MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Continuous Relaxation for Discrete DC Programming

Takanori MAEHARA, Naoki MARUMO,
and Kazuo MUROTA

METR 2015–26 August 2015

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: http://www.keisu.t.u-tokyo.ac.jp/research/techrep/index.html

Continuous Relaxation for Discrete DC Programming *

Takanori Maehara

Department of Mathematical and Systems Engineering,
Shizuoka University
maehara.takanori@shizuoka.ac.jp

Naoki Marumo and Kazuo Murota

Department of Mathematical Informatics,
University of Tokyo
naoki_marumo@mist.i.u-tokyo.ac.jp,
murota@mist.i.u-tokyo.ac.jp

August 2015

Abstract

Discrete DC programming with convex extensible functions is studied. A natural approach for this problem is a continuous relaxation that extends the problem to a continuous domain and applies the algorithm in continuous DC programming. By employing a special form of continuous relaxation, which is named "lin-vex extension," the optimal solution of the continuous relaxation coincides with the original discrete problem. The proposed method is demonstrated for the degree-concentrated spanning tree problem, the unfair flow problem, and the correlated knapsack problem.

*A preliminary version of this paper is included in the Proceedings of the 3rd International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences (MCO 2015, Metz, May 13–15) — Part I, Advances in Intelligent Systems and Computing, vol.359, Springer, 2015, pp. 181–190.

1 Introduction

DC programming, minimization of a difference of two convex functions, is an established area in nonconvex optimization. Most of the optimization problems can be represented as a DC programming problem [9, 27–29] and many practically efficient algorithms are developed. Two different approaches can be distinguished in the solution method for DC programming problems. The first is the *combinatorial approach* [8, 9, 28], which employs techniques such as branch-and-bound and cutting plane method for global optimization. The second is the *convex analysis approach* [26, 27], which utilizes the concepts in convex analysis, such as biconjugacy and subgradient. Combinatorial or discrete optimization problems have also been treated via DC programming [7, 23, 24].

Recently, Maehara and Murota [17] proposed a framework of discrete DC programming using discrete convex analysis [4, 20, 21]. A function $f : \mathbb{Z}^n \to \mathbb{Z} \cup \{-\infty, +\infty\}$ is defined to be a discrete DC function if it can be represented as $f = g - h$ with two discrete convex ($\mathrm{M}^\natural$-convex and/or $\mathrm{L}^\natural$-convex) functions $g, h : \mathbb{Z}^n \to \mathbb{Z} \cup \{+\infty\}$. This framework contains minimization of a difference of two submodular functions, which often appears in machine learning [11, 13, 22]. The methods of Narasimhan and Bilmes [22], Iyer, Jegelka, and Bilmes [11], and Maehara and Murota [17] are categorized as convex analysis approach, whereas that of Kawahara and Washio [13] as combinatorial approach.

In this paper, we are dealing with a larger class of discrete convex functions, convex extensible functions. A discrete-variable real-valued function $g : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ is said to be *convex extensible* if it can be interpolated by a convex function $\hat{g} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ in continuous variables. $\mathrm{M}^\natural$-convex and $\mathrm{L}^\natural$-convex functions are known to be convex extensible. Convex extensibility is a natural property required of discrete convex functions, but it is considered too weak for a rich theory. In fact, any function g defined on a unit cube $\{0, 1\}^n$ is convex extensible. Not much theory has been developed so far for convex extensible functions.

Here, we study a discrete DC programming problem

$$\text{minimize } f(x) := g(x) - h(x) \text{ s.t. } x \in \mathbb{Z}^n \tag{1.1}$$

with convex extensible functions g and h. This class of discrete optimization problems arise in many applications. For example, a discrete optimization problem with a continuous objective function restricted to the integer domain usually falls into this category.

A natural approach for such problem is *continuous relaxation*, which extends the discrete functions to the continuous domain and applies a continuous optimization method. As the continuous extension of $f = g - h$, we employ a special form, which we call a *lin-vex extension* $\tilde{f} = \overline{g} - \hat{h}$, where $\overline{g}$ is the convex closure (largest convex extension, usually piecewise linear) of g and $\hat{h}$ is any (often smooth) convex extension of h. A crucial fact (Theorem 3.6) is that no integral gap exists between the discrete optimization problem for f and the continuous optimization problem for its lin-vex extension $\tilde{f}$. This approach is useful in solving a discrete optimization problem having a "nice" DC representation. If an

objective function f is represented as $f = g - h$ such that the discrete optimization problem with g is efficiently solved and the subgradient of $\hat{h}$ is efficiently obtained, then the continuous DC algorithm for the lin-vex extension can be efficiently implemented and the obtained solution for the continuous relaxation is guaranteed to be an integral solution.

Use of continuous extension for discrete optimization problems is a standard technique. Integer programming problems are solved successfully via linear programing. In discrete convex analysis [4, 20, 21], in particular, we can design theoretically and practically faster algorithms by using continuous extensions and proximity theorems for $\mathrm{M}^\natural$-convex and $\mathrm{L}^\natural$-convex functions [18, 19].

To demonstrate the use of the proposed framework, we consider three problems. The first problem is a variant of the spanning tree problem, to be called *degree-concentrated spanning tree problem*, which finds a spanning tree with the maximum variance of degrees. Our experiment for a real-world network shows that the proposed DC algorithm works pretty well for this problem. The second problem is a variant of the minimum cost flow problem, to be called *unfair flow problem*, which finds a sparse supply vector with smaller cost value. This problem is an opposite of the fair flow problem [4, 10, 12], which aims at finding a supply vector x whose component values are distributed as evenly as possible. We formulate this problem in a concave-regularized minimum cost flow problem, which is a DC programming problem. Our experiment for random graphs shows that the DC algorithm successfully finds a desired solution. The third problem is a *correlated knapsack problem*, which is a knapsack problem in which utility (or profit) of chosen items is pairwise correlated. This problem can be formulated as a DC programming problem of the form of (1.1), but it turns out that the proposed DC algorithm is not better than the greedy algorithm.

2 Existing studies of DC programming

2.1 Continuous DC programming

Let $g : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a convex function. The effective domain of g is defined by $\mathrm{dom}_{\mathbb{R}}\, g := \{x \in \mathbb{R}^n : g(x) < +\infty\}$. Throughout the paper, we always consider functions with $\mathrm{dom}_{\mathbb{R}}\, g \neq \emptyset$. A vector $p \in \mathbb{R}^n$ is a *subgradient* of g at $x \in \mathrm{dom}_{\mathbb{R}}\, g$ if

$$g(y) \geq g(x) + \langle p, y - x \rangle \quad (y \in \mathbb{R}^n), \tag{2.1}$$

where $\langle p, x \rangle = \sum_{i=1}^n p_i x_i$ denotes the inner product. The set of all subgradients of g at x is called the *subdifferential* of g at x and denoted by $\partial_{\mathbb{R}} g(x)$. Every convex function g has a subgradient at each $x \in \mathrm{relint}(\mathrm{dom}_{\mathbb{R}}\, g)$, where relint denotes the relative interior.

The *Fenchel conjugate* $g^* : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ of a convex function $g : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is defined by

$$g^*(p) := \sup_{x \in \mathbb{R}^n} \{\langle p, x \rangle - g(x)\}, \tag{2.2}$$

Algorithm 1 DC algorithm.

1: Let $x \in \mathrm{dom}_{\mathbb{R}}\, g$ be an initial solution.
2: **repeat**
3: Find $p \in \partial_{\mathbb{R}} h(x)$
4: Find $x \in \partial_{\mathbb{R}} g^*(p)$
5: **until** convergence

which is a convex function. When g is a closed proper convex function (with all level sets closed), we have $g^{**} = g$. This property is called *biconjugacy*.

A function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty, -\infty\}$ is called a *DC function* if it can be represented as a difference of two convex functions g and h, i.e., $f = g - h$. To guarantee $f > -\infty$, we always assume $\mathrm{dom}_{\mathbb{R}}\, g \subseteq \mathrm{dom}_{\mathbb{R}}\, h$, and define $(+\infty) - (+\infty) = +\infty$. A *DC programming problem* is a minimization problem for a DC function.

In the convex analysis approach to DC programming, the most important fact is the Toland–Singer duality, which can be established by a direct calculation using biconjugacy.

Theorem 2.1 (Toland–Singer duality)**.** For closed convex functions $g, h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, we have

$$\inf_{x \in \mathbb{R}^n} \{g(x) - h(x)\} = \inf_{p \in \mathbb{R}^n} \{h^*(p) - g^*(p)\}. \tag{2.3}$$

DC algorithm [27,28] is a practically efficient algorithm for finding a local optimal solution of a DC programming problem. It starts from an initial solution $x^{(0)} \in \mathrm{dom}_{\mathbb{R}}\, g$, and repeats the following process until convergence. Let $x^{(\nu)}$ be the ν-th solution. DC algorithm approximates the function h for the concave part by its subgradient, $h(x) \approx h(x^{(\nu)}) + \langle p, x - x^{(\nu)} \rangle$, and minimize the convex function $g(x) - h(x^{(\nu)}) - \langle p, x - x^{(\nu)} \rangle$ to determine the next solution $x^{(\nu+1)}$. Since $x \in \mathrm{argmin}_{y \in \mathbb{R}^n} \left(g(y) - h(x^{(\nu)}) - \langle p, y - x^{(\nu)} \rangle \right)$ is equivalent to $x \in \partial_{\mathbb{R}} g^*(p)$, the algorithm is expressed simply as in Algorithm 1. When the algorithm terminates, we obtain a pair of vectors (x, p) such that $p \in \partial_{\mathbb{R}} g(x) \cap \partial_{\mathbb{R}} h(x)$. If both g and h are differentiable, this condition is equivalent to $p = \nabla g(x) = \nabla h(x)$, which implies $\nabla f(x) = \nabla g(x) - \nabla h(x) = 0$. Thus the DC algorithm terminates at a stationary point.

2.2 Discrete DC programming

Extending DC programming to discrete setting is a natural idea to conceive. But we must specify what we mean by "convex functions" in a discrete space. Moreover, such discrete convex functions should satisfy "subdifferentiability (existence of subgradients)" and "biconjugacy." Here, discrete versions of the subgradient and Fenchel conjugate are defined similarly to (2.1) and (2.2) with "$\mathbb{R}$" replaced by "$\mathbb{Z}$." In particular, we denote by $\partial_{\mathbb{Z}} g(x)$ the set of all integral subgradients of g at x.

Table 1: Complexity of discrete DC programming $\min\{g(x) - h(x)\}$ [17].

(a) $x \in \mathbb{Z}^n$

$g\backslash h$	$\mathrm{M}^\natural$	$\mathrm{L}^\natural$
$\mathrm{M}^\natural$	NP-hard	NP-hard
$\mathrm{L}^\natural$	open	NP-hard

(b) $x \in \{0,1\}^n$

$g\backslash h$	$\mathrm{M}^\natural$	$\mathrm{L}^\natural$
$\mathrm{M}^\natural$	NP-hard [14]	NP-hard
$\mathrm{L}^\natural$	P	NP-hard

A theory of discrete DC programming has been proposed recently by Maehara and Murota [17] using discrete convex analysis [4, 20, 21]. A function $g : \mathbb{Z}^n \to \mathbb{Z} \cup \{+\infty\}$ is called *$M^\natural$-convex* if it satisfies a certain exchange axiom. A linear function on a (poly)matroid is a typical example of $\mathrm{M}^\natural$-convex functions, and a matroid rank function is an $\mathrm{M}^\natural$-concave function. A function $g : \mathbb{Z}^n \to \mathbb{Z} \cup \{+\infty\}$ is called *$L^\natural$-convex* if it satisfies the translation submodularity. A submodular set function is a typical example of $\mathrm{L}^\natural$-convex functions. $\mathrm{M}^\natural$-convex and $\mathrm{L}^\natural$-convex functions are endowed with nice properties related to subgradient and biconjugacy. A *discrete DC function* means a function $f : \mathbb{Z}^n \to \mathbb{Z} \cup \{+\infty\}$ that can be represented as a difference of two discrete convex functions g and h, i.e., $f = g - h$. Since there are two classes of discrete convex functions ($\mathrm{M}^\natural$-convex functions and $\mathrm{L}^\natural$-convex functions), there are four types of discrete DC functions (an $\mathrm{M}^\natural$-convex function minus an $\mathrm{M}^\natural$-convex function, an $\mathrm{M}^\natural$-convex function minus an $\mathrm{L}^\natural$-convex function, and so on).

Minimization problems of discrete DC functions are referred to as *discrete DC programming problems*. According to the four classes of discrete DC functions, we have four classes of discrete DC programming problems. The computational complexity of these four classes is summarized in Table 1. It is noted that the NP-hardness of $\mathrm{M}^\natural - \mathrm{M}^\natural$ DC programming has been shown recently [14] through a reduction from the maximum clique problem.

The Toland–Singer duality is extended to the discrete case.

Theorem 2.2 (Discrete Toland–Singer duality [17])**.** For $\mathrm{M}^\natural$- and/or $\mathrm{L}^\natural$-convex functions $g, h : \mathbb{Z}^n \to \mathbb{Z} \cup \{+\infty\}$, we have

$$\inf_{x \in \mathbb{Z}^n} \{g(x) - h(x)\} = \inf_{p \in \mathbb{Z}^n} \{h^*(p) - g^*(p)\}. \tag{2.4}$$

A discrete version of the DC algorithm can also be defined similarly with $\mathbb{R}$ in Algorithm 1 replaced by "$\mathbb{Z}$." Each step of the algorithm, $p \in \partial_{\mathbb{Z}} h(x)$ and $x \in \partial_{\mathbb{Z}} g^*(p)$, can be executed efficiently by using the existing algorithms in discrete convex analysis. Moreover, by exploiting polyhedral properties of $\mathrm{M}^\natural$-convex and $\mathrm{L}^\natural$-convex functions, we can guarantee a stronger local optimality condition. See [17] for more details of discrete DC programming.

3 Continuous relaxation for discrete DC programming

A continuous relaxation framework for minimizing the difference of two convex extensible discrete functions is presented in this section.

3.1 Continuous extension of discrete DC function

Let $g : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be a real-valued function in discrete variables. A *convex extension* of g means a convex function $\hat{g} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ (in continuous variables) that satisfies

$$\hat{g}(x) = g(x) \quad (x \in \mathbb{Z}^n). \tag{3.1}$$

We say that g is *convex extensible* if it admits a convex extension.

The following examples demonstrate some typical convex extensible functions.

Example 3.1. Often a discrete function $g : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ is defined in terms of some continuous convex function $\hat{g} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ as $g(x) := \hat{g}(x)$. Such function g is obviously convex extensible.

Example 3.2 ([20])**.** A univariate discrete function $g : \mathbb{Z} \to \mathbb{R} \cup \{+\infty\}$ is convex extensible if (and only if) it satisfies

$$g(x-1) + g(x+1) \geq 2g(x) \quad (x \in \mathbb{Z}). \tag{3.2}$$

Example 3.3. Any function g on $\{0,1\}$-vectors, $g : \{0,1\}^n \to \mathbb{R} \cup \{+\infty\}$, is convex extensible.

Example 3.4. L-convex, $\mathrm{L}^\natural$-convex, M-convex, and $\mathrm{M}^\natural$-convex functions in discrete convex analysis [20] are convex extensible. Integrally convex functions [3, 20] and BS-convex functions [5] are convex extensible.

Example 3.5. A sum of convex extensible functions is also convex extensible.

Note that there are (possibly) many convex extensions for a discrete function g. The *convex closure* $\overline{g} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ of g is the point-wise maximum of all affine functions that are global underestimators of g, i.e.,

$$\overline{g}(x) := \sup\{\ell(x) : \ell \text{ affine}, \ell(y) \leq g(y) \ (y \in \mathbb{Z}^n)\}. \tag{3.3}$$

Under mild assumptions (e.g., if the effective domain is bounded), the convex closure of a function on $\mathbb{Z}^n$ is a piecewise linear function. In this paper, we always assume that, if g is convex extensible, then $g(x) = \overline{g}(x)$ for $x \in \mathbb{Z}^n$ and the supremum in (3.3) is attained for each x. Figure 1 illustrates the difference between a convex extension and the convex closure.

(a) Convex extension. (b) Convex closure.

Figure 1: Convex extension and convex closure.

3.2 Lin-vex extension of discrete DC function

We consider a discrete DC programming problem:

$$\text{minimize } g(x) - h(x) \text{ s.t. } x \in \mathbb{Z}^n, \tag{3.4}$$

which is represented in terms of two convex extensible functions g and h. A natural approach to this problem is continuous relaxation that extends the objective function to the continuous domain and solves the continuous optimization problem by some existing method in continuous optimization.

A special form of continuous relaxation is particularly convenient in this context. Let $\overline{g}$ be the convex closure of g and $\hat{h}$ be any convex extension of h that is continuous on $\operatorname{dom} \hat{h}$. The function defined by

$$\tilde{f}(x) = \overline{g}(x) - \hat{h}(x) \tag{3.5}$$

is named here a *lin-vex extension* of $f = g - h$, where "lin-vex" is intended to mean "piecewise linear for g and general convex for h." By definition, we have $f(x) = \tilde{f}(x)$ for all $x \in \mathbb{Z}^n$; therefore

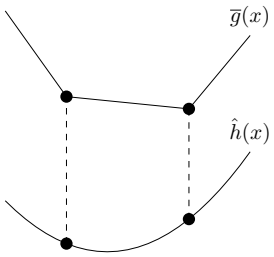
$$\inf_{x \in \mathbb{Z}^n} f(x) \geq \inf_{x \in \mathbb{R}^n} \tilde{f}(x). \tag{3.6}$$

In discrete (or integer) optimization, in general, the optimal values of the original problem and that of a continuous relaxation are different, and the discrepancy between these optimal values are referred to as the *integrality gap*. Fortunately, however, our continuous relaxation based on lin-vex extension does not suffer from integrality gap (see Figure 2).

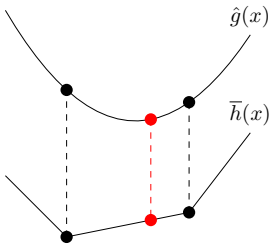
Theorem 3.6. For convex extensible functions $g, h : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ with $\operatorname{dom}_{\mathbb{Z}} g$ bounded and $\operatorname{dom}_{\mathbb{Z}} g \subseteq \operatorname{dom}_{\mathbb{Z}} h$, we have

$$\inf_{x \in \mathbb{Z}^n} \{g(x) - h(x)\} = \inf_{x \in \mathbb{R}^n} \{\overline{g}(x) - \hat{h}(x)\}. \tag{3.7}$$

Proof. By our assumptions, $\overline{g}$ and $\hat{h}$ are continuous on their effective domains. Moreover, since $\operatorname{dom}_{\mathbb{Z}} g$ is bounded, $\operatorname{dom}_{\mathbb{R}} \overline{g}$ is compact. Therefore $\overline{g} - \hat{h}$ attains the minimum in $\operatorname{dom}_{\mathbb{R}} \overline{g}$. Let $x^* \in \operatorname{dom}_{\mathbb{R}} \overline{g}$ be a minimizer of $\overline{g} - \hat{h}$. Since $\overline{g}$ is



(a) Lin-vex extension.



(b) Vex-lin extension

Figure 2: Difference between lin-vex extension $\overline{g}(x)-\hat{h}(x)$ and vex-lin extension $\hat{g}(x)-\overline{h}(x)$. In lin-vex extension, (a), the minimum is attained at an integral point; however, in vex-lin extension, (b), the minimum is attained at a non-integral point.

a piecewise linear function, we can take a convex polyhedron R such that $\overline{g}$ is linear on R and $x^* \in R$. Since $\overline{g}$ is linear on R, $\overline{g}-\hat{h}$ is concave on R; therefore its minimum is attained at an extreme point of R, which is integral. □

The lin-vex extension of f has two kinds of freedoms. First, it depends on the DC representation $f = g - h$. For an arbitrary convex extensible function $k : \mathbb{Z}^n \to \mathbb{R}$, we can obtain another DC representation $f = (g + k) - (h + k)$, and the corresponding lin-vex extension may change. Second, it depends on the choice of convex extension $\hat{h}$ of h. The convex closure $\overline{h}$ is eligible for $\hat{h}$, but in some cases, there can be a more suitable choice for $\hat{h}$. For example, if h is defined by the restriction of a continuous (smooth) convex function $\varphi : \mathbb{R}^n \to \mathbb{R}\cup\{+\infty\}$, i.e., $h(x) = \varphi(x)$ for $x \in \mathbb{Z}^n$, then φ is a reasonable candidate for $\hat{h}$. We intend to make use of these freedoms to design an efficient algorithm.

Remark 3.7. Theorem 3.6 does not hold for a continuous extension of the form $\hat{g} - \hat{h}$. For example, let us consider $\hat{g}(x) = (x - 1/2)^2$ and $\hat{h}(x) = 0$ for $x \in \mathbb{R}$, and $g(x) = \hat{g}(x)$ and $h(x) = \hat{h}(x)$ for $x \in \mathbb{Z}$. Then we have $\inf_{x\in\mathbb{Z}}\{g(x) - h(x)\} = 1/4 \neq 0 = \inf_{x\in\mathbb{R}}\{\hat{g}(x) - \hat{h}(x)\}$. See also Figure 2b, which demonstrates that "vex-lin" extension $\hat{g} - \overline{h}$ does not work, either.

Remark 3.8. In convex analysis, the *Legendre–Fenchel duality*

$$\inf_{x\in\mathbb{R}^n}\{g(x) + h(x)\} = -\inf_{p\in\mathbb{R}^n}\{g^*(p) + h^*(-p)\} \tag{3.8}$$

is frequently used and a discrete version of (3.8) is also known in discrete convex analysis. It should be clear that the Toland–Singer duality (2.3) deals with the infimum of $g-h$, but the Legendre–Fenchel duality (3.8) deals with the infimum of $g + h$. For the Legendre–Fenchel duality, there is an integrality gap, i.e.,

$$\inf_{x\in\mathbb{Z}^n}\{g(x) + h(x)\} \neq \inf_{x\in\mathbb{R}^n}\{\overline{g}(x) + \hat{h}(x)\}, \tag{3.9}$$

in general. For example, let $g(x_1, x_2) = |x_1 + x_2 - 1|$ and $h(x_1, x_2) = |x_1 - x_2|$ for $(x_1, x_2) \in \mathbb{Z}^2$, and $\overline{g}(x_1, x_2) = |x_1 + x_2 - 1|$ and $\hat{h}(x_1, x_2) = |x_1 - x_2|$ for $(x_1, x_2) \in \mathbb{R}^2$. Then, the left-hand side of (3.9) is $\inf\{g(x_1, x_2) + h(x_1, x_2)\} = 1$ attained at $(x_1, x_2) = (1, 0), (0, 1)$, and the right-hand side is $\inf\{\overline{g}(x_1, x_2) + \hat{h}(x_1, x_2)\} = 0$ attained at $(x_1, x_2) = (1/2, 1/2)$.

3.3 DC algorithm for lin-vex extension

By Theorem 3.6, solving a convex extensible DC problem (left-hand side of (3.7)) is equivalent to solving its lin-vex relaxation problem (right-hand side of (3.7)), which is a continuous DC programming problem. Here, we consider how to carry out the DC algorithm for this continuous DC programming problem.

As shown in Algorithm 1, the DC algorithm for $\inf_{x \in \mathbb{R}^n}\{\overline{g}(x) - \hat{h}(x)\}$ repeats the dual step $p \in \partial_{\mathbb{R}} \hat{h}(x)$ and the primal step $x \in \partial_{\mathbb{R}} \overline{g}^*(p)$. In some cases these two steps can be carried out easily and efficiently. The dual step of finding a subgradient p of $\hat{h}$ at x can be done efficiently, if the convex extension $\hat{h}$ can be chosen to be a smooth function. For example, if $h : \mathbb{Z}^n \to \mathbb{R}$ is given by $h(x) = x^\top A x$ $(x \in \mathbb{Z}^n)$ for some positive semidefinite matrix A, we can take $\hat{h}(x) = x^\top A x$ $(x \in \mathbb{R}^n)$, whose subgradient is explicitly obtained as $\partial_{\mathbb{R}} \hat{h}(x) = \{2Ax\}$. For the primal step, recall that $x \in \partial_{\mathbb{R}} \overline{g}^*(p)$ is equivalent to

$$x \in \operatorname*{argmin}_{y \in \mathbb{R}^n}\{\overline{g}(y) - \langle p, y\rangle\}. \tag{3.10}$$

Since $\overline{g}$ is a piecewise linear function and its linearity domain is an integral polyhedron, the problem (3.10) has an integral optimal solution, provided that $\mathrm{dom}_{\mathbb{Z}}\, g$ is bounded. Therefore, the problem (3.10) is essentially equivalent to the discrete optimization problem

$$x \in \operatorname*{argmin}_{y \in \mathbb{Z}^n}\{g(y) - \langle p, y\rangle\}. \tag{3.11}$$

Here, we emphasize that considering the convex closure of g is conceptually important, but we do not really need its specific form, because we can minimize the convex closure $\overline{g}$ by minimizing the original discrete function g.

We assume that the minimization problem (3.11) can be solved efficiently. This is the case, for example, if g is an $\mathrm{M}^\natural$-convex or $\mathrm{L}^\natural$-convex function, or if the problem is a (practically) solvable problem such as matching problem on a graph, maximum independent set problem on a tree, knapsack problem, etc.

The proposed method based on lin-vex extension is summarized in Algorithm 2. Recall that we consider a discrete DC programming problem:

$$\text{minimize } g(x) - h(x) \text{ s.t. } x \in \mathbb{Z}^n$$

in (3.4), where g and h are convex extensible. It is emphasized again that the convex closure $\overline{g}$ is uniquely determined but a choice can be made for the convex extension $\hat{h}$.

Algorithm 2 DC algorithm based on lin-vex extension.

1: Choose a convex extension $\hat{h}$ for h.
2: Let $x \in \mathrm{dom}_{\mathbb{Z}} g$ be an initial solution.
3: **repeat**
4: Find $p \in \partial_{\mathbb{R}} \hat{h}(x)$,
5: Find $x \in \mathbb{Z}^n$ that minimizes $g(x) - \langle p, x \rangle$.
6: **until** convergence

Under mild assumptions (e.g., if the effective domain is bounded), after finite iterations, Algorithm 2 terminates with a pair of vectors (x^*, p^*) such that $p^* \in \partial_{\mathbb{R}} \hat{h}(x^*)$ and $x^* \in \partial_{\mathbb{R}} \bar{g}^*(p^*)$. Thus we have

$$\partial_{\mathbb{R}} \hat{h}(x^*) \cap \partial_{\mathbb{R}} \bar{g}(x^*) \neq \emptyset. \tag{3.12}$$

If $\hat{h}$ is differentiable, i.e., its subgradient consists of a single element as $\partial_{\mathbb{R}} \hat{h}(x) = \{\nabla \hat{h}(x)\}$, the above condition is reduced to

$$\nabla \hat{h}(x^*) \in \partial_{\mathbb{R}} \bar{g}(x^*), \tag{3.13}$$

which implies

$$g(x) - g(x^*) \geq \langle \nabla \hat{h}(x^*), x - x^* \rangle \tag{3.14}$$

for any $x \in \mathbb{Z}^n$.

Remark 3.9. As mentioned in Introduction, combinatorial or discrete optimization problems have been treated via continuous DC programming [7,23,24], but the existing methods are completely different from ours. Specifically, the method proposed in [23] minimizes a discrete function $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ that has a convex extension $\hat{f} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ as follows. Let

$$\phi(x) := \sum_{i=1}^{n} (1 - \cos(2\pi x_i))^2 .$$

Then $\phi(x) = 0$ if and only if $x \in \mathbb{Z}^n$. Therefore, for a sufficiently large $\mu > 0$, we have

$$\min_{x \in \mathbb{Z}^n} f(x) = \min_{x \in \mathbb{R}^n} \hat{f}(x) + \mu \phi(x).$$

Since the Hessian of $\phi(x)$ is bounded, it has a DC representation $\phi(x) = (\lambda \|x\|^2 + \phi(x)) - \lambda \|x\|^2$ for a sufficiently large $\lambda > 0$. Therefore, if $\hat{f}$ has a DC representation, the above problem can be expressed as a DC programming problem.

4 Applications

4.1 Degree-concentrated spanning tree problem

We consider a problem in network routing [25]. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E, where V represents a set of computers and E the connection of the computers: $(i, j) \in E$ if computer i communicates with computer j. The *spanning tree routing* is a routing system such that all packets are sent along a spanning tree.

Let us consider monitoring of network communications by observing packets on vertices. Naturally, much information can be obtained at high-degree vertices. Thus, for efficient monitoring, we want to construct a spanning tree with high-degree vertices. This problem can be formulated as constructing a spanning tree T that has the maximum sum of squares of the degrees; note that the sum of the degrees is constant for spanning trees. That is, we consider the following problem, to be named *degree-concentrated spanning tree problem*:

$$\text{maximize} \sum_{v \in V} \deg_T(v)^2 \text{ s.t. } T : \text{spanning tree}, \tag{4.1}$$

where $\deg_T(v)$ is the degree of a vertex v in the spanning tree T.

This problem is NP-hard [6] as follows.

Proposition 4.1. The degree-concentrated spanning tree problem for cubic graphs is NP-hard.

Proof. We prove the claim by a reduction from the *maximum leaf spanning tree problem*, which is known to be NP-hard for cubic graphs [6, 16].

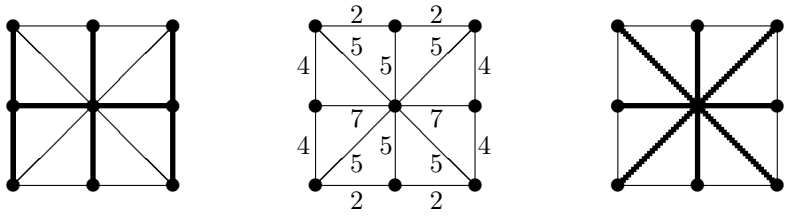
Let T be a spanning tree on a cubic graph with n vertices, and let $x(T)$ denote the sum of squares of degrees on T. Let $a(T)$, $b(T)$, and $c(T)$ be the numbers of vertices of degrees 1, 2, and 3 on T, respectively. Then we have $a(T) + b(T) + c(T) = n$, $a(T) + 2b(T) + 3c(T) = 2n - 2$, and $a(T) + 4b(T) + 9c(T) = x(T)$. From these equations, we obtain $a(T) = x(T)/2 + 5 - 2n$. Therefore, maximizing $a(T)$, which is the maximum leaf spanning tree problem, is equivalent to maximizing $x(T)$, which is the degree-concentrated spanning tree problem. □

The above problem can be formulated in a DC programming problem as follows. Let $B \in \mathbb{R}^{|V| \times |E|}$ be the incidence matrix of graph G, i.e., $B_{ie} = 1$ if an edge e is incident to a vertex i. Then, for $x \in \{0, 1\}^{|E|}$ and $T = \{e \in E : x_e = 1\}$, we have

$$Bx = (\deg_T(v_1), \ldots, \deg_T(v_{|V|}))^\top. \tag{4.2}$$

Therefore, for $A = B^\top B \in \mathbb{R}^{|E| \times |E|}$, we have

$$x^\top A x = \sum_{v \in V} \deg_T(v)^2. \tag{4.3}$$



(a) Current solution. (b) Gradient (divided by 2). (c) Updated solution.

Figure 3: Illustration of the DC algorithm for the degree-concentrated spanning tree problem. For a current solution shown in (a), the gradient p is given as (b), and the updated solution (c) is a maximum spanning tree with respect to p.

We define the concave part by $h(x) = x^\top Ax$ and the convex part by

$$g(x) = \begin{cases} 0, & \{e \in E : x_e = 1\} \text{ is a spanning tree,} \\ +\infty, & \text{otherwise.} \end{cases} \tag{4.4}$$

Then we have

$$g(x) - h(x) = \begin{cases} -\sum_{v \in V} \deg_T(v)^2, & T = \{e \in E : x_e = 1\} \text{ is a spanning tree,} \\ +\infty, & \text{otherwise.} \end{cases} \tag{4.5}$$

Thus the minimization problem for $f(x) = g(x) - h(x)$ coincides with the degree-concentrated spanning tree problem. It is known that $g(x)$ in (4.4) is an M-convex function [20, Example 6.27], whereas $h(x) = x^\top Ax$ is neither $\mathrm{L}^\natural$-convex nor $\mathrm{M}^\natural$-convex.

In this representation, both g and h are convex extensible. The gradient of $\hat{h}(x) = x^\top Ax$ is explicitly obtained as $\partial_{\mathbb{R}} \hat{h}(x) = \{p\} = \{2Ax\}$; here $(Ax)_{(u,v)} = \deg_T(u) + \deg_T(v)$. The minimization of $g(x) - \langle p, x \rangle$ is performed by solving a maximum spanning tree problem with edge weight p_e for $e \in E$; see Figure 3 for the illustration of the algorithm. If there are two or more optimal solutions in this minimization problem, we randomly choose one of them. The algorithm terminates when the objective value is not improved, i.e., $f(x^{(\nu)}) = f(x^{(\nu+1)})$. Thus, the local optimal solution for the degree-concentrated spanning tree problem can be found efficiently by the DC algorithm. The complexity of the algorithm is $O(|E| \log |E|)$ for each iteration.

To evaluate the performance of the above DC algorithm and the quality of solutions, we conducted the following experiment. We used a real-world network, `p2p-Gnutella08`, obtained from Stanford Large Network Dataset Collection,[1] representing a network of a peer-to-peer communication network.

We performed the proposed algorithm 1000 times. The number of iterations for convergence and the obtained objective values are shown in Figures 4a and

[1] `http://snap.stanford.edu/data/` For other real-world networks in this collection, we performed the same experiments, to obtain similar results.

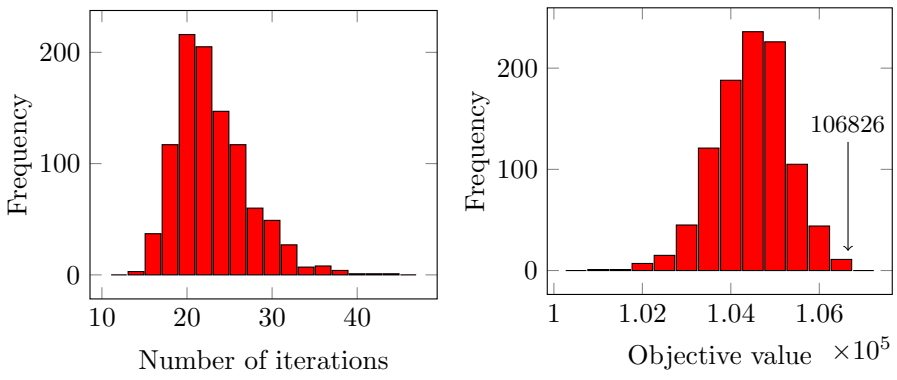


Figure 4: Distribution of the number of iterations (a) and objective values (b) of the DC algorithm for the degree-concentrated spanning tree problem.

4b, respectively. The number of iterations stays in the range $[14, 44]$, and the average is 23.49 with the standard deviation of 4.30. This shows that, since the number of iterations is small, the DC algorithm is efficient and scales to large instances. The objective values are contained in the range $[101420, 106826]$, and the average is 104717.79 with the standard deviation of 818.51. With probability $\geq 95\%$, the objective value is greater than 104192, which is 97% of the maximum objective value.

To see the detail of the convergence of the DC algorithm, we select four runs and plot the objective values in the first 10-iterations in Figure 5; each plot corresponds to a single run of the algorithm. This shows that a solution of the DC algorithm quickly approaches the optimal solution in the first few iterations.

For comparison, we also implemented the greedy algorithm that iteratively selects an edge e^* randomly from $\mathrm{argmin}_{e \notin T} f(T \cup \{e\})$ and updates the solution T to $T \cup \{e^*\}$. We performed the greedy algorithm 1000 times. The best greedy solution has objective value 87016, which is also shown in Figure 5 as a horizontal line. The greedy solution is outperformed by the DC algorithm in the second iteration. Note that the greedy solution is out-of-range in Figure 4b because it is too small, i.e., all solutions obtained by the DC algorithm outperformed the greedy solution. Thus, the DC algorithm works pretty well for the degree-concentrated spanning tree problem.

Remark 4.2. In contrast to (4.1) we may also consider

$$\text{minimize} \sum_{v \in V} \deg_T(v)^2 \text{ s.t. } T : \text{spanning tree}, \tag{4.6}$$

which is named *degree-balanced minimum spanning tree problem*. This problem is also NP-hard because the optimal solution is a Hamilton path if it exists.

This problem can also be formulated as a DC programming problem as

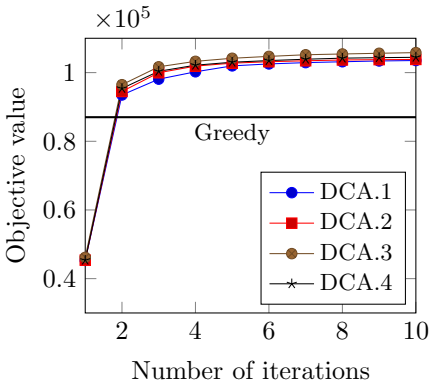


Figure 5: Objective values vs iterations for the DC algorithm for the degree-concentrated spanning tree problem.

follows. By introducing an L_2-regularization factor we define g and h as

$$g(x) = \begin{cases} \mu\|x\|^2, & \{e \in E : x_e = 1\} \text{ is a spanning tree,} \\ +\infty, & \text{otherwise,} \end{cases}$$
$$h(x) = -x^\top Ax + \mu\|x\|^2,$$

where $A = B^\top B$ as above, and $\mu > 0$ is a constant large enough to make $\mu I - A$ positive semidefinite.

The minimization of $g(x) - h(x)$ coincides with the degree-balanced spanning tree problem. In our computational experiment, however, the DC algorithm for this formulation did not work well. This phenomenon can be explained theoretically for regular graphs as follows.

Let G be an r-regular graph. To make $\mu I - A$ positive semidefinite, the regularization parameter μ must satisfy $\mu \geq \lambda_{\max}(A)$, where $\lambda_{\max}(A)$ is the maximum eigenvalue of A. With the estimate $\lambda_{\max}(A) \geq u^\top Au/\|u\|^2 = \|Bu\|^2/\|u\|^2 = 2r$ for $u = (1, \ldots, 1)^\top$, we obtain $\mu \geq 2r$. Now we can see that such μ is too large for the primal step. Consider a gradient vector $p = \nabla h(x) = 2(\mu x - Ax)$ at any solution x. If $x_i = 0$, then $p_i = -2e_i Ax < 0$ because $e_i Ax$ is the number of edges in x that touch edge i. If $x_i = 1$, then $p_i = 2(\mu - 2r) \geq 0$. This means that the primal step for $p = \nabla h(x)$ does not change the solution from x, i.e., the DC algorithm does not improve an initial solution.

4.2 Unfair flow problem

We consider a network flow problem. Let $G = (V, E)$ be a graph and $\gamma_e : \mathbb{Z} \to \mathbb{R} \cup \{+\infty\}$ be convex functions associated with edges $e \in E$. For an integral flow $\phi : E \to \mathbb{Z}$, its cost $\Gamma(\phi)$ is defined by

$$\Gamma(\phi) := \sum_{e \in E} \gamma_e(\phi(e)). \tag{4.7}$$

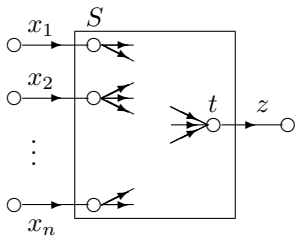


Figure 6: A network with n input terminals S and a single output terminal t.

Suppose that an input terminal set $S \subset V$, an output terminal $t \in V \setminus S$, and the flow demand $z \geq 0$ at the output terminal are given (Figure 6). The flow supply at the input terminals is specified by a vector $x \in \mathbb{Z}^S$, where $\sum_{s \in S} x_s = z$ and $x_s \geq 0$ $(s \in S)$ are assumed. There are possibly many integral flows ϕ that satisfy the supply/demand condition:

$$\sum\{\phi(e) : \text{tail of } e \text{ is } s\} - \sum\{\phi(e) : \text{head of } e \text{ is } s\} = x_s \quad (s \in S), \tag{4.8}$$

$$\sum\{\phi(e) : \text{tail of } e \text{ is } t\} - \sum\{\phi(e) : \text{head of } e \text{ is } t\} = -z, \tag{4.9}$$

as well as the conservation condition at vertices in $V \setminus (S \cup \{t\})$. We define a function $g : \mathbb{Z}^S \to \mathbb{R} \cup \{+\infty\}$ as the minimum cost of such a flow:

$$g(x) := \min\{\Gamma(\phi) : \phi \text{ is an integral flow that satisfies (4.8), (4.9)}\}. \tag{4.10}$$

It is known that $g(x)$ is an M-convex function [20, Chapter 2]. This implies, in particular, that the minimization of $g(x)$ can be done via a greedy algorithm, where the value of $g(x)$ itself can be computed by solving a minimum cost flow problem.

Here, we consider a *sparse optimization problem* [1,2] associated with g: Find a sparse vector x that has a smaller value of $g(x)$. We call this problem *unfair flow problem*, as it can be viewed as an opposite of the *fair flow problem* [4,10,12], which finds a vector x whose component values are distributed as evenly as possible.

Remark 4.3. The unfair flow problem admits an interpretation in the context of electrical circuits. Imagine an electrical circuit consisting of nonlinear resistors, with n input terminals and a single output terminal represented by a network $G = (V, E)$ as Figure 6. When a current of value I flows through an edge $e \in E$, the power consumption is represented by a convex function, say, $\gamma_e(I)$. For example, $\gamma_e(I) = RI^2/2$ for a linear (ohmic) resistor of resistance R. When the currents on E are represented by ϕ, the total power consumption in the circuit is given by $\Gamma(\phi)$ in (4.7). Suppose that a vector x of currents on the input terminals is given. Then, physically, the currents in the circuits are determined according to a variational principle that the current distribution ϕ should minimize the total power consumption $\Gamma(\phi)$. Therefore $g(x)$ defined in (4.10)

is equal to the physically realized power consumption for x. By Kirchhoff's current law, the output current is equal to $x_1 + \cdots + x_n$.

In this context the unfair flow problem may be regarded as a discrete version of the following problem: Find input currents x that satisfy the following conditions: (1) the output current is equal to a specified value z, (2) the power consumption is small, and (3) the number of used terminals is small. The third condition is the sparsity on the solution, which is helpful to simplify circuit operation.

Let us formulate the unfair flow problem as a DC programming problem:

$$\text{minimize } f_\lambda(x) := g(x) - h_\lambda(x) \text{ s.t. } x \in \mathbb{Z}^n \tag{4.11}$$

with

$$h_\lambda(x) := \lambda \|x\|^2 = \lambda \sum_{i=1}^{n} x_i^2, \tag{4.12}$$

where $n = |S|$ and $\lambda \geq 0$ is a regularization parameter. This is a minimization of the cost $g(x)$ regularized by a concave function $-h_\lambda(x)$. Since the concave regularization tends to give a sparse solution [15], the solution to the DC programming problem (4.11) is expected to be sparse. Note that $h_\lambda(x)$ is a separable convex function; thus it is an $\mathrm{L}^\natural$-convex and $\mathrm{M}^\natural$-convex function. Therefore, the problem (4.11) is a discrete DC programming problem in the sense of [17].

In the representation (4.11), both g and h_λ are convex extensible and the gradient of $\hat{h}_\lambda(x) = \lambda\|x\|^2$ is explicitly obtained as $\partial\hat{h}_\lambda(x) = \{p\} = \{2\lambda x\}$. The minimization of $g(x) - \langle p, x\rangle$ is performed by solving a minimum cost flow problem, where the cost of an input terminal s is specified by $-p_s$. Hence, if $x_s > x_{s'}$ for two terminals s and s', we have $p_s > p_{s'}$, and at the next step, a larger amount of flow will go through terminal s than through s'. Thus we can expect that the obtained solution tends to be concentrated (i.e., sparse).

To evaluate the performance of this algorithm, we conducted the following experiment. We constructed an Erdős-Rényi random graph $G = (V, E)$ with $|V| = 1000$ and $|E| = 3042$; each edge $e \in E$ having the same cost function given by $\gamma_e(I) = I^2$. We randomly varied 100 input terminals $S \subset V$ and a single output terminal $t \in V \setminus S$ and set the demand z at the output terminal to be 200. We solved (4.11) for various $\lambda \geq 0$ to find a sparse input flow x by using the DC algorithm. For comparison, we also implemented the greedy algorithm that iteratively selects an input terminal s from $\operatorname{argmin}_{s\in S} f_\lambda(x+e_s)$. We varied $\lambda \in \{0.00, 0.01, \ldots, 0.40\}$ for both algorithms. For the DC algorithm, we additionally varied $\lambda \in \{0.140, 0.141, \ldots, 0.150\}$ to observe the changes in the sensitive range.

Figures 7a and 7b respectively show the objective value $f_\lambda(x)$ and the number of nonzero elements of the solutions obtained by the algorithms for each λ. For $\lambda \leq 0.140$, both algorithms produce almost the same objective values and sparsities; therefore the plots are overlapped. For $\lambda > 0.140$, on the other hand, the DC algorithm outperforms the greedy algorithm in the sense that the DC

algorithm produces lower objective values and sparser solutions than the greedy algorithm. In particular, the number of nonzero elements of the solutions obtained by the DC algorithm quickly decreases to one from $\lambda = 0.140$ to 0.150. Compatibly with our original objective to find a sparse vector x with low cost $g(x)$, we show the relation between the number of nonzero elements and the cost value in Figure 8. The solutions obtained by the DC algorithm are located at the lower-left of the solutions obtained by the greedy algorithm. Therefore, the DC algorithm gives lower cost and sparser solution than the greedy algorithm.

4.3 Correlated knapsack problem

We recall the knapsack problem. Let $V = \{1, \dots, n\}$ be a set of items, $u_j \in \mathbb{R}$ ($u_j > 0$) be the utility of item $j \in V$, and $w_j \in \mathbb{Z}$ ($w_j > 0$) be the weight of item j. The knapsack problem maximizes the total utility $\sum_{j \in S} u_j$ by selecting a subset of items $S \subseteq V$ under the capacity constraint $\sum_{j \in S} w_j \leq W$ for a given capacity W.

Here, we consider the case that some items have *correlations*, i.e., if two items i and j are simultaneously selected, we get additional utility c_{ij} ($= c_{ji}$). Here, c_{ij} can be positive or negative; if $c_{ij} > 0$, these two items are *complementary goods* (e.g., coffee and sugar), and if $c_{ij} < 0$, they are *substitute goods* (e.g., coffee and tea). For simplicity, we assume that correlations c_{ij} are much smaller than the individual utility u_i. The correlated knapsack problem is to maximize the total utility by taking this correlation into account:

$$\begin{aligned} &\text{maximize} && \sum_{j \in V} u_j x_j + \sum_{i \neq j} c_{ij} x_i x_j \\ &\text{subject to} && \sum_{j \in V} w_j x_j \leq W, \end{aligned} \tag{4.13}$$

where $x \in \{0, 1\}^V$ represents the characteristic vector of a selected subset S.

Let $C = (c_{ij})$ be an $n \times n$ matrix whose entries represent correlations of items, where $c_{ii} = 0$ ($i = 1, \dots, n$), and let $h(x) = x^\top C x$. Define g as

$$g(x) = \begin{cases} 0, & \displaystyle\sum_{j \in V} w_j x_j \leq W, \\ +\infty, & \text{otherwise.} \end{cases} \tag{4.14}$$

Then the minimization of $f(x) = g(x) - \langle u, x \rangle - h(x)$ coincides with the correlated knapsack problem (4.13). However, $h(x)$ is not necessarily convex extensible.

We consider three different DC programming formulations. First, we set

$$g_1(x) = g(x) - \langle u, x \rangle + \mu \|x\|^2, \quad h_1(x) = x^\top (C + \mu I) x \tag{4.15}$$

for $\mu \geq \max\{0, -\lambda_{\min}(C)\}$, where $\lambda_{\min}(C)$ denotes the smallest eigenvalue of C. As the second choice, we set

$$g_2(x) = g(x) + \mu \|x\|^2, \quad h_2(x) = x^\top (C + \operatorname{diag}(u) + \mu I) x, \tag{4.16}$$

Table 2: Computational results for the correlated knapsack problem. "#Iterations" means the average number of iterations and "Objective values" are shown in the format of "(average) ± (standard deviation)" in 1000 trials.

Formulation	Initial solution	#Iterations	Objective value
DC (4.15): $g_1 - h_1$	zero vector	2.3	4737.81 ± 88.92
	uncorrelated opt.	2.0	4736.60 ± 80.90
DC (4.16): $g_2 - h_2$	zero vector	2.0	4447.30 ± 69.50
	uncorrelated opt.	2.0	4722.17 ± 77.20
DC (4.17): $g_3 - h_3$	zero vector	2.6	4699.44 ± 23.29
	uncorrelated opt.	2.2	$\mathbf{4886.64} \pm 85.61$
greedy	—	—	$\mathbf{4914.87} \pm 83.60$

where $\mu \geq \max\{0, -\lambda_{\min}(C + \mathrm{diag}(u))\}$. Note that $\langle u, x\rangle = x^\top \mathrm{diag}(u)\, x$ for a zero-one vector x. Finally, we set

$$g_3(x) = g(x), \quad h_3(x) = x^\top (C + \mathrm{diag}(d))x + \langle u - d, x\rangle \tag{4.17}$$

with $d_i = \sum_{j \neq i} |c_{ij}|$. Note that $C + \mathrm{diag}(d)$ is diagonally dominant, and hence positive semidefinte. For $k = 1, 2, 3$ we have $f(x) = g_k(x) - h_k(x)$, and $g_k(x)$ and $h_k(x)$ have natural convex extensions according to their representations. We also mention that $g_k(x)$ and $h_k(x)$ are neither $\mathrm{L}^\natural$-convex nor $\mathrm{M}^\natural$-convex.

The primal step (3.11) can be performed in $O(|V|W)$ time by dynamic programming, where we randomly perturb the utilities of items for tie-breaking. We try with two kinds of initial solutions: the zero vector and the optimal solution to the corresponding uncorrelated knapsack problem, which is obtained by dynamic programming. We refer to the latter by an abbreviation "uncorrelated optimal."

To generate problem instances we employed the standard generator by David Pisinger[2] with parameters `100 200 3 1 1000`, which means that we obtain 1000 instances, each of which has $n = 100$ items and strongly correlated profit and capacity with $u_j, w_j \in [1, 200]$. Next, we defined correlated cost as follows. For each $i < j$, with probability $p = 0.99$, we set $c_{ij} = 0$. Otherwise, we set $c_{ij} \in [-u_i/5, u_i/5]$. The budget is set as $W = 5000$.

We evaluate the above three DC formulations, with two kinds of initial solutions, in terms of the quality of the obtained solutions. For comparison, we also implemented a greedy algorithm that greedily selects item j to maximize $f(x + e_j)$ under the budget constraint. The result is summarized in Table 2. Overall, the greedy algorithm outperforms the DC algorithms. Among the DC algorithms, the third formulation (4.17) with "uncorrelated optimal" as an initial solution gives the best performance. The comparison between these two algorithms are given in Figure 9, which indicates that the DC algorithm sometimes outperforms the greedy algorithm, buy, statistically, it gives slightly worse performance than the greedy algorithm. Thus we may safely say that DC

[2] `http://www.diku.dk/~pisinger/codes.html`

programming approach is not suited for this problem, although we could not identify the reason for that.

Next, we discuss the behavior of the DC algorithms. First, for all formulations with both initial solutions, the DC algorithms terminate in two or three iterations. The first formulation (4.15) with the zero initial solution and "uncorrelated optimal" initial solution give comparable results. For the second formulation (4.16), we have $\nabla h_2(x) = 0$ at $x = 0$ and hence the DC algorithm gives a meaningless solution at the first step when the zero initial solution is used. Accordingly, the initial solution of "uncorrelated optimal" is better than the zero initial solution. The third formulation (4.17) with the zero initial solution gives solutions with significantly small standard deviation. However, the quality of the solution is worse than that for the first and second formulations with "uncorrelated optimal" initial solution.

As observed above, even for the same problem, different DC formulations and different initial solutions exhibit different behaviors with different qualities of solutions. Our preliminary experiments have demonstrated that the choice of DC formulations as well as initial solutions is an important issue in solving discrete optimization problem by DC programming approach. But not much insight has been obtained so far, and further investigation is left for the future.

Acknowledgement

This work is supported by JSPS/MEXT KAKENHI Grant Number 26280004 and by CREST, JST.

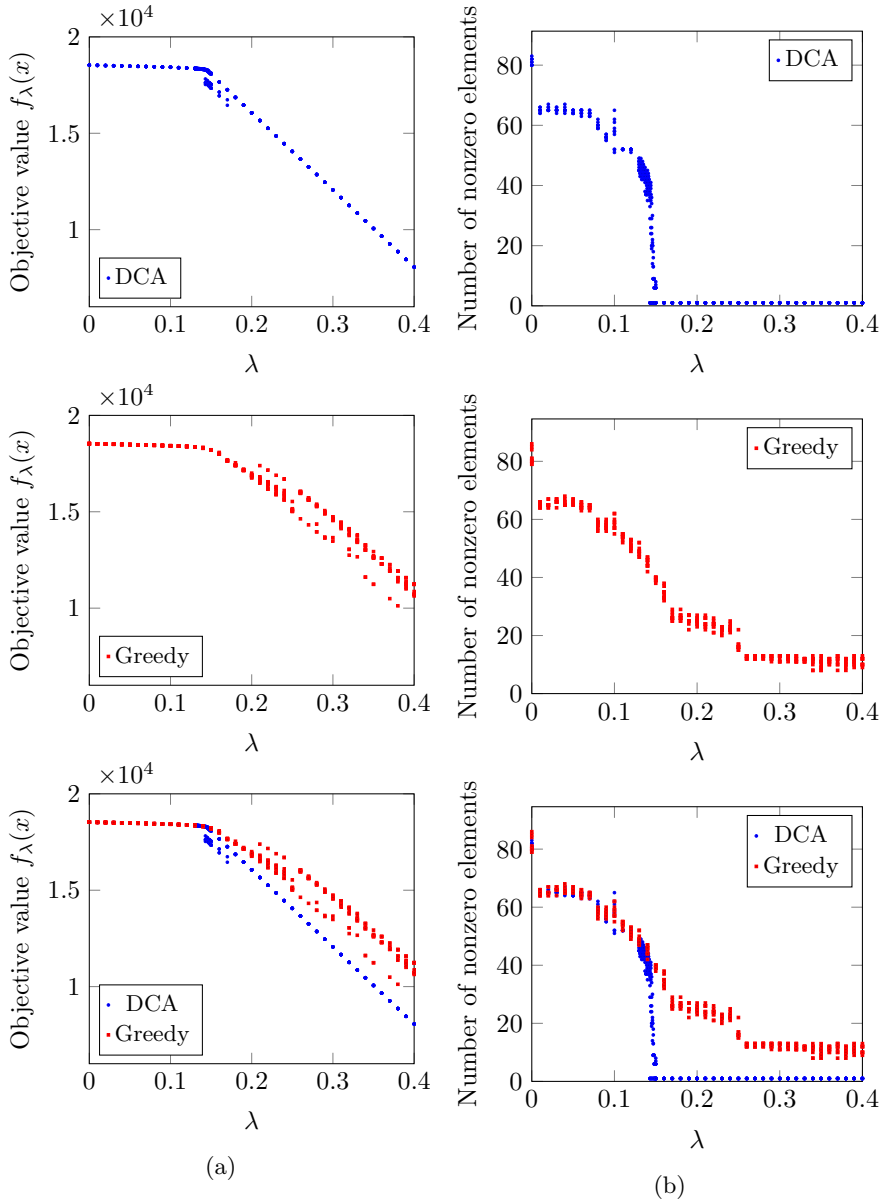


Figure 7: Comparison of the DC algorithm and the greedy algorithm for the unfair flow problem: (a) Objective value $f_\lambda(x)$ versus regularization parameter λ. (b) Number of nonzero elements in x versus λ.

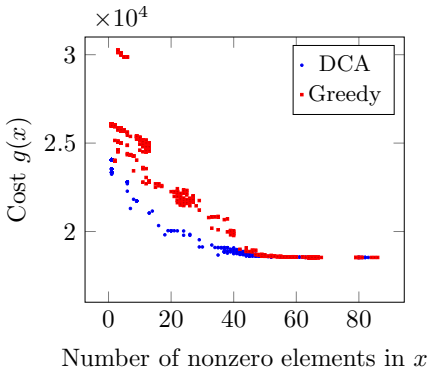


Figure 8: Number of nonzero elements versus cost $g(x)$ of the solution x for the unfair flow problem.

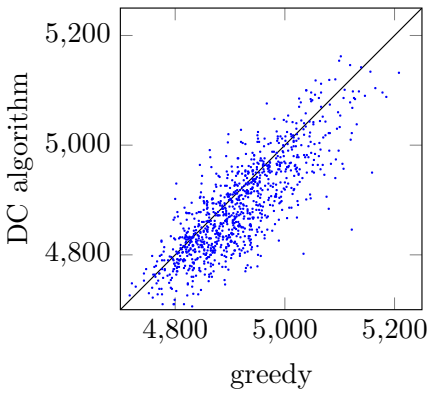


Figure 9: Comparison of objective values of the greedy algorithm and the DC algorithm for $g_3(x) - h_3(x)$ in (4.17) with "uncorrelated optimal" as an initial solution.

References

[1] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski (2012): Optimization with sparsity-inducing penalties. Foundations and Trends in Machine Learning, vol. 4, pp. 1–106.

[2] D. L. Donoho (2006): Compressed sensing. IEEE Transactions on Information Theory, vol. 52, pp. 1289–1306.

[3] P. Favati and F. Tardella (1990): Convexity in nonlinear integer programming. Ricerca Operativa, vol. 53, pp. 3–44.

[4] S. Fujishige (2005): Submodular Functions and Optimization. 2nd ed., Annals of Discrete Mathematics, vol. 58, Elsevier, Amsterdam.

[5] S. Fujishige (2014): Bisubmodular polyhedra, simplicial divisions, and discrete convexity. Discrete Optimization. vol. 12, pp. 115–120.

[6] M. R. Garey and D. S. Johnson (1979): Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, San Francisco.

[7] L. T. Hoai An and P. D. Tao (2001): A continuous approach for globally solving linearly constrained quadratic zero-one programming problems. Optimization, vol. 50, pp. 93-120.

[8] R. Horst, N. V. Thoai (1996): Global Optimization: Deterministic Approaches. Springer.

[9] R. Horst, N. V. Thoai (1999): DC Programming: Overview. Journal of Optimization Theory and Applications, vol. 103, pp. 1–43.

[10] T. Ibaraki and N. Katoh (1988): Resource Allocation Problems: Algorithmic Approaches. MIT Press, Cambridge, Mass, USA.

[11] R. Iyer, S. Jegelka, and J. Bilmes (2013): Fast semidifferential-based submodular function optimization. In Proceedings of the 30th International Conference on Machine Learning, pp. 855–863.

[12] N. Katoh, A. Shioura, and T. Ibaraki (2013): Resource Allocation Problems. In: P. M. Pardalos, D.-Z. Du, and R. L. Graham, eds., Handbook of Combinatorial Optimization (2nd ed.), Springer, vol. 5, pp. 2897–2988.

[13] Y. Kawahara and T. Washio (2011): Prismatic algorithm for discrete D.C. programming problem. In Proceedings of the 25th Annual Conference on Neural Information Processing Systems, pp. 2106–2114.

[14] Y. Kobayashi (2014): The complexity of maximizing the difference of two matroid rank functions. METR2014-42, University of Tokyo.

[15] M. O. Larsson and J. Ugander (2011): A concave regularization technique for sparse mixture models. Advances in Neural Information Processing Systems, vol. 24, pp. 1890–1898.

[16] P. Lemke (1988): The maximum leaf spanning tree problem for cubic graphs is NP-complete, IMA Preprint Series #428, University of Minnesota.

[17] T. Maehara and K. Murota (2015): A framework of discrete DC programming by discrete convex analysis. Mathematical Programming, Series A, vol. 152, pp. 435–466.

[18] S. Moriguchi, A. Shioura, and N. Tsuchimura (2011): M-convex function minimization by continuous relaxation approach—Proximity theorem and algorithm. SIAM Journal on Optimization, vol. 21, pp. 633–668.

[19] S. Moriguchi and N. Tsuchimura (2009): Discrete L-convex function minimization based on continuous relaxation. Pacific Journal of Optimization, vol. 5, pp. 227–236.

[20] K. Murota (2003): Discrete Convex Analysis. Society for Industrial and Applied Mathematics, Philadelphia.

[21] K. Murota (2009): Recent developments in discrete convex analysis. In: W. Cook., L. Lovász, J. Vygen, eds., Research Trends in Combinatorial Optimization, Springer, Berlin, Chapter 11, pp. 219–260.

[22] M. Narasimhan and J. Bilmes (2005): A submodular-supermodular procedure with applications to discriminative structure learning. In Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, pp. 404–412.

[23] Y. S. Niu and P. D. Tao (2008): A DC programming approach for mixed-integer linear programs. Modelling, Computation and Optimization in Information Systems and Management Sciences, Communications in Computer and Information Science, vol. 14, pp. 244–253.

[24] T. Schüle, C. Schnörr, S. Weber, J. Hornegger (2005): Discrete tomography by convex-concave regularization and D.C. programming. Discrete Applied Mathematics, vol. 151, pp. 229–243.

[25] A. S. Tanenbaum (2010): Computer Networks. 5th ed., Prentice Hall, Upper Saddle River, New Jersey.

[26] P. D. Tao and S. El Bernoussi (1987): Duality in D.C. (difference of convex functions) optimization: Subgradient methods. In: K. H. Hoffman, J. Zowe, J. B. Hiriart-Urruty, and C. Lemaréchal, eds., Trends in Mathematical Optimization, International Series of Numerical Mathematics, vol. 84, Birkhäuser, Basel, pp. 277–293.

[27] P. D. Tao and L. T. Hoai An (1997): Convex analysis approach to D.C. programming: Theory, algorithms and applications. Acta Mathematica Vietnamica, vol. 22, pp. 289–355.

[28] H. Tuy (1995): D.C. optimization: Theory, methods and algorithms. In: R. Horst and P. M. Pardalos, eds., Handbook of Global Optimization, Kluwer Academic Publishers, Dordrecht, pp. 149–216.

[29] A. L. Yuille and A. Rangarajan (2003): The concave-convex procedure. Neural Computation, vol. 15, pp. 915–936.