Disciplined Convex-Concave Programming

Xinyue Shen[1] Steven Diamond[2] Yuantao Gu[1] Stephen Boyd[2]

***Abstract*— In this paper we introduce *disciplined convex-concave programming* (DCCP), which combines the ideas of disciplined convex programming (DCP) with convex-concave programming (CCP). Convex-concave programming is an organized heuristic for solving nonconvex problems that involve objective and constraint functions that are a sum of a convex and a concave term. DCP is a structured way to define convex optimization problems, based on a family of basic convex and concave functions and a few rules for combining them. Problems expressed using DCP can be automatically converted to standard form and solved by a generic solver; widely used implementations include `YALMIP`, `CVX`, `CVXPY`, and `Convex.jl`. In this paper we propose a framework that combines the two ideas, and includes two improvements over previously published work on convex-concave programming, specifically the handling of domains of the functions, and the issue of subdifferentiability on the boundary of the domains. We describe a Python implementation called `DCCP`, which extends `CVXPY`, and give examples.**

I. Disciplined convex-concave programming

A. Difference of convex programming

Difference of convex (DC) problems have the form

$$\begin{array}{ll} \text{minimize} & f_0(x) - g_0(x) \\ \text{subject to} & f_i(x) - g_i(x) \le 0, \quad i = 1, \dots, m, \end{array} \qquad (1)$$

where $x \in \mathbf{R}^n$ is the optimization variable, and the functions $f_i : \mathbf{R}^n \to \mathbf{R}$ and $g_i : \mathbf{R}^n \to \mathbf{R}$ for $i = 0, \dots, m$ are convex. The DC problem (1) can also include equality constraints of the form $p_i(x) = q_i(x)$, where p_i and q_i are convex; we simply express these as the pair of inequality constraints

$$p_i(x) - q_i(x) \le 0, \qquad q_i(x) - p_i(x) \le 0,$$

which have the difference of convex form in (1). When the functions g_i are all affine, the problem (1) is a convex optimization problem, and easily solved [1].

The broad class of DC functions includes all C^2 functions [2], so the DC problem (1) is very general. A special case is Boolean linear programs, which can represent many problems, such as the traveling salesman problem, that are widely believed to be hard to solve [3]. DC programs arise in many applications in fields such as image processing [4], machine learning [5], [6], [7], [8], computer vision [9], control [10], [11], discrete tomography [12], subgraph matching [13], and statistics [14]. As a more specific example, optimizing over the efficient set of a multiple objective problem can be formulated as a DC program [15].

[1]Department of Electronic Engineering, Tsinghua University, Beijing 100084, China

[2]Department of Electrical Engineering, Stanford University, CA 94305, USA

DC problems can be solved globally by methods such as branch and bound [16], [17], [18], which can be slow in practice. Good overviews of solving DC programs globally can be found in [19], [20] and the references therein. A locally optimal (approximate) solution can be found instead through the many techniques of general nonlinear optimization [21].

The convex-concave procedure (CCP) [22], [23] is another heuristic algorithm for finding a local optimum of (1), which leverages our ability to efficiently solve convex optimization problems. In its basic form, it replaces concave terms with a convex upper bound, and then solves the resulting convex problem, which is a restriction of the original DC problem. Basic CCP can thus be viewed as an instance of majorization minimization (MM) algorithms [24], in which a minimization problem is approximated by an easier to solve upper bound created around the current point (a step called majorization) and then minimized. Many MM extensions have been developed over the years and more can be found in [25], [26], [27]. CCP can also be viewed as a version of DCA [28] which instead of explicitly linearizing the concave terms, solves a (convex) dual problem; see, *e.g.*, [29] and the references therein.

A recent overview of CCP, with some extensions, can be found in [30], where the issue of infeasibility is handled (heuristically) by an increasing penalty on constraint violations. The method we present in this paper is an extension of the penalty CCP method introduced in [30], given as algorithm I.1 below.

```
Algorithm I.1 Penalty CCP.
given an initial point $x_0$, $\tau_0 > 0$, $\tau_{\max} > 0$, and $\mu > 1$.
$k := 0$.
repeat
    1. Convexify. For $i = 0, \dots, m$, form
           $\hat{g}_i(x; x_k) = g_i(x_k) + \nabla g_i(x_k)^T (x - x_k)$.
    2. Solve. Set the value of $x_{k+1}$ to a solution of
           minimize     $f_0(x) - \hat{g}_0(x; x_k) + \tau_k \sum_{i=1}^m s_i$
           subject to   $f_i(x) - \hat{g}_i(x; x_k) \le s_i, \quad i = 1, \dots, m$
                        $s_i \ge 0, \quad i = 1, \dots, m$.
    3. Update $\tau$. $\tau_{k+1} := \min(\mu \tau_k, \tau_{\max})$.
    4. Update iteration. $k := k + 1$.
until stopping criterion is satisfied.
```

See [30] for discussion of a few variations on the penalty CCP algorithm, such as not using slack variables for constraints that are convex, *i.e.*, the case when g_i is affine. Here it is assumed that g_i are differentiable, and have full domain (*i.e.*, $\mathbf{R}^n$). The first condition is not critical; we can replace $\nabla g_i(x_k)$ with a subgradient of g_i at x_k, if it is not differentiable. The linearization with a subgradient instead of the gradient is still a lower bound on g_i.

In some practical applications, the second assumption, that g_i have full domain, does not hold, in which case the penalty CCP algorithm can fail, by arriving at a point x_k not in the domain of g_i, so the convexification step fails. This is one of the issues we will address in this paper.

B. Disciplined convex programming

Disciplined convex programming (DCP) is a methodology introduced by Grant et al. [31] that imposes a set of conventions that must be followed when constructing (or specifying or defining) convex programs. Conforming problems are called *disciplined convex programs*.

The conventions of DCP restrict the set of functions that can appear in a problem and the way functions can be composed. Every function in a disciplined convex program must come from a set of atomic functions with known curvature and graph implementation, or representation as partial optimization over a cone program [32], [33]. Every composition of functions $f(g_1(x), \dots, g_p(x))$, where $f : \mathbf{R}^p \to \mathbf{R}$ is convex and $g_1, \dots, g_p : \mathbf{R}^n \to \mathbf{R}$, must satisfy the following composition rule, which ensures the composition is convex. Let $\tilde{f} : \mathbf{R}^p \to \mathbf{R} \to \mathbf{R} \cup \{\infty\}$ be the extended-value extension of f [1, Chap. 3]. One of the following conditions must hold for each $i = 1, \dots, p$:

- g_i is convex and $\tilde{f}$ is nondecreasing in argument i.
- g_i is concave and $\tilde{f}$ is nonincreasing in argument i.
- g_i is affine.

The composition rule for concave functions is analogous.

A disciplined convex program has the specific form

$$\begin{array}{ll} \text{minimize/maximize} & o(x) \\ \text{subject to} & l_i(x) \sim r_i(x), \quad i = 1, \dots, m, \end{array} \tag{2}$$

where o (the objective), l_i (lefthand sides), and r_i (righthand sides) are expressions (functions of the variable x) with curvature known from the DCP rules, and $\sim$ denotes one of the relational operators $=$, $\le$, or $\ge$. In DCP this problem must be convex, which imposes conditions on the curvature of the expressions, listed below.

- For a minimization problem, o must be convex; for a maximization problem, o must be concave.
- When the relational operator is $=$, l_i and r_i must both be affine.
- When the relational operator is $\le$, l_i must be convex, and r_i must be concave.
- When the relational operator is $\ge$, l_i must be concave, and r_i must be convex.

Functions that are affine (*i.e.*, are both convex and concave) can match either curvature requirement; for example, we can minimize or maximize an affine expression.

A disciplined convex program can be transformed into an equivalent cone program by replacing each function with its graph implementation. The convex optimization modeling systems `YALMIP` [34], `CVX` [35], `CVXPY` [36], and `Convex.jl` [37] use DCP to verify problem convexity and automatically convert convex programs into cone programs, which can then be solved using generic solvers.

C. Disciplined convex-concave programming

We refer to a problem as a *disciplined convex-concave program* (DCCP) if it has the form (2), with o, l_i, and r_i all having known DCP-verified curvature, *but the DCP curvature conditions for the objective and constraints need not hold*. Such problems include DCP as a special case, but it includes many other nonconvex problems as well. In DCCP we can, for example, maximize a convex function, subject to nonaffine equality constraints, and nonconvex inequality constraints between convex and concave expressions.

The general DC program (1) and the DCCP standard form (2) are equivalent. To express (1) as (2), we express it as

$$\begin{array}{ll} \text{minimize} & f_0(x) - t \\ \text{subject to} & t = g_0(x) \\ & f_i(x) \le g_i(x), \quad i = 1, \dots, m, \end{array}$$

where x is the original optimization variable, and t is a new optimization variable. The objective here is convex. We have one (nonconvex) equality constraint, and the inequality constraints are all nonconvex (except for some special cases when g_i is affine). It is straighforward to express the DCCP form (2) in the form (1), by identifying the functions o_i, l_i, and r_i as $\pm f_i$ or $\pm g_i$ depending on their curvatures.

DCCP is a convenient and simple standard form for DC programming, because the linearized problem in algorithm I.1 is DCP whenever the original problem is DCCP. The linearized problem can thus be automatically converted into a cone program and solved using generic solvers.

II. Domain and subdifferentiability

In this section we delve deeper into an issue that is 'assumed away' in the standard treatments and discussions of DC programming, specifically, how to handle the case when the functions g_i do not have full domain. (The functions f_i can have non-full domains, but this is handled automatically by the conversion into a cone program.)

a) An example: Suppose the domain of g_i is $\mathcal{D}_i$, for $i = 0, \dots, m$. If $\mathcal{D}_i \ne \mathbf{R}^n$, simply defining the linearization $\hat{g}_i(x; z)$ as the first order Taylor expansion of g_i at the point z can lead to failure. The following simple problem gives an example:

$$\begin{array}{ll} \text{minimize} & \sqrt{x} \\ \text{subject to} & x \ge -1, \end{array}$$

where $x \in \mathbf{R}$ is the optimization variable. The objective has domain $\mathbf{R}_+$, and the solution is evidently $x^\star = 0$. The linearized problem in the first iteration of CCP is

$$\begin{array}{ll} \text{minimize} & x_0 + \frac{1}{2\sqrt{x_0}}(x - x_0) \\ \text{subject to} & x \ge -1, \end{array}$$

which has solution $x_1 = -1$. The CCP algorithm will fail in the first step of the next iteration, since the original objective function is not defined at $x_1 = -1$.

If we add the domain constraint directly into the linearized problem, we obtain $x_1 = 0$, but the first step of the next iteration also fails here, in a different way. While x_1 is in the domain of the objective function, the objective is not subdifferentiable (or superdifferentiable) at x_1, so the linearization does not exist. This phenomenon of non-subdifferentiability or non-superdifferentiability can only occur at a point on the boundary of the domain.

A. *Linearization with domain*

Suppose that the intersection of domains of all g_i in problem (1) is $\mathcal{D} = \cap_{i=0}^{m} \mathcal{D}_i$. The correct way to handle the domain is to define the linearization of g_i at point z to be

$$\hat{g}_i(x; z) = g_i(z) + \nabla g_i(z)^T (x - z) - \mathcal{I}_i(x), \qquad (3)$$

where the indicator function is

$$\mathcal{I}_i(x) = \begin{cases} 0 & x \in \mathcal{D}_i \\ \infty & x \notin \mathcal{D}_i, \end{cases}$$

so any feasible point for the linearized problem is in the domain $\mathcal{D}$.

Since g_i is convex, $\mathcal{D}_i$ is a convex set and $\mathcal{I}_i$ is a convex function. Therefore the 'linearization' (3) is a concave function; it follows that if we replace the standard linearization in algorithm I.1 with the domain-restricted linearization (3), the linearized problem is still convex.

B. *Domain in DCCP*

Recall that we defined DCCP to ensure that the linearized problem in algorithm I.1 is a DCP problem. It is not obvious that if we replace the standard linearization with the domain modified version (3), the linearized problem is still DCP. In this section we prove that the linearized problem still satisfies the rules of DCP, or equivalently that each $\mathcal{I}_i(x)$ has a known graph implementation or satisfies the DCP composition rule.

If g_i is an atomic function, then we assume that

$$\mathcal{D}_i = \cap_{j=1}^{p} \{x \mid A_j x + b_j \in \mathcal{K}_j\},$$

for some cone constraints $\mathcal{K}_1, \ldots, \mathcal{K}_p$. The assumption is reasonable since g_i itself can be represented as partial optimization over a cone program. The graph implementation of $\mathcal{I}_i(x)$ is simply

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & A_j x + b_j \in \mathcal{K}_j, \quad j = 1, \ldots, p. \end{array}$$

The other possibility is that g_i is a composition of atomic functions. Since the original problem is DCCP, we may assume that $g_i(x) = f(h_1(x), \ldots, h_p(x))$ for some convex atomic function $f : \mathbf{R}^p \to \mathbf{R}$ and DCP compliant $h_1, \ldots, h_p : \mathbf{R}^n \to \mathbf{R}$ such that $f(h_1(x), \ldots, h_p(x))$ satisfies the DCP composition rule. Then we have

$$\mathcal{I}_i(x) = \mathcal{I}_f(h_1(x), \ldots, h_p(x)) + \sum_{j=1}^{p} \mathcal{I}_{h_j}(x),$$

where $\mathcal{I}_f$ is the indicator function for the domain of f and $\mathcal{I}_{h_1}, \ldots, \mathcal{I}_{h_p}$ are defined similarly.

Since f is convex, $\mathcal{I}_f$ is convex. Moreover, $\mathcal{I}_f(h_1(x), \ldots, h_p(x))$ satisfies the DCP composition rule. To see why, observe that for $j = 1, \ldots, p$, if h_j is convex then by assumption the extended-value extension $\tilde{f}$ is nondecreasing in argument j. It follows that $\mathcal{I}_f$ is nondecreasing in argument j. Similarly, if h_j is concave then $\mathcal{I}_f$ is nonincreasing in argument j.

An inductive argument shows that $\mathcal{I}_{h_1}, \ldots, \mathcal{I}_{h_p}$ are convex and satisfy the DCP rules. We conclude that $\mathcal{I}_i$ satisfies the DCP composition rule.

C. *Chain rule for a subgradient*

When linearizing a nondifferentiable convex function, we need to calculate a subgradient at a certain point. (We say 'a' subgradient since it is not in general unique.) The gradient of a differentiable expression is readily computed recursively by the chain rule. The chain rule for compositions of functions does not generally apply for the calculation of the subdifferential, but it does apply when we seek a subgradient when the function conforms to the DCP rules.

Suppose that $h(x) = f(g_1(x), \ldots, g_p(x))$, where f is convex, and the DCP rules hold, *i.e.*, for each i, g_i is affine, or g_i is convex and f is nondecreasing in argument i, or g_i is concave and f is nonincreasing in argument i. We can compute a subgradient of h at x as follows. Choose any $q \in \partial f(g_1(x), \ldots, g_p(x))$. By the monotonicity assumption $q_i \geq 0$ when f_i is nondecreasing in argument i and $q_i \leq 0$ when f_i is nonincreasing in argument i. Choose any $s_i \in \partial g_i(x)$, $i = 1, \ldots, p$. Then we have $r = q_1 s_1 + \cdots + q_p s_p \in \partial h(x)$.

This can be directly proved as follows. For any $x, y \in \mathbf{R}^n$,

$$\begin{aligned} h(x) &= f(g_1(x), \ldots, g_p(x)) \\ &\geq f(g_1(y), \ldots, g_p(y)) + \textstyle\sum_{i=1}^{p} q_i (g_i(x) - g_i(y)) \\ &\geq f(g_1(y), \ldots, g_p(y)) + \textstyle\sum_{i=1}^{p} q_i s_i^T (x - y) \\ &= h(y) + r^T (x - y). \end{aligned}$$

D. *Sub-differentiability on boundary*

When $\mathcal{D} \neq \mathbf{R}^n$, a solution to the linearized problem $\hat{x}_k$ at iteration k can be on the boundary of the closure of $\mathcal{D}$. It is possible (as our simple example above shows) that the convex function g_i is not subdifferentiable at $\hat{x}_k$, which means the linearization does not exist and the algorithm fails. This pathology can and does occur in practical problems.

In order to handle this, at each iteration, when the subgradient $\nabla g_i(\hat{x}_k)$ for any function g_i does not exist, we simply take a damped step,

$$x_k = \alpha \hat{x}_k + (1 - \alpha) x_{k-1},$$

where $0 < \alpha < 1$. If x_0 is in the interior of the domain, then x_k will be in the interior for all $k \geq 0$, and $\nabla g_i(x_k)$ will be guaranteed to exist. The algorithm can (and does, for our simple example) converge to a point on the boundary of the the domain, but each iterate is in the interior of the domain, which is enough to guarantee that the linearization exists.

III. Implementation

The proposed methods described above have been implemented as the Python package `DCCP`, which extends the package `CVXPY`. New methods were added to `CVXPY` to return the domain of a DCP expression (as a list of constraints), and gradients (or sub/supergradients) were added to the atoms. The linearization, damping, and initialization are handled by the package `DCCP`. Users can form any problem of the form (2) conforming to the DCCP rules, with each expression composed of functions in the `CVXPY` library.

When the `solve(method = 'dccp')` method is called on a problem object, `DCCP` first verifies that the problem satisfies the DCCP rules. Then it splits each non-affine equality constraint $l_i = r_i$ into $l_i \leq r_i$ and $l_i \geq r_i$. The curvature of the objective and the left and righthand sides of each constraint is checked, and if needed, linearized. In the linearization the function value and gradient are `CVXPY` parameters, which are constants whose value can change without reconstructing the problem. For each constraint in which the left or righthand side is linearized, a slack variable is introduced, and added to the objective. For any expression that is linearized, the domain of the original expression is added as a constraint.

If a valid initial value for the variable x is given by the user, it is used. Otherwise several random points are generated from the standard normal distribution (or the uniform distribution on $[0, 1]$ if the variable is declared as nonnegative) and projected onto the domain $\mathcal{D}$ (which, according to the discussion in §II-B, is a DCP problem). The projected points are then averaged to give an initial value for x. This random initialization is a generic heuristic for finding a starting point in the interior of $\mathcal{D}$, and seems to work well in practice.

Algorithm I.1 is then applied to the convexified problem. In each iteration the parameters in the linearizations (which are function and gradient values) are updated based on the current value of the variables. If a gradient (super/subgradient) with respect to any variable does not exist, damping is applied to all variables. The convexified problem at each iteration is solved using `CVXPY`.

Some useful functions and attributes in the `DCCP` package are listed below.

- Function `is_dccp(problem)` returns a boolean indicating if an optimization problem is DCCP.
- Attribute `expression.gradient` returns a dictionary of the gradients (or sub/supergradient) of a DCP expression with respect to its variables at the points specified by `variable.value`. (This attribute is also in the core `CVXPY` package.)
- Function `linearize(expression)` returns the linearization (3) of a DCP expression.
- Attribute `expression.domain` returns a list of constraints describing the domain of a DCP expression. (This attribute is also in the core `CVXPY` package.)
- Function `convexify(constraint)` returns the transformed constraint (without slack variables) satisfying DCP of a DCCP constraint.
- Method `problem.solve(method = 'dccp')` carries out the proposed algorithm, and returns the value of the transformed cost function, the value of the weight μ_k, and the maximum value of slack variables at each iteration k. An optional parameter `ccp_times` is used to set the number of times to run the CCP based algorithm, using the randomized initialization.

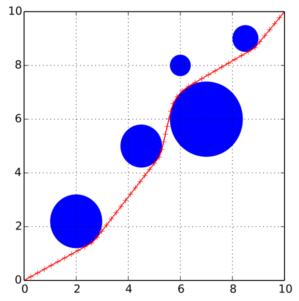

Fig. 1: Path planning.

IV. Examples

In this section we describe some simple examples, show how they can be expressed using `DCCP`, and give the results. In each case we run the default solve method, with no tuning or adjustment of algorithm parameters.

A. Path planning

The goal is to find the shortest path connecting points a and b in $\mathbf{R}^d$ that avoids m circles, centered at p_j with radius r_j, $j = 1, \ldots, m$ [38]. After discretizing the arc length parametrized path into points $x_0, \ldots, x_n$, the problem is posed as

$$\begin{array}{ll} \text{minimize} & L \\ \text{subject to} & x_0 = a, \quad x_n = b \\ & \|x_i - x_{i-1}\|_2 \leq L/n, \quad i = 1, \ldots, n \\ & \|x_i - p_j\|_2 \geq r_j, \quad i = 1, \ldots, n, \quad j = 1, \ldots, m, \end{array}$$

where L and x_i are variables, and a, b, p_j, and r_j are given. The code is given below.

```
x = Variable(d,n+1)
L = Variable()
constr = [x[:,0] == a, x[:,n] == b]
for i in range(1,n+1):
    constr += [norm(x[:,i]-x[:,i-1])
    <= L/n]
    for j in range(m):
        constr += [norm(x[:,i]
        - center[:,j]) >= r[j]]
prob = Problem(Minimize(L), constr)
prob.solve(method = 'dccp')
```

An example with $d = 2$ and $n = 50$ is shown in figure 1.

B. Control with collision avoidance

We have n linear dynamic systems, given by

$$x_{t+1}^i = A^i x_t^i + B^i u_t^i, \quad y_t^i = C^i x_t^i, \quad i = 1, \ldots, n,$$

where $t = 0, 1, \ldots, T$ denotes (discrete) time, x_t^i are the states, and y_t^i are the outputs. At each time t for $t = 0, \ldots, T$ the n outputs y_t^i are required to keep a distance of at least

$d_{\min}$ from each other [39]. The initial states x_0^i and ending states x_n^i are given by x_{init}^i and x_{end}^i, and the inputs are limited by $\|u_t^i\|_\infty \leq f_{\max}$. We will minimize a sum of the ℓ_1 norms of the inputs, an approximation of fuel use. (Of course we can have any convex state and input constraints, and any convex objective.) This gives the problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^n \sum_{t=0}^{T-1} \|u_t^i\|_1 \\ \text{subject to} & x_0^i = x_{\text{init}}^i, \quad x_T^i = x_{\text{end}}^i, \quad i = 1, \ldots, n \\ & x_{t+1}^i = A^i x_t^i + B^i u_t^i, \\ & t = 0, \ldots, T-1, \quad i = 1, \ldots, n \\ & \|y_t^i - y_t^j\|_2 \geq d_{\min}, \\ & t = 0, \ldots, T, \quad 1 \leq i < j \leq n \\ & y_t^i = C^i x_t^i, \quad \|u_t^i\|_\infty \leq f_{\max}, \\ & t = 0, \ldots, T-1, \quad i = 1, \ldots, n, \end{array}$$

where x_t^i, y_t^i, and u_t^i are variables.

The code can be written as follows.

```
constr = []
cost = 0
for i in range(n):
    for t in range(T):
        u[i] += [Variable(d)]
        constr += [norm(u[i][-1],
        'inf') <= f_max]
        cost += norm(u[i][-1],1)
        y[i] += [Variable(d)]
        x[i] += [Variable(2*d)]
        constr += [y[i][-1] ==
        C[i]*x[i][-1]]
for i in range(n):
    constr += [x[i][0] == x_ini[i]]
    constr += [x[i][-1] == x_end[i]]
    for t in range(T-1):
        constr += [x[i][t+1] ==
        A[i]*x[i][t] + B[i]*u[i][t]]
for t in range(T):
    for i in range(n-1):
        for j in range(i+1,n):
            constr += [norm(y[i][t] -
            y[j][t],2) >= d_min]
prob = Problem(Minimize(cost), constr)
prob.solve(method = 'dccp')
```

We consider an instance with $n = 2$, with outputs (positions) $y_t^i \in \mathbf{R}^2$, $d_{\min} = 0.6$, $f_{\max} = 0.5$, $T = 100$. The linear dynamic system matrices are

$$A^i = \begin{bmatrix} 1 & 0 & 0.1 & 0 \\ 0 & 1 & 0 & 0.1 \\ 0 & 0 & 0.95 & 0 \\ 0 & 0 & 0 & 0.95 \end{bmatrix},$$

$$B^i = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad C^i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

The results are in figure 2, where the black arrows in the first two figures show initial and final states (position and velocity), and the black dashed line in the third figure shows $d_{\min}$. The left plot shows the output trajectory without collision avoidance, the middle shows it with collision avoidance, and the right plot shows the distance between outputs versus time.

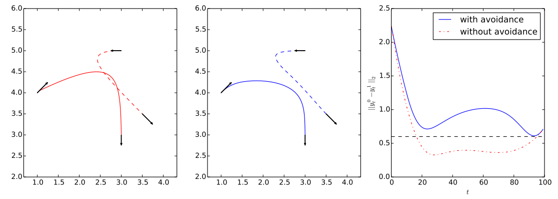

Fig. 2: Optimal control with collision avoidance.

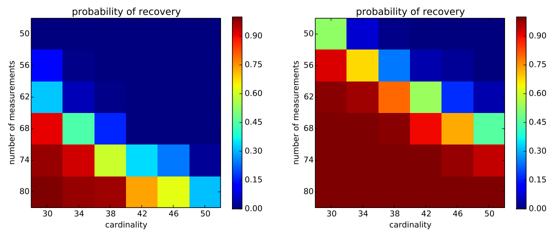

Fig. 3: Sparse recovery. *Left*: l_1 norm. *Right*: Sqrt of $\ell_{1/2}$ 'norm'.

C. Sparse recovery using $\ell_{1/2}$ 'norm'

The aim is to recover a sparse nonnegative signal $x_0 \in \mathbf{R}^n$ from a measurement vector $y = Ax_0$, where $A \in \mathbf{R}^{m \times n}$ (with $m < n$) is a known sensing matrix [40]. A common heuristic based on convex optimization is to minimize the ℓ_1 norm of x (which reduces here to the sum of entries of x) subject to $y = Ax_0$ (and here, $x \geq 0$). It has been proposed to minimize the sum of the *squareroots* of the entries of x, which since $x \geq 0$ is the same as minimizing the squareroot of the $\ell_{1/2}$ 'norm' (which is not convex, and therefore not a norm), to obtain better recovery. The optimization problem is

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^n \sqrt{x_i} \\ \text{subject to} & y = Ax, \end{array}$$

where x is the variable. (The constraint $x \geq 0$ is implicit, since this is the objective domain.) This is a nonconvex problem, directly in DCCP form.

The corresponding code is as follows.

```
x = Variable(n,1)
x.value = np.ones((n,1))
obj = Minimize(sum_entries(sqrt(x)))
prob = Problem(obj, [A*x == y])
prob.solve(method = 'dccp')
```

In a numerical simulation, we take $n = 100$, $A_{ij} \sim \mathcal{N}(0,1)$, the positions of the nonzero entries in x_0 are from uniform distribution, and the nonzero values are the absolute values of $\mathcal{N}(0,100)$ random variables. To count the probability of recovery, 100 independent instances are tested, and a recovery is successful if the relative error $\|\hat{x} - x_0\|_2 / \|x_0\|_2$ is less than 0.01. In each instance, the cardinality takes 6 values from 30 to 50, according to which x_0 is generated, and A is generated for each m taking one of the 6 values from 50 to 80. The results in figure 3 verify that nonconvex recovery is more effective than convex recovery.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-114747, by the DARPA X-DATA and SIMPLEX programs, and by the CSC State Scholarship Fund.

References

[1] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[2] P. Hartman, "On functions representable as a difference of convex functions," *Pacific Journal of Math*, vol. 9, no. 3, pp. 707–713, 1959.

[3] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computation*, R. E. Miller and J. W. Thatcher, Eds. Plenum, 1972, pp. 85–104.

[4] Y. Lou, S. Osher, and J. Xin, *Computational aspects of constrained L1-L2 minimization for compressive sensing*, ser. Advances in Intelligent Systems and Computing, 2015, vol. 359, pp. 169–180.

[5] L. T. H. An, H. M. Le, V. V. Nguyen, and P. D. Tao, "A DC programming approach for feature selection in support vector machines learning," *Advances in Data Analysis and Classification*, vol. 2, no. 3, pp. 259–278, 2008.

[6] C. J. Yu and T. Joachims, "Learning structural svms with latent variables," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. ACM, 2009, pp. 1169–1176.

[7] L. T. Hoai An, H. M. Le, and P. D. Tao, "Feature selection in machine learning: An exact penalty approach using a difference of convex function algorithm," *Machine Learning*, vol. 101, no. 1-3, pp. 163–186, 2015.

[8] R. Collobert, F. Sinz, J. Weston, and L. Bottou, "Trading convexity for scalability," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. ACM, 2006, pp. 201–208.

[9] Y. Lou, T. Zeng, S. Osher, and J. Xin, "A weighted difference of anisotropic and isotropic total variation model for image processing," *SIAM Journal on Imaging Sciences*, vol. 8, no. 3, pp. 1798–1823, 2015.

[10] T. Lipp and S. Boyd, "Antagonistic control," http://web.stanford.edu/~boyd/papers/antag_control.html, March 2015.

[11] S. Boyd, M. Hast, and K. J. Åström, "Mimo pid tuning via iterated lmi restriction," *International Journal of Robust and Nonlinear Control*, 2015.

[12] T. Schüle, C. Schnörr, S. Weber, and J. Hornegger, "Discrete tomography by convex-concave regularization and D.C. programming," *Discrete Applied Mathematics*, vol. 151, no. 1-3, pp. 229–243, 2005.

[13] Z. Liu and H. Qiao, "A convex-concave relaxation procedure based subgraph matching algorithm," in *ACML*, 2012, pp. 237–252.

[14] J. Thai, T. Hunter, A. K. Akametalu, C. J. Tomlin, and A. M. Bayen, "Inverse covariance estimation from data with missing values using the concave-convex procedure," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 2014, pp. 5736–5742.

[15] L. D. Muu, "A convex-concave programming method for optimizing over the efficient set," *Acta Mathematica Vietnamica*, vol. 25, no. 1, pp. 67–85, 2000.

[16] N. Agin, "Optimum seeking with branch and bound," *Management Science*, vol. 13, pp. 176–185, 1966.

[17] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: a survey," *Operations Research*, vol. 14, pp. 699–719, 1966.

[18] L. D. Muu, "An algorithm for solving convex programs with an additional convex-concave constraint," *Mathematical Programming*, vol. 61, no. 1, pp. 75–87, 1993.

[19] R. Horst, P. M. Pardalos, and N. V. Thoai, *Introduction to Global Optimization*. Dordrecht, Netherlands: Kluwer Academic Publishers, 1995.

[20] R. Horst and N. V. Thoai, "DC programming: overview," *Journal of Optimization Theory and Applications*, vol. 103, no. 1, pp. 1–43, 1999.

[21] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006.

[22] A. L. Yuille and A. Rangarajan, "The concave-convex procedure," *Neural Computation*, vol. 15, no. 4, pp. 915–936, 2003.

[23] G. R. Lanckriet and B. K. Sriperumbudur, "On the convergence of the concave-convex procedure," in *Advances in neural information processing systems*, 2009, pp. 1759–1767.

[24] K. Lange, D. R. Hunter, and I. Yang, "Optimization transfer using surrogate objective functions," *Journal of Computational and Graphical Statistics*, vol. 9, no. 1, pp. 1–20, 2000.

[25] R. J. A. Little and D. B. Rubin, *Statistical Analysis with Missing Data*. New York, New York: John Wiley & Sons, 1987.

[26] K. Lange, *Optimization*, ser. Springer Texts in Statistics. New York, New York: Springer, 2004.

[27] G. McLachlan and T. Krishnan, *The EM algorithm and extensions*. John Wiley & Sons, 2007.

[28] P. D. Tao and E. B. Souad, "Algorithms for solving a class of nonconvex optimization problems. Methods of subgradients," in *FERMAT Days 85: Mathematics for Optimization*, J.-B. Hiriart-Urruty, Ed. Elsevier Scince Publishers B. V., 1986, pp. 249–271.

[29] L. T. H. An, "DC programming and DCA: local and global approaches - theory, algorithms and applications," http://lita.sciences.univ-metz.fr/~lethi/DCA.html, 2015.

[30] T. Lipp and S. Boyd, "Variations and extension of the convex–concave procedure," *Optimization and Engineering*, pp. 1–25, 2015.

[31] M. Grant, S. Boyd, and Y. Ye, "Disciplined convex programming," in *Global Optimization: From Theory to Implementation*, ser. Nonconvex Optimization and its Applications, L. Liberti and N. Maculan, Eds. Springer, 2006, pp. 155–210.

[32] M. Grant and S. Boyd, "Graph implementation for nonsmooth convex programs," in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences, V. Blondel, S. Boyd, and H. Kimura, Eds. Springer-Verlag, 2008.

[33] Y. Nesterov and A. Nemirovsky, "Conic formulation of a convex programming problem and duality," *Optimization Methods and Software*, vol. 1, no. 2, pp. 95–115, 1992.

[34] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proceedings of the IEEE International Symposium on Computed Aided Control Systems Design*, Sep. 2004, pp. 294–289.

[35] CVX Research, Inc., "CVX: Matlab software for disciplined convex programming, version 2.0," http://cvxr.com.cvx, Aug. 2012.

[36] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *To appear, Journal of Machine Learning Research*, 2016.

[37] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd, "Convex optimization in Julia," in *Proceedings of the Workshop for High Performance Technical Computing in Dynamic Languages*, 2014, pp. 18–28.

[38] J. C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[39] A. Miele, T. Wang, C. S. Chao, and J. B. Dabney, "Optimal control of a ship for collision avoidance maneuvers," *Journal of Optimization Theory and Applications*, vol. 103, no. 3, pp. 495–519, 1999.

[40] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, 2008.