Dual Lagrangian Learning for Conic Optimization

Mathieu Tanneau, Pascal Van Hentenryck
H. Milton Steward School of Industrial and Systems Engineering
NSF AI Institute for Advances in Optimization
Georgia Institute of Technology
{mathieu.tanneau,pascal.vanhentenryck}@isye.gatech.edu

Abstract

This paper presents Dual Lagrangian Learning (DLL), a principled learning methodology for dual conic optimization proxies. DLL leverages conic duality and the representation power of ML models to provide high-duality, dual-feasible solutions, and therefore valid Lagrangian dual bounds, for linear and nonlinear conic optimization problems. The paper introduces a systematic dual completion procedure, differentiable conic projection layers, and a self-supervised learning framework based on Lagrangian duality. It also provides closed-form dual completion formulae for broad classes of conic problems, which eliminate the need for costly implicit layers. The effectiveness of DLL is demonstrated on linear and nonlinear conic optimization problems. The proposed methodology significantly outperforms a state-of-the-art learning-based method, and achieves 1000x speedups over commercial interior-point solvers with optimality gaps under 0.5% on average.

1 Introduction

From power systems and manufacturing to supply chain management, logistics and healthcare, optimization technology underlies most aspects of the economy and society. Over recent years, the substantial achievements of Machine Learning (ML) have spurred significant interest in combining the two methodologies. This integration has led to the development of new optimization algorithms (and the revival of old ones) taylored to ML problems, as well as new ML techniques for improving the resolution of hard optimization problems [1]. This paper focuses on the latter (ML for optimization), specifically, the development of so-called *optimization proxies*, i.e., ML models that provide approximate solutions to parametric optimization problems, see e.g., [2].

In that context, considerable progress has been made in learning *primal* solutions for a broad range of problems, from linear to discrete and nonlinear, non-convex optimization problems. State-of-the-art methods can now predict high-quality, feasible or close-to-feasible solutions for various applications [2]. This paper complements these methods by learning *dual* solutions which, in turn, certify the (sub)optimality of learned primal solutions. Despite the fundamental role of duality in optimization, there is no dedicated framework for dual optimization proxies, which have seldom received any attention in the literature. The paper addresses this gap by proposing, for the first time, a principled learning methodology that combines conic duality theory with Machine Learning. As a result, it becomes possible, for a large class of optimization problems, to design a primal proxy to deliver a high-quality primal solution and an associated dual proxy to obtain a quality certificate.

1.1 Contributions and outline

The core contribution of the paper is the Dual Lagrangian Learning (DLL) methodology for learning dual-feasible solutions for parametric conic optimization problems. DLL leverages conic duality to design a self-supervised Lagrangian loss for training dual conic optimization proxies. In addition, the

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

 https://doi.org/10.52202/079017-1764

paper proposes a general dual conic completion using differential conic projections and implicit layers to guarantee dual feasibility, which yields stronger guarantees than existing methods for constrained optimization learning. Furthermore, it presents closed-form analytical solutions for conic projections, and for dual conic completion across broad classes of problems. This eliminates the need for implicit layers in practice. Finally, numerical results on linear and nonlinear conic problems demonstrate the effectiveness of DLL, which a outperforms state-of-the-art learning baseline, and yields significant speedups over interior-point solvers.

The rest of the paper is organized as follows. Section 2 presents the relevant literature. Section 3 introduces notations and background material. Section 4 presents the DLL methodology, which comprises the Lagrangian loss, dual completion strategy, and conic projections. Section 5 reports numerical results. Section 6 discusses possible the limitations of DLL and possible extensions, and Section 7 concludes the paper.

2 Related works

Constrained Optimization Learning The vast majority of existing works on optimization proxies focuses on learning *primal* solutions and, especially, on ensuring their feasibility. This includes, for instance, physics-informed training loss [3, 4, 5], mimicking the steps of an optimization algorithm [6, 7, 8], using masking operations [9, 10], or designing custom projections and feasibility layers [11, 12]. The reader is referred to [2] for an extensive survey of constrained optimization learning.

Only a handful of methods offer feasibility guarantees, and only for convex constraints; this latter point is to be expected since satisfying non-convex constraints is NP-hard in general. Implicit layers [13] have a high computational cost, and are therefore impractical unless closed-form solutions are available. DC3 [5] uses equality completion and inequality correction, and is only guaranteed to converge for convex constraints and given enough correction steps. LOOP-LC [14] uses a gauge mapping to ensure feasibility for bounded polyhedral domains. RAYEN [12] and the similar work in [15] use a line search-based projection mechanism to handle convex constraints. All above methods employ equality completion, and the latter three [14, 12, 15] assume knowledge of a strictly feasible point, which is not always available.

Dual Optimization Learning To the authors' knowledge, dual predictions have received very little attention, with most works using them to warm-start an optimization algorithm. In [16], a primal-dual prediction is used to warm-start an ADMM algorithm, while These works consider specific applications, and do not provide dual feasibility guarantees. More recently, [6, 8] attempt to mimic the (dual) steps of an augmented Lagrangian method, however with the goal of obtaining high-quality primal solutions.

In the mixed-integer programming (MIP) setting, [17] and [18] use a dual prediction as warm-start in a column-generation algorithm, for cutting-stock and unit-commitment problems, respectively. In a similar fashion, [19] consider a (combinatorial) Lagrangian relaxation of Traveling Salesperson Problem (TSP) Most recently, [20] consider learning Lagrangian multipliers for mixed-integer linear programs. Therein, a machine learning model predicts Lagrange multipliers, and a Lagrangian sub-problem is solved to obtained a Lagrangian dual bound. This approach only supports linear constraints for the Lagrangian, and it requires an external combinatorial solver to solve the subproblem, which may be NP-hard in general.

The first work to explicitly consider dual proxies in the context of conic optimization, and to offer dual feasibility guarantees, is [21], which learns a dual proxy for a second-order cone relaxation of the AC Optimal Power Flow. Klamkin et al. [22] later introduce a dual interior-point learning algorithm to speed-up the training of dual proxies for bounded linear programming problems. In contrast, this paper proposes a general methodology for conic optimization problems, thus generalizing the approach in [21], and provides more extensive theoretical results. The dual completion procedure used in [22, Lemma 1] is a special case of the one proposed in this paper.

3 Background

This section introduces relevant notations and standard results on conic optimization and duality, which lay the basis for the proposed learning methodology. The reader is referred to [23] for a thorough overview of conic optimization.

3.1 Notations

Unless specified otherwise, the Euclidean norm of a vector $x \in \mathbb{R}^n$ is denoted by $\|x\| = \sqrt{x^\top x}$. The positive and negative part of $x \in \mathbb{R}$ are denoted by $x^+ = \max(0, x)$ and $x^- = \max(0, -x)$. The identity matrix of order n is denoted by I_n, and $\mathbf{e}$ denotes the vector of all ones. The smallest eigenvalue of a real symmetric matrix X is $\lambda_{\min}(X)$.

Given a set $\mathcal{X} \subseteq \mathbb{R}^n$, the *interior* and *closure* of $\mathcal{X}$ are denoted by $\operatorname{int} \mathcal{X}$ and by $\operatorname{cl} \mathcal{X}$, respectively. The Euclidean projection onto convex set $\mathcal{C}$ is denoted by $\Pi_{\mathcal{C}}$, where

$$\Pi_{\mathcal{C}}(\bar{x}) = \arg\min_{x \in \mathcal{C}} \|x - \bar{x}\|^2 . \tag{1}$$

The set $\mathcal{K} \subseteq \mathbb{R}^n$ is a *cone* if $x \in \mathcal{K}, \lambda \geq 0 \Rightarrow \lambda x \in \mathcal{K}$. The *dual cone* of $\mathcal{K}$ is

$$\mathcal{K}^* = \{y \in \mathbb{R}^n : y^\top x \geq 0, \forall x \in \mathcal{K}\}, \tag{2}$$

whose negative $\mathcal{K}^\circ = -\mathcal{K}^*$ is the *polar* cone of $\mathcal{K}$. A cone $\mathcal{K}$ is self-dual if $\mathcal{K} = \mathcal{K}^*$, and it is *pointed* if $\mathcal{K} \cap (-\mathcal{K}) = \{0\}$. All cones considered in the paper are *proper* cones, i.e., closed, convex, pointed cones with non-empty interior. A proper cone $\mathcal{K}$ defines *conic inequalities* $\succeq_{\mathcal{K}}$ and $\succ_{\mathcal{K}}$ as

$$\forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n,\ x \succeq_{\mathcal{K}} y \ \Leftrightarrow\ x - y \in \mathcal{K}, \tag{3a}$$

$$\forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n,\ x \succ_{\mathcal{K}} y \ \Leftrightarrow\ x - y \in \operatorname{int} \mathcal{K}. \tag{3b}$$

3.2 Conic optimization

Consider a (convex) conic optimization problem of the form

$$\min_x \quad \left\{ c^\top x \;\middle|\; Ax \succeq_{\mathcal{K}} b \right\}, \tag{4}$$

where $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$, and $\mathcal{K}$ is a proper cone. All convex optimization problems can be formulated in conic form. A desirable property of conic formulations is that is enables the use of principled conic duality theory [23]. Namely, the conic dual problem reads

$$\max_y \quad \left\{ b^\top y \;\middle|\; A^\top y = c, y \in \mathcal{K}^* \right\}. \tag{5}$$

The dual problem (5) is a conic problem, and the dual of (5) is (4). Weak conic duality always holds, i.e., any dual-feasible solution provides a valid lower bound on the optimal value of (4), and vice-versa. When strong conic duality holds, e.g., under Slater's condition, both primal/dual problems have the same optimal value and a primal-dual optimal solution exists [23].

Conic optimization encompasses broad classes of problems such linear and semi-definite programming. Most real-life convex optimization problems can be represented in conic form using only a small number of cones [24], which are supported by off-the-shelf solvers such as Mosek, ECOS, or SCS. These so-called "standard" cones comprise the non-negative orthant $\mathbb{R}_+$, the second-order cone $\mathcal{Q}$ and rotated second-order cone $\mathcal{Q}_r$, the positive semi-definite cone $\mathcal{S}_+$, the power cone $\mathcal{P}$ and the exponential cone $\mathcal{E}$; see Appendix B for algebraic definitions.

4 Dual Lagrangian Learning (DLL)

This section presents the *Dual Lagrangian Learning* (DLL) methodology, illustrated in Figure 1, for learning dual solutions for conic optimization problems. DLL combines the representation power of artificial neural networks (or, more generally, any differentiable program), with conic duality theory, thus providing valid Lagrangian dual bounds for general conic optimization problems. *To the best of the authors' knowledge, this paper is the first to propose a principled self-supervised framework with dual guarantees for general conic optimization problems.*

 https://doi.org/10.52202/079017-1764

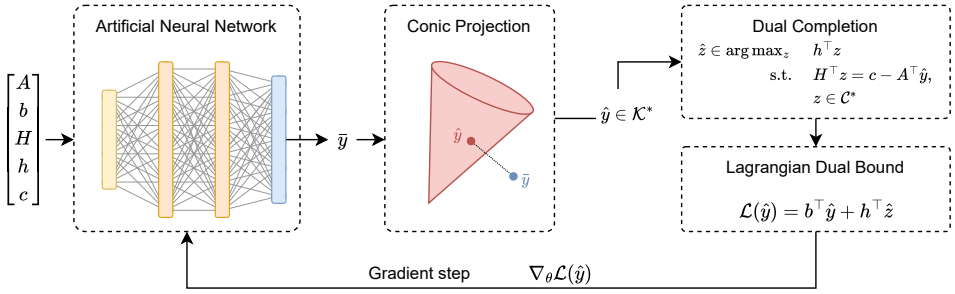

Figure 1: Illustration of the proposed DLL scheme. Given input data (A, b, H, h, c), a neural network first predicts $\bar{y} \in \mathbb{R}^n$. Next, a conic projection layer computes a conic-feasible $\hat{y} \in \mathcal{K}^*$, which is then completed into a full dual-feasible solution $(\hat{y}, \hat{z})$. The model is trained in a self-supervised fashion, by updating the weights θ to maximize the Lagrangian dual bound $\mathcal{L}(\hat{y})$.

DLL exploits three fundamental building blocks: (1) a dual conic completion procedure that provides dual-feasible solutions and, hence, valid Lagrangian dual bounds; (2) fast and differentiable conic projection layers; and (3) a self-supervised learning algorithm that emulates the steps of a dual Lagrangian ascent algorithm.

4.1 Dual Conic Completion

Consider a conic optimization problem in primal-dual form

$$\min_x \quad c^\top x \tag{6a}$$
$$\text{s.t.} \quad Ax \succeq_{\mathcal{K}} b, \tag{6b}$$
$$Hx \succeq_{\mathcal{C}} h, \tag{6c}$$

$$\max_{y,z} \quad b^\top y + h^\top z \tag{7a}$$
$$s.t. \quad A^\top y + H^\top z = c, \tag{7b}$$
$$y \in \mathcal{K}^*, z \in \mathcal{C}^*. \tag{7c}$$

where $y \in \mathcal{K}^*$ and $z \in \mathcal{C}^*$ are the dual variables associated to constraints (6b) and (6c), respectively. The proposed dual conic completion, outlined in Theorems 1 and 2 below, takes as input $\hat{y} \in \mathcal{K}^*$, and recovers $\hat{z} \in \mathcal{C}^*$ such that $(\hat{y}, \hat{z})$ is feasible for (7). The initial assumption that $\hat{y} \in \mathcal{K}^*$ can be enforced through a projection step, which will be described in Section 4.2.

Theorem 1 (Dual conic completion)**.** *Assume that* $\forall \hat{y} \in \mathcal{K}^*, \exists x : Hx \succ_{\mathcal{C}} h$ *and the problem*

$$\min_x \quad \left\{ c^\top x + (b - Ax)^\top \hat{y} \;\middle|\; Hx \succeq_{\mathcal{C}} h \right\} \tag{8}$$

is bounded. Then, $\forall \hat{y} \in \mathcal{K}^*, \exists \hat{z} \in \mathcal{C}^* : A^\top \hat{y} + H^\top \hat{z} = c$*, i.e.,* $(\hat{y}, \hat{z})$ *is feasible for* (7)*.*

Theorem 2 (Optimal dual completion)**.** *Let* $\hat{y} \in \mathcal{K}^*$*, and let* $\hat{z}$ *be dual-optimal for* (8)*. Then,* $\mathcal{L}(\hat{y}, \hat{z}) = b^\top \hat{y} + h^\top \hat{z}$ *is a valid dual bound on the optimal value of* (6)*, and* $\mathcal{L}(\hat{y}, \hat{z})$ *is the strongest dual bound that can be obtained after fixing* $y = \hat{y}$ *in* (7)*.*

It is important to note the theoretical differences between the proposed dual completion, and applying a generic method, e.g., DC3 [5], LOOP-LC [14] or RAYEN [12], to the dual problem (7). First, LOOP-LC is not applicable here, because it only handles linear constraints and requires a compact feasible set, which is not the case in general for (7). Second, unlike RAYEN, Theorem 1 does not require an explicit characterization of the affine hull of the (dual) feasible set, nor does it assume knowledge of a strictly feasible point. In fact, Theorem 1 applies even if the feasible set of (7) has an empty interior. Third, the proposed dual completion enforces both linear equality constraints (7b) and conic constraints (7c). In contrast, the equality completion schemes used in DC3 and RAYEN enforce equality constraints but need an additional mechanism to handle inequality constraints. Fourth, the optimal completion outlined in Theorem 2 provides guarantees on the strength of the Lagrangian dual bound $\mathcal{L}(\hat{y}, \hat{z})$. This is a major difference with DC3 and RAYEN, whose correction mechanism does not provide any guarantee of solution quality. *Overall, the fundamental difference between generic methods and the proposed optimal dual completion, is that the former only exploit dual feasibility constraints* (7b)–(7c)*, whereas DLL also exploits (dual) optimality conditions, thus providing additional guarantees.*

Another desirable property of the proposed dual completion procedure, is that it does not require the user to formulate the dual problem (7) explicitly, as would be the case for DC3 or RAYEN. Instead,

the user only needs to identify a set of *primal* constraints that satisfy the conditions of Theorem 1. For instance, it suffices to identify constraints that bound the set of primal-feasible solutions. This is advantageous because practitioners typically work with primal problems rather than their dual. The optimal dual completion can then be implemented via an implicit optimization layer. Thereby, in a forward pass, $\hat{z}$ is computed by solving the primal-dual pair (8)–(27) and, in a backward pass, gradient information is obtained via the implicit function theorem [25].

The main limitations of implicit layers are their numerical instability and their computational cost, both in the forward and backward passes. To eliminate these issues, closed-form analytical solutions are presented next for broad classes of conic optimization problems; other examples are presented in the numerical experiments of Section 5.

Example 1 (Bounded variables)**.** *Consider a conic optimization problem with bounded variables*

$$\min_{x} \quad \left\{ c^\top x \;\middle|\; Ax \succeq_{\mathcal{K}} b, l \leq x \leq u \right\} \tag{9}$$

where $l < u$ are finite lower and upper bounds on all variables x. The dual problem is

$$\min_{y, z^l, z^u} \quad \left\{ b^\top y + l^\top z^l - u^\top z^u \;\middle|\; A^\top y + z^l - z^u = c, y \in \mathcal{K}^*, z^l \geq 0, z^u \geq 0 \right\} \tag{10}$$

and the optimal dual completion is $\hat{z}^l = |c - A^\top \hat{y}|^+, \hat{z}^u = |c - A^\top \hat{y}|^-$.

The assumption in Example 1 that all variables have finite bounds holds in most –if not all– real-life settings, where decision variables are physical quantities (e.g. budgets or production levels) that are naturally bounded. The resulting completion procedure is a generalization of that used in [22] for linear programming (LP) problems.

Example 2 (Trust region)**.** *Consider the trust region problem [26]*

$$\min_{x} \quad \left\{ c^\top x \;\middle|\; Ax \succeq_{\mathcal{K}} b, \|x\| \leq r \right\} \tag{11}$$

where $r \geq 0$, $\|\cdot\|$ is a norm, and $\|x\| \leq r \Leftrightarrow (r, x) \in \mathcal{C} = \{(t, x) \mid t \geq \|x\|\}$. The dual problem is

$$\max_{y, z_0, z} \quad \left\{ b^\top y - r z_0 \;\middle|\; A^\top y + z = c, y \in \mathcal{K}^*, (z_0, z) \in \mathcal{C}^* \right\} \tag{12}$$

where $\|\cdot\|_*$ is the dual norm and $\mathcal{C}^* = \{(t, x) \mid t \geq \|x\|_*\}$ [27]. The optimal dual completion is $\hat{z} = c - A^\top \hat{y}$, $\hat{z}_0 = \|\hat{z}\|_*$.

Example 3 (Convex quadratic objective)**.** *Consider the convex quadratic conic problem*

$$\min_{x} \quad \left\{ 1/2 \times x^\top Q x + c^\top x \;\middle|\; Ax \succeq_{\mathcal{K}} b \right\}, \tag{13}$$

where $Q = F^\top F$ is positive definite. The problem can be formulated as the conic problem

$$\min_{x} \quad \left\{ q + c^\top x \;\middle|\; Ax \succeq_{\mathcal{K}} b, (1, q, Fx) \in \mathcal{Q}_r^{2+n} \right\} \tag{14}$$

whose dual is

$$\max_{y, z_0, z} \quad \left\{ b^\top y - z_0 \;\middle|\; A^\top y + F^\top z = c, (1, z_0, z) \in \mathcal{Q}_r^{2+n} \right\}. \tag{15}$$

The optimal dual completion is $\hat{z} = F^{-\top}(c - A^\top \hat{y})$, $\hat{z}_0 = 1/2 \|\hat{z}\|_2^2$.

4.2 Conic Projections

The second building block of DLL are differentiable conic projection layers. Note that DLL only requires a valid projection onto $\mathcal{K}^*$, which need not be the Euclidean projection $\Pi_{\mathcal{K}^*}$. Indeed, the latter may be computationally expensive and cumbersome to differentiate. For completeness, the paper presents Euclidean and non-Euclidean projection operators, where the latter are simple to implement, computationally fast, and differentiable almost everywhere. Closed-form formulae are presented for each standard cone in Appendix B, and an overview is presented in Table 1.

https://doi.org/10.52202/079017-1764

Table 1: Overview of conic projections for standard cones

Cone	Definition	Euclidean projection	Radial projection
$\mathbb{R}_+$	Appendix B.1	(35)	(35)
$\mathcal{Q}$	Appendix B.2	(38)	(39)
$\mathcal{S}_+$	Appendix B.3	(41)	(42)
$\mathcal{E}$	Appendix B.4	no closed form	(45) and (47)
$\mathcal{P}$	Appendix B.5	no closed form	(50)

4.2.1 Euclidean projection

Let $\mathcal{K}$ be a proper cone, and $\bar{x} \in \mathbb{R}^n$. By Moreau's decomposition [28],

$$\bar{x} = \Pi_{\mathcal{K}}(\bar{x}) + \Pi_{\mathcal{K}^\circ}(\bar{x}), \tag{16}$$

which is a reformulation of the KKT conditions of the projection problem (1), i.e.,

$$\bar{x} = p - q, \quad p \in \mathcal{K}, \quad q \in \mathcal{K}^*, \quad p^\top q = 0. \tag{17}$$

It then follows that $\Pi_{\mathcal{K}^*}(\bar{x}) = -\Pi_{\mathcal{K}^\circ}(-\bar{x})$, by invariance of Moreau's decomposition under orthogonal transformations. Thus, it is sufficient to know how to project onto $\mathcal{K}$ to be able to project onto $\mathcal{K}^*$ and $\mathcal{K}^\circ$. Furthermore, (16) shows that $\Pi_{\mathcal{K}}$ is identically zero on the polar cone $\mathcal{K}^\circ$. In a machine learning context, this may cause gradient vanishing issues and slow down training.

4.2.2 Radial projection

Given an interior ray $\rho \succ_{\mathcal{K}} 0$, the radial projection operator $\Pi^\rho_{\mathcal{K}}$ is defined as

$$\Pi^\rho_{\mathcal{K}}(\bar{x}) = \bar{x} + \lambda\rho \quad \text{where} \quad \lambda = \min_{\lambda \geq 0}\{\lambda \,|\, \bar{x} + \lambda\rho \in \mathcal{K}\}. \tag{18}$$

The name stems from the fact that $\Pi^\rho_{\mathcal{K}}$ traces ray ρ from $\bar{x}$ until $\mathcal{K}$ is reached. Unlike the Euclidean projection, it requires an interior ray, which however only needs to be determined once *per cone*. The radial projection can then be computed, in general, via a line search on λ or via an implicit layer. Closed-form formulae for standard cones and their duals are presented in Appendix B.

4.3 Self-Supervised Dual Lagrangian Training

The third building block of DLL is a self-supervised learning framework for training dual conic optimization proxies. In all that follows, let $\xi = (A, b, H, h, c)$ denote the data of an instance (6), and assume a distribution of instances $\xi \sim \Xi$. Next, let $\mathcal{M}_\theta$ be a differentiable program parametrized by θ, e.g., an artificial neural network, which takes as input ξ and outputs a dual-feasible solution $(\hat{y}, \hat{z})$. Recall that dual feasibility of $(\hat{y}, \hat{z})$ can be enforced by combining the dual conic projection presented in Section 4.2, and the optimal dual completion outlined in Theorem 2.

The proposed self-supervised dual lagrangian training is formulated as

$$\max_{\theta} \quad \mathbb{E}_{\xi\sim\Xi}\left[\mathcal{L}(\hat{y}, \hat{z}, \xi)\right] \tag{19a}$$

$$\text{s.t.} \quad (\hat{y}, \hat{z}) = \mathcal{M}_\theta(\xi), \tag{19b}$$

where $\mathcal{L}(\hat{y}, \hat{z}, \xi) = b^\top \hat{y} + h^\top \hat{z}$ is the Lagrangian dual bound obtained from $(\hat{y}, \hat{z})$ by weak duality. Thereby, the training problem (19) seeks the value of θ that maximizes the expected Lagrangian dual bound over the distribution of instances Ξ, effectively mimicking the steps of a (sub)gradient algorithm. Note that, instead of updating $(\hat{y}, \hat{z})$ directly, the training procedure computes a (sub)gradient $\partial_\theta \mathcal{L}(\hat{y}, \hat{z}, \xi)$ to update θ, and then obtains a new prediction $(\hat{y}, \hat{z})$ through $\mathcal{M}_\theta$. Also note that formulation (19) does not required labeled data, i.e., it does not require pre-computed dual-optimal solutions. Furthermore, it applies to any architecture that guarantees dual feasibility of $(\hat{y}, \hat{z})$, i.e., it does not assume any specific projection nor completion procedure.

5 Numerical experiments

This section presents numerical experiments on linear and nonlinear optimization problems; detailed problem formulations, model architectures, and other experiment settings, are reported in Appendix

Table 2: Comparison of optimality gaps on linear programming instances.

			DC3			DLL		
m	n	Opt val*	avg.	std	max	avg.	std	max
5	100	14811.9	19.58	1.86	41.42	**0.36**	0.20	1.36
	200	29660.4	20.58	1.41	49.47	**0.18**	0.10	0.84
	500	74267.0	33.70	1.29	41.54	**0.07**	0.04	0.30
10	100	14675.8	41.85	2.51	69.58	**0.68**	0.25	2.15
	200	29450.7	36.88	2.28	100.90	**0.34**	0.13	0.96
	500	73777.5	100.04	3.38	104.00	**0.14**	0.06	0.46
30	100	14441.5	159.49	5.54	166.31	**1.93**	0.37	3.31
	200	29156.1	255.24	8.42	259.25	**0.96**	0.20	1.83
	500	73314.3	274.78	7.91	277.40	**0.38**	0.09	0.75

All gaps are in %; best values are in bold. *Mean optimal value on test set; obtained with Gurobi.

C. The code used for experiments is available under an open-source license.[1] The proposed DLL methodology is evaluated against applying DC3 to the dual problem (7) as a baseline. Thereby, linear equality constraints (7b) and conic inequality constraints (7c) are handled by DC3's equality completion and inequality correction mechanisms, respectively. The two approaches (DLL and DC3) are evaluated in terms of dual optimality gap and training/inference time. The dual optimality gap is defined as $(\mathcal{L}^* - \mathcal{L}(\hat{y}, \hat{z}))/\mathcal{L}^*$, where $\mathcal{L}^*$ is the optimal value obtained from a state-of-the-art interior-point solver.

5.1 Linear Programming Problems

5.1.1 Problem formulation and dual completion

The first set of experiments considers continuous relaxations of multi-dimensional knapsack problems [29, 30], which are of the form

$$\min_{x} \quad \left\{ -p^\top x \;\middle|\; Wx \leq b, x \in [0,1]^n \right\} \tag{20}$$

where $p \in \mathbb{R}^n_+$, $W \in \mathbb{R}^{m \times n}_+$, and $b \in \mathbb{R}^m_+$. The dual problem reads

$$\max_{y, z^l, z^u} \quad \left\{ b^\top y - \mathbf{e}^\top z^u \;\middle|\; W^\top y + z^l - z^u = -p, y \leq 0, z^l \geq 0, z^u \geq 0, \right\} \tag{21}$$

where $y \in \mathbb{R}^m$ and $z^l, z^u \in \mathbb{R}^n$. Since variables x is bounded, the closed-form completion presented in Example 1 applies. Namely, $\hat{z}^l = |-p - W^\top \hat{y}|^+$ and $\hat{z}^u = |-p - W^\top \hat{y}|^-$, where $\hat{y} \in \mathbb{R}^m_-$.

5.1.2 Numerical results

Table 2 reports, for each combination of m, n: the average optimal value obtained by Gurobi (Opt val), as well as the average (avg), standard-deviation (std) and maximum (max) optimality gaps achieved by DC3 and DLL on the test set. First, DLL significantly outperforms DC3, with average gaps ranging from 0.07% to 1.93%, compared with 19.58%–274.78% for DC3, an improvement of about two orders of magnitude. A similar behavior is observed for maximum optimality gaps. The rest of the analysis thus focuses on DLL. Second, an interesting trend can be identified: optimality gaps tend to increase with m and decrease with n. This effect may be explained by the fact that increasing m increases the output dimension of the FCNN; larger output dimensions are typically harder to predict. In addition, a larger n likely provides a smoothing effect on the dual, whose solution becomes easier to predict. The reader is referred to [30] for probabilistic results on properties of multi-knapsack problems.

Next, Table 3 reports computing time statistics for Gurobi, DC3 and DLL. Namely, the table reports, for each combination of m, n, the time it takes to execute each method on all instances in the test set. First, DLL is 3-10x faster than DC3, which is caused by DC3's larger output dimension ($m+n$, compared to m for DLL), and its correction steps. Furthermore, unsurprisingly, both DC3 and DLL

[1] `https://github.com/AI4OPT/DualLagrangianLearning`

https://doi.org/10.52202/079017-1764

Table 3: Computing time statistics for linear programming instances

m	n	Gurobi†	DC3‡	DLL‡
5	100	2.8 CPU.s	2.1 GPU.ms	0.3 GPU.ms
	200	4.1 CPU.s	4.0 GPU.ms	0.7 GPU.ms
	500	6.6 CPU.s	13.2 GPU.ms	3.0 GPU.ms
10	100	3.7 CPU.s	2.3 GPU.ms	0.4 GPU.ms
	200	6.1 CPU.s	4.9 GPU.ms	1.1 GPU.ms
	500	11.9 CPU.s	17.2 GPU.ms	4.7 GPU.ms
30	100	14.0 CPU.s	4.6 GPU.ms	0.9 GPU.ms
	200	21.3 CPU.s	10.4 GPU.ms	2.5 GPU.ms
	500	40.0 CPU.s	39.5 GPU.ms	13.6 GPU.ms

†Time to solve all instances in the test set, using one CPU core. ‡Time to run inference on all instances in the test set, using one V100 GPU.

yield substantial speedups compared to Gurobi, of about 3 orders of magnitude. Note however that Gurobi's timings could be improved given additional CPU cores, although both ML-based methods remain significantly faster using a single GPU.

5.2 Nonlinear Production and Inventory Planning Problems

5.2.1 Problem formulation and dual completion

The second set of experiments considers the nonlinear resource-constrained production and inventory planning problem [31, 32]. In primal-dual form, the problem reads

$$\min_{x,t} \quad d^\top x + f^\top t \tag{22a}$$
$$\text{s.t.} \quad r^\top x \le b, \tag{22b}$$
$$(x_j, t_j, \sqrt{2}) \in \mathcal{Q}_r^3,\ j{=}1, ..., n \tag{22c}$$

$$\max_{y,\pi,\tau,\sigma} \quad by - \sqrt{2}\mathbf{e}^\top \sigma_j \tag{23a}$$
$$s.t. \quad ry + \pi = d, \tag{23b}$$
$$\tau = f, \tag{23c}$$
$$y \le 0, \tag{23d}$$
$$(\pi_j, \tau_j, \sigma_j) \in \mathcal{Q}_r^3,\ j{=}1, ..., n \tag{23e}$$

where $r, d, f \in \mathbb{R}^n$ are positive vectors, and $b > 0$. Primal variables are $x, t \in \mathbb{R}^n$, and the dual variables associated to constraints (22b) and (22c) are $y \in \mathbb{R}_-$, and $\pi, \sigma, \tau \in \mathbb{R}^n$, respectively.

Note that (22c) implies $x, t \ge 0$. Next, let $y \le 0$ be fixed, and consider the problem

$$\min_{x,t} \quad \left\{ (d - yr)^\top x + f^\top t + by \;\middle|\; (22c) \right\}. \tag{24}$$

Problem (24) is immediately strictly feasible, and bounded since $(d - yr), f > 0$ and $x, t \ge 0$. Hence, Theorems 1 and 2 apply, and there exists a dual-optimal completion to recover π, σ, τ. A closed-form completion is then achieved as follows. First, constraints (23b) and (23c) yield $\pi = d - ry$ and $\tau = f$. Next, note that σ only appears in constraint (23e) and has negative objective coefficient. Further noting that (23e) can be written as $\sigma_j^2 \le 2\pi_j\tau_j$, it follows that $\sigma_j = -\sqrt{2\pi_j\tau_j}$ at the optimum.

5.2.2 Numerical Results

Table 4 reports optimality gap statistics for DC3 and DLL. Similar to the linear programming setting, DLL substantially outperforms DC3, with average optimality gaps ranging from 0.23% to 1.03%, compared with 70.76%–87.01% for DC3. In addition, DLL exhibits smaller standard deviation and maximum optimality gaps than DC3. These results can be explained by several factors. First, the neural network architecture used in DC3 has output size $n+1$, compared to 1 for DLL; this is because DLL leverages a more efficient dual completion procedure. Second, a closer examination of DC3's output reveals that it often fails to satisfy the (conic) inequality constraints (23d) and (23e). More generally, DC3 was found to have much slower convergence than DLL during training. While the performance of DC3 may benefit from more exhaustive hypertuning, doing so comes at a significant computational and environmental cost. This further highlights the benefits of DLL, which requires minimal tuning and is efficient to train.

Table 4: Comparison of optimality gaps on production planning instances.

		DC3			DLL		
n	Opt val*	avg.	std	max	avg.	std	max
10	3441.8	70.76	9.42	90.23	**0.23**	0.57	17.05
20	6988.2	78.52	6.67	92.31	**0.41**	0.69	9.04
50	17667.4	81.70	5.41	92.69	**1.03**	1.69	21.68
100	35400.2	83.25	4.78	93.31	**0.37**	0.57	6.69
200	70889.5	84.06	4.20	93.44	**0.29**	0.46	4.81
500	177060.0	86.74	3.80	93.74	**0.46**	0.73	9.92
1000	354037.5	87.01	3.71	93.80	**0.36**	0.48	4.44

All gaps are in %; best values are in bold. *Mean optimal value on test set; obtained with Mosek.

Table 5: Computing time statistics for nonlinear instances

n	Mosek†	DC3‡	DLL‡
10	73.5 CPU.s	2.7 GPU.ms	0.2 GPU.ms
20	75.3 CPU.s	2.7 GPU.ms	0.2 GPU.ms
50	15.4 CPU.s	2.7 GPU.ms	0.2 GPU.ms
100	24.9 CPU.s	2.7 GPU.ms	0.4 GPU.ms
200	49.9 CPU.s	5.1 GPU.ms	1.0 GPU.ms
500	98.8 CPU.s	15.9 GPU.ms	5.1 GPU.ms
1000	203.0 CPU.s	41.5 GPU.ms	19.0 GPU.ms

†Time to solve all instances in the test set, using one CPU core. ‡Time to run inference on all instances in the test set, using one V100 GPU.

Finally, Table 5 reports computing time statistics for Mosek, a state-of-the-art conic interior-point solver, DC3 and DLL. Abnormally high times are observed for Mosek and n=10, 20. These are most likely caused by congestion on the computing nodes used in the experiments, and are discarded in the analysis. Again, DC3 and DLL outperform Mosek by about three orders of magnitude. Furthermore, DLL is about 10x faster than DC3 for smaller instances ($n\leq$100), and about 2x faster for the largest instances (n=1000). This is caused by DC3's larger output dimension and correction steps.

6 Discussion

6.1 Mixed-Integer Nonlinear Programming Setting

The proposed Dual Lagrangian Learning framework directly extends to the mixed-integer nonlinear programming (MINLP) setting. Consider a general MINLP problem of the form

$$\min_{x\in\mathcal{X}} \quad \{f(x) \mid h(x)=0, g(x)\geq 0\}, \tag{25}$$

where $\mathcal{X}\subseteq\mathbb{R}^n$ denotes a possibly discrete domain. Given Lagrange multipliers $\lambda\in\mathbb{R}^m$ and $\mu\in\mathbb{R}^p_+$ associated to equality and inequality constraints, the corresponding Lagrangian dual bound is

$$\mathcal{L}(\lambda,\mu)=\min_{x\in\mathcal{X}} \quad f(x)-\lambda^\top h(x)-\mu^\top g(x). \tag{26}$$

Note that (26) is the MINLP counterpart of (8) in the conic setting.

The dual Lagrangian function $\mathcal{L}(\lambda,\mu)$ is concave, non-smooth, and admits sub-differentials of the form $\partial_\lambda\mathcal{L}=-h(\bar{x})$, $\partial_\mu\mathcal{L}=-g(\bar{x})$, where $\bar{x}$ is an *optimal* solution of (26). The self-supervised learning framework of Section 4.3 can then be applied out of the box, wherein an ML model predicting (λ,μ) is trained in a self-supervised fashion by maximizing the dual bound $\mathcal{L}(\lambda,\mu)$. This approach is followed in, e.g., [20] for mixed-integer linear problems, and [6, 8] for nonlinear problems.

Despite natural similarities between the (convex) conic and MINLP settings, several intrinsic limitations appear in the latter. First, although the domain of $(\lambda,\mu)\in\mathbb{R}^m\times\mathbb{R}^p_+$ is simple, and can be enforced via, e.g., ReLU activations, evaluating $\mathcal{L}(\lambda,\mu)$ is not. Indeed, this requires solving the MINLP problem (26) *to optimality*, which is NP-hard in general. In contrast, the proposed dual conic completion can be performed efficiently, and closed-form solutions are available for broad classes of problems. Second, (sub)gradient information $\partial\mathcal{L}$ is obtained from an *optimal* solution of (26), which poses obvious limitations if (26) is solved approximately. Third, arbitrary values of λ, μ may result in

https://doi.org/10.52202/079017-1764

(26) being unbounded, yielding a dual bound of $-\infty$ and no usable gradient information. In contrast, in the conic setting, Theorem 1 provides sufficient conditions under which dual completion is always possible. Finally, an intrinsic limitation in the MINLP setting is the absence, in general, of strong duality. Therefore, even predicting a dual-optimal (λ, μ) may be insufficient to prove optimality, thus requiring additional computation such as branching. In contrast, the strong conic duality theorem [23] offers a robust foundation to obtain high-quality dual bounds efficiently.

6.2 Limitations

The main theoretical limitation of the paper is that it considers convex conic optimization problems, and therefore does not consider discrete decisions nor general non-convex constraints. Since convex relaxations are typically used to solve non-convex problems to global optimality, the proposed approach is nonetheless still useful in non-convex settings. Furthermore, as pointed out in Section 6.1, the DLL framework extends naturally to the MINLP setting, by leveraging Lagrangian duality for discrete and/or nonlinear problems. However, this approach suffers from several theoretical and computational limitations.

On the practical side, the optimal dual completion presented in Section 4.1 requires, in general, the use of an implicit layer, which is typically not tractable for large-scale problems. In the absence of a known closed-form optimal dual completion, it may still be possible to design efficient completion strategies that at least ensure dual feasibility. One such strategy is to introduce artificial large bounds on all primal variables, and use the completion outlined in Example 1. Finally, all neural network architectures considered in the experiments are fully-connected neural networks. Thus, a separate model is trained for each input dimension. Nevertheless, the DLL methodology is applicable to graph neural network architectures, which would support arbitrary problem size. The use of GNN models in the DLL context is a promising avenue for future research.

7 Conclusion

The paper has proposed Dual Lagrangian Learning (DLL), a principled methodology for learning dual conic optimization proxies. Thereby, a systematic dual conic completion, differentiable conic projection layers, and a self-supervised dual Lagrangian training framework have been proposed. The effectiveness of DLL has been demonstrated on numerical experiments that consider linear and nonlinear conic problems, where DLL significantly outperforms DC3 [5], and achieves 1000x speedups over commercial interior-point solvers.

One of the main advantages of DLL is its simplicity. The proposed dual completion can be stated only in terms of *primal* constraints, thus relieving users from the need to explicitly write the dual problem. DLL introduces very few hyper-parameters, and requires minimal tuning to achieve good performance. This results in simpler models and improved performance, thus delivering computational and environmental benefits.

DLL opens the door to multiple avenues for future research, at the intersection of ML and optimization. The availability of high-quality dual-feasible solutions naturally calls for the integration of DLL in existing optimization algorithms, either as a warm-start, or to obtain good dual bounds fast. Multiple optimization algorithms have been proposed to optimize Lagrangian functions, which may yield more efficient training algorithms in DLL. Finally, given the importance of conic optimization in numerous real-life applications, DLL can provide a useful complement to existing primal proxies.

Acknowledgments

This research was partially supported by NSF awards 2007164 and 2112533, and ARPA-E PERFORM award DE-AR0001280.

References

[1] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.07.063.

[2] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4475–4482. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/610. URL https://doi.org/10.24963/ijcai.2021/610. Survey Track.

[3] Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. Predicting AC Optimal Power Flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 630–637, 2020. URL https://doi.org/10.1609/aaai.v34i01.5403.

[4] Xiang Pan, Tianyu Zhao, Minghua Chen, and Shengyu Zhang. DeepOPF: A deep neural network approach for security-constrained DC optimal power flow. *IEEE Transactions on Power Systems*, 36(3):1725–1735, 2020.

[5] Priya L. Donti, David Rolnick, and J. Zico Kolter. DC3: A learning method for optimization with hard constraints. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021. URL https://iclr.cc/virtual/2021/poster/2868.

[6] Seonho Park and Pascal Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4052–4060, 2023. doi: 10.1609/aaai.v37i4.25520. URL https://ojs.aaai.org/index.php/AAAI/article/view/25520.

[7] Chendi Qian, Didier Chételat, and Christopher Morris. Exploring the power of graph neural networks in solving linear optimization problems. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 1432–1440. PMLR, 02–04 May 2024. URL https://proceedings.mlr.press/v238/qian24a.html.

[8] James Kotary and Ferdinando Fioretto. Learning constrained optimization with deep augmented lagrangian methods, 2024.

[9] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Bk9mxlSFx.

[10] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[11] Wenbo Chen, Mathieu Tanneau, and Pascal Van Hentenryck. End-to-End Feasible Optimization Proxies for Large-Scale Economic Dispatch. *IEEE Transactions on Power Systems*, pages 1–12, 2023. doi: 10.1109/TPWRS.2023.3317352.

[12] Jesus Tordesillas, Jonathan P How, and Marco Hutter. Rayen: Imposition of hard convex constraints on neural networks, 2023.

[13] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/9ce3c52fc54362e22053399d3181c638-Paper.pdf.

[14] Meiyi Li, Soheil Kolouri, and Javad Mohammadi. Learning to Solve Optimization Problems With Hard Linear Constraints. *IEEE Access*, 11:59995–60004, 2023. doi: 10.1109/ACCESS.2023.3285199.

[15] Andrei V Konstantinov and Lev V Utkin. A new computationally simple approach for implementing neural networks with output hard constraints. In *Doklady Mathematics*, pages 1–9. Springer, 2024.

[16] Terrence W.K. Mak, Minas Chatzos, Mathieu Tanneau, and Pascal Van Hentenryck. Learning regionally decentralized ac optimal power flows with admm. *IEEE Transactions on Smart Grid*, 14(6):4863–4876, 2023.

[17] Sebastian Kraul, Markus Seizinger, and Jens O. Brunner. Machine learning–supported prediction of dual variables for the cutting stock problem with an application in stabilized column generation. *INFORMS Journal on Computing*, 35(3):692–709, 2023. doi: 10.1287/ijoc.2023.1277.

[18] Nagisa Sugishita, Andreas Grothey, and Ken McKinnon. Use of machine learning models to warmstart column generation for unit commitment. *INFORMS Journal on Computing*, 2024. doi: 10.1287/ijoc.2022.0140.

[19] Augustin Parjadis, Quentin Cappart, Bistra Dilkina, Aaron Ferber, and Louis-Martin Rousseau. Learning Lagrangian Multipliers for the Travelling Salesman Problem, 2023.

[20] Francesco Demelas, Joseph Le Roux, Mathieu Lacroix, and Axel Parmentier. Predicting lagrangian multipliers for mixed integer linear programs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 10368–10384. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/demelas24a.html`.

[21] Guancheng Qiu, Mathieu Tanneau, and Pascal Van Hentenryck. Dual Conic Proxies for AC Optimal Power Flow. In *Power Systems Computations Conference*, 2024.

[22] Michael Klamkin, Mathieu Tanneau, and Pascal Van Hentenryck. Dual interior-point optimization learning, 2024.

[23] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[24] M. Lubin, E. Yamangil, R. Bent, and J. P. Vielma. Extended Formulations in Mixed-Integer Convex Programming. In Q. Louveaux and M. Skutella, editors, *Proceedings of the 18th Conference on Integer Programming and Combinatorial Optimization (IPCO 2016)*, volume 9682 of *Lecture Notes in Computer Science*, pages 102–113, 2016.

[25] Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M. Moursi. Differentiating through a Cone Program. *Journal of Applied and Numerical Optimization*, 1(2):107–115, August 2019.

[26] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[27] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[28] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.

[29] Arnaud Freville and Gérard Plateau. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete Applied Mathematics*, 49(1):189–212, 1994. ISSN 0166-218X. doi: 10.1016/0166-218X(94)90209-7. Special Volume Viewpoints on Optimization.

[30] Arnaud Freville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004. ISSN 0377-2217. doi: 10.1016/S0377-2217(03)00274-1.

[31] Hans Ziegler. Solving certain singly constrained convex optimization problems in production planning. *Operations Research Letters*, 1(6):246–252, 1982. ISSN 0167-6377. doi: 10.1016/0167-6377(82)90030-X.

[32] MOSEK Aps. *The MOSEK Modeling Cookbook*, 2023. URL `https://docs.mosek.com/modeling-cookbook/index.html`.

 https://doi.org/10.52202/079017-1764

A Proofs for Section 4 (Dual Lagrangian Learning (DLL))

Theorem 1 (Dual conic completion)**.** *Assume that* $\forall \hat{y} \in \mathcal{K}^*, \exists x : Hx \succ_{\mathcal{C}} h$ *and the problem*

$$\min_{x} \quad \left\{ c^\top x + (b - Ax)^\top \hat{y} \;\middle|\; Hx \succeq_{\mathcal{C}} h \right\} \tag{8}$$

is bounded. Then, $\forall \hat{y} \in \mathcal{K}^*, \exists \hat{z} \in \mathcal{C}^* : A^\top \hat{y} + H^\top \hat{z} = c$*, i.e.,* $(\hat{y}, \hat{z})$ *is feasible for* (7)*.*

Proof. Let $\hat{y} \in \mathcal{K}^*$, and recall that (8) is bounded and strictly feasible. By strong duality, its dual

$$\max_{z} \quad \left\{ h^\top z + b^\top \hat{y} \;\middle|\; H^\top z = c - A^\top \hat{y},\ z \in \mathcal{C}^* \right\} \tag{27}$$

is solvable [BTN01]. Therefore, there exists a feasible solution $\hat{z}$ for (27). By construction, $\hat{z} \in \mathcal{C}^*$ and $A^\top \hat{y} + H^\top \hat{z} = c$, hence $(\hat{y}, \hat{z})$ is feasible for (7). □

Theorem 2 (Optimal dual completion)**.** *Let* $\hat{y} \in \mathcal{K}^*$*, and let* $\hat{z}$ *be dual-optimal for* (8)*. Then,* $\mathcal{L}(\hat{y}, \hat{z}) = b^\top \hat{y} + h^\top \hat{z}$ *is a valid dual bound on the optimal value of* (6)*, and* $\mathcal{L}(\hat{y}, \hat{z})$ *is the strongest dual bound that can be obtained after fixing* $y = \hat{y}$ *in* (7)*.*

Proof. First, recall that $\hat{z}$ exists by strong conic duality; see proof of Theorem 1. Furthermore, $(\hat{y}, \hat{z})$ is feasible for (7) by construction. Thus, by weak duality, the Lagrangian bound $\mathcal{L}(\hat{y}) = b^\top \hat{y} + h^\top \hat{z}$ is a valid dual bound on the optimal value of (6). Finally, fixing $y = \hat{y}$ in (7) yields

$$\max_{y,z} \quad \{(7a) \mid (7b), (7c), y = \hat{y}\}, \tag{28}$$

which is equivalent to (27). Hence, its optimal value is $b^\top \hat{y} + h^\top \hat{z} = \mathcal{L}(\hat{y}, \hat{z})$ by definition of $\hat{z}$. □

Example 1 (Bounded variables)**.** *Consider a conic optimization problem with bounded variables*

$$\min_{x} \quad \left\{ c^\top x \;\middle|\; Ax \succeq_{\mathcal{K}} b, l \leq x \leq u \right\} \tag{9}$$

where $l < u$ *are finite lower and upper bounds on all variables* x*. The dual problem is*

$$\min_{y, z^l, z^u} \quad \left\{ b^\top y + l^\top z^l - u^\top z^u \;\middle|\; A^\top y + z^l - z^u = c, y \in \mathcal{K}^*, z^l \geq 0, z^u \geq 0 \right\} \tag{10}$$

and the optimal dual completion is $\hat{z}^l = |c - A^\top \hat{y}|^+, \hat{z}^u = |c - A^\top \hat{y}|^-$*.*

Proof. Let $\hat{y} \in \mathcal{K}^*$ be fixed. Fixing $y = \hat{y}$ in the dual problem yields

$$\max_{z^l, z^u} \quad l^\top z^l - u^\top z^u \tag{29}$$

$$\text{s.t.} \quad z^l - z^u = c - A^\top \hat{y} \tag{30}$$

$$z^l, z^u \geq 0. \tag{31}$$

Eliminating $z^l = z^u + (c - A^\top \hat{y})$, the problem becomes

$$\max_{z^u} \quad (l - u)^\top z^u + l^\top (c - A^\top \hat{y}) \tag{32}$$

$$\text{s.t.} \quad z^u \geq -(c - A^\top \hat{y}) \tag{33}$$

$$z^u \geq 0. \tag{34}$$

Since $l < u$, i.e., the objective coefficient of z^u is negative, and the problem is a maximization problem, it follows that z^u must be as small as possible in any optimal solution. Hence, at the optimum, $\hat{z}^u = \max(0, -(c - A^\top y)) = |c - A^\top \hat{y}|^-$, and $\hat{z}^l = |c - A^\top \hat{y}|^+$. □

Example 2 (Trust region)**.** *Consider the trust region problem [NW99]*

$$\min_{x} \quad \left\{ c^\top x \;\middle|\; Ax \succeq_{\mathcal{K}} b, \|x\| \leq r \right\} \tag{11}$$

where $r \geq 0$*,* $\|\cdot\|$ *is a norm, and* $\|x\| \leq r \Leftrightarrow (r, x) \in \mathcal{C} = \{(t, x) \mid t \geq \|x\|\}$*. The dual problem is*

$$\max_{y, z_0, z} \quad \left\{ b^\top y - r z_0 \;\middle|\; A^\top y + z = c, y \in \mathcal{K}^*, (z_0, z) \in \mathcal{C}^* \right\} \tag{12}$$

where $\|\cdot\|_*$ *is the dual norm and* $\mathcal{C}^* = \{(t, x) \mid t \geq \|x\|_*\}$ *[BV04]. The optimal dual completion is* $\hat{z} = c - A^\top \hat{y}$*,* $\hat{z}_0 = \|\hat{z}\|_*$*.*

Proof. The relation $\hat{z} = c - A^\top \hat{y}$ is immediate from the dual equality constraint $A^\top y + z = c$. Next, observe that z_0 appears only in the constraint $(z_0, z) \in \mathcal{C}^*$, and has negative objective coeffient. Hence, z_0 must be as small as possible in any optimal solution. This yields $\hat{z}_0 = \|\hat{z}\|_*$. □

Example 3 (Convex quadratic objective)**.** *Consider the convex quadratic conic problem*

$$\min_x \quad \left\{ 1/2 \times x^\top Q x + c^\top x \;\middle|\; Ax \succeq_\mathcal{K} b \right\}, \tag{13}$$

where $Q = F^\top F$ is positive definite. The problem can be formulated as the conic problem

$$\min_x \quad \left\{ q + c^\top x \;\middle|\; Ax \succeq_\mathcal{K} b, (1, q, Fx) \in \mathcal{Q}_r^{2+n} \right\} \tag{14}$$

whose dual is

$$\max_{y,z_0,z} \quad \left\{ b^\top y - z_0 \;\middle|\; A^\top y + F^\top z = c, (1, z_0, z) \in \mathcal{Q}_r^{2+n} \right\}. \tag{15}$$

The optimal dual completion is $\hat{z} = F^{-\top}(c - A^\top \hat{y}), \hat{z}_0 = 1/2 \|\hat{z}\|_2^2$.

Proof. The proof uses the same argument as the proof of Example 2. Namely, $\hat{z} = F^{-\top}(c - A^\top \hat{y})$ is immediate from the dual equality constraint $A^\top y + F^\top z = c$. Note that $F^\top$ is non-singular because Q is positive definite. Finally, z_0 has negative objective coefficient, and only appears in the conic constraint $(1, z_0, z) \in \mathcal{Q}_r^{2+n}$. Therefore, at the optimum, one must have $2\hat{z}_0 = \|\hat{z}\|_2^2$, which concludes the proof. □

B Standard cones

This section presents standard cones and their duals, as well as corresponding Euclidean and radial projections. The reader is referred to [CKV22] for a more exhaustive list of non-standard cones, and to [PB$^+$14, Sec. 6.3] for an overview of Euclidean projections onto standard cones.

B.1 Non-negative orthant

The non-negative orthant is defined as $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x \geq 0\}$. It is a self-dual cone, and forms the basis of linear programming [BTN01].

Euclidean projection The Euclidean projection on $\mathbb{R}^n_+$ is

$$\Pi_{\mathbb{R}^n_+}(\bar{y}) = \max(0, \bar{y}) = \mathrm{ReLU}(\bar{y}), \tag{35}$$

where the $\max$ and ReLU operations are performed element-wise.

Radial projection The radial projection with ray $\mathbf{e}$, applied coordinate-wise, is equivalent to the Euclidean projection.

B.2 Conic quadratic cones

Conic quadratic cones include the second-order cone (SOC)

$$\mathcal{Q}^n = \{x \in \mathbb{R}^n : x_1 \geq \sqrt{x_2^2 + \cdots + x_n^2}\} \tag{36}$$

and the rotated second-order cone (RSOC)

$$\mathcal{Q}_r^n = \{x \in \mathbb{R}^n : 2x_1 x_2 \geq x_3^2 + \cdots + x_n^2, x_1, x_2 \geq 0\}. \tag{37}$$

Both cones are self-dual, i.e., $\mathcal{Q}^* = \mathcal{Q}$ and $\mathcal{Q}_r^* = \mathcal{Q}_r$. The RSOC is the main building block of conic formulations of convex quadratically-constrained optimization problems.

Euclidean projection The Euclidean projection on $\mathcal{Q}^n$ is given by

$$\Pi_{\mathcal{Q}^n}(\bar{x}) = \begin{cases} \bar{x} & \text{if } \bar{x} \in \mathcal{Q}^n \\ 0 & \text{if } \bar{x} \in -\mathcal{Q}^n \\ \frac{\bar{x}_1 + \delta}{2\delta}(\delta, \bar{x}_2, ..., \bar{x}_n) & \text{otherwise} \end{cases} \tag{38}$$

where $\delta = \|(\bar{x}_2, ..., \bar{x}_n)\|_2$.

https://doi.org/10.52202/079017-1764

Radial projection Given interior ray $\rho = (1, 0, ..., 0) \succ_{\mathcal{Q}^n} 0$, the radial projection is

$$\Pi^{\rho}_{\mathcal{Q}^n}(\bar{x}) = (\hat{x}_1, \bar{x}_2, ..., \bar{x}_n), \qquad \hat{x}_1 = \max(\bar{x}_1, \|(\bar{x}_2, ..., \bar{x}_n)\|_2). \tag{39}$$

Note that, in the worst case, computing $\Pi_{\mathcal{Q}}(\bar{x})$ requires $\mathcal{O}(2n)$ operations, and modifies all coordinates of $\bar{x}$. In contrast, computing $\Pi^{\rho}_{\mathcal{Q}}(\bar{x})$ requires only $\mathcal{O}(n)$ operations, and only modifies the first coordinate of $\bar{x}$.

Closed-form formulae for Euclidean and radial projections onto $\mathcal{Q}^n_r$ are derived from (38) and (39).

B.3 Positive Semi-Definite cone

The cone of positive semi-definite (PSD) matrices of order n is defined as

$$\mathcal{S}^n_+ = \{X \in \mathbb{R}^{n \times n} : X = X^\top, \lambda_{\min}(X) \geq 0\}. \tag{40}$$

Note that all matrices in $\mathcal{S}^n_+$ are symmetric, hence all their eigenvalues are real. The PSD cone is self-dual, and generalizes the non-negative orthant and SOC cones [BV04].

Euclidean projection The Euclidean projection onto $\mathcal{S}^n_+$ is given by

$$\Pi_{\mathcal{S}^n_+}(\bar{X}) = \sum_i \max(0, \lambda_i) v_i v_i^\top, \tag{41}$$

where $\bar{X} \in \mathbb{R}^{n \times n}$ is symmetric with eigenvalue decomposition $\bar{X} = \sum_i \lambda_i v_i v_i^\top$. Note that the Euclidean projection onto the PSD code thus requires a full eigenvalue decomposition, which has complexity $\mathcal{O}(n^3)$.

Radial projection The radial projection considered in the paper uses $\rho = I_n \in \text{int}(\mathcal{S}^n_+)$. This yields the closed-form projection

$$\Pi^{\rho}_{\mathcal{S}^n_+}(\bar{X}) = \bar{X} + \min(0, |\lambda_{\min}(\bar{X})|) I_n. \tag{42}$$

Note that the radial projection only requires computing the smallest eigenvalue of $\bar{X}$, which is typically much faster than a full eigenvalue decomposition, and only modifies the diagonal of $\bar{X}$.

B.4 Exponential Cone

The 3-dimensional exponential cone is a non-symmetric cone defined as

$$\mathcal{E} = \text{cl}\left\{x \in \mathbb{R}^3 : x_1 \geq x_2 e^{x_3/x_2}, x_2 > 0\right\}, \tag{43}$$

whose dual cone is

$$\mathcal{E}^* = \text{cl}\left\{y \in \mathbb{R}^3 : \frac{-y_1}{y_3} \geq e^{\frac{y_2}{y_3} - 1}, y_1 > 0, y_3 < 0\right\}. \tag{44}$$

The exponential cone is useful to model exponential and logarithmic terms, which occur in, e.g., relative entropy, logistic regression, or logarithmic utility functions.

Euclidean projection To the best of the authors' knowledge, there is no closed-form, analytical formula for evaluating $\Pi_{\mathcal{E}}$ nor $\Pi_{\mathcal{E}^*}$, which instead require a numerical method, see, e.g., [PB$^+$14] and [Fri23] for completeness.

Radial projection To avoid any root-finding operation, the paper leverages the fact that $x_1, x_2 > 0, \forall (x_1, x_2, x_3) \in \mathcal{E}$. Note that one can enforce $\bar{x}_1, \bar{x}_2 > 0$ via, e.g., softplus activation. A radial projection is then obtained using $\rho = (0, 0, 1)$, which yields

$$\Pi^{\rho}_{\mathcal{E}}(\bar{x}_1, \bar{x}_2, \bar{x}_3) = \left(\bar{x}_1, \bar{x}_2, \min\left(\bar{x}_3, \bar{x}_2 \log \frac{\bar{x}_1}{\bar{x}_2}\right)\right). \tag{45}$$

This approach does not require any root-finding, and is therefore more amenable to automatic differentiation. The validity of Eq. (45) is immediate from the representation

$$\mathcal{E} = \text{cl}\{x \in \mathbb{R}^3 : \frac{x_3}{x_2} \leq \log \frac{x_1}{x_2}, x_1, x_2 > 0\}. \tag{46}$$

Similarly, assuming $\bar{y}_1 > 0$ and $\bar{y}_3 < 0$, the radial projection onto $\mathcal{E}^*$ reads

$$\Pi^{\rho}_{\mathcal{E}^*}(\bar{x}_1, \bar{x}_2, \bar{x}_3) = \left(\bar{y}_1, \max\left(\bar{y}_2, \bar{y}_3 + \bar{y}_3 \ln \frac{\bar{y}_1}{-\bar{y}_3}\right), \bar{y}_3\right). \tag{47}$$

B.5 Power Cone

Given $0 < \alpha < 1$, the 3-dimensional power cone is defined as

$$\mathcal{P}_\alpha = \left\{ x \in \mathbb{R}^3 : x_1^\alpha x_2^{1-\alpha} \geq |x_3|, x_1, x_2 \geq 0 \right\}. \tag{48}$$

Power cones are non-symmetric cones, which allow to express power other than 2, e.g., p-norms with $p \geq 1$. Note that $\mathcal{P}_{1/2}$ is a scaled version of the rotated second-order cone $\mathcal{Q}_r^3$. The 3-dimensional power cone $\mathcal{P}_\alpha$ is sufficient to express more general, high-dimensional power cones. The dual power cone is

$$\mathcal{P}_\alpha^* = \left\{ y \in \mathbb{R}^3 : \left(\frac{y_1}{\alpha}, \frac{y_2}{1-\alpha}, y_3 \right) \in \mathcal{P}_\alpha \right\}. \tag{49}$$

Euclidean projection The Euclidean projection onto the power cone $\mathcal{P}_\alpha$ is described in [Hie15]. Similar to the exponential cone, it requires a root-finding operation.

Radial projection The proposed radial projection is similar to the one proposed for $\mathcal{E}$. Assuming $\bar{x}_2, \bar{x}_3 > 0$, and using $\rho = (1, 0, 0)$, the radial projection reads

$$\Pi_{\mathcal{P}_\alpha}^\rho(\bar{x}_1, \bar{x}_2, \bar{x}_3) = \left(\max\left(\bar{x}_1, \bar{x}_2^{\frac{\alpha-1}{\alpha}} |\bar{x}_3|^{\frac{1}{\alpha}}\right), \bar{x}_2, \bar{x}_3 \right). \tag{50}$$

A similar approach is done to recover $y \in \mathcal{P}_\alpha^*$ after scaling the first two coordinates of y. This technique can be extended to the more general power cones.

C Experiment Details

C.1 Common experiment settings

All experiments are conducted on the Phoenix cluster [PAC17] with Intel Xeon Gold 6226@2.70GHz + Tesla V100 GPU nodes; each job was allocated 1 GPU, 12 CPU cores and 64GB of RAM. All ML models are formulated and trained using Flux [ISF+18]; unless specified otherwise, all (sub)gradients are computed using the auto-differentiation backend Zygote [Inn18]. All linear problems are solved with Gurobi v10 [GO18]. All nonlinear conic problems are solved with Mosek [MOS23b].

All neural network architectures considered here are fully-connected neural networks (FCNNs). Thus, a separate model is trained for each input dimension. Note that the proposed DLL methodology is applicable to graph neural network architectures, which would support arbitrary problem size. The use GNN models in the DLL context is a promising avenue for future research; a systematic comparison of the performance of GNN and FCNN architectures is, however, beyond the scope of this work.

All ML models are trained in a self-supervised fashion following the training scheme outlined in Section 4.3, and training is performed using the Adam optimizer [KB15]. The training scheme uses a patience mechanism where the learning rate η is decreased by a factor 2 if the validation loss does not improve for more than N_p epochs. The initial learning rate is $\eta = 10^{-4}$. Training is stopped if either the learning rate reaches $\eta_{\text{min}} = 10^{-7}$, or a maximum N_{max} epochs is reached. Every ML model considered in the experiments was trained in under an hour.

A limited, manual, hypertuning was performed by the authors during preliminary experiments. It was found that DLL models require very little hypertuning, if any, to achieve satisfactory performance. In contrast, DC3 was found to require very careful hypertuning, even just to ensure its numerical stability. It is also important to note that DC3 introduces multiple additional hyperparameters, such as the number of correction steps, learning rate for the correction steps, penalty coefficient for the soft penalty loss, etc. These additional hyperparameters complicate the hypertuning task, and result in additional computational needs. Given the corresponding economical and environmental costs, only limited hypertuning of DC3 was performed.

Finally, it was observed that DC3 often fail to output dual-feasible solutions, which therefore do not valid dual bounds. Therefore, to ensure a fair comparison, the dual solution produced by DC3 is fed to the dual optimal completion of DLL, thus ensuring dual feasibility and a valid dual bound. This is only performed at test time, with a negligible overhead since the dual completion uses a closed-form formula. All optimality gaps for DC3 are reported for this valid dual bound.

https://doi.org/10.52202/079017-1764

C.2 Linear programming problems

C.2.1 Problem formulation

The first set of experiments considers the continuous relaxation of multi-dimensional knapsack problems [FP94, Fre04], which are of the form

$$\min_{x} \quad \left\{ -p^\top x \;\middle|\; Wx \leq b, x \in [0,1]^n \right\}, \tag{51}$$

where n denotes the number of items, m denotes the number of resources, $p \in \mathbb{R}^n_+$ is the value of each item, b_i is the amount of resource i, and W_{ij} denotes the amount of resource i used by item j. The dual problem reads

$$\max_{y,z^l,z^u} \quad \left\{ b^\top y - \mathbf{e}^\top z^u \;\middle|\; W^\top y + z^l - z^u = -p, y \leq 0, z^l \geq 0, z^u \geq 0, \right\} \tag{52}$$

C.2.2 Data generation

For each number of items $n \in \{100, 200, 500\}$ and number of resources $m \in \{5, 10, 30\}$, a total of 16384 instances are generated using the same procedure as the MIPLearn library [SXQG+23]. Each instance is solved with Gurobi, and the optimal dual solution is recorded for evaluation purposes. This dataset is split in training, validation and testing sets, which contain 8192, 4096 and 4096 instances, respectively.

C.2.3 DLL implementation

The DLL architecture considered here is a fully-connected neural network (FCNN); a separate model is trained for each combination (m, n). The FCNN model takes as input the flatted problem data $(b, p, W) \in \mathbb{R}^{1+n+n\times m}$, and outputs $y \in \mathbb{R}^m$. The FCNN has two hidden layers of size $2(m+n)$ and sigmoid activation; the output layer uses a negated softplus activation to ensure $y \leq 0$. The dual completion procedure follows Example (1).

Hyperparameters The patience parameter is $N_p = 32$, and the maximum number of training epochs is $N_{\text{max}} = 1024$.

C.2.4 DC3 implementation

The DC3 architecture consists of an initial FCNN which takes as input (b, p, W), and outputs y, z^l. Then, z^u is recovered by equality completion as $z^u = p + W^\top y + z^l$. The correction then applies gradient steps $(y, z^l) \leftarrow (y, z^l) - \gamma \nabla \phi(y, z^l)$ where

$$\phi(y, z^l) = \|\max(0, y)\|^2 + \|\min(0, z^l)\|^2 + \|\min(0, z^u)\|^2$$

The corresponding gradients $\nabla\phi(y, z^l)$ were computed analytically. After applying corrections, the dual equality completion is applied one more time to recover z^u, and the final soft loss is

$$b^\top y - \mathbf{e}^\top z^u + \rho \times \phi(y, z^l) \tag{53}$$

which considers both the dual objective value, and the violation of inequality constraints. Note that the dual objective $b^\top y - \mathbf{e}^\top z^u$ is not a valid dual bound in general, because y, z^l, z^u may not be dual-feasible.

Hyperparameters The maximum number of correction steps is 10, the learning rate for correction is $\gamma = 10^{-4}$. The soft penalty weight is set to $\rho = 10$; this parameter was found to have a high impact on the numerical stability of training. The patience parameter is $N_p = 32$, and the maximum number of training epochs is $N_{\text{max}} = 1024$.

C.3 Nonlinear Production and Inventory Planning Problems

C.3.1 Problem formulation

The original presentation of the resource-constrained production and inventory planning problem [Zie82] uses the nonlinear convex formulation

$$\min_{x} \quad \sum_{j} d_j x_j + f_j \frac{1}{x_j} \tag{54a}$$

$$s.t. \quad r^\top x \leq b, \tag{54b}$$

$$x \geq 0, \tag{54c}$$

where n is the number of items to be produced, $x \in \mathbb{R}^n$, $b \in \mathbb{R}$ denotes the available resource amount, and $r_j > 0$ denotes the resource consumption rate of item j. The objective function captures production and inventory costs. Namely, $d_j = \frac{1}{2} c_j^p c_j^r$ and $f_j = c_j^o D_j$, where $c_j^p, c_j^r, c_j^o > 0$ and $D_j > 0$ denote per-unit holding cost, rate of holding cost, ordering cost, and total demand for item j, respectively.

The problem is reformulated in conic form [MOS23a] as

$$\min_{x,t} \quad d^\top x + f^\top t \tag{55a}$$

$$s.t. \quad r^\top x \leq b, \tag{55b}$$

$$(x_j, t_j, \sqrt{2}) \in \mathcal{Q}_r^3, \quad \forall j = 1, ..., n. \tag{55c}$$

whose dual problem is

$$\max_{y,\pi,\tau,\sigma} \quad by - \sqrt{2}\mathbf{e}^\top \sigma_j \tag{56a}$$

$$s.t. \quad ry + \pi = d, \tag{56b}$$

$$\tau = f, \tag{56c}$$

$$y \leq 0, \tag{56d}$$

$$(\pi_j, \tau_j, \sigma_j) \in \mathcal{Q}_r^3, \quad \forall j = 1...n. \tag{56e}$$

Note that the dual problem contains $1 + 3n$ variables, $2n$ equality constraints, 1 linear inequality constraints, and n conic inequality constraints. Therefore, DC3 must predict $n + 1$ dual variables, then recover $2n$ variables by equality completion, and correct for $n + 1$ inequality constraints. In contrast, by exploiting dual optimality conditions, DLL eliminates $3n$ dual variables, thus reducing the output dimension of the initial prediction from $n + 1$ to 1, and eliminates the need for correction.

C.3.2 Data generation

For each $n \in \{10, 20, 50, 100, 200, 500, 1000\}$, 16384 instances are generated using the procedure of [Zie82]. First, D_j is sampled from a uniform distribution $U[1, 100]$, c_j^p is sampled from $U[1, 10]$, and c_j^r is sampled from $U[0.05, 0.2]$. Then, $c_j^o = \alpha_j c_j^p$ and $r_j = \beta_j c_j^p$, where α, β are sampled from $U[0.1, 1.5]$ and $U[0.1, 2]$, respectively. Finally, the right-hand side is $b = \eta \sum_j r_j$ where η is sampled from $U[0.25, 0.75]$.

Each instance is solved with Mosek, and its solution is recorded for evaluation purposes. The dataset is split into training, validation and testing sets comprising 8192, 4096 and 4096 instances, respectively.

C.3.3 DLL implementation

The DLL architecture consists of an initial FCNN that takes as input $(d, f, r, b) \in \mathbb{R}^{1+3n}$, and output $y \in \mathbb{R}$. The FCNNs have two hidden layers of size $\max(128, 4n)$ and sigmoid activation. For the output layer, a negated softplus activation ensures $y \leq 0$. The dual completion outlined in Section 5.2 then recovers (π, σ, τ).

https://doi.org/10.52202/079017-1764

Hyperparameters The patience parameter is $N_p = 128$, and the maximum number of training epochs is $N_{\max} = 4096$. The patience mechanism is deactivated for the first 1024 epochs; this latter setting has little impact of the performance of DLL, and was introduced to avoid premature termination for DC3.

C.3.4 DC3 implementation

The DLL architecture consists of an initial FCNN that takes as input $(d, f, r, b) \in \mathbb{R}^{1+3n}$, and outputs $y, \sigma \in \mathbb{R}$. The FCNNs have two hidden layers of size $\max(128, 4n)$ and sigmoid activation, and the output layer has linear activation.

The equality completion step recovers $\pi = d - ry$ and $\tau = f$. The correction step then apply gradient steps to minimize the violations $\phi(y, \sigma) = \phi_y(y, \sigma) + \phi_\pi(y, \sigma) + \phi_\sigma(y, \sigma)$, where

$$\phi_y(y, \sigma) = \max(0, y)^2, \tag{57}$$

$$\phi_\pi(y, \sigma) = \min(0, \pi)^2, \tag{58}$$

$$\phi_\sigma(y, \sigma) = \sum_j \max(0, \sigma_j^2 - 2\pi_j \tau_j)^2. \tag{59}$$

Note that, to express $\phi_\sigma(y, \sigma)$, conic constraints (23e) are converted to their nonlinear programming equality, because DC3 does not handle conic constraints. Gradients for ϕ are computed analytically, and implemented directly in the inequality correction procedure. The final soft loss is then

$$by - \sqrt{2}\mathbf{e}^\top \sigma_j + \rho\phi(y, \sigma). \tag{60}$$

Hyperparameters The maximum number of correction steps is 10, the learning rate for correction is $\gamma = 10^{-5}$, and the soft loss penalty parameter is $\rho = 10$. The patience parameter is $N_p = 128$, and the maximum number of training epochs is $N_{\max} = 4096$. The patience mechanism is deactivated for the first 1024 epochs; this latter setting has little impact of the performance of DLL, and was introduced to avoid premature termination for DC3.

Overall, DC3 was found to experience substantial numerical instability, and failed to produce dual-feasible solutions on a majority of instances. Increasing the number of correction steps helps alleviate this issue, at the cost of more expensive inference and back-propagation. Increasing the learning rate for correction (γ) was also found to yield smaller violations, yet resulted in degraded numerical stability. Finally, increasing the number of correction steps also increases GPU memory requirements, which can further affect training performance.

C.3.5 Convergence plots

Figures 2 and 3 show the progress of the Lagrangian dual bound obtained by DLL and DC3 throughout training. The figures report the average Lagrangian dual bound on the training and validation set, as a function of the number of epochs (Figure 2) and training time (Figure 3). The difference in training times, for a same number of epochs, is explained by the longer inference and back-propagation times for DC3 (see also Table 5).

Both figures show that DLL exhibits a faster convergence, with performance plateau-ing after about 1,000 training epochs. The performance of DC3 degrades as the instances become larger (from $n = 10$ to $n = 500$): the plots exhibit a more erratic behavior, especially in the first 500 training epochs. On the smallest instances (n=10 and n=20), the behavior stabilizes after about 500 epochs, yet progress remains slow compared to DLL.

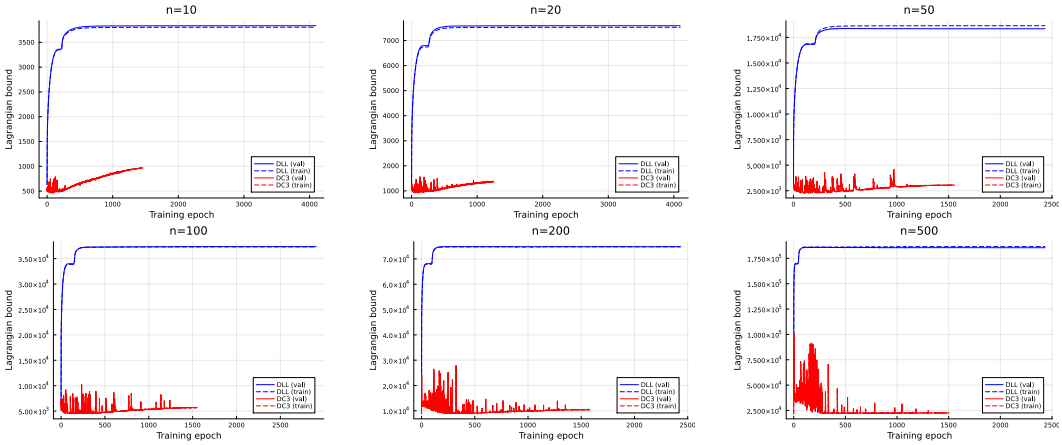

Figure 2: Production planning instances: convergence plots of average Lagrangian dual bound on training and validation sets for DLL and DC3 models, as a function of the number of training epochs.

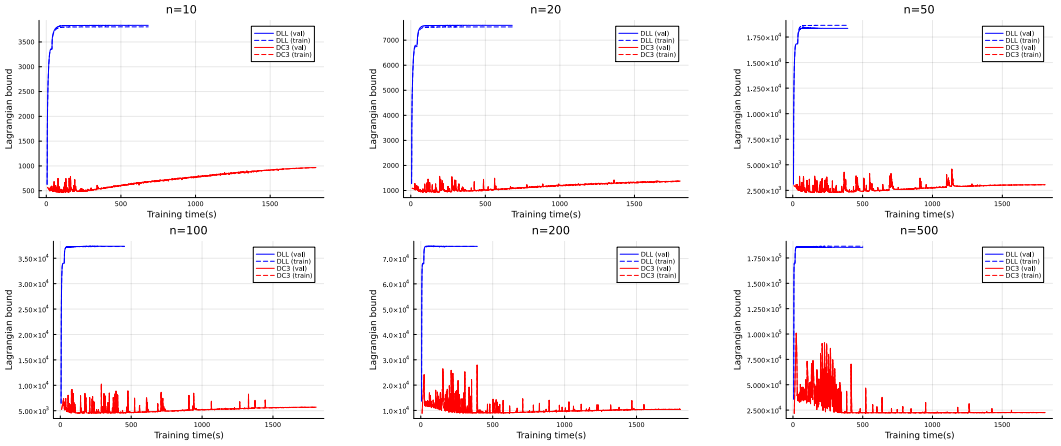

Figure 3: Production planning instances: convergence plots of average Lagrangian dual bound on training and validation sets for DLL and DC3 models, as a function of training time.

https://doi.org/10.52202/079017-1764

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract mentions the 3 building blocks of the proposed methodology, which are described in Section 4. The numerical results that are mentioned in the abstract reflect the results presented in Section 5.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Theoretical and practical limitations are discussed in Section 6.2.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Assumptions are stated in the paper and in theorem, and proofs are provided in Appendix.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Experiment details are provided in Appendix C. These include:

   - Computing resources used for experiments (CPU and GPU models)
   - Problem formulations and data-generation procedures
   - Neural architecture used in the experiments
   - Detailed training scheme
   - Hyper-parameters used for the final results

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: All the data used in the experiments is publicly available and/or synthetically generated. We have cited sources whenever using a data-generation procedure proposed elsewhere. We intend to release our code upon acceptance of the paper.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Experimental details are provided in Appendix C

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: When reporting optimality gaps in Section 5 (Tables 2 and 4), we report averages, standard deviations, and maximum across the test set. Computing times reported in Tables 3 and 5 were evaluated over multiple runs.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute resources and training time are reported in Appendix C.

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the code of ethics, and do not see any deviation to report.

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Broader impact is discussed in Section 6 (Limitations) and Section 7 (Conclusion).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Not applicable.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All prior codes / methods have been cited.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: [NA]

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: [NA]

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

https://doi.org/10.52202/079017-1764

Answer: [NA]

Justification: [NA]

References for the Appendix

[BTN01] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[BV04] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[CKV22] Chris Coey, Lea Kapelevich, and Juan Pablo Vielma. Solving natural conic formulations with hypatia.jl. *INFORMS Journal on Computing*, 34(5):2686–2699, 2022.

[FP94] Arnaud Freville and Gérard Plateau. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete Applied Mathematics*, 49(1):189–212, 1994. Special Volume Viewpoints on Optimization.

[Fre04] Arnaud Freville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.

[Fri23] Henrik A. Friberg. Projection onto the exponential cone: a univariate root-finding problem. *Optimization Methods and Software*, 38(3):457–473, 2023.

[GO18] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[Hie15] Le Thi Khanh Hien. Differential properties of euclidean projection onto power cone. *Mathematical Methods of Operations Research*, 82:265–284, 2015.

[Inn18] Michael Innes. Don't unroll adjoint: Differentiating ssa-form programs. *CoRR*, abs/1810.07951, 2018.

[ISF+18] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[MOS23a] MOSEK Aps. *The MOSEK Modeling Cookbook*, 2023.

[MOS23b] MOSEK Aps. *The MOSEK optimization toolbox for Julia manual. Version 10.1.23.*, 2023.

[NW99] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[PAC17] PACE. *Partnership for an Advanced Computing Environment (PACE)*, 2017.

[PB+14] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.

[SXQG+23] Alinson Santos Xavier, Feng Qiu, Xiaoyi Gu, Berkay Becu, and Santanu S. Dey. MIPLearn: An Extensible Framework for Learning- Enhanced Optimization, June 2023.

[Zie82] Hans Ziegler. Solving certain singly constrained convex optimization problems in production planning. *Operations Research Letters*, 1(6):246–252, 1982.