_k and T_k be the simplex and prism at k-th iteration in the algorithm, respectively. And, let α be an upper bound of $t - g(A)$, which is the smallest value of $t - g(A)$ attained at a feasible point known so far in the algorithm. Moreover, let P_k be a polyhedral convex set which contains $\tilde{D}$ and be represented as

$$P_k = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : A_k \boldsymbol{x} + \boldsymbol{a}_k t \le \boldsymbol{b}_k\}, \tag{4}$$

where A_k is a real $(m \times n)$-matrix and $\boldsymbol{a}_k, \boldsymbol{b}_k \in \mathbb{R}^m$.[1] Now, a lower bound $\beta(T_k, P_k, \alpha)$ of $t - g(A)$ over $T_k \cap \tilde{D}$ can be computed as follows.

First, let $\boldsymbol{v}_k^i$ $(i = 1, \ldots, n+1)$ denote the vertices of S_k, and define $I(S_k) = \{i \in \{1, \ldots, n+1\} : \boldsymbol{v}_k^i \in \mathbb{B}^n\}$ and

$$\mu = \begin{cases} \min\{\alpha, \min\{\hat{f}(\boldsymbol{v}_k^i) - \hat{g}(\boldsymbol{v}_k^i) : i \in I(S)\}\}, & \text{if } I(S) \ne \emptyset, \\ \alpha, & \text{if } I(S) = \emptyset. \end{cases}$$

For each $i = 1, \ldots, n+1$, consider the point $(\boldsymbol{v}_k^i, t_k^i)$ where the edge of T_k passing through $\boldsymbol{v}_k^i$ intersects the level set $\{(\boldsymbol{x}, t) : t - \hat{g}(\boldsymbol{x}) = \mu\}$, i.e.,

$$t_k^i = \hat{g}(\boldsymbol{v}_k^i) + \mu \quad (i = 1, \ldots, n+1).$$

Then, let us denote the uniquely defined hyperplane through the points $(\boldsymbol{v}_k^i, t_k^i)$ by $H = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{p}^T \boldsymbol{x} - t = \gamma\}$, where $\boldsymbol{p} \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}$. Consider the upper and lower halfspace generated by H, i.e., $H_+ = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{p}^T \boldsymbol{x} - t \le \gamma\}$ and $H_- = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{p}^T \boldsymbol{x} - t \ge \gamma\}$. If $T_k \cap \tilde{D} \subseteq H_+$, then we see from the supermodularity of $g(A)$ (the concavity of $\hat{g}(\boldsymbol{x})$) that

$$\begin{aligned} \min\{t - g(A) : (A, t) \in (A, t)(T_k \cap \tilde{D})\} &\ge \min\{t - g(A) : (A, t) \in (A, t)(T_k \cap H_+)\} \\ &\ge \min\{t - \hat{g}(\boldsymbol{x}) : (\boldsymbol{x}, t) \in T_k \cap H_+\} \\ &= t_k^i - \hat{g}(\boldsymbol{x}_k^i)(i = 1, \ldots, n+1) = \mu. \end{aligned}$$

Otherwise, we shift the hyperplane H (downward with respect to t) until it reaches a point $\boldsymbol{z} = (\boldsymbol{x}^*, t^*)$ $(\in T_k \cap P_k \cap H_-, \boldsymbol{x}^* \in \mathbb{B}^n)$ $((\boldsymbol{x}^*, t^*)$ is a point with the largest distance to H and the corresponding pair (A, t) (since $\boldsymbol{x}^* \in \mathbb{B}^n$) is in $(A, t)(T_k \cap P_k \cap H_-))$. Let $\bar{H}$ denote the resulting supporting hyperplane, and denote by $\bar{H}_+$ the upper halfspace generated by $\bar{H}$. Moreover, for each $i = 1, \ldots, n+1$, let $\boldsymbol{z}^i = (\boldsymbol{v}_k^i, \bar{t}_k^i)$ be the point where the edge of T passing through $\boldsymbol{v}_k^i$ intersects $\bar{H}$. Then, it follows $(A, t)(T_k \cap \tilde{D}) \subset (A, t)(T_k \cap P_k) \subset (A, t)(T_k \cap \bar{H}_+)$, and hence

$$\begin{aligned} \min\{t - g(A) : (A, t) \in (A, t)(T_k \cap \tilde{D})\} &> \min\{t - g(A) : (A, t) \in (A, t)(T_k \cap \bar{H}_+)\} \\ &= \min\{\bar{t}_k^i - \hat{g}(\boldsymbol{v}_k^i) : i = 1, \ldots, n+1\}. \end{aligned}$$

Now, the above consideration leads to the following BILP in $(\boldsymbol{\lambda}, \boldsymbol{x}, t)$:

$$\begin{aligned} \max_{\boldsymbol{\lambda}, \boldsymbol{x}, t} \quad & \left(\sum_{i=1}^{n+1} t_i \lambda_i - t\right) \quad \text{s.t. } A_k \boldsymbol{x} + \boldsymbol{a}_k t \le \boldsymbol{b}_k, \ \boldsymbol{x} = \sum_{i=1}^{n+1} \lambda_i \boldsymbol{v}_k^i, \ \boldsymbol{x} \in \mathbb{B}^n, \\ & \sum_{i=1}^{n+1} \lambda_i = 1, \ \lambda_i \ge 0 \ (i = 1, \ldots, n+1), \end{aligned} \tag{5}$$

where A, $\boldsymbol{a}$ and $\boldsymbol{b}$ are given in Eq. (4).

Proposition 1. *(a) If the system* (5) *has no solution, then intersection* $(A, t)(T_k \cap \tilde{D})$ *is empty.*
(b) Otherwise, let $(\boldsymbol{\lambda}^*, \boldsymbol{x}^*, t^*)$ *be an optimal solution of BILP* (5) *and* $c^* = \sum_{i=1}^{n+1} t_i \lambda_i^* - t^*$ *its optimal value, respectively. Then, the following statements hold:*
(b1) If $c^* \le 0$, *then* $(A, t)(T_k \cap \tilde{D}) \subset (A, t)(H_+)$.
(b2) If $c^* > 0$, *then* $\boldsymbol{z} = (\sum_{i=1}^{n+1} \lambda_i \boldsymbol{v}_k^i, t_k^*)$, $\boldsymbol{z}^i = (\boldsymbol{v}_k^i, \bar{t}_k^i) = (\boldsymbol{v}_k^i, t_k^i - c^*)$ *and* $\bar{t}_k^i - \hat{g}(\boldsymbol{v}_k^i) = \mu - c^*$ $(i = 1, \ldots, n+1)$.

Proof. First, we prove part (a). Since every point in S_k is uniquely representable as $\boldsymbol{x} = \sum_{i=1}^{n+1} \lambda_i \boldsymbol{v}^i$, we see from Eq. (4) that the set $(A, t)(T_k \cap P_k)$ coincide with the feasible set of problem (5). Therefore, if the system (5) has no solution, then $(A, t)(T_k \cap P_k) = \emptyset$, and hence $(A, t)(T_k \cap \tilde{D}) = \emptyset$ (because $\tilde{D} \subset P_k$). Next, we move to part (b). Since the equation of H is $\boldsymbol{p}^T \boldsymbol{x} - t = \gamma$, it follows

---
[1] Note that P_k is updated at each iteration, which does not depend on S_k, as described in Section 4.4.

5

that determining the hyperplane $\bar{H}$ and the point $\boldsymbol{z}$ amounts to solving the binary integer linear programming problem:

$$\max \quad \boldsymbol{p}^T \boldsymbol{x} - t \quad \text{s.t.} \ (\boldsymbol{x}, t) \in T_k \cap P_k, \ \boldsymbol{x} \in \mathbb{B}^n. \tag{6}$$

Here, we note that the objective of the above can be represented as

$$\boldsymbol{p}^T \boldsymbol{x} - t = \boldsymbol{p}^T \left(\sum_{i=1}^{n+1} \lambda_i \boldsymbol{v}_k^i \right) - t = \sum_{i=1}^{n+1} \lambda_i \boldsymbol{p}^T \boldsymbol{v}_k^i - t.$$

On the other hand, since $(\boldsymbol{v}_k^i, t_k^i) \in H$, we have $\boldsymbol{p}^T \boldsymbol{v}_k^i - t_k^i = \gamma \ (i = 1, \dots, n+1)$, and hence

$$\boldsymbol{p}^T \boldsymbol{x} - t = \sum_{i=1}^{n+1} \lambda_i (\gamma + t_k^i) - t = \sum_{i=1}^{n+1} t_k^i \lambda_i - t + \gamma.$$

Thus, the two BILPs (5) and (6) are equivalent. And, if γ^* denotes the optimal objective function value in Eq. (6), then $\gamma^* = c^* + \gamma$. If $\gamma^* \leq \gamma$, then it follows from the definition of H_+ that $\bar{H}$ is obtained by a parallel shift of H in the direction H_+. Therefore, $c^* \leq 0$ implies $(A, t)(T_k \cap P_k) \subset (A, t)(H_+)$, and hence $(A, t)(T_k \cap \tilde{D}) \subset (A, t)(H_+)$.

Since $\bar{H} = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{p}^T \boldsymbol{x} - t = \gamma^*\}$ and $H = \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : \boldsymbol{p}^T \boldsymbol{x} - t = \gamma\}$ we see that for each intersection point $(\boldsymbol{v}_k^i, \bar{t}_k^i)$ (and $(\boldsymbol{v}_k^i, t_k^i)$) of the edge of T_k passing through $\boldsymbol{v}_k^i$ with $\bar{H}$ (and H), we have $\boldsymbol{p}^T \boldsymbol{v}_k^i - \bar{t}_k^i = \gamma^*$ and $\boldsymbol{p}^T \boldsymbol{v}_k^i - t_k^i = \gamma$, respectively. This implies that $\bar{t}_k^i = t_k^i + \gamma - \gamma^* = t_k^i - c^*$, and (using $t_k^i = \hat{g}(\boldsymbol{v}_k^i) + \mu$) that $\bar{t}_k^i = \hat{g}(\boldsymbol{v}_k^i) + \mu - c^*$. $\qquad\square$

From the above, we see that, in the case (b1), μ constitutes a lower bound of $(t - g(A))$ wheres, in the case (b2), such a lower bound is given by $\min\{\bar{t}_k^i - \hat{g}(\boldsymbol{v}_k^i) : i = 1, \dots, n+1\}$. Thus, Proposition 1 provides the lower bound

$$\beta_k(T_k, P_k, \alpha) = \begin{cases} +\infty, & \text{if BILP (5) has no feasible point,} \\ \mu, & \text{if } c^* \leq 0, \\ \mu - c^* & \text{if } c^* > 0. \end{cases} \tag{7}$$

As stated in Section 4.5, T_k can be deleted from further consideration when $\beta_k = \infty$ or μ.

4.4 Outer approximation

The polyhedral convex set $P_k \supset \tilde{D}$ used in the preceding section is updated in each iteration, *i.e.*, a sequence $P_0, P_1, \cdots$ is constructed such that $P_0 \supset P_1 \supset \cdots \supset \tilde{D}$. The update from P_k to P_{k+1} $(k = 0, 1, \dots)$ is done in a way which is standard for pure outer approximation methods [12]. That is, a certain linear inequality $l_k(\boldsymbol{x}, t) \leq 0$ is added to the constraint set defining P_k, *i.e.*, we set

$$P_{k+1} = P_k \cap \{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : l_k(\boldsymbol{x}, t) \leq 0\}.$$

The function $l_k(\boldsymbol{x}, t)$ is constructed as follows. At iteration k, we have a lower bound β_k of $t - g(A)$ as defined in Eq. (7), and a point $(\bar{\boldsymbol{v}}_k, \bar{t}_k)$ satisfying $\bar{t}_k - \hat{g}(\bar{\boldsymbol{v}}_k) = \beta_k$. We update the outer approximation only in the case $(\bar{\boldsymbol{v}}_k, \bar{t}_k) \notin \tilde{D}$. Then, we can set

$$l_k(\boldsymbol{x}, t) = \boldsymbol{s}_k^T[(\boldsymbol{x}, t) - \boldsymbol{z}_k] + (\hat{f}(\boldsymbol{x}_k^*) - t_k^*), \tag{8}$$

where $\boldsymbol{s}_k$ is a subgradient of $\hat{f}(\boldsymbol{x}) - t$ at $\boldsymbol{z}_k$. The subgradient can be calculated as, for example, stated in [9] (see also [7]).

Proposition 2. *The hyperplane* $\{(\boldsymbol{x}, t) \in \mathbb{R}^n \times \mathbb{R} : l_k(\boldsymbol{x}, t) = 0\}$ *strictly separates* $\boldsymbol{z}_k$ *from* $\tilde{D}$*, i.e.,* $l_k(\boldsymbol{z}_k) > 0$*, and* $l_k(\boldsymbol{x}, t) \leq 0$ *for* $^\forall(\boldsymbol{x}, t) \in \tilde{D}$*.*

Proof. Since we assume that $\boldsymbol{z}_k \notin \tilde{D}$, we have $l_k(\boldsymbol{z}_k) = (\hat{f}(\boldsymbol{x}_k^*) - t_k^*)$. And, the latter inequality is an immediate consequence of the definition of a subgradient. $\qquad\square$

4.5 Deletion rules

At each iteration of the algorithm, we try to delete certain subprisms that contain no optimal solution. To this end, we adopt the following two deletion rules:

(DR1) Delete T_k if BILP (5) has no feasible solution.

6

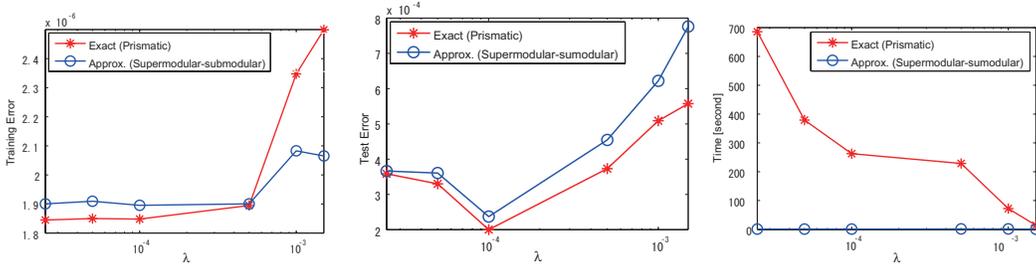



Figure 2: Training errors, test errors and computational time versus λ for the prismatic algorithm and the supermodular-sumodular procedure.

p	n	k	exact(PRISM)	SSP	greedy	lasso
120	150	5	1.8e-4 (192.6)	1.9e-4 (0.93)	1.8e-4 (0.45)	1.9e-4 (0.78)
120	150	10	2.0e-4 (262.7)	2.4e-4 (0.81)	2.3e-4 (0.56)	2.4e-4 (0.84)
120	150	20	7.3e-4 (339.2)	7.8e-4 (1.43)	8.3e-4 (0.59)	7.7e-4 (0.91)
120	150	40	1.7e-3 (467.6)	2.1e-3 (1.17)	2.9e-3 (0.63)	1.9e-3 (0.87)

Table 1: Normalized mean-square prediction errors of training and test data by the prismatic algorithm, the supermodular-submodular procedure, the greedy algorithm and the lasso.

(DR2) Delete T_k if the optimal value c^* of BILP (5) satisfies $c^* \leq 0$.

The feasibility of these rules can be seen from Proposition 1 as well as the D.C. programing problem [11]. That is, (DR1) follows from Proposition 1 that in this case $T_k \cap \tilde{D} = \emptyset$, *i.e.*, the prism T_k is infeasible, and (DR2) from Proposition 1 and from the definition of μ that the current best feasible solution cannot be improved in T.

5   Experimental Results

We first provide illustrations of the proposed algorithm and its solution on toy examples from feature selection in Section 5.1, and then apply the algorithm to an application of discriminative structure learning using the UCI repository data in Section 5.2. The experiments below were run on a 2.8 GHz 64-bit workstation using Matlab and IBM ILOG CPLEX ver. 12.1.

5.1   Application to feature selection

We compared the performance and solutions by the proposed prismatic algorithm (PRISM), the supermodular-submodular procedure (SSP) [21], the greedy method and the LASSO. To this end, we generated data as follows: Given p, n and k, the design matrix $X \in \mathbb{R}^{n \times p}$ is a matrix of i.i.d. Gaussian components. A feature set J of cardinality k is chosen at random and the weights on the selected features are sampled from a standard multivariate Gaussian distribution. The weights on other features are 0. We then take $y = X\boldsymbol{w} + n^{-1/2}\|X\boldsymbol{w}\|_2\boldsymbol{\epsilon}$, where $\boldsymbol{w}$ is the weights on features and $\boldsymbol{\epsilon}$ is a standard Gaussian vector. In the experiment, we used the trace norm of the submatrix corresponding to J, X_J, *i.e.*, $\mathrm{tr}(X_J^T X_J)^{1/2}$. Thus, our problem is $\min_{\boldsymbol{w} \in \mathbb{R}^p} \frac{1}{2n}\|\boldsymbol{y} - X\boldsymbol{w}\|_2^2 + \lambda \cdot \mathrm{tr}(X_J^T X_J)^{1/2}$, where J is the support of $\boldsymbol{w}$. Or equivalently, $\min_{A \in V} g(A) + \lambda \cdot \mathrm{tr}(X_A^T X_A)^{1/2}$, where $g(A) := \min_{\boldsymbol{w}_A \in \mathbb{R}^{|A|}} \|\boldsymbol{y} - X_A\boldsymbol{w}_A\|^2$. Since the first term is a supermodular function [4] and the second is a submodular function, this problem is the D.S. programming problem.

First, the graphs in Figure 2 show the training errors, test errors and computational time versus λ for PRISM and SSP (for $p = 120$, $n = 150$ and $k = 10$). The values in the graphs are averaged over 20 datasets. For the test errors, we generated another 100 data from the same model and applied the estimated model to the data. And, for all methods, we tried several possible regularization parameters. From the graphs, we can see the following: First, exact solutions (by PRISM) always outperform approximate ones (by SSP). This would show the significance of optimizing the submodular-norm. That is, we could obtain the better solutions (in the sense of prediction error) by optimizing the objective with the submodular norm more exactly. And, our algorithm took longer especially when

7

Data	Attr.	Class	exact (PRISM)	approx. (SSP)	generative
Chess	36	2	96.6 ($\pm$0.69)	94.4 ($\pm$0.71)	92.3 ($\pm$0.79)
German	20	2	70.0 ($\pm$0.43)	69.9 ($\pm$0.43)	69.1 ($\pm$0.49)
Census-income	40	2	73.2 ($\pm$0.64)	71.2 ($\pm$0.74)	70.3 ($\pm$0.74)
Hepatitis	19	2	86.9 ($\pm$1.89)	84.3 ($\pm$2.31)	84.2 ($\pm$2.11)

Table 2: Empirical accuracy of the classifiers in [%] with standard deviation by the TANs discriminatively learned with PRISM or SSP and generatively learned with a submodular minimization solver. The numbers in parentheses are computational time in seconds.

λ smaller. This would be because smaller λ basically gives a larger size subset (solution). Also, Table 1 shows normalized-mean prediction errors by the prismatic algorithm, the supermodular-submodular procedure, the greedy method and the lasso for several k. The values are averaged over 10 datasets. This result also seems to show that optimizing the objective with the submodular norm exactly is significant in the meaning of prediction errors.

5.2 Application to discriminative structure learning

Our second application is discriminative structure learning using the UCI machine learning repository.[2] Here, we used CHESS, GERMAN, CENSUS-INCOME (KDD) and HEPATITIS, which have two classes. The Bayesian network topology used was the tree augmented naive Bayes (TAN) [22]. We estimated TANs from data both in generative and discriminative manners. To this end, we used the procedure described in [20] with a submodular minimization solver (for the generative case), and the one [21] combined with our prismatic algorithm (PRISM) or the supermodular-submodular procedure (SSP) (for the discriminative case). Once the structures have been estimated, the parameters were learned based on the maximum likelihood method.

Table 2 shows the empirical accuracy of the classifier in [%] with standard deviation for these datasets. We used the train/test scheme described in [6, 22]. Also, we removed instances with missing values. The results seem to show that optimizing the EAR measure more exactly could improve the performance of classification (which would mean that the EAR is significant as the measure of discriminative structure learning in the sense of classification).

6 Conclusions

In this paper, we proposed a prismatic algorithm for the D.S. programming problem (1), which is the first exact algorithm for this problem and is a branch-and-bound method responding to the structure of this problem. We developed the algorithm based on the analogy with the D.C. programming problem through the continuous relaxation of solution spaces and objective functions with the help of the Lovász extension. We applied the proposed algorithm to several situations of feature selection and discriminative structure learning using artificial and real-world datasets.

The D.S. programming problem addressed in this paper covers a broad range of applications in machine learning. In future works, we will develop a series of the presented framework specialized to the specific structure of each problem. Also, it would be interesting to investigate the extension of our method to enumerate solutions, which could make the framework more useful in practice.

Acknowledgments

This research was supported in part by JST PRESTO PROGRAM (Synthesis of Knowledge for Information Oriented Society), JST ERATO PROGRAM (Minato Discrete Structure Manipulation System Project) and KAKENHI (22700147). Also, we are very grateful to the reviewers for helpful comments.

---

[2]http://archive.ics.uci.edu/ml/index.html

8

References

[1] F. Bach. Structured sparsity-inducing norms through submodular functions. In *Advances in Neural Information Processing Systems 23*, pages 118–126, 2010.

[2] J. A. Bilmes. Dynamic Bayesian multinets. In *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence (UAI'00)*, pages 38–45, 2000.

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

[4] A. Das and D. Kempe. Algorithms for subset selection in linear regression. In *Proc. of the 40th annual ACM symp. on Theory of computing (STOC'08)*, pages 45–54, 2008.

[5] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial structures and their applications*, pages 69–87, 1970.

[6] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifier. 29:131–163, 1997.

[7] S. Fujishige. *Submodular Functions and Optimization*. Elsevier, 2 edition, 2005.

[8] S. Fujishige, T. Hayashi, and S. Isotani. The minimum-norm-point algorithm applied submodular function minimization and linear programming. Technical report, Research Institute for Mathematical Sciences, Kyoto University, 2006.

[9] E. Hazan and S. Kale. Beyond convexity: online submodular minimization. In *Advances in Neural Information Processing Systems 22*, pages 700–708, 2009.

[10] S. Hoi, R. Jin, J. Zhu, and M. Lyu. Batch mode active learning and its application to medical image classification. In *Proc. of the 23rd Int'l Conf. on Machine learning (ICML'06)*, pages 417–424, 2006.

[11] R. Horst, T. Q. Phong, Ng. V. Thoai, and J. de Vries. On solving a D.C. programming problem by a sequence of linear programs. *Journal of Global Optimization*, 1:183–203, 1991.

[12] R. Horst and H. Tuy. *Global Optimization (Deterministic Approaches)*. Springer, 3 edition, 1996.

[13] T. Ibaraki. Enumerative approaches to combinatorial optimization. In J.C. Baltzer and A.G. Basel, editors, *Annals of Operations Research*, volume 10 and 11. 1987.

[14] Y. Kawahara, K. Nagano, K. Tsuda, and J. A. Bilmes. Submodularity cuts and applications. In *Advances in Neural Information Processing Systems 22*, pages 916–924. MIT Press, 2009.

[15] A. Krause and V. Cevher. Submodular dictionary selection for sparse representation. In *Proc. of the 27th Int'l Conf. on Machine learning (ICML'10)*, pages 567–574. Omnipress, 2010.

[16] A. Krause, H. B. McMahan, C. Guestrin, and A. Gupta. Robust submodular observation selection. *Journal of Machine Learning Research*, 9:2761–2801, 2008.

[17] L. Lovász. Submodular functions and convexity. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming – The State of the Art*, pages 235–257. 1983.

[18] K. Murota. *Discrete Convex Analysis*. Monographs on Discrete Math and Applications. SIAM, 2003.

[19] K. Nagano, Y. Kawahara, and S. Iwata. Minimum average cost clustering. In *Advances in Neural Information Processing Systems 23*, pages 1759–1767, 2010.

[20] M. Narasimhan and J. A. Bilmes. PAC-learning bounded tree-width graphical models. In *Proc. of the 20th Ann. Conf. on Uncertainty in Artificial Intelligence (UAI'04)*, pages 410–417, 2004.

[21] M. Narasimhan and J. A. Bilmes. A submodular-supermodular procedure with applications to discriminative structure learning. In *Proc. of the 21st Ann. Conf. on Uncertainty in Artificial Intelligence (UAI'05)*, pages 404–412, 2005.

[22] F. Pernkopf and J. A. Bilmes. Discriminative versus generative parameter and structure learning of bayesian network classifiers. In *Proc. of the 22nd Int'l Conf. on Machine Learning (ICML'05)*, pages 657–664, 2005.

[23] M. Queyranne. Minimizing symmetric submodular functions. *Math. Prog.*, 82(1):3–12, 1998.

[24] C. Rother, T. Minka, A. Blake, and V. Kolmogorov. Cosegmentation of image pairs by histogram matching-incorporating a global constraint into mrfs. In *Proc. of the 2006 IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition (CVPR'06)*, pages 993–1000, 2006.

[25] A. Shekhovtsov. Supermodular decomposition of structural labeling problem. *Control Systems and Computers*, 20(1):39–48, 2006.

[26] A. Shekhovtsov, V. Kolmogorov, P. Kohli, V. Hlav c, C. Rother, and P. Torr. Lp-relaxation of binarized energy minimization. Technical Report CTU-CMP-2007-27, Czech Technical University, 2007.

[27] M. Thoma, H. Cheng, A. Gretton, H. Han, H. P. Kriegel, A. J. Smola, S. Y. Le Song Philip, X. Yan, and K. Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *Proc. of the 2009 SIAM Conf. on Data Mining (SDM'09)*, pages 1076–1087, 2008.

9