Verification of Sequential Convex Programming for Parametric Non-convex Optimization

Rajiv Sambharya, Nikolai Matni, George Pappas

University of Pennsylvania

December 1, 2025

Abstract

We introduce a verification framework to exactly verify the worst-case performance of sequential convex programming (SCP) algorithms for parametric non-convex optimization. The verification problem is formulated as an optimization problem that maximizes a performance metric (*e.g.*, the suboptimality after a given number of iterations) over parameters constrained to be in a parameter set and iterate sequences consistent with the SCP update rules. Our framework is general, extending the notion of SCP to include both conventional variants such as trust-region, convex-concave, and prox-linear methods, and algorithms that combine convex subproblems with rounding steps, as in relaxing and rounding schemes. Unlike existing analyses that may only provide local guarantees under limited conditions, our framework delivers global worst-case guarantees—quantifying how well an SCP algorithm performs across all problem instances in the specified family. Applications in control, signal processing, and operations research demonstrate that our framework provides, for the first time, global worst-case guarantees for SCP algorithms in the parametric setting.

1 Introduction

In this paper, we are interested in solving the parametric non-convex optimization problem

$$\begin{array}{ll} \text{minimize} & f(z,x) \\ \text{subject to} & z \in \Omega(x), \end{array} \tag{1}$$

where $z \in \mathbf{R}^n$ is the decision variable, $x \in \mathcal{X} \subseteq \mathbf{R}^d$ is the *problem parameter*, $\mathcal{X}$ is a known set, $\Omega(x) \subseteq \mathbf{R}^n$ is the possibly non-convex feasible region, and $f : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}$ is the possibly non-convex objective with respect to z. In many settings, we must repeatedly solve problem (1) with a varying problem parameter x. For instance, in model predictive control (Borrelli et al., 2017; Chen et al., 2022a), we repeatedly solve optimal control problems with varying initial states. In energy systems, we repeatedly solve optimal power flow problems with varying loads and renewable generation (Hentenryck, 2021; Baker, 2019). In signal processing, we repeatedly solve sparse coding problems with varying noisy measurements (Gregor and LeCun, 2010; Liu et al., 2019). In operations research, such parametric structure arises in optimization problems over networks (*e.g.*, traffic assignment, routing, or graph-based flow problems) where the underlying topology remains fixed but demands or costs vary across instances (Still, 2018).

Global approaches such as branch-and-bound (Lawler and Wood, 1966; Belotti et al., 2013) and branch-and-cut (Padberg and Rinaldi, 1991; Stubbs and Mehrotra, 1999) can, in principle, provide globally optimal solutions to problem (1). However, these methods scale poorly with problem size and are often not practical in time-sensitive settings where real-time optimization is required (Bertsimas and Stellato, 2022; Bengio et al., 2021). Instead, it is common to rely on faster, heuristic approaches such as *sequential convex programming* (Gill et al., 2005; Dinh and Diehl, 2010; Nocedal and Wright, 2006; Boyd et al., 2008) in which a sequence of convex problems is solved. Convex optimization enjoys both a rich theoretical foundation, with strong guarantees of global optimality (Boyd and Vandenberghe, 2004), and a mature ecosystem of efficient algorithms (Boyd et al., 2011; Parikh and Boyd, 2014; Ryu and Boyd, 2016; Ryu and Yin, 2022) and solvers (Goulart and Chen, 2024; Stellato et al., 2020; O'Donoghue, 2021) that make it practical for applications where real-time optimization is required.

Many methods fall under this broad category of sequential convex programming. One widely used approach is the trust-region method (Nocedal and Wright, 2006; Byrd et al., 2000), where at each iteration a local convex model is optimized within a bounded region around the current iterate. Another example is the convex-concave procedure (CCP) for difference-of-convex (DC) programming problems (Lipp and Boyd, 2016; Yuille and Rangarajan, 2001; Shen et al., 2016), where the concave part of the objective or constraints is linearized to yield a convex subproblem. For composite optimization, the prox-linear method (Drusvyatskiy and Paquette, 2019; Drusvyatskiy, 2017) is a popular approach. At each iteration, it replaces the smooth inner map with its linearization and evaluates the convex (possibly non-smooth) outer function on this approximation, thereby reducing each step to a convex subproblem. It is common to use variants of these approaches, for example, by using penalized procedures or hyperparameters (*e.g.*, the trust-region size) that vary over the iterations (Lipp and Boyd, 2016).

Relaxing and rounding schemes are another popular way to solve problems of the form (1) when the feasible set includes discrete constraints. A common example is to relax a non-convex constraint to obtain a convex problem and then round the resulting solution (Goemans and Williamson, 1995; Ageev and Sviridenko, 2004). The work of Diamond et al. (2018) systematizes this approach with the relax-round-polish method: first, a relaxation of the non-convex problem is solved, then the discrete variables are rounded, and finally the remaining continuous variables are re-optimized by solving a convex problem. Like SCP, these methods rely on solving a sequence of convex optimization problems, but they differ in that they include explicit non-convex projection steps.

While both conventional SCP methods and related heuristic methods that combine convex subproblems with rounding steps are widely used in practice, they lack guarantees on their global worst-case performance (Diamond et al., 2018; Boyd et al., 2008; Lipp and Boyd, 2016). Such limitations can be unacceptable in systems where it is essential to *verify* the global quality of the candidate solution of these algorithms over all admissible parameters in the set $\mathcal{X}$. At the same time, existing analyses typically do not take advantage of the parametric structure of problem (1). Exploiting this structure creates an opportunity to construct global guarantees in cases where none are otherwise available.

1.1 Contributions

In this paper, we develop a framework to verify the global worst-case performance of sequential convex programming methods for parametric non-convex optimization. While SCP is traditionally understood to involve only convex subproblems, we adopt a broader definition that also encompasses algorithms combining convex subproblems with simple non-convex projection steps (*e.g.*, in rounding schemes). Our verification method is meant to be solved offline (*i.e.*, in a time-insensitive setting), so that the algorithm can then be used online when the problem parameter is seen (*i.e.*, in a time-sensitive setting) with certified worst-case guarantees. In this work, we focus on the setting of non-convex quadratically-constrained quadratic programs with potentially additional discrete constraints (*i.e.*, binary or sparsity constraints). Our key contributions are as follows:

- **Verification framework for global performance guarantees.** We introduce a verification framework to certify the global worst-case performance of SCP methods for parametric non-convex optimization. In this framework, verification is posed as an optimization problem that maximizes a performance metric over problem parameters constrained to a specified set and iterate sequences consistent with the SCP update rules for a given number of iterations. We focus on three complementary aspects of performance: (i) suboptimality with respect to the optimal value of problem (1) in cases where feasibility of the candidate solution can be guaranteed, (ii) the worst-case level of constraint violation of the candidate solution when feasibility cannot be guaranteed, and (iii) feasibility of downstream convex subproblems which contain only linear constraints.

- **Applicability across sequential convex programming algorithms.** Our framework can be applied across a range of SCP algorithms including conventional variants such as trust-region, convex-concave, and prox-linear methods, and algorithms that combine convex subproblems with rounding steps such as relax-round-polish. The key idea to our approach is to encode the algorithm's steps—both convex quadratic programs (QPs) and rounding operations—as constraints in the verification problem through their optimality conditions. In the most general case, this yields a verification problem that is a mixed-integer quadratically-constrained quadratic program that can be solved using modern solvers.

- **Numerical examples.** Through a broad variety of numerical examples from control, signal processing, and operations research, we demonstrate the utility of our framework in certifying worst-case guarantees on the performance of SCP algorithms. Our approach is able to provide exact numerical guarantees where no known analytic methods are able to. Our results reveal a broad spectrum of algorithmic behaviors, from certifiable convergence to optimality to cases where the worst-case guarantee plateaus short of global optimality, and even complete failure to make progress toward finding a feasible solution beyond the first iteration. These examples also show how the framework can provide further insights to design algorithms, such as hyperparameter selection and initialization choice.

Layout of the paper. In Section 2, we review related work. In Section 3, we introduce several SCP algorithms including the trust-region method, the CCP, the prox-linear method, and the relax-round-polish method. In Section 4, we present our verification framework to obtain global worst-case performance guarantees. In Section 5, we showcase the utility of this verification framework with many numerical examples. Finally, in Section 6, we conclude.

Notation. We use $\mathbf{R}^n$ to denote the space of n-dimensional real vectors, $\mathbf{R}^n_+$ the set of n-dimensional with non-negative entries, and $\mathbf{R}^n_{++}$ the set of n-dimensional with positive real numbers. We use $\mathbf{S}^n$ to denote the space of $n \times n$-symmetric matrices, $\mathbf{S}^n_+$ the set of $n \times n$-positive semidefinite symmetric matrices, and $\mathbf{S}^n_{++}$ the set of $n \times n$-positive definite symmetric matrices. We denote the all ones vector of length m with $\mathbf{1_m}$, the all zeros vector of length m with $\mathbf{0_m}$, and the $n \times n$ identity matrix with I_n. For a set $S \subseteq \mathbf{R}^n$ and point $v \in \mathbf{R}^n$, we define the indicator function $\mathcal{I}_S(v) = 0$ if $v \in S$ and ∞ otherwise. For a vector $v \in \mathbf{R}^n$, we let $v_+ = \max(v, 0)$ element-wise.

2 Related work

Conventional sequential convex programming. Sequential convex programming (Gill et al., 2005; Dinh and Diehl, 2010; Nocedal and Wright, 2006; Boyd et al., 2008; Byrd et al., 2000; Bonalli et al., 2019) conventionally refers to an algorithmic approach that solves non-convex problems by repeatedly constructing and solving convex approximations. Classical examples include the trust-region method (Nocedal and Wright, 2006; Byrd et al., 2000; Boyd et al., 2008), the CCP for DC programming (Tao and An, 1997; Lipp and Boyd, 2016; Yuille and Rangarajan, 2001), and the prox-linear method for composite minimization (Drusvyatskiy and Paquette, 2019; Lewis and Wright, 2016). In certain cases, some of these methods come with local guarantees (Drusvyatskiy and Paquette, 2019; Bonalli et al., 2019), but in all cases, constructing tight global worst-case guarantees remains a challenge. Our framework can exactly certify the worst-case performance of such SCP methods in the parametric setting of (1).

Relaxation and rounding methods. Relaxation and rounding schemes are widely used to tackle non-convex problems by replacing difficult discrete constraints with relaxed convex constraints, followed by either deterministic (Ageev and Sviridenko, 2004; Nannicini and Belotti, 2012) or randomized (Goemans and Williamson, 1995; Raghavan and Tompson, 1987) rounding to produce candidate solutions. A systematic formulation of this approach for deterministic rounding is the relax-round-polish framework (Diamond et al., 2018), which combines a convex relaxation, a non-convex rounding step, and a subsequent polishing step that re-optimizes the continuous, convex variables. Such methods often yield strong empirical performance, but they lack worst-case guarantees on suboptimality and feasibility (Diamond et al., 2018). While SCP is traditionally understood to involve only convex subproblems, we adopt a broader definition that also encompasses algorithms combining convex subproblems with rounding steps (*e.g.*, in relax-round-polish). Our framework also provides a way to exactly certify the worst-case performance of these methods.

Guarantees in parametric optimization. Recently, several works have studied tight performance bounds for algorithms that solve parametric optimization problems. The works of Ranjan et al. (2024); Ranjan and Stellato (2025) are most similar to ours: they verify the worst-case performance of first-order methods for parametric convex optimization problems by encoding the algorithm steps as constraints in the verification problem. In contrast, our work verifies the performance of SCP algorithms for parametric non-convex optimization problems. Another key distinction is that in our setting, the algorithmic steps are defined *implicitly* through convex subproblems rather than through explicit update rules. Moreover, while global worst-case guarantees exist in convex optimization, no such guarantees are available in our non-convex setting.

Other methods provide *probabilistic guarantees* on the performance of algorithms to solve parametric optimization problems (Sambharya and Stellato, 2025; Huang et al., 2025); however, these methods rely on the assumption that the problem parameters are independently and identically distributed (i.i.d.). This assumption is often hard to verify and may not be realistic in many scenarios (*e.g.*, when problems are solved sequentially as in control applications (Borrelli et al., 2017)). Moreover, these guarantees only hold with high-probability—meaning that performance can still be arbitrarily poor even if it occurs rarely.

Finite-iteration guarantees in learning to optimize. An adjacent research area, learning to optimize (L2O) (Chen et al., 2022b; Amos, 2023; Bengio et al., 2021), focuses on learning to accelerate algorithms for parametric optimization. L2O methods have been applied to a range of non-convex settings—for example, learning branching rules or the discrete variables for mixed-integer optimization (Bengio et al., 2021; Khalil et al., 2016; Bertsimas and Stellato, 2022; Cauligi et al., 2021), and learning warm starts for SCP (Banerjee et al., 2020)—where they often deliver impressive empirical results. A central challenge in L2O, however, is guaranteeing performance after a finite number of iterations of an algorithm (Chen et al., 2022b; Amos, 2023). One line of work addresses this gap with probabilistic generalization bounds, showing that a learned optimizer likely performs well on unseen problem instances (Sambharya and Stellato, 2024; Sucker et al., 2024; Sambharya et al., 2024; Saravanos et al., 2025; Sambharya and Stellato, 2025; Balcan et al., 2021). Another line develops worst-case, finite-iteration guarantees, but these results hold only for convex problems (Sambharya et al., 2025; Banert et al., 2024; Martin et al., 2025). Although our work also considers the parametric setting, it differs fundamentally in that we do not learn an optimizer. Instead, we exactly verify the worst-case finite-iteration performance of SCP methods for parametric non-convex optimization—providing guarantees in a regime where existing L2O worst-case bounds do not apply.

Neural network verification. Our approach is closely related in spirit to neural network verification, which seeks to certify that for every admissible input (*e.g.*, an image within a bounded perturbation set), the network's output satisfies a desired property (*e.g.*, correct classification) (Tjeng et al., 2017; Ceccon et al., 2022; Anderson et al., 2020). The key similarity between the two lies in formulating verification as an optimization problem that searches for the most adversarial instance while encoding the computation steps (either the forward pass of a neural network or an SCP algorithm) as constraints.

3 Sequential convex programming algorithms

In this section, we introduce several SCP algorithms that fit within our verification framework. We focus on the trust-region method in Subsection 3.1, the penalized CCP in Subsection 3.2, the prox-linear method in Subsection 3.3, and the relax-round-polish method in Subsection 3.4. In each type of algorithm, we first write the non-convex optimization problem under consideration (which is a special case of problem (1)), and then present the algorithm steps. All methods can be described by the iterations $z^{k+1} \in s^k(z^k, x)$, where z^k is the k-th iterate and $s^k : \mathbf{R}^n \times \mathbf{R}^d \rightrightarrows \mathbf{R}^n$ is a (possibly multi-valued) operator. In this paper, we focus on the quite general case where the non-convex problems are mixed-integer quadratically-constrained quadratic programs. For the algorithms we consider[1], the choice of initialization z^0 and hyperparameters can significantly influence the quality of the candidate solution produced by the algorithm.

[1]except for the relax-round-polish method which has no initial point

3.1 Trust-region method

In this subsection, we consider the trust-region method to solve the problem

$$
\begin{array}{ll}
\text{minimize} & f(z,x) \\
\text{subject to} & g(z,x) \leq 0, \quad h(z,x) = 0,
\end{array}
\tag{2}
$$

where $z \in \mathbf{R}^n$ is the decision variable, $x \in \mathbf{R}^d$ is the problem parameter, the functions $f : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}$ and $g : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}^m$ may be non-convex in z, and $h : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}^p$ may be non-affine in z. The trust-region method iteratively builds local convex approximations around the current iterate and solves the resulting subproblems subject to a trust-region constraint (Boyd et al., 2008). Formally, given a current iterate z^k, we construct the trust-region subproblem

$$
\begin{array}{lll}
z^{k+1} \in s(z^k, x) = \operatorname{argmin} & \hat{f}(z,x,z^k) \\
& \text{subject to} & \hat{g}(z,x,z^k) \leq 0, \quad \hat{h}(z,x,z^k) = 0 \\
& & \|z - z^k\|_\infty \leq \rho_k,
\end{array}
\tag{3}
$$

where $\hat{f}(z,x,z^k)$ and $\hat{g}(z,x,z^k)$ are convex approximations of f and g around z^k, $\hat{h}(z,x,z^k)$ is an affine approximation of h around z^k, and $\rho_k > 0$ is the trust-region radius at the k-th iteration. The trust-region constraint prevents the next iterate from moving too far from the current iterate, ensuring that the convexified problem is a meaningful local model of the non-convex problem.

3.2 Penalized convex-concave procedure

In this subsection, we consider the penalized CCP to solve the DC program

$$
\begin{array}{ll}
\text{minimize} & f_0(z,x) - g_0(z,x) \\
\text{subject to} & f_i(z,x) - g_i(z,x) \leq 0, \quad i = 1, \ldots, m,
\end{array}
\tag{4}
$$

where $z \in \mathbf{R}^n$ is the decision variable and $f_i : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}$ and $g_i : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}$ are convex functions in z for $i = 0, \ldots, m$. Problem (4) is convex only if the g_i functions are *affine* in z. The penalized CCP involves solving a series of optimization subproblems of the form

$$
\begin{array}{lll}
z^{k+1} \in s(z^k, x) = \operatorname{argmin} & f_0(z,x) - \hat{g}_0(z,x,z^k) + \tau_k \mathbf{1}_m^T s \\
& \text{subject to} & f_i(z,x) - \hat{g}_i(z,x,z^k) \leq s_i, \quad i = 1, \ldots, m, \\
& & s \geq 0,
\end{array}
\tag{5}
$$

where $z \in \mathbf{R}^n$ (the original decision variable) and $s \in \mathbf{R}^m$ (introduced slack variables) are the decision variables, and $\tau_k > 0$ is a hyperparameter that weights the constraint violations at the k-th iteration. At each iterate z^k, the non-affine functions are replaced with their local affine approximations $\hat{g}_i(z,x,z^k) = g_i(z,x) + \nabla_z g_i(z^k,x)^T(z - z^k)$ for $i = 0, \ldots, m$. By introducing slacks and penalizing their violation in the objective, the penalized CCP ensures that each subproblem remains feasible while still driving the iterates toward satisfaction of the original non-convex constraints.

3.3 Prox-linear method

In this subsection, we focus on the prox-linear method, an algorithm that is specialized to minimize composite functions of the form

$$\text{minimize} \quad g(z,x) + h(c(z,x),x), \tag{6}$$

where $g : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R} \cup \{+\infty\}$ is convex in z, $h : \mathbf{R}^p \times \mathbf{R}^d \to \mathbf{R}$ is convex in z, and $c : \mathbf{R}^n \times \mathbf{R}^d \to \mathbf{R}^p$ is smooth with respect to z. The prox-linear method (Drusvyatskiy, 2017; Drusvyatskiy and Paquette, 2019) consists of solving the following convex subproblem at each iteration:

$$z^{k+1} \in s(z^k, x) = \ \text{argmin} \quad g(z,x) + h(c(z^k,x) + \nabla c(z^k,x)^T(z - z^k)) + \tfrac{\rho}{2}\|z - z^k\|_2^2, \tag{7}$$

where z is the decision variable and $\rho > 0$ is a regularization hyperparameter. Problem (7) is convex in z since convex functions composed with affine mappings are convex (Boyd and Vandenberghe, 2004, Subsection 3.3.2). We focus on the case where h is the absolute value function and c is a non-convex quadratic function—a case which has many applications (Drusvyatskiy, 2017). Under certain assumptions, the iterates of the prox-linear method are known to converge to a point that is nearly stationary (Drusvyatskiy and Paquette, 2019).

3.4 Relax-round-polish method

We now turn to an algorithm that is not typically viewed as an SCP method, due to the presence of a non-convex projection step: the relax-round-polish algorithm (Diamond et al., 2018). We apply it to the non-convex problem

$$\begin{array}{ll} \text{minimize} & (1/2)w^T P_w w + c_w^T w + (1/2)v^T P_v v + c_v^T v \\ \text{subject to} & Aw + Bv \leq d, \quad v \in \mathcal{Z}, \end{array} \tag{8}$$

where the decision variable is $z = (w, v)$ with $w \in \mathbf{R}^{n_w}$ and $v \in \mathbf{R}^{n_v}$. The problem parameter is given by all of the problem data: *i.e.*, $x = (P_w, c_w, P_v, c_v, A, B, d)$. Moreover, $P_w \succeq 0$ and $P_v \succeq 0$, so problem (8) is a convex problem except for the non-convex constraint $v \in \mathcal{Z}$. In this paper, we focus on two non-convex sets:

- Binary constraints: $\mathcal{Z} = \{0,1\}^{n_v}$.
- Sparsity constraints: $\mathcal{Z} = \{v \in \mathbf{R}^{n_v} \mid \|v\|_0 \leq k\}$ for a given integer k.

The relax-round-polish method consists of the following three steps.

- **Relax**: Solve a convex relaxation of problem (8). For binary constraints, we replace $v \in \{0,1\}^{n_v}$ with $v \in [0,1]^{n_v}$. For sparsity constraints, we soften the condition by solving problem (8) without the non-convex constraint but with an added penalty of $\lambda\|v\|_1$ for a chosen hyperparameter $\lambda > 0$.
- **Round**: Take the solution of the relaxed problem and project the variable v to the non-convex set. For binary variables, this means rounding each entry of v to either 0 or 1. For sparsity constraints, this means keeping the k entries of largest magnitude and setting the other entries to 0.

- **Polish**: With the discrete variables v fixed, solve the resulting convex subproblem in the decision variable w. This step sharpens the continuous variables w while enforcing consistency with the discrete variables v.

Each of these three steps can be written in the form of the iterations $z^{k+1} \in s^k(z^k, x)$, where each operator s^k is different. In this case of relax-round-polish, there is no need for an initialization z^0. While the relax and round steps are feasible optimization problems, there is no guarantee that the polish convex subproblem is feasible (Diamond et al., 2018).

4 Verification framework

In this section, we introduce our verification framework for certifying the global worst-case performance of SCP methods for parametric non-convex optimization. In Subsection 4.1, we formulate the verification problem as an optimization problem that maximizes a performance metric subject to constraints that i) encode the SCP steps, ii) enforce the parameter is in a set $\mathcal{X}$, iii) enforce that the initial point is in a set S, and iv) impose optimality on the true solution to problem (1). In Subsection 4.2, we show how to encode the algorithm steps (the convex QPs and rounding steps) as constraints in the verification problem. Then in the next three subsections, we focus on three specific cases: verifying suboptimality when feasibility of the final iterate can be guaranteed in Subsection 4.3, verifying the level of constraint violation when feasibility cannot be guaranteed in Subsection 4.4, and verifying the feasibility of convex subproblems with linear constraints in Subsection 4.5. Finally, in Subsection 4.6, we show how to incorporate inexact solves to the convex subproblems into our framework.

4.1 Formulating the verification problem

Before presenting the verification formulations, we detail the algorithmic updates, initialization sets, and performance metrics that together specify how the algorithms are run and evaluated.

Algorithm steps. All of the algorithms outlined in Section 3 can be represented by the iterations

$$z^{k+1} \in s^k(z^k, x), \quad k = 0, \dots, K-1,$$

where $s^k : \mathbf{R}^n \times \mathbf{R}^d \rightrightarrows \mathbf{R}^n$ is a (possibly multi-valued) operator and K is the number of iterations. The number of steps K is either dictated by i) the prescribed number of steps within an algorithm (*e.g.*, exactly three in relax-round-polish) or ii) the number of iterations one has time to run (*e.g.*, in the case of trust-region and prox-linear methods). The operators s^k need not be identical across iterations: different steps in an algorithm may correspond to solving different convex subproblems or applying different rounding rules. Moreover, we model each s^k as a multi-valued operator since all of the algorithm steps we consider may have more than one solutions. Throughout, we assume that each subproblem admits at least one solution; issues of subproblem infeasibility are treated separately later in Subsection 4.5.

Initial points. Most of the algorithms we consider require an initial point z^0 which can significantly affect the quality of the resulting candidate solution (Boyd et al., 2008). We model the initialization by assuming that z^0 belongs to some known non-empty set S. For cold-started algorithms, the set of initial points is $S = \{\mathbf{0}_n\}$. When a heuristic warm start $z^{\text{ws}} \in \mathbf{R}^n$ is available, we simply take $S = \{z^{\text{ws}}\}$. For algorithms whose first step does not depend on z^0 (*e.g.*, relax-round-polish), the specification of S is omitted.

Performance metrics. We are interested in analyzing the worst-case performance in terms of some performance metric $\phi(z^K, z^\star, x)$ where z^K is a candidate solution and $z^\star$ is the optimal solution to the original non-convex problem. We focus on three key aspects of performance in our verification framework: suboptimality, the level constraint violation, and subproblem feasibility.

- **Suboptimality.** To provide guarantees on suboptimality, we let

$$\phi(z^K, z^\star, x) = f(z^K, x) - f(z^\star, x), \tag{9}$$

where $f(z, x)$ is the objective function of the original non-convex problem (1), and $z^\star$ is the solution parametrized by parameter x. In this case, it is necessary that the final iterate z^K be feasible, which is the case in many scenarios (see Subsection 4.3 for details).

- **Level of constraint violation.** Given constraints of the form $\Omega(x) = \{z \mid g(z, x) \leq 0\}$, the square of the ℓ_2-norm of violations is quantified as

$$\phi(z^K, z^\star, x) = \sum_i (\max\{g_i(z^K, x), 0\})^2. \tag{10}$$

We could easily adapt this metric to other norms, such as the ℓ_1-norm or the ℓ_∞-norm of violations. Moreover, the metric (10) could be adapted to handle equality constraints.

- **Subproblem feasibility.** Sometimes, we cannot guarantee that the convex subproblems are feasible. For example, in the relax-round-polish method, once the discrete variables are fixed, there is no guarantee that there exists continuous variables that can satisfy the constraints. We show how to use the Farkas Lemma in Subsection 4.5 to check feasibility of downstream convex QPs.

The verification problem. Using the algorithm steps, initial points, and performance metrics from above, we formulate the verification problem as

$$\begin{array}{lll} \text{maximize} & \text{(performance metric)} & \phi(z^K, z^\star, x) \\ \text{subject to} & \text{(algorithm steps)} & z^{k+1} \in s^k(z^k, x) \quad k = 0, \dots, K-1 \\ & \text{(parameter)} & x \in \mathcal{X} \\ & \text{(initial point)} & z^0 \in S \\ & \text{(optimality)} & z^\star \in \operatorname{argmin}_{z \in \Omega(x)} f(z, x). \end{array} \tag{11}$$

We let δ denote the optimal value of problem (11). In order for verification problem (11) to be well-posed, we rely on the following three assumptions.

Assumption 1 (Existence of optimal solution)**.** *For all parameters $x \in \mathcal{X}$, there exists a minimizer $z^\star(x)$ for problem* (1)*.*

Assumption 2 (Non-empty sets)**.** *The parameter set $\mathcal{X}$ and initial point set S are both non-empty.*

Assumption 3 (Well-posed subproblems)**.** *For every parameter $x \in \mathcal{X}$ and initialization $z^0 \in S$, each update subproblem that defines z^{k+1} is well-posed. That is, whenever the algorithm reaches iteration k with iterate z^k, the corresponding update operator satisfies $s^k(z^k, x) \neq \emptyset$ for all $k = 0, \ldots, K-1$.*

Assumption 1 is mild and amounts to requiring that the parametric problem (1) always admits at least one minimizer. Assumption 2 is also mild, requiring that there is at least one feasible parameter and initial point. Throughout the paper, we assume that these two assumptions hold.

Assumption 3 requires more care. In many algorithmic settings this condition is natural—*e.g.*, in the trust-region method where the constraints are convex and every admissible initial point is feasible for all parameters $x \in \mathcal{X}$. However, there are also important situations in which the assumption may fail, such as when downstream convex subproblems become infeasible (*e.g.*, the final polish step in relax-round-polish). We remark that this assumption concerns only iterates generated by the algorithm; it does not impose feasibility of the k-th subproblem (corresponding to $s^k(z^k, x)$) for arbitrary z^k, but only for those z^k that arise during a valid run of the method.

Efficiently encoding the algorithms steps and optimality conditions in problem (11) is a challenge since the steps are *implicitly* defined by optimization subproblems. Moreover, the presence of tie-breakers introduces pessimism: the verification framework must account for the worst-case outcome of any tie when representing the algorithm's behavior. In Subsection 4.2 we show how to encode algorithm steps via their optimality conditions. To verify suboptimality, we show how to handle the optimality condition in problem (11) in Subsection 4.3.

4.2 Encoding algorithm steps

In this subsection, we show how to encode the algorithm steps in SCP methods as constraints in the verification problem (11). These building blocks are the components that we use to construct the verification problem for various algorithms. We show how to encode convex QPs in Subsection 4.2.1 and projections onto some non-convex sets in Subsection 4.2.2.

4.2.1 Convex quadratic problems

The main SCP steps we consider are convex QPs that take the primal and dual formulations of the form

$$\begin{array}{llcll} \text{minimize} & (1/2)u^T P u + c^T u & \qquad & \text{maximize} & -(1/2)u^T P u - b^T y \\ \text{subject to} & Au + s = b & & \text{subject to} & Pu + A^T y + c = 0 \\ & s \geq 0 & & & y \geq 0, \end{array} \tag{12}$$

where the primal variables are $u \in \mathbf{R}^{n_u}$ and $s \in \mathbf{R}^{n_s}$ and the dual variable is $y \in \mathbf{R}^{n_y}$. The Karush-Kuhn-Tucker (KKT) conditions of problem (12) are (O'Donoghue, 2021)

$$Au + s = b, \quad s \geq 0, \quad Pu + A^T y + c = 0, \quad y \geq 0, \quad s^T y = 0.$$

Since problem (12) is convex, the KKT conditions are always sufficient for optimality, and are necessary if Slater's condition holds (Boyd and Vandenberghe, 2004, Section 5.5.3). Since we assume that each QP is feasible (under Assumption 3) and the feasible region is polyhedral, Slater's condition is always satisfied (Boyd and Vandenberghe, 2004, Section 5.2.3). Hence, the KKT conditions are necessary and sufficient for optimality.

As is standard in the bilevel programming literature (Sinha et al., 2018), we avoid writing nested optimization problems in the verification problem (11) explicitly by reformulating the QPs through their KKT conditions. In the most general case of the verification problem, the iterates $\{z^k\}_{k=0}^{K-1}$, problem parameter x, and the problem data of each QP (P, A, c, b) (which depends on the parameter x and the iterate z^k) all serve as decision variables. The resulting KKT constraints introduce non-convexities (*e.g.*, the complementarity constraint $s^T y = 0$ is a non-convex bilinear constraint).

4.2.2 Projections onto non-convex sets

We now consider the case where the algorithm step is a projection onto a non-convex set $\mathcal{Z} \subseteq \mathbf{R}^n$ for rounding steps outlined in Subsection 3.4. We denote this projection with the operator Π.

Binary variables. To formalize how the projection step onto the set $\{0,1\}^n$ can be enforced within our framework, we state the following proposition.

Proposition 4 (Encoding binary variables)**.** *Let $\mathcal{Z} = \{0,1\}^n$ and suppose $u \in [0,1]^n$. Then*

$$v = \Pi(u) \iff v \in \{0,1\}^n, \quad v - u \leq 1/2, \quad u - v \leq 1/2.$$

See Appendix Subsection A.1 for the proof.

Sparsity constraints. To formalize how the projection step onto the set $\{z \in \mathbf{R}^n \mid \|z\|_0 \leq k\}$ can be enforced within our framework, we state the following proposition.

Proposition 5 (Encoding sparsity constraints)**.** *Let $\mathcal{Z} = \{z \in \mathbf{R}^n \mid \|z\|_0 \leq k\}$ and suppose $u \in \mathbf{R}^n$ with $w = |u|$. Then the projection step $v = \Pi(u)$, which keeps the k entries of largest magnitude and sets the rest to zero, can be equivalently encoded as*

$$\begin{aligned} \exists\, \alpha \in \{0,1\}^n,\ \tau \geq 0 \ \ s.t.\ \ \mathbf{1}^T\alpha = k, \quad \alpha_i = 0 \implies w_i \leq \tau\ \forall i, \quad \alpha_i = 1 \implies w_i \geq \tau\ \forall i, \\ \alpha_i = 0 \implies v_i = 0\ \forall i, \quad \alpha_i = 1 \implies v_i = u_i,\ \forall i. \end{aligned}$$

See Appendix Subsection A.2 for the proof. To model the logical implication constraints, we use the standard big-M technique (Vielma, 2015), which is often supported in modern solvers (Gurobi Optimization, 2025). We assume access to the absolute value of the vector u since that variable is a natural outcome of the penalized version of the previous problem (in the relax-round-polish method).[2]

4.3 Verifying suboptimality

In this subsection, we show how to verify the suboptimality of the candidate solution in cases where feasibility of the final iterate can be guaranteed. In this subsection, we assume that the following assumption holds.

Assumption 6 (Feasibility of final iterate)**.** *For every problem parameter $x \in \mathcal{X}$, the SCP algorithm produces a final iterate z^K that lies in the feasible set $\Omega(x)$.*

[2] If the absolute value is unavailable, the sparsity constraint can still easily be encoded.

There are many settings where Assumption 6 is satisfied.

- **Unconstrained settings:** $\Omega(x) = \mathbf{R}^n$, so feasibility holds automatically.
- **Convex feasible regions with feasibility-preserving steps:** If $\Omega(x)$ is convex for all $x \in \mathcal{X}$, the initial point is feasible for all parameters $x \in \mathcal{X}$, and each step preserves feasibility, then the final iterate remains feasible.
- **Non-convex feasible regions with explicit enforcement:** In some cases of simpler non-convex feasible regions, feasibility is ensured by the algorithm itself—for example, when $\Omega(x) = \{0,1\}^n$ and the final step is a rounding operation.
- **Verified feasibility with our constraint satisfaction framework:** In some cases, none of the above three conditions may hold. However, we can use the framework from Subsection 4.4 to first verify that the final iterate is always feasible after a given number of iterations. If feasibility can be verified, we can then solve a second verification problem for the worst-case suboptimality.

The primary difficulty in solving problem (11) for worst-case suboptimality lies in enforcing the optimality condition. In Subsection 4.2.1 we showed how to encode the minimizer of a convex QP into the verification problem by directly writing the KKT conditions as constraints. However, we cannot rely on this reformulation for the optimality condition in problem (11) since problem (1) is non-convex. This means that i) the KKT conditions only encode stationary points for non-convex problems and ii) in order for the KKT conditions to be necessary for even a stationary point, additional constraint qualifications (like the linear independent constraint qualification) (Nocedal and Wright, 2006, Chapter 12) must be satisfied.

It turns out that in this case where the performance metric is the suboptimality, *we do not need to encode the optimality constraint.* Instead, we relax it to a feasible constraint and formulate the problem

$$\begin{array}{lll} \text{maximize} & \text{(performance metric)} & f(z^K, x) - f(z^\star, x) \\ \text{subject to} & \text{(algorithm steps)} & z^{k+1} \in s^k(z^k, x) \quad k = 0, \dots, K-1 \\ & \text{(parameter)} & x \in \mathcal{X} \\ & \text{(initial point)} & z^0 \in S \\ & \text{(feasibility)} & z^\star \in \Omega(x). \end{array} \tag{13}$$

Let $\bar{\delta}$ denote the optimal value of problem (13). Observe that problem (11) with objective $\phi(z^K, z^\star, x) = f(z^K, x) - f(z^\star, x)$ and problem (13) are the same except that the optimality constraint in (11) is replaced with a feasibility constraint in (13). The following theorem shows that *the relaxation from problem* (11) *to problem* (13) *is in fact tight.*

Theorem 7. *Under Assumptions 1, 2, 3 and 6, and with the performance metric defined as in Equation* (9)*,* $\phi(z^K, z^\star, x) = f(z^K, x) - f(z^\star, x)$*, we obtain* $\delta = \bar{\delta}$*. That is, the optimal value of problem* (11) *is equal to the optimal value of problem* (13)*.*

See Appendix Subsection A.3 for the proof. Intuitively, the relaxation of the optimality constraint to a feasibility constraint is tight because maximizing the suboptimality *forces* decision variable $z^\star$ to minimize $f(z^\star, x)$ for parameter x. The tightness of the relaxation from Theorem 7 plays a pivotal role—it is what renders the verification of suboptimality doable, as it is not obvious how one would encode the optimality condition directly in problem (11). The approach outlined in this subsection makes it possible to obtain global worst-case guarantees on suboptimality in cases where feasibility can be certified.

4.4 Verifying the level of constraint violation

In the previous subsection, we assumed that the feasibility of the candidate solution z^K could be guaranteed; yet in some cases this assumption is not realistic. In this subsection, we discuss how to measure the worst-case constraint violation (as measured by the square of the ℓ_2-norm of the violations) in such cases. Assuming that the feasible region $\Omega(x)$ is defined by inequality constraints $g_i(z, x) \leq 0$, $i = 1, \ldots, m$, we can verify the worst-case constraint violation of the candidate solution z^K by solving the verification problem

$$\begin{array}{lll} \text{maximize} & \text{(performance metric)} & \sum_i (\max\{g_i(z^K, x), 0\})^2 \\ \text{subject to} & \text{(algorithm steps)} & z^{k+1} \in s^k(z^k, x) \quad k = 0, \ldots, K-1 \\ & \text{(parameter)} & x \in \mathcal{X} \\ & \text{(initial point)} & z^0 \in S. \end{array} \tag{14}$$

If the optimal value of problem (14) is zero, then the candidate solution z^K is guaranteed to be feasible for any admissible parameter $x \in \mathcal{X}$. If the optimal value is positive, then there exists a parameter $x \in \mathcal{X}$ for which the candidate solution z^K is infeasible, and the optimal value quantifies the corresponding level of constraint violation. In problem (14) we penalize the square of the ℓ_2-norm of violations, but we could easily adjust it to penalize other metrics (*e.g.*, the ℓ_∞ or ℓ_1 norms) of violations.

4.5 Verifying feasibility of subproblems with linear constraints

Convex subproblems arising within the SCP procedure may, in some cases, be infeasible. In this subsection, we show how our verification framework can certify whether feasibility of a convex QP at iteration $K-1$ is guaranteed for all admissible parameters and iterates consistent with the SCP update rules. Suppose that the convex QP at the final iteration has constraints $A^{K-1}u \leq b^{K-1}$ where the problem data A^{K-1} and b^{K-1} are functions of the previous iterate z^{K-1} and parameter x: *i.e.*, $(A^{K-1}, b^{K-1}) = \psi^{K-1}(z^{K-1}, x)$. Feasibility of these linear constraints can be characterized using the Farkas Lemma (Schrijver, 1986, Section 7.3).

Lemma 8 (Farkas Lemma). *Given $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$, exactly one of the following is true:*

- *There exists $u \in \mathbf{R}^n$ such that $Au \leq b$.*
- *There exists $y \in \mathbf{R}^m$ such that $A^T y = 0$, $y \geq 0$, and $b^T y < 0$.*

Using this characterization, we embed feasibility checking into our verification framework through the following optimization problem:

$$\begin{array}{lll} \text{maximize} & \text{(performance metric)} & -(b^{K-1})^T y^K \\ \text{subject to} & \text{(algorithm steps)} & z^{k+1} \in s^k(z^k, x) \quad k = 0, \dots, K-2 \\ & \text{(parameter)} & x \in \mathcal{X} \\ & \text{(initial point)} & z^0 \in S \\ & \text{(polyhedral data)} & (A^{K-1}, b^{K-1}) = \psi^{K-1}(z^{K-1}, x) \\ & \text{(Farkas constraints)} & y^K \geq 0, \quad (A^{K-1})^T y^K = 0, \end{array} \tag{15}$$

where $y^K \in \mathbf{R}^m$ enters as an additional decision variable. Let γ be the optimal value of problem (15).

Theorem 9. *Suppose that Assumptions 1 and 2 hold and the operators $s^0, \dots, s^{K-2}$ are well-posed (i.e., Assumption 3 holds). Then, the convex QP at iteration $K-1$ is feasible for all problem parameters $x \in \mathcal{X}$ and iterates consistent with SCP update rules if and only if $\gamma \leq 0$.*

See Appendix Subsection A.4 for the proof which uses the Farkas Lemma 8. The significance of Theorem 9 is that it enables us to certify whether or not a downstream QP is feasible for all admissible parameters and all iterates consistent with the SCP update rules. This is particularly useful for verifying feasibility of the polish subproblem in the relax-round-polish method described in Subsection 3.4.

4.6 Inexact solves

In practice, QPs are solved using iterative QP solvers (Stellato et al., 2020; O'Donoghue, 2021; Goulart and Chen, 2024), which are typically solved up to a given tolerance. In this subsection, we describe how to handle inexact solves to problem (12) in our verification framework. We consider two representative models of inexactness:

- **Distance to optimality inexactness**: A common modeling assumption is that the computed solution is within an ϵ-distance of the true subproblem solution (Nocedal and Wright, 2006, Section A.2). In this case, we include variables for the QP's optimal solution u^{exact} that exactly satisfies the KKT conditions and the inexact solution u^{inexact} with the constraint $\|u^{\text{exact}} - u^{\text{inexact}}\|_\infty \leq \epsilon$.

- **KKT inexactness**: Many QP solvers, such as SCS (O'Donoghue, 2021) and OSQP (Stellato et al., 2020), report accuracy in terms of primal and dual residuals rather than distance to the exact solution. In this case, the iterates automatically satisfy some of the optimality conditions from Subsection 4.2 (see for example, (Stellato et al., 2020, Subsection 3.3)), but not the primal residuals $\|Au + s - b\|_\infty \leq \epsilon$ and dual residuals $\|Pu + A^T y + c\|_\infty \leq \epsilon$. In this case, we replace the exact optimality conditions with these constraints.

We use the infinity norm since this is the standard convergence criterion in many QP solvers (O'Donoghue, 2021; Stellato et al., 2020; Ranjan and Stellato, 2025). These two models allow our framework to reliably capture the kinds of inexactness encountered in practice, ensuring that the resulting worst-case guarantees remain meaningful even when subproblems are not solved to full precision.

5 Numerical experiments

In this section, we showcase the utility of our framework to verify the performance of SCP methods with numerical examples. We focus on the trust-region method in Subsection 5.1, the penalized CCP in Subsection 5.2, the prox-linear method in Subsection 5.3, and the relax-round-polish algorithm in Subsection 5.4. The code to reproduce our results is available at

`https://github.com/rajivsambharya/verify_scp`.

We solve the verification problems (which in the most general case are mixed-integer quadratically-constrained quadratic programs) to a 2% optimality gap (unless otherwise indicated) using Gurobi 12.0 (Gurobi Optimization, 2025). In some examples, we use scalability enhancements described in Appendix Subsection B based on optimization-based bound tightening and solving the verification problems *sequentially*. We include all of the timing results to solve the verification problems in Appendix Section C. Our results exhibit a wide range of algorithmic behaviors—from provable convergence to a globally optimal point, to scenarios where significant progress is made but plateaus short of optimality, and cases where the algorithm immediately stalls and fails to progress. In all examples, Assumptions 1, 2, and 3 which ensure the existence of an optimal solution for every admissible parameter, the parameter sets and initial point sets are non-empty, and the SCP update rules are well-posed, are satisfied.

Baselines. To the best of our knowledge, no general baseline methods exist to generate worst-case guarantees for these SCP algorithms. For validation, in all experiments we report the *sample maximum*, obtained by uniformly sampling 500 parameter instances from the set $\mathcal{X}$ (unless stated otherwise) and taking the worst-case value. We formulate the convex QPs in CVXPY (Diamond and Boyd, 2016; Agrawal et al., 2018) and solve them using OSQP (Stellato et al., 2020).

5.1 Trust-region method

In this subsection, we apply our verification framework to the trust-region method from Subsection 3.1, on box-constrained quadratic minimization in Subsection 5.1.1 and network utility maximization in Subsection 5.1.2.

5.1.1 Box-constrained quadratic minimization

We first consider the non-convex box-constrained quadratic optimization problem

$$\begin{array}{ll} \text{minimize} & (1/2)z^T P z + x^T z \\ \text{subject to} & -1 \leq z \leq 1, \end{array} \tag{16}$$

where $z \in \mathbf{R}^n$ is the decision variable, $x \in \mathbf{R}^n$ is the problem parameter, and the matrix $P \in \mathbf{S}^n$ which has both positive and negative eigenvalues is shared across all problem instances.

Numerical example. We take $P = \bar{P} + \bar{P}^T$, where the entries of $\bar{P}$ are generated by sampling from a standard Gaussian distribution in an i.i.d. fashion. In this example, we compare against two different sets for the parameters: $\mathcal{X}_1 = [2, 4]^n$ and $\mathcal{X}_2 = [5, 8]^n$. We fix the trust-region size to be $\rho = 0.2$ and solve the verification problem with a cold start $S = \{\mathbf{0}_n\}$ and with a warm start

$S = \{s^{\text{ws}}\}$. To calculate the warm start point s^{ws}, we solve the non-convex problem (16) with x fixed to the point at the center of the parameter set (*e.g.*, for $\mathcal{X}_2$, we fix $x = (6.5)\mathbf{1}_n$). Assumption 6 is satisfied since the initial point is always feasible and the SCP steps preserve feasibility. We take $n = 10$ in this example. We use only 10 samples to compute the sample maximum, as using more samples causes the curves to overlap and become difficult to distinguish.

Results. We illustrate our results in Figure 1. In parameter set $\mathcal{X}_1$, the cold-started initialization is certifiably optimal, while the warm-started initialization is not; in parameter set $\mathcal{X}_2$, the reverse holds. This shows that guarantees are highly dependent on both the initialization and the parameter set.

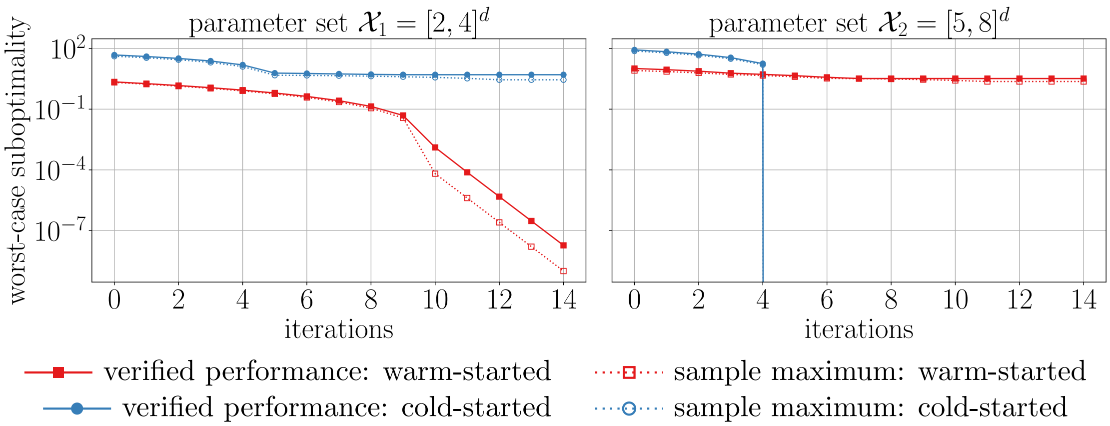

Figure 1: Box QP results. **Left**: Results for $\mathcal{X}_1 = [2,4]^d$. The warm-started initialization is guaranteed to achieve global optimality after 14 iterations (within tolerance 10^{-7}), but the cold-started initialization is not guaranteed to reach a globally optimal solution. **Right**: Results for $\mathcal{X}_2 = [5,8]^d$. The cold-started initialization is guaranteed to exactly achieve global optimality after 5 iterations (the blue curve is not visible because the suboptimality is below 10^{-15}), but the warm-started initialization is not guaranteed to reach a globally optimal solution. **Takeaway message**: It is not obvious a priori whether warm-starting yields better results than cold-starting, or whether the trust-region method reaches global optimality. Our framework provides definitive guarantees that can answer these questions by explicitly accounting for the initialization, parameter set, and the trust-region size.

5.1.2 Network utility maximization

We now turn to the resource allocation setting, modeled as a network utility maximization problem (Mattingley and Boyd, 2010). We model a communication network comprising d edges and n flows. Each edge i has capacity d_i, and each flow j carries a non-negative rate z_j. The routing structure is encoded in a binary matrix $R \in \{0,1\}^{d \times n}$, where $R_{ij} = 1$ indicates that flow j passes through edge i. The total load on edge i is then given by the sum of the rates of all flows that traverse it, compactly expressed as Rz which must satisfy the parametric capacity constraints $Rz \leq x$. We consider a variant with quadratic objective (Fazel and Chiang, 2005; Guisewite and Pardalos, 1990)

$$\begin{array}{ll} \text{maximize} & (1/2)z^T Dz + d^T z \\ \text{subject to} & Rz \leq x, \quad 0 \leq z, \end{array} \tag{17}$$

where $z \in \mathbf{R}^n$ is the decision variable, $x \in \mathbf{R}^d$ is the problem parameter, and $R \in \{0,1\}^{d \times n}$, $D \in \mathbf{S}^n_+$, and $d \in \mathbf{R}^n$ are problem data fixed for all instances. Problem (17) is non-convex since the objective function being maximized is convex.

Numerical example. We set the cost problem data to be $d = \mathbf{0}_n$ and $D = I_n$. We generate the routing matrix R as in (Ranjan and Stellato, 2025), where we select 1 with 0 for each entry i.i.d. with probability 0.5. We let $\mathcal{X} = [7,8]^d$. We use a cold start (initialized from a random point in $[0,1]^d$)[3]. This cold start is feasible for all problems with parameter $x \in \mathcal{X}$ which means that Assumption 6 is met. We take $d = 10$ and $n = 5$ as in (Ranjan and Stellato, 2025).

Results. The results are illustrated in Figure 2, where we test different trust-region size values ρ. Our framework reveals that among the tested sizes $\{1,2,5\}$, the largest trust-region size 5 provides the strongest worst-case guarantees. For all three cases, the worst-case guarantee plateaus within 5 iterations.

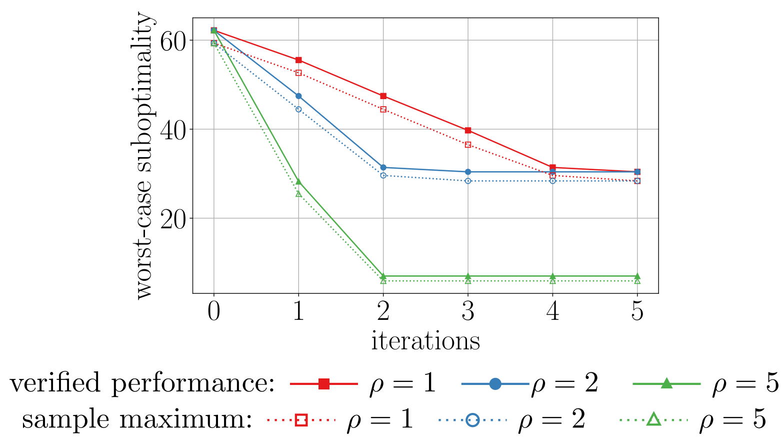

Figure 2: Network utility results. A larger trust-region size enables the trust-region method to escape poorer local minima that the variants with a smaller trust-region size converge to—an interesting outcome, since larger steps are often associated with instability rather than improved convergence.

5.2 Penalized convex-concave procedure

In this subsection, we apply our verification framework to the penalized CCP from Subsection 3.2, focusing on power converter control in Subsection 5.2.1 and the knapsack problem in Subsection 5.2.2.

[3]We cold start from a random point instead of the zero vector because the trust-region method gets stuck at the zero vector

5.2.1 Power converter control

We consider the following optimal control problem for power electronic converters, following a similar setup from (Takapoui et al., 2020):

$$\begin{array}{ll} \text{minimize} & (1/2)\sum_{t=1}^{T}(s_t - s_t^{\text{ref}})^T Q_t (s_t - s_t^{\text{ref}}) + (1/2)\sum_{t=1}^{T-1} u_t^T R_t u_t, \\ \text{subject to} & s_{t+1} = As_t + Bu_t, \quad u_t \in \{-1,1\}^{n_u}, \quad t = 0,\dots,T-1 \\ & s_0 = s^{\text{init}}, \end{array} \tag{18}$$

where the decision variables are the states $\{s_t\}_{t=0}^{T-1}$ where $s_t \in \mathbf{R}^{n_s}$ and the controls $\{u_t\}_{t=0}^{T-1}$ where $u_t \in \mathbf{R}^{n_u}$. The problem parameter is the initial state $x = s^{\text{init}}$. The dynamics matrices $A \in \mathbf{R}^{n_s \times n_s}$, $B \in \mathbf{R}^{n_s \times n_u}$, the reference trajectory $\{s_t^{\text{ref}}\}_{t=1}^{T}$, and the cost matrices $Q_t \in \mathbf{S}_+^{n_s}$ and $R_t \in \mathbf{S}_+^{n_u}$ are fixed for all instances. The control inputs represent the switching states of the converter, taking values in $\{-1,1\}$. After eliminating the state variables, problem (18) can be written in the condensed form

$$\begin{array}{ll} \text{minimize} & (1/2)u^T P u + u^T(Kx + c_0), \\ \text{subject to} & u \in \{-1,1\}^{Tn_u}, \end{array} \tag{19}$$

where the decision variable u is the stacked controls over the time indices. The matrices $P \in \mathbf{S}^{Tn_u}$ and $K \in \mathbf{R}^{Tn_u \times n_s}$ and the vector $c_0 \in \mathbf{R}^{Tn_u}$ are derived from the problem data. At the end of the penalized CCP, we round the solution to the nearest vector in the discrete set $\{-1,1\}^{Tn_u}$ to obtain a feasible point, thus satisfying Assumption 6 (see Subsection 4.2.2 for how to encode this step as constraints).

Numerical example. We generate the dynamics matrices A and B by discretizing the continuous dynamics as described in (Takapoui et al., 2020, Subsection 3.3). We have $n_s = 4$ and $n_u = 1$. We set Q_t to be the all zeroes matrix except for the bottom right entry which is 1. We set $R_t = 0$ and $s_t^{\text{ref}} = (0,0,0,1)$ for $t = 0,\dots,T$. We discretize with $\Delta t = 10^{-6}$ seconds and take $T = 10$ time steps. We set $\mathcal{X} = [-2,2]^{n_s}$ and use the hyperparameter $\tau_k = 0.2$ for the penalties in problem (5). We compare the two different inexactness criteria for terminating the penalized CCP from Subsection 4.6 for different tolerances. We use the same warm-start initialization heuristic as described in Subsection 5.1.1.

Results. We illustrate our guarantees in Figure 3. We observe a sizeable gap between the verified worst-case performance and the sample maximum for the KKT inexactness criteria, indicating that the verification framework finds particularly adversarial inexact solutions that do not naturally arise from the OSQP solver.

5.2.2 Knapsack

We now consider the knapsack problem formulated as

$$\begin{array}{ll} \text{maximize} & x^T z \\ \text{subject to} & a^T z \le b, \quad z \in \{0,1\}^n, \end{array} \tag{20}$$

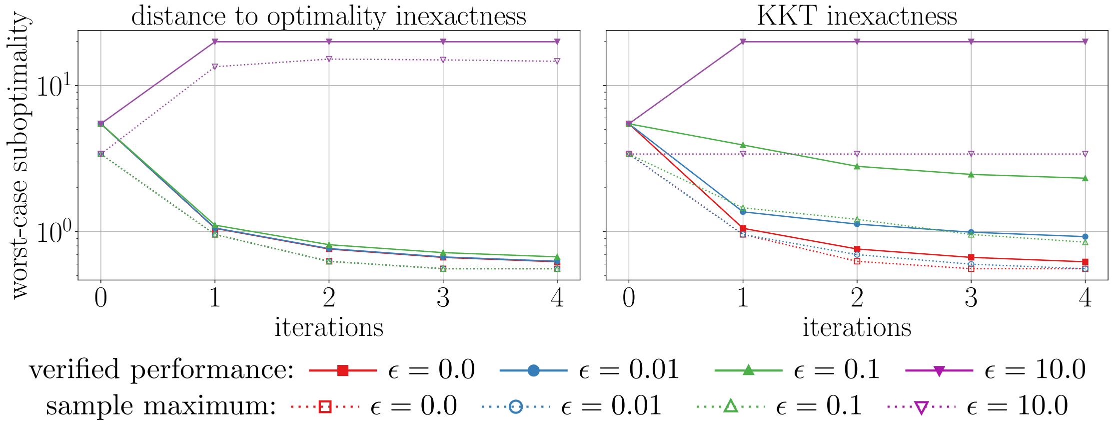

Figure 3: Power converter control results. Left: Results for distance to optimality inexactness. Right: Results for KKT inexactness. Our framework allows us to quantitatively compare these two common stopping criteria. Provided a specific ϵ value, the distance to optimality inexactness criterion yields the stronger guarantees of the two.

where $z \in \mathbf{R}^n$ is the decision variable, $x \in \mathbf{R}^n$ is the problem parameter, and $a \in \mathbf{R}^n_{++}$ and $b \in \mathbf{R}_{++}$ are problem data that are fixed for all problem instances. The goal is to select a subset of items (given by binary variables z) that maximizes the total value $x^T z$ while ensuring the total occupied space $a^T z$ does not exceed capacity b.

Numerical example. We set $n = 10$, randomly sample the vector a from a uniform distribution in $[0, 1]$ in an i.i.d. fashion, and set b to be half the sum of the entries of a. We set the penalty hyperparameters with $\tau_k = \tau_0 \kappa^k$ and test across $\tau_0 \in \{0.01, 1, 100\}$ and $\kappa \in \{1, 2\}$. We initialize from the vector $(0.5)\mathbf{1}_n$ (the middle of the set $\{0, 1\}^n$) and take the parameter set $\mathcal{X} = [5, 7]^n$.

Results. We showcase our bounds on the level of constraint violation in Figure 4. Across all choices of hyperparameters τ_0 and κ, the penalized CCP fails to make any progress beyond the first iteration, and no feasible solution can be verified in any case.

5.3 Prox-linear method

In this subsection, we apply our verification framework to the prox-linear method from Subsection 3.3 for phase retrieval in Subsection 5.3.1.

5.3.1 Phase retrieval

We consider the phase retrieval problem

$$\text{minimize} \quad (1/d) \textstyle\sum_{i=1}^d |(a_i^T z)^2 - x_i|, \tag{21}$$

where $z \in \mathbf{R}^n$ is the decision variable, $x \in \mathbf{R}^d$ is the problem parameter, and the dictionary vectors $\{a_i\}_{i=1}^d$ (where $a_i \in \mathbf{R}^n$) are fixed across all instances. Phase retrieval seeks to reconstruct a signal z from quadratic measurements, which arises in applications such as optics, imaging, and crystallography (Drusvyatskiy, 2017).

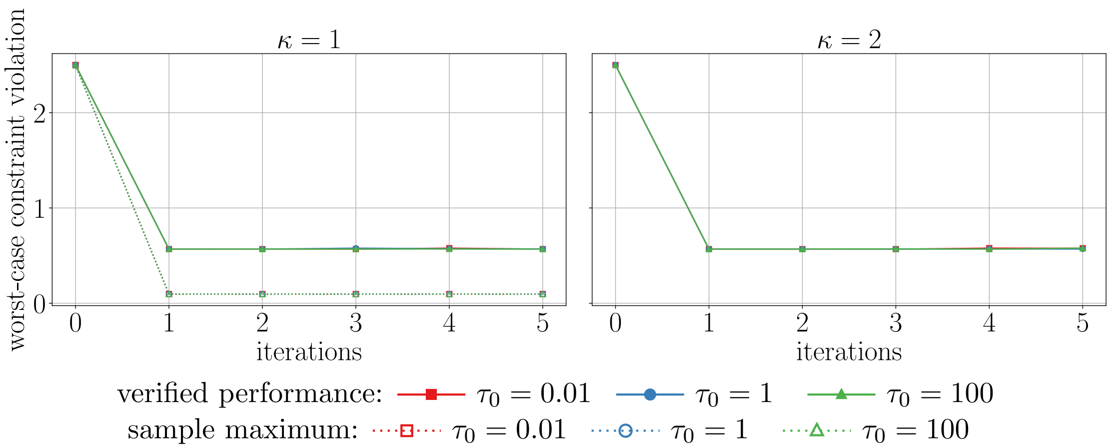

Figure 4: Knapsack results. All of the verification curves overlap with each other across all τ_0 and κ values. In all cases, the algorithm makes progress for 1 iteration and then stalls. When $\kappa = 2$, the final penalty value becomes very large (which is designed to prioritize feasibility (Lipp and Boyd, 2016, Subsection 3.1)), yet the algorithm still fails to find a feasible solution. This example shows how our framework can reveal cases in which the SCP algorithm is ineffective. Finally, for all $\kappa = 2$ settings, the verified performance matches the sample maximum exactly.

Numerical example. We consider a small phase retrieval problem where $d = 15$ and $n = 5$. We use hyperparameter $\rho = 1$ and compare against two different parameter sets: $\mathcal{X}_1 = [6, 7]^d$ and $\mathcal{X}_2 = [6.5, 7]^d$. We generate the entries of the a_i vectors i.i.d. from a standard Gaussian distribution with probability 0.25 and otherwise set it to 0. We use the same warm-start initialization heuristic as described in Subsection 5.1.1. Since problem (21) is unconstrained, Assumption 6 is met.

Results. The results for this example are illustrated in Figure 5. For the smaller parameter set, we can obtain significantly stronger worst-case guarantees.

5.4 Relax-round-polish

In this subsection, we apply our framework to verify the relax-round-polish method from Subsection 3.4 on sparse coding in Subsection 5.4.1 and hybrid vehicle control in Subsection 5.4.2.

5.4.1 Sparse coding

We consider the cardinality-constrained version of a sparse coding problem

$$\begin{array}{ll} \text{minimize} & (1/2)\|Az - x\|_2^2 \\ \text{subject to} & \|z\|_0 \leq k, \end{array} \tag{22}$$

where $z \in \mathbf{R}^n$ is the decision variable, $x \in \mathbf{R}^d$ is the problem parameter, and $A \in \mathbf{R}^{d \times n}$ is problem data that is fixed for all instances. In the relax-round-polish method, recall that our relax step consists in solving the lasso problem $\min_z (1/2)\|Az - x\|_2^2 + \lambda\|z\|_1$ with hyperparameter λ. The subsequent round step retains only the k largest entries of z in magnitude, setting all others to zero. Finally, the polish step fixes this support and solves a least-squares problem over the selected variables. We use our verification framework to upper bound the suboptimality.

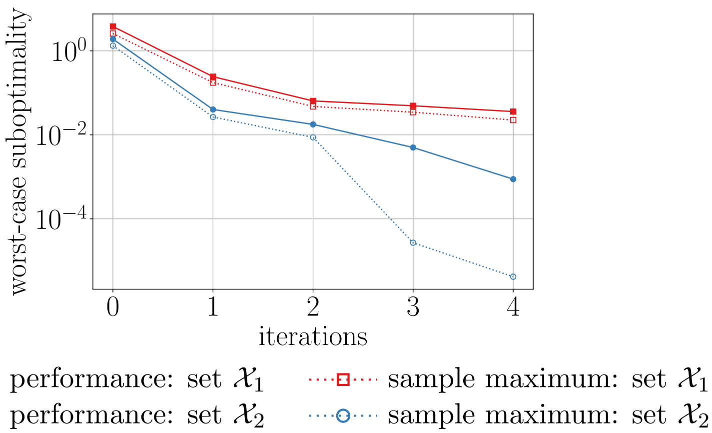

Figure 5: Phase retrieval results. With the smaller parameter set $\mathcal{X}_2$, we can certify global optimality (up to tolerance 0.001) after 4 iterations. The worst-case guarantee for the larger parameter set $\mathcal{X}_1$ is significantly worse. For $K = 4$ and $\mathcal{X}_2$, Gurobi fails to reach a 2% optimality gap within the 2 hour time limit, so we report the best upper bound obtained.

Numerical example. We follow a similar setup from (Ranjan et al., 2024) to generate a random A matrix. With probability 0.2, we sample the entries of $A \in \mathbf{R}^{d\times n}$ i.i.d. from a standard normal distribution and otherwise set each entry to 0. We then normalize the columns to have unit norm. We take the set $\mathcal{X} = [5, 8]^d$ and use $d = 20$, $n = 15$, and $k = 5$. We then solve the verification problem for a range of λ values spaced evenly on a log-scale between 0.01 and 100.

Results. Figure 6 shows the results for different λ values. The verification results indicate that a globally optimal solution is not guaranteed for any value of λ. Moreover, our framework can be used to select λ to optimize worst-case performance.

5.4.2 Hybrid vehicle control

We study a hybrid vehicle control problem that captures the task of meeting a power demand while managing battery energy, limiting engine switching, and enforcing operational constraints. Formally, we consider

$$\begin{array}{ll}
\text{minimize} & \eta(E_T - E^{\max})^2 + \sum_{t=0}^{T-1}(\alpha(P_t^{\text{eng}})^2 + \beta P_t^{\text{eng}} + \gamma z_t + c(z_{t+1} - z_t)_+) \\
\text{subject to} & E_{t+1} = E_t - \tau P_t^{\text{batt}}, \quad t = 0, \dots, T-1 \\
& P_t^{\text{batt}} + P_t^{\text{eng}} = P_t^{\text{load}}, \quad t = 0, \dots, T-1 \\
& E^{\min} \le E_t \le E^{\max}, \quad 0 \le P_t^{\text{eng}} \le z_t P^{\max} \quad t = 0, \dots, T-1 \\
& E_0 = E^{\text{init}}, \quad z_{-1} = z^{\text{prev}} \\
& z_t \in \{0, 1\}, \quad t = 0, \dots, T-1,
\end{array} \tag{23}$$

where the decision variables are $\{P_t^{\text{batt}}\}_{t=0}^{T-1}$, $\{P_t^{\text{eng}}\}_{t=0}^{T-1}$, $\{E_t\}_{t=0}^{T}$, and $\{z_t\}_{t=-1}^{T-1}$. The problem parameter is $x = (E^{\text{init}}, \{P_t^{\text{load}}\}_{t=0}^{T-1}, z^{\text{prev}}) \in \mathbf{R} \times \mathbf{R}^T \times \{0, 1\}$. The scalars α and β penalize engine use, η encourages the final energy level to be high, γ penalizes keeping the engine on, and c penalizes switching the engine on and off. The scalars $E^{\min}$, $E^{\max}$, and $P^{\max}$ give the operating limits of the

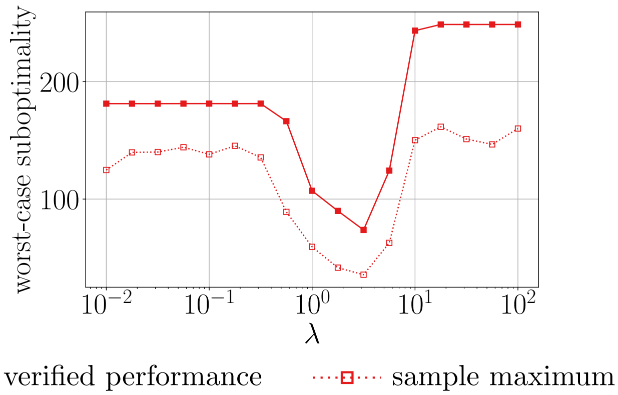

Figure 6: Sparse coding results. Among the set of hyperparameter options, $\lambda = 3.16$ yields the lowest worst-case suboptimality. We observe a sizeable gap between the verified bound and the sample maximum, which we attribute to the method's combinatorial structure. By selecting an adversarial parameter, the verification procedure can (i) induce ties in the relaxation that make multiple rounding outcomes equally valid, and (ii) break those ties in the most adverse way to maximize suboptimality. In contrast, the sampling procedure breaks such ties at random, leading to milder observed outcomes.

battery and engine. The horizon length is given by T and the sampling time by τ. In this example, we focus on certifying the feasibility of the polish step in relax-round-polish.

Numerical example. We take $\alpha = 1$, $\beta = 10$, $\gamma = 1.5$, $\eta = 2$, $c = 1$, $P^{\max} = 1$, $E^{\min} = 5$, $E^{\max} = 10$, $\tau = 2$, and $T = 5$. We take the parameter set to be $\mathcal{X} = [E^{\text{mid}} - \Delta E, E^{\text{mid}} + \Delta E] \times [P^{\text{load}}_{\text{lb}}, P^{\text{load}}_{\text{ub}}]^T \times \{0, 1\}$, where $E^{\text{mid}} = 7.5$. We conduct three different experiments with $\Delta E \in \{0, 0.5, 1\}$. Within each experiment, we conduct a grid search on $P^{\text{load}}_{\text{lb}}$ and $P^{\text{load}}_{\text{ub}}$ (for $P^{\text{load}}_{\text{lb}} \leq P^{\text{load}}_{\text{ub}}$). In this example, we solve the verification problems to the optimality gap tolerance of 10^{-9}.

Results. Figure 7 illustrates the certificates returned for different values of ΔE, $P^{\text{load}}_{\text{lb}}$, and $P^{\text{load}}_{\text{ub}}$. In a few cases, the solver hits the time limit, but feasibility can still be inferred from neighboring certificates. The plots show that our method cleanly separates feasible and infeasible regions for different parameter sets $\mathcal{X}$.

6 Conclusion

We introduce a verification framework to certify the worst-case performance of sequential convex programming methods for parametric non-convex optimization. The verification problem is formulated as an optimization problem that maximizes a performance metric over a given parameter set subject to constraints representing the SCP steps. Our framework is general and applies to many SCP algorithms and extends naturally to inexact subproblem solves. Applications in control, signal processing, and operations research demonstrate that the framework provides worst-case guarantees in settings where they are otherwise unavailable.

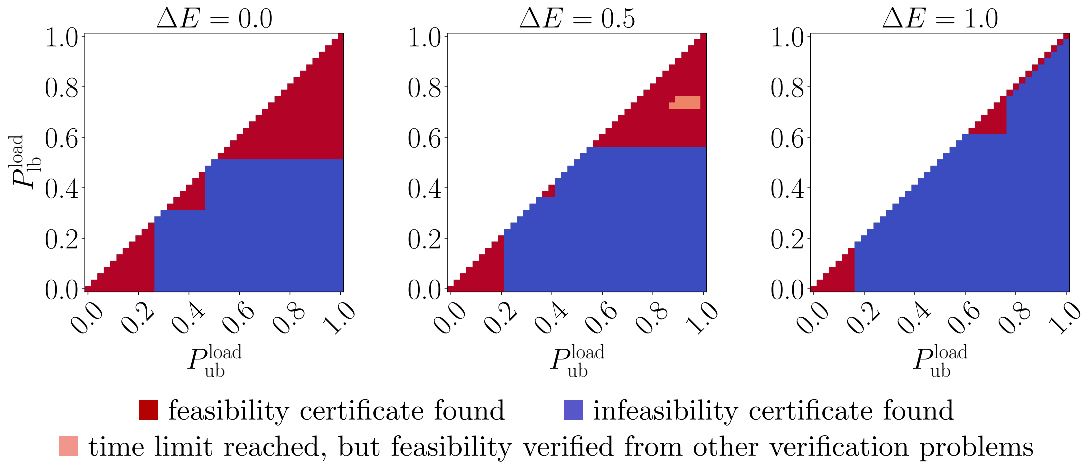

Figure 7: Hybrid vehicle results. Left: $\Delta E = 0.0$. Middle: $\Delta E = 0.5$. Right: $\Delta E = 1.0$. As we go from i) right-to-left across the three plots, ii) right-to-left within any plot, or iii) down-to-up within any plot, the size of the parameter set $\mathcal{X}$ strictly decreases, so it is strictly easier to find the feasibility certificate. We use this fact to verify feasibility in cases, where the time limit is reached.

There are several interesting directions for future work. First, it would be interesting to extend our framework to handle semidefinite program relaxations of non-convex optimization problems and low-rank rounding schemes (as done in (Goemans and Williamson, 1995)). Second, adapting the framework to analyze randomized rounding schemes would enable certification of algorithms that incorporate stochastic elements. Third, improving the scalability of the approach remains an important challenge, for example through more efficient numerical implementations or problem-specific simplifications.

Acknowledgements

NM is supported in part by NSF Award SLES-2331880, NSF CAREER award ECCS-2045834, and AFOSR Award FA9550-24-1-0102.

References

Alexander A Ageev and Maxim I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3): 307–328, 2004.

Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, 16(5):592–732, 2023.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma.

Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1):3–39, 2020.

Kyri Baker. Learning warm-start points for ac optimal power flow. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2019.

Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design. In *Proceedings of the Symposium on Theory of Computing*, pages 919–932, 2021.

Somrita Banerjee, Thomas Lew, Riccardo Bonalli, Abdulaziz Alfaadhel, Ibrahim Abdulaziz Alomar, Hesham M Shageer, and Marco Pavone. Learning-based warm-starting for fast sequential convex programming and trajectory optimization. In *2020 IEEE Aerospace Conference*, pages 1–8. IEEE, 2020.

Sebastian Banert, Jevgenija Rudzusika, Ozan Öktem, and Jonas Adler. Accelerated forward-backward optimization using deep learning. *SIAM Journal on Optimization*, 34(2):1236–1263, 2024.

Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

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.

Dimitris Bertsimas and Bartolomeo Stellato. Online Mixed-Integer Optimization in Milliseconds. *INFORMS Journal on Computing*, 34(4):2229–2248, 2022.

Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, and Marco Pavone. Gusto: Guaranteed sequential trajectory optimization via sequential convex programming. In *2019 International conference on robotics and automation (ICRA)*, pages 6741–6747. IEEE, 2019.

Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Stephen Boyd, L Xiao, A Mutapic, and J Mattingley. Sequential convex programming. *Lecture Notes, Stanford University*, 2008.

Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

Richard H. Byrd, Jean Charles Gilbert, and Jorge Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000.

Abhishek Cauligi, Preston Culbertson, Edward Schmerling, Mac Schwager, Bartolomeo Stellato, and Marco Pavone. Coco: Online mixed-integer control via supervised learning. *IEEE Robotics and Automation Letters*, 7(2):1447–1454, 2021.

Francesco Ceccon, Jordan Jalving, Joshua Haddad, Alexander Thebelt, Calvin Tsay, Carl D Laird, and Ruth Misener. Omlt: Optimization & machine learning toolkit. *Journal of Machine Learning Research*, 23(349):1–8, 2022.

Steven W. Chen, Tianyu Wang, Nikolay Atanasov, Vijay Kumar, and Manfred Morari. Large scale model predictive control with neural networks and primal active sets. *Automatica*, 135:109947, 2022a.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Zhangyang Wang, Howard Heaton, Jialin Liu, and Wotao Yin. Learning to optimize: a primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022b.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Steven Diamond, Reza Takapoui, and Stephen Boyd. A general system for heuristic minimization of convex functions over non-convex sets. *Optimization Methods and Software*, 33(1):165–193, 2018.

Quoc Tran Dinh and Moritz Diehl. Local convergence of sequential convex programming for non-convex optimization. In *Recent Advances in Optimization and its Applications in Engineering: The 14th Belgian-French-German Conference on Optimization*, pages 93–102. Springer, 2010.

Dmitriy Drusvyatskiy. The proximal point method revisited. *arXiv preprint arXiv:1712.06038*, 2017.

Dmitriy Drusvyatskiy and Courtney Paquette. Efficiency of minimizing compositions of convex functions and smooth maps. *Mathematical Programming*, 178(1):503–558, 2019.

Maryam Fazel and Mung Chiang. Network utility maximization with nonconcave utilities using sum-of-squares method. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 1867–1874. IEEE, 2005.

Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

Ambros M Gleixner, Timo Berthold, Benjamin Müller, and Stefan Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, 67(4):731–757, 2017.

Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6): 1115–1145, 1995.

Paul J Goulart and Yuwen Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives. *arXiv preprint arXiv:2405.12762*, 2024.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, Madison, WI, USA, 2010. Omnipress.

G. M. Guisewite and P. M. Pardalos. Minimum concave-cost network flow problems: Applications, complexity, and algorithms. *Annals of Operations Research*, 25(1):75–99, 1990.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2025.

Pascal Van Hentenryck. *Machine Learning for Optimal Power Flows*, chapter 3, pages 62–82. 2021.

Jingyi Huang, Paul Goulart, and Kostas Margellos. Data-driven performance guarantees for parametric optimization problems. *arXiv preprint arXiv:2506.23819*, 2025.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to Branch in Mixed Integer Programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016. Section: Technical Papers: Heuristic Search and Optimization.

Eugene L. Lawler and Donald E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

Adrian S Lewis and Stephen J Wright. A proximal method for composite minimization. *Mathematical Programming*, 158(1):501–546, 2016.

Thomas Lipp and Stephen Boyd. Variations and extension of the convex–concave procedure. *Optimization and Engineering*, 17(2):263–287, 2016.

Jialin Liu, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. ALISTA: Analytic weights are as good as learned weights in LISTA. In *International Conference on Learning Representations*, 2019.

Andrea Martin, Ian R Manchester, and Luca Furieri. Learning to optimize with guarantees: a complete characterization of linearly convergent algorithms. *arXiv preprint arXiv:2508.00775*, 2025.

Jacob Mattingley and Stephen Boyd. Automatic code generation for real-time convex optimization. In Daniel P. Palomar and Yonina C. Eldar, editors, *Convex Optimization in Signal Processing and Communications*, pages 1–41. Cambridge University Press, 2010.

Giacomo Nannicini and Pietro Belotti. Rounding-based heuristics for nonconvex MINLPs. *Mathematical Programming Computation*, 4(1):1–31, 2012.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2006.

Brendan O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31(3):1999–2023, 2021.

Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1 (3), 2014.

Prabhakar Raghavan and Clark D Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

Vinit Ranjan and Bartolomeo Stellato. Verification of first-order methods for parametric quadratic optimization. *Mathematical Programming*, pages 1–57, 2025.

Vinit Ranjan, Jisun Park, Stefano Gualandi, Andrea Lodi, and Bartolomeo Stellato. Exact verification of first-order methods via mixed-integer linear programming. *arXiv preprint arXiv:2412.11330*, 2024.

Ernest Ryu and Wotao Yin. *Large-Scale Convex Optimization: Algorithms & Analyses via Monotone Operators*. Cambridge University Press, 2022.

Ernest K. Ryu and Stephen Boyd. A primer on monotone operator methods (survey). *Applied and Computational Mathematics*, 15(1):3–43, 2016.

Rajiv Sambharya and Bartolomeo Stellato. Learning algorithm hyperparameters for fast parametric convex optimization. *arXiv preprint arXiv:2411.15717*, 2024.

Rajiv Sambharya and Bartolomeo Stellato. Data-driven performance guarantees for classical and learned optimizers. *Journal of Machine Learning Research*, 26(171):1–49, 2025.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. Learning to warm-start fixed-point optimization algorithms. *Journal of Machine Learning Research*, 25(166):1–46, 2024.

Rajiv Sambharya, Jinho Bok, Nikolai Matni, and George Pappas. Learning acceleration algorithms for fast parametric convex optimization with certified robustness. *arXiv preprint arXiv:2507.16264*, 2025.

Augustinos D Saravanos, Hunter Kuperman, Alex Oshin, Arshiya Taj Abdul, Vincent Pacelli, and Evangelos A Theodorou. Deep distributed optimization for large-scale quadratic programming. In *International Conference on Learning Representations*, 2025.

Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, UK, 1986.

Xinyue Shen, Steven Diamond, Yuantao Gu, and Stephen Boyd. Disciplined convex-concave programming. In *2016 IEEE 55th conference on decision and control (CDC)*, pages 1009–1014. IEEE, 2016.

Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2018.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

Georg Still. Lectures on parametric optimization: An introduction. *Optimization Online*, page 2, 2018.

R. Stubbs and S. Mehrotra. A branch-and-cut method for 0–1 mixed convex programming. *Mathematical Programming*, 86(3):515–532, 1999.

Michael Sucker, Jalal Fadili, and Peter Ochs. Learning-to-optimize with PAC-Bayesian guarantees: Theoretical considerations and practical implementation. *arXiv preprint arXiv:2404.03290*, 2024.

Reza Takapoui, Nicholas Moehle, Stephen Boyd, and Alberto Bemporad. A simple effective heuristic for embedded mixed-integer quadratic programming. *International journal of control*, 93(1):2–12, 2020.

Pham Dinh Tao and LT Hoai An. Convex analysis approach to dc programming: theory, algorithms and applications. *Acta mathematica vietnamica*, 22(1):289–355, 1997.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57 (1):3–57, 2015.

Alan L Yuille and Anand Rangarajan. The concave-convex procedure (cccp). *Advances in neural information processing systems*, 14, 2001.

A Proofs

A.1 Proof of Proposition 4

Proof. Consider a single coordinate $i \in \{1, \dots, n\}$. Since the argument holds independently for each coordinate, the claim follows for all $i = 1, \dots, n$.

($\Longrightarrow$) If $v_i = \Pi(u_i)$, then by definition $v_i = 0$ when $u_i \leq 1/2$ and $v_i = 1$ when $u_i \geq 1/2$, so the inequalities are satisfied.

($\Longleftarrow$) If $v_i \in \{0, 1\}$ and the inequalities hold, then $u_i \leq 1/2$ forces $v_i = 0$, and $u_i \geq 1/2$ forces $v_i = 1$, which is exactly $v_i = \Pi(u_i)$.

■

A.2 Proof of Proposition 5

Proof. Let $u \in \mathbf{R}^n$ and $w = |u|$. Denote by $|w|_{[1]} \geq \cdots \geq |w|_{[n]}$ the order statistics of $\{w_i\}_{i=1}^n$.

($\Longrightarrow$) From $v = \Pi(u)$ to the encoding. Let $\tau = w_{[k]}$ denote the k-th largest value among $\{w_i\}_{i=1}^n$. We consider two cases.

Case 1: $\|u\|_0 \geq k$. By definition of the Euclidean projection onto $\mathcal{Z}$, v keeps exactly k indices of (one choice of) largest magnitude and zeros the rest. Let $S = \{i_{[1]}, \dots, i_{[k]}\}$; *i.e.*, S contains the indices of the k largest-magnitude entries of u. Set $\alpha_i = \mathcal{I}_S(i)$. Observe that $\alpha^T \mathbf{1}_n = k$. Then $\min_{i \in S} w_i \geq \tau \geq \max_{i \notin S} w_i$ and $v_i = u_i$ if $\alpha_i = 1$ and $v_i = 0$ if $\alpha_i = 0$, so the implications in the encoding hold.

Case 2: $\|u\|_0 < k$. Here $\Pi(u) = u$. Let S contain all nonzero indices of u (so $|S| = \|u\|_0$) and add any $(k - \|u\|_0)$ indices with $w_i = 0$ to reach $|S| = k$. Set $\alpha_i = \mathcal{I}_S(i)$ and choose $t = 0$. Then for $i \in S$, $w_i \geq 0 = t$ and $v_i = u_i$; for $i \notin S$, $w_i \leq 0 = t$ and $v_i = 0$. Hence the encoding holds.

In both cases (with ties handled by the freedom in choosing T and t), the existence of (α, t) satisfying the stated conditions follows from $v = \Pi(u)$.

($\Longleftarrow$) **From the encoding to $v = \Pi(u)$.** Conversely, suppose there exist $\alpha \in \{0,1\}^n$ and $\tau \geq 0$ such that

$$\mathbf{1}^\top \alpha = k, \;\; \alpha_i = 1 \Rightarrow w_i \geq \tau, \;\; \alpha_i = 0 \Rightarrow w_i \leq \tau, \;\; \alpha_i = 1 \Rightarrow v_i = u_i, \;\; \alpha_i = 0 \Rightarrow v_i = 0.$$

Let $S = \{i \mid \alpha_i = 1\}$; then $|S| = k$, $v_i = u_i$ for $i \in S$, and $v_i = 0$ for $i \notin S$. The threshold conditions imply $\min_{i \in S} w_i \geq \tau \geq \max_{i \notin S} w_i$, so every selected index has magnitude at least as large as any unselected index. Hence S indexes (one choice of) the k largest magnitudes of u, and v keeps exactly those entries while zeroing the rest. This is precisely the projection $v = \Pi(u)$ onto $\mathcal{Z}$ (up to tie-breaking). ■

A.3 Proof of Theorem 7

Proof. For notational convenience, we denote the optimal solutions in problem (11) with a subscript "opt" (*e.g.*, $z^\star_{\text{opt}}$) and the optimal values for the decision variables in problem (13) with a subscript "feas" (*e.g.*, $z^\star_{\text{feas}}$). Under Assumptions 1, 2, and 3 both problems are feasible. Under Assumption 6, $z^K_{\text{opt}} \in \Omega(x_{\text{opt}})$ and $z^K_{\text{feas}} \in \Omega(x_{\text{feas}})$. It is obvious that $\delta \leq \bar{\delta}$ since the optimality constraint implies the feasibility constraint. We must now prove that $\delta \geq \bar{\delta}$ under Assumption 3 and choice of ϕ. To do so, it is sufficient to prove that $z^\star_{\text{feas}}$ is optimal for problem (1) with parameter x_{feas}. Suppose it was not true. Then there exists a point $\bar{z} \in \mathbf{R}^n$ that is feasible in problem (1) with parameter x_{feas} such that $f(\bar{z}, x_{\text{feas}}) \leq f(z^\star_{\text{feas}}, x_{\text{feas}})$. But if this was true, then $z^\star_{\text{feas}}$ cannot be optimal to (13) since replacing it with $\bar{z}$ improves the objective. This finishes the proof by contradiction. ■

A.4 Proof of Theorem 9

Proof. Under Assumptions 1, 2, and 3, problem (15) is feasible. By the Farkas Lemma (Lemma 8), the constraint set $\{u \mid A^{K-1}u \leq b^{K-1}\}$ is non-empty if and only if there is no certificate y^K satisfying $(A^{K-1})^\top y^K = 0, y^K \geq 0, (b^{K-1})^\top y^K < 0$. In the verification problem (15), we maximize $-(b^{K-1})^\top y^K$ over all parameters $x \in \mathcal{X}$, iterates consistent with the SCP update rules, and dual variables y^K satisfying $(A^{K-1})^\top y^K = 0, y^K \geq 0$. Hence, the optimal value γ is the largest attainable value of $-(b^{K-1})^\top y^K$. If $\gamma > 0$, an infeasibility certificate exists. Conversely, if $\gamma \leq 0$, no infeasibility certificate can be constructed, and the QP is always feasible. ■

B Scalability improvements

We now present scalability enhancements that enable solving the verification problem over a greater number of iterations. We use this strategy for two examples: phase retrieval and sparse coding.

Optimization-based bound tightening. We take advantage of optimization-based bound tightening (OBBT) (Gleixner et al., 2017) to tighten the feasible region of the verification problem. The key idea is to tighten the lower and upper bounds on each decision variable by solving auxiliary optimization problems that minimize or maximize that variable subject to the verification constraints. Although this procedure can be applied iteratively to further tighten the bounds, we perform only a single pass in our examples. Each bound-tightening subproblem is solved with a time limit of 5 seconds.

Sequential solution strategy. For algorithms with a variable number of iterations (*i.e.*, all methods except relax-round-polish), we solve the verification problems sequentially, starting from iteration 0 and proceeding up to K. The bounds obtained from the OBBT procedure (if used) above serve as valid upper and lower bounds for each variable and are reused across iterations to improve solver efficiency.

C Timing results

We report solver times for the verification problems across all examples: box QP (Figure 8), network utility (Figure 9), power converter control (Figure 10), knapsack (Figure 11), phase retrieval (Figure 12), sparse coding (Figure 13), and hybrid vehicle control (Figure 14).

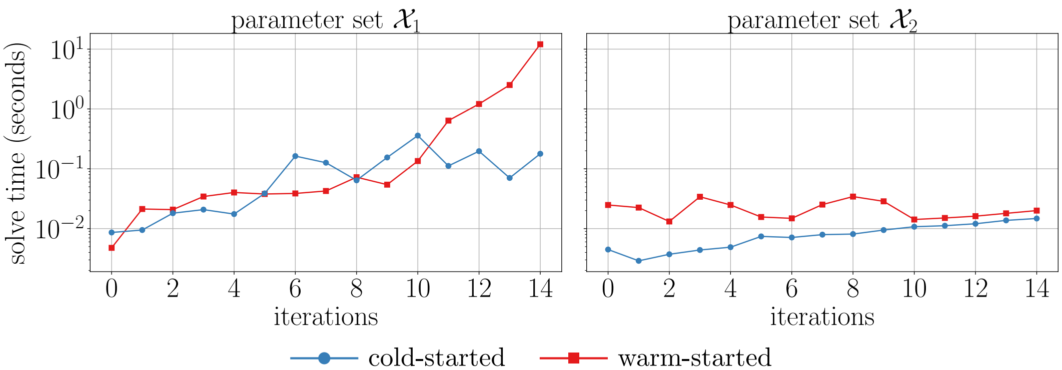

Figure 8: Timing results for the box QP from Subsection 5.1.1.

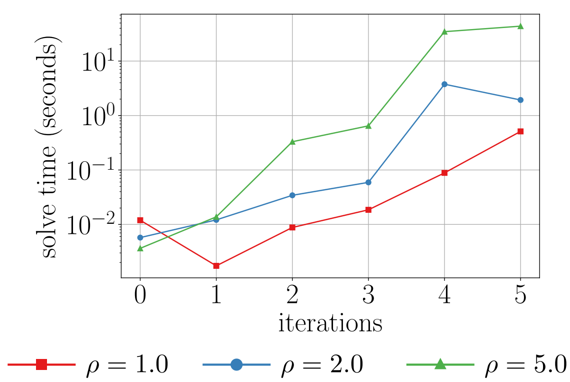

Figure 9: Timing results for the network utility maximization problem from Subsection 5.1.2.

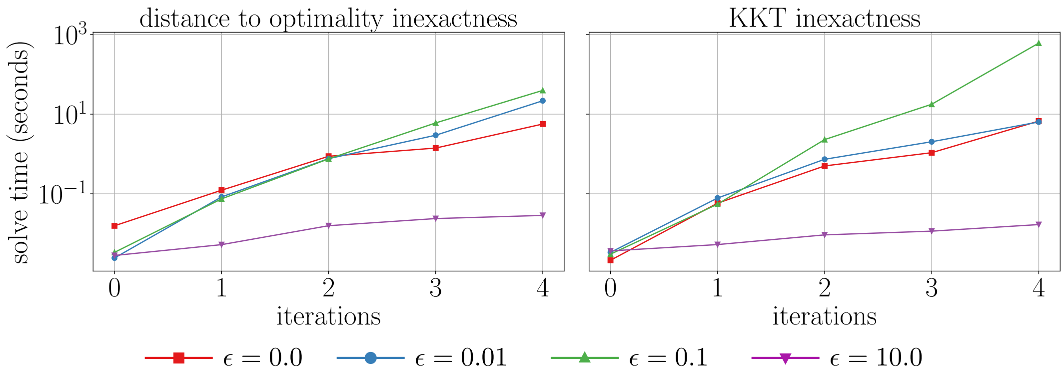

Figure 10: Timing results for the power converter problem from Subsection 5.2.1.

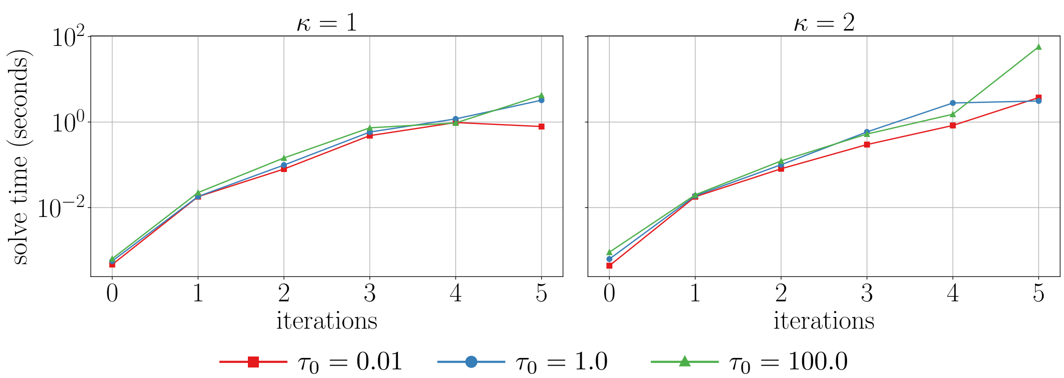

Figure 11: Timing results for the knapsack problem from Subsection 5.2.2.

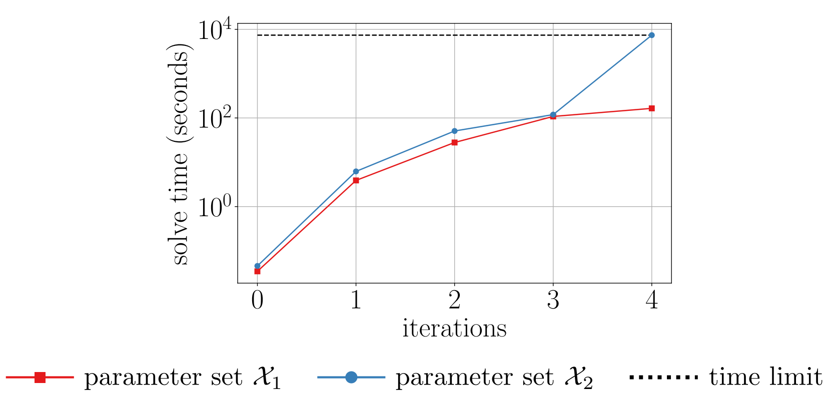

Figure 12: Timing results for phase retrieval from Subsection 5.3.1. The verification problem for the final iterate of $\mathcal{X}_2$ reaches the time limit of 7200 seconds (excluding time for OBBT).

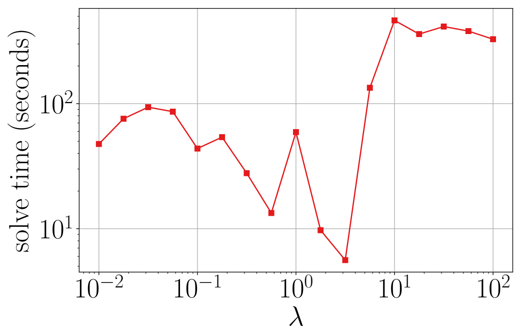

Figure 13: Timing results for sparse coding from Subsection 5.4.1.

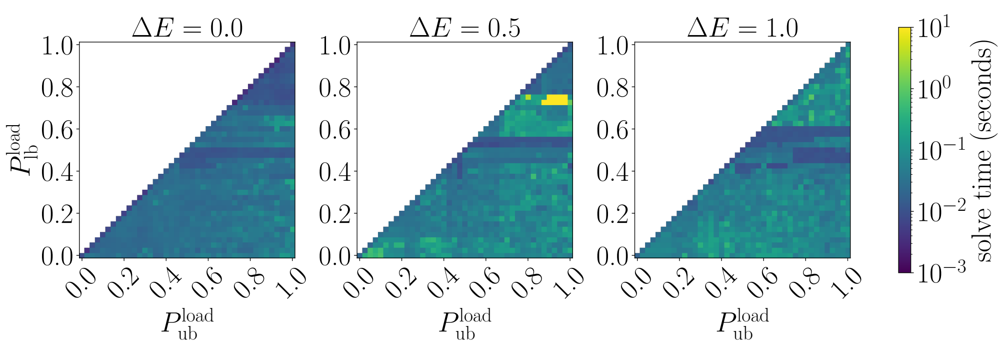

Figure 14: Timing results for hybrid vehicle control from Subsection 5.4.2. The time limit is 10 seconds.