Signal Temporal Logic Meets Convex-Concave Programming: A Structure-Exploiting SQP Algorithm for STL Specifications

Yoshinari Takayama[1], Kazumune Hashimoto[2], and Toshiyuki Ohtsuka[3]

Abstract— This study considers the control problem with signal temporal logic (STL) specifications. Prior works have adopted smoothing techniques to address this problem within a feasible time frame and solve the problem by applying sequential quadratic programming (SQP) methods naively. However, one of the drawbacks of this approach is that solutions can easily become trapped in local minima that do not satisfy the specification. In this study, we propose a new optimization method, termed *CCP-based SQP*, based on the convex-concave procedure (CCP). Our framework includes a new *robustness decomposition* method that decomposes the robustness function into a set of constraints, resulting in a form of difference of convex (DC) program that can be solved efficiently. We solve this DC program sequentially as a quadratic program by only approximating the disjunctive parts of the specifications. Our experimental results demonstrate that our method has a superior performance compared to the state-of-the-art SQP methods in terms of both robustness and computational time.

I. INTRODUCTION

As technology becomes more and more powerful, autonomous systems—such as drones and self-driving cars—are required to execute increasingly complex control tasks. Signal temporal logic (STL) has been progressively used as a specification language for these complex robotic tasks in various situations, due to their expressiveness and closeness to natural language [1]. However, the resulting optimization problem is a mixed integer program (MIP) [2, 3], which can easily be computationally intractable and scales poorly with the number of integer variables. To avoid this complexity, recent work formulates the problem as nonlinear programs (NLP) and solve it by sequential quadratic programming (SQP) methods [4, 5]. Although these methods can find a locally optimal solution in a short time compared to the MIP-based methods in general, the major difficulty of this formulation is that the solution can get stuck in a local minimum far from the global optimum as this method approximates the highly non-convex programs to quadratic programs. Moreover, the popular formulation makes this method more inefficient because all nonconvex functions are compressed into the objective function—which results in a coarse approximation by the naive SQP method.

[1]Y. Takayama is with Laboratoire des Signaux et Systèmes, Université Paris-Saclay, CNRS, CentraleSupélec, Gif-sur-Yvette, France. `yoshinari.takayama@l2s.centralesupelec.fr`
[2]K. Hashimoto is with the Graduate School of Engineering, Osaka University, Suita, Japan. `hashimoto@eei.eng.osaka-u.ac.jp`
[3]T. Ohtsuka is with the Graduate School of Informatics, Kyoto University, Kyoto, Japan. `ohtsuka@i.kyoto-u.ac.jp`

In this study, we address the aforementioned challenges by leveraging the structure of STL control problems. We introduce a novel optimization framework, termed the *convex-concave procedure based sequential quadratic programming (CCP-based SQP)*, which is a variant of the SQP method that solves quadratic programs sequentially. However, unlike traditional SQP methods, this framework approximates the original program at each iteration differently. It is based on the iterative optimization approach of CCP [6, 7]. Stated differently, only the concave portion is linearized at each iteration, allowing the method to retain the complete information of the convex parts. This is in contrast to naive SQP, which can only retain second-order derivative information at each iteration. As a result, our approach was able to synthesize a trajectory that is not only more robust (in terms of robustness score) but is also produced in a shorter time than the naive SQP methods.

The remainder of this paper is organized as follows. We first provide the notation, preliminaries, and problem formulation in Section II. Then, Section III provides our robustness decomposition method that transforms the optimization problem to an efficient DC form. Section IV provides some properties of the resulting CCP-based SQP framework. Section V demonstrates the effectiveness of our method over a state-of-the-art naive SQP method.

II. PRELIMINARIES

$\mathbb{R}$, and $\mathbb{Z}$ are defined as the fields of real and integer numbers, respectively. Given $a, b \in \mathbb{Z}$ with $a < b$, $[a, b]$ denotes a set of integers from a to b. True and false are denoted by $\top$ and $\bot$. I_n denotes the identity matrix of size n. $0_{n \times m}$ denotes the zero matrix of size $n \times m$.

A. System description

Throughout this paper, we consider a discrete-time linear system:

$$x_{t+1} = Ax_t + Bu_t, \tag{1}$$

where $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times m}$, and $x_t \in \mathcal{X} \subseteq \mathbb{R}^n, u_t \in \mathcal{U} \subseteq \mathbb{R}^m$ denote the state and input. We assume that $\mathcal{X}$ and $\mathcal{U}$ are a conjunction of polyhedra. Given an initial state $x_0 \in \mathcal{X}$ and a sequence of control inputs $\boldsymbol{u} = (u_0, \ldots, u_{T-1})$, the sequence of states $\boldsymbol{x} = (x_0, \ldots, x_T)$, which we call a *trajectory*, is uniquely generated.

B. Specification by Signal Temporal Logic

In this study, we consider the specifications given in STL. STL is a predicate logic defined for specifying properties for

continuous signals [1]. STL consists of predicates μ that can be obtained through a function $g^\mu(\cdot)$ as follows:

$$\mu = \begin{cases} \top & \text{if } g^\mu(\boldsymbol{x}) \leq 0 \\ \bot & \text{if } g^\mu(\boldsymbol{x}) > 0. \end{cases} \tag{2}$$

In this study, we consider affine functions for g^μ of the form $g^\mu = a^\top x_t - b : \mathcal{X} \to \mathbb{R}$, where $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. In addition to standard boolean operators $\wedge, \vee$, in STL, temporal operators $\square$ (*always*), $\lozenge$ (*eventually*), and $\boldsymbol{U}$ (*until*) are also considered. Each temporal operator has an associated bounded time interval $[t_1, t_2]$ where $0 \leq t_1 < t_2 < \infty$. We assume that STL formulae are written in negation normal form (NNF) without loss of generality [8]. STL formulae in NNF have negations only in front of the predicates [9] and we omit such negations from STL syntax. We refer to this negation-free STL as STL, which is defined by [10]:

$$\varphi := \mu | \vee \varphi | \wedge \varphi | \square_{[t_1, t_2]}\varphi | \lozenge_{[t_1, t_2]}\varphi \mid \varphi_1 \boldsymbol{U}_{[t_1, t_2]}\varphi_2.$$

The notion of *robustness* is a useful semantic defined for STL formulae, which is a real-valued function that describes how much a trajectory satisfies an STL formula. Let $(\boldsymbol{x}, t)$ denote a trajectory starting at timestep t, i.e., $(\boldsymbol{x}, t) = x_t, x_{t+1}, ..., x_T$. The trajectory length T should be selected large enough to calculate the satisfaction of a formula (see e.g., [9]). Given an STL formula φ, we define the *reversed* robustness with respect to a trajectory $\boldsymbol{x}$ and a time t that can be obtained recursively according to the following quantitative semantics:

Definition 1: (reversed-robustness)

$$\rho_{\text{rev}}^\mu((\boldsymbol{x}, t)) = g^\mu(x_t) \tag{3a}$$

$$\rho_{\text{rev}}^{\varphi_1 \wedge \varphi_2}((\boldsymbol{x}, t)) = \max\left(\rho_{\text{rev}}^{\varphi_1}((\boldsymbol{x}, t)), \rho_{\text{rev}}^{\varphi_2}((\boldsymbol{x}, t))\right) \tag{3b}$$

$$\rho_{\text{rev}}^{\varphi_1 \vee \varphi_2}((\boldsymbol{x}, t)) = \min\left(\rho_{\text{rev}}^{\varphi_1}((\boldsymbol{x}, t)), \rho_{\text{rev}}^{\varphi_2}((\boldsymbol{x}, t))\right) \tag{3c}$$

$$\rho_{\text{rev}}^{\square_{[t_1, t_2]}\varphi}((\boldsymbol{x}, t)) = \max_{t' \in [t+t_1, t+t_2]}\left(\rho_{\text{rev}}^{\varphi}((\boldsymbol{x}, t'))\right) \tag{3d}$$

$$\rho_{\text{rev}}^{\lozenge_{[t_1, t_2]}\varphi}((\boldsymbol{x}, t)) = \min_{t' \in [t+t_1, t+t_2]}\left(\rho_{\text{rev}}^{\varphi}((\boldsymbol{x}, t'))\right) \tag{3e}$$

$$\rho_{\text{rev}}^{\varphi_1 \boldsymbol{U}_{[t_1, t_2]}\varphi_2}((\boldsymbol{x}, t)) = \max_{t' \in [t+t_1, t+t_2]}\left(\min\left(\left[\rho_{\text{rev}}^{\varphi_1}((\boldsymbol{x}, t')), \right.\right.\right.$$
$$\left.\left.\left. \min_{t'' \in [t+t_1, t']}\left(\rho_{\text{rev}}^{\varphi_2}((\boldsymbol{x}, t''))\right)\right]\right)\right) \tag{3f}$$

Note that we define the recursive semantics for the *reversed* robustness function, i.e., the minus of the original robustness $\rho_{\text{orig}}^\varphi$ defined in the literature (see e.g., [11, Definition 2]) as $\rho_{\text{rev}}^\varphi = -\rho_{\text{orig}}^\varphi$. We reverse the sign of the robustness functions ρ_{rev}^μ of predicates μ, and we interchange the max and min operators. The reversed-robustness function is introduced as a notational simplification within our robustness decomposition framework. This approach removes unfavorable minus sign symbols in the decomposition and establishes a correspondence between convex parts in the optimization and conjunctive operators in the specification, and similarly for concave parts. It is worth mentioning that this modification does not change all the properties of the original robustness as we simply reverse the signs in the definition of the robustness function.

STL can be represented as a tree [12, 13]. An example is shown below.

Example 1: This example deals with a specification called two-target specifications, which is borrowed from [12]. The formula of this specification is given as:

$$\varphi = \lozenge_{[0, T-T_d]}\left(\square_{[0, T_d]}B_1 \vee \square_{[0, T_d]}B_2\right) \wedge \square_{[0, T]}\neg O \wedge \lozenge_{[0, T]}G. \tag{4}$$
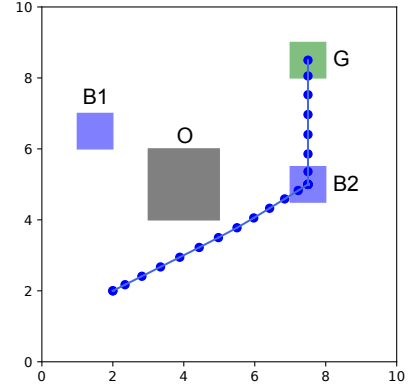

Fig. 1: Two-target scenario when $T = 25, T_d = 5$ with a trajectory generated by an MIP-based optimization method [11].

Figure 1 shows the graphical representation of the state trajectory satisfying the two-target specification with $T = 25, T_d = 5$. The robot must remain in one of the two regions, either B_1 or B_2, for a dwelling time of T_d, and it must reach the goal region G within the trajectory length of T while avoiding the obstacle region O.

Figure 2 shows a tree description of the two-target specification for reversed-robustness functions ρ^φ with $T = 1$ and $T_d = 1$. While logically equivalent STL formulas can have different tree descriptions, we adopt the *simplified form* of the tree description, which is uniquely determined per STL formula (for details, see [12]). The tree nodes marked "LP" at the bottom represent linear predicates, while the top (outermost) node is marked "and." The depth of this tree is four and the height of the predicates is either "and-or-and-predicate" or "and-or-predicate" regardless of the value of T and T_d. Note that although this figure represents the specification with $T = 1, T_d = 1$ for visibility, the depth and height of the tree do not change even when we vary these parameters because changing them will only increase the number of predicates in each depth.

C. Smooth approximation

The robustness function that results from (3) is not differentiable due to the max and min operators, and thus, we cannot calculate the gradients of this function. Recent papers such as [4] smooth these functions in order to apply gradient-based methods. However, because smoothing both the max and min of the function leads to a problem that deviates from the original, it is preferable to minimize the parts that are smoothed. Our method requires to smooth *only* min functions. This is because max functions in the robustness
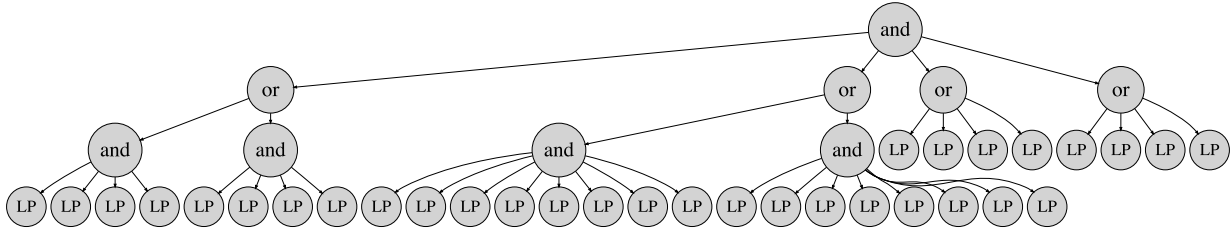
Fig. 2: A tree description of the two-target formula with $T = 1$ and $T_d = 1$

are rather decomposed in our method as shown in Section III. The popular smooth approximation of the max and min operators is by using the log-sum-exp (LSE) function. This approximation for min operators is given as:

$$\overline{\min}_k(a_1, ..., a_r) := -\frac{1}{k} \ln \sum_{i=1}^{r} e^{-ka_i}, \tag{5}$$

where $a_i \in \mathbb{R}$ and k is the smooth parameter. Now, we define a new robustness measure that will be used throughout this paper.

Definition 2: (Smoothed reversed-robustness) The smoothed reversed-robustness $\overline{\rho}_{\text{rev}}^{\varphi}(\boldsymbol{x})$ is defined by the quantitative semantics (3) where every min operator is replaced by $\overline{\min}_k$.

D. Problem statement

This study considers the problem of synthesizing the trajectory that satisfies the STL specification in a maximally robust way. Given the system (1) and the specification φ and the initial state x_0, the optimization problem is as follows:

$$\min_{\boldsymbol{x}, \boldsymbol{u}} \quad \overline{\rho}_{\text{rev}}^{\varphi}(\boldsymbol{x}) \tag{6a}$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t \tag{6b}$$

$$x_t \in \mathcal{X}, u_t \in \mathcal{U} \tag{6c}$$

$$\overline{\rho}_{\text{rev}}^{\varphi}(\boldsymbol{x}) \le 0 \tag{6d}$$

Note that the reversed-robustness $\rho_{\text{rev}}^{\varphi}(\boldsymbol{x})$ is replaced with the smoothed reversed-robustness $\overline{\rho}_{\text{rev}}^{\varphi}(\boldsymbol{x})$ in (6a) and (6d). Thus, although a feasible solution of the non-smoothed problem always satisfies the specification, a feasible solution of (6) does not necessarily satisfy the specification (as it under-approximates the non-smoothed robustness). However, the approximation error can be made arbitrarily small by making the smooth parameter k sufficiently large [4, 5].

III. A STRUCTURE-AWARE DECOMPOSITION OF STL FORMULAE

In this section, we present our robustness decomposition framework. The goal of our robustness decomposition is to transform the problem (6) in order to apply a convex-concave procedure (CCP)-based algorithm.

A. Convex-concave procedure (CCP)

The convex-concave procedure [6, 7] is a heuristic method for finding a local optimum of a class of optimization problems called the difference of convex (DC) programs, which is defined as follows:

Definition 3: (Difference of convex (DC) programs)

$$\text{minimize} \quad f_0(\boldsymbol{z}) - g_0(\boldsymbol{z}) \tag{7a}$$

$$\text{subject to} \quad f_i(\boldsymbol{z}) - g_i(\boldsymbol{z}) \le 0, \quad i \in \{1, \dots, m\}, \tag{7b}$$

where $\boldsymbol{z} \in \mathbb{R}^h$ is the vector of h optimization variables and the functions $f_i : \mathbb{R}^h \to \mathbb{R}$ and $g_i : \mathbb{R}^h \to \mathbb{R}$ are convex for $i \in \{0, \dots, m\}$. The DC problem (7) can also include equality constraints

$$p_i(\boldsymbol{z}) = q_i(\boldsymbol{z}), \tag{8}$$

where p_i and q_i are convex. These equality constraints are expressed as the pair of inequality constraints

$$p_i(\boldsymbol{z}) - q_i(\boldsymbol{z}) \le 0, \quad q_i(\boldsymbol{z}) - p_i(\boldsymbol{z}) \le 0. \tag{9}$$

The basic idea of CCP is simple; at each iteration of the optimization of the DC program (7), we replace concave terms with a convex upper bound obtained by linearization and then solve the resulting convex problem using convex optimization algorithms. It can be shown that all of the iterates are feasible if the starting point lies within the feasible set and the objective value converges (possibly to negative infinity, see [6, 14] for proof).

Note that either of the functions $p_i(\boldsymbol{z}), q_i(\boldsymbol{z})$ in equality constraints (8) are also approximated by the CCP. This is because an equality constraint is transformed into two inequalities (9) and both convex functions can make either inequality concave. Therefore, we should avoid introducing equality constraints if possible.

B. Robustness decomposition

Our decomposition framework alters the problem into an efficient form of a DC program considering the aforementioned features of CCP-based algorithms. This decomposition not only transforms the problem into a DC program but also enhances the efficiency of the overall algorithm. As a result, the resulting program approximates only the truly concave parts, and our approach results in a new SQP method.

To this end, we need to know whether each function in the program (6) is convex or concave. In this program, the robustness function $\overline{\rho}_{\text{rev}}^{\varphi}$ is the only part whose curvature (convex or concave) cannot be determined. Thus, if the robustness function $\overline{\rho}_{\text{rev}}^{\varphi}$, which is a composite function consisting of max and min operators, can be rewritten as a combination of convex and concave functions, the entire program can be reduced to a DC program. The proposed framework achieves this by recursively decomposing the

outermost operator of the robustness function until all the arguments of the functions are affine. As this study considers only affine functions for predicates $g^\mu(\cdot)$, if the arguments of a function are all predicates, the convexity or concavity of the function can be determined by examining the operator preceding it, which may be either a max or min function. The recursive decomposition of a nonconvex robustness function into predicates corresponds to a decomposition of the robustness tree from the top to the bottom, in accordance with the tree structure.

Our method utilizes the idea of the epigraphic reformulation methods. This method transforms the convex (usually linear) cost function into the corresponding epigraph constraints. However, our approach is different from the normal epigraphic reformulation in the sense that we deal with *nonconvex* cost functions and we have to apply the reformulation *recursively*. Our robustness decomposition transforms (6) into a more efficient form of DC program. For the remainder of this paper, we will focus on the simplified two-target specification in Example 1 due to page limitations. However, this decomposition can be generalized to the whole STL specifications in a similar manner.

As the robustness function $\overline{\rho}_{\text{rev}}^{\varphi}$ is in the cost function, the first procedure is to decompose the function from the cost function into a set of constraints. As we consider the two-target specification, the outermost operator of the robustness function $\overline{\rho}_{\text{rev}}^{\varphi}$ is max, i.e., $\overline{\rho}_{\text{rev}}^{\varphi} = \max(\overline{\rho}_{\text{rev}}^{\varphi_1}, \overline{\rho}_{\text{rev}}^{\varphi_2}, ..., \overline{\rho}_{\text{rev}}^{\varphi_r})$. We introduce a new variable s_ξ, and reformulate the program as follows.

$$\min_{\boldsymbol{x},\boldsymbol{u},s_\xi} s_\xi \tag{10a}$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t \tag{10b}$$

$$x_t \in \mathcal{X}, u_t \in \mathcal{U} \tag{10c}$$

$$s_\xi \leq 0 \tag{10d}$$

$$\overline{\rho}_{\text{rev}}^{\varphi_1}(\boldsymbol{x}) \leq s_\xi \ldots \overline{\rho}_{\text{rev}}^{\varphi_r}(\boldsymbol{x}) \leq s_\xi \tag{10e}$$

Note that the set of inequalities (10e) is equivalent to

$$\max(\overline{\rho}_{\text{rev}}^{\varphi_1}, \overline{\rho}_{\text{rev}}^{\varphi_2}, ..., \overline{\rho}_{\text{rev}}^{\varphi_r}) \leq s_\xi \tag{11}$$

although the original semantics of the variable s_ξ is the *equality* constraint:

$$\max(\overline{\rho}_{\text{rev}}^{\varphi_1}, \overline{\rho}_{\text{rev}}^{\varphi_2}, ..., \overline{\rho}_{\text{rev}}^{\varphi_r}) = s_\xi. \tag{12}$$

However, it is well known that this kind of max transformation is known to be equivalent, particularly for the convex case. Although (10) is nonconvex, we can also show that both (6) and (10) are equivalent (see [15] and [16, Section 8.3.4.4]). The proof of this equivalence is provided in Appendix A for a later explanation (in particular (17)).

As each $\overline{\rho}_{\text{rev}}^{\varphi_i}$ for $i = 1, \ldots, r$ is a robustness function, each inequality in (10e) can be restated as a constraint in one of the following two forms, depending on whether the outermost operator is max or min:

$$\max(\overline{\rho}_{\text{rev}}^{\Phi_1}, ..., \overline{\rho}_{\text{rev}}^{\Phi_{y_{\max}}}) \leq s_\xi, \tag{13}$$

$$\overline{\min}(\overline{\rho}_{\text{rev}}^{\Psi_1}, ..., \overline{\rho}_{\text{rev}}^{\Psi_{y_{\min}}}) \leq s_\xi, \tag{14}$$

where functions $\overline{\rho}_{\text{rev}}^{\Phi_j}(j \in \{1, ..., y_{\max}\})$ (resp. $\overline{\rho}_{\text{rev}}^{\Psi_j}(j \in \{1, ..., y_{\min}\})$) are robustness functions associated with Φ_j (resp. Ψ_j), which are the subformulas of $\varphi_i(i \in \{1, ..., r\})$. Note that these arguments of the max and min functions are still not necessarily predicates. We continue the following steps until all the arguments in each function become the predicates.

For inequality constraints of the form (13), we transform it as follows:

$$\overline{\rho}_{\text{rev}}^{\Phi_1} \leq s_\xi, \overline{\rho}_{\text{rev}}^{\Phi_2} \leq s_\xi, ..., \overline{\rho}_{\text{rev}}^{\Phi_{y_{\max}}} \leq s_\xi. \tag{15}$$

This is of course equivalent to (13).

For constraints of the forms (14), we first check whether each argument of the $\overline{\min}$ function is a predicate or not. If not, we replace such an argument with a new variable. Because we consider the simplified form, the outermost operator of the function $\overline{\rho}_{\text{rev}}^{\Psi_j}(j \in \{1, ..., y_{\min}\})$ is max, i.e., $\overline{\rho}_{\text{rev}}^{\Psi_j} = \max(\overline{\rho}_{\text{rev}}^{\psi_1}, ..., \overline{\rho}_{\text{rev}}^{\psi_h})$. We first replace function $\overline{\rho}_{\text{rev}}^{\Psi_j}$ in (14) by a new variable s_{new} as

$$\overline{\min}(\overline{\rho}_{\text{rev}}^{\Psi_1}, ..., s_{\text{new}}, ..., \overline{\rho}_{\text{rev}}^{\Psi_{y_{\min}}}) \leq s_\xi. \tag{16}$$

Then, we add the following constraints:

$$\max(\overline{\rho}_{\text{rev}}^{\psi_1}, ..., \overline{\rho}_{\text{rev}}^{\psi_h}) \leq s_{\text{new}}. \tag{17}$$

Although the inequality constraint (17) should be equality, this modification is justified by the similar discussion for (10e) even though the equivalence of this transformation is not immediately obvious (see the proof of Theorem 1).

All the inequalities in (15) and (17) are either in the max form (13) or the min form (14). Thus, subsequent to these steps, the constraints of the forms (13), (14) are decomposed into several constraints in the same forms again. We repeat this procedure until we reach the predicates, which is the bottom of the tree in Figure 2. When we finish the above recursive manipulations, (6) is transformed into an DC program as follows:

$$\min_{\boldsymbol{x},\boldsymbol{u},s_\xi,\boldsymbol{s}_{\max},\boldsymbol{s}_{\min}} s_\xi \tag{18a}$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t \tag{18b}$$

$$x_t \in \mathcal{X}, u_t \in \mathcal{U} \tag{18c}$$

$$s_\xi \leq 0, \tag{18d}$$

$$\left.\begin{array}{ccc} \overline{\rho}_{\text{rev}}^{\lambda_1^{(1)}} \leq s_{\max}^{(1)}, ..., \overline{\rho}_{\text{rev}}^{\lambda_{y_{\max}}^{(1)}} \leq s_{\max}^{(1)} \\ \vdots \qquad\qquad \vdots \\ \overline{\rho}_{\text{rev}}^{\lambda_1^{(v)}} \leq s_{\max}^{(v)}, ..., \overline{\rho}_{\text{rev}}^{\lambda_{y_{\max}}^{(v)}} \leq s_{\max}^{(v)} \end{array}\right\} \begin{array}{c} \text{from max} \\ \text{functions} \end{array}$$
$$\tag{18e}$$

$$\left.\begin{array}{c} \overline{\min}(\overline{\rho}_{\text{rev}}^{\mu_1^{(1)}}, ..., \overline{\rho}_{\text{rev}}^{\mu_{y_{\min}}^{(1)}}) \leq s_{\min}^{(1)} \\ \vdots \\ \overline{\min}(\overline{\rho}_{\text{rev}}^{\mu_1^{(w)}}, ..., \overline{\rho}_{\text{rev}}^{\mu_{y_{\min}}^{(w)}}) \leq s_{\min}^{(w)}, \end{array}\right\} \begin{array}{c} \text{from min} \\ \text{functions} \end{array}$$
$$\tag{18f}$$

where all the formulas $\lambda_1^{(\cdot)}, ..., \lambda_{y_{\max}}^{(\cdot)}, \mu_1^{(\cdot)}, ..., \mu_{y_{\min}}^{(\cdot)}$ in (18e) and (18f) are predicates and $\boldsymbol{s}_{\max} = (s_{\max}^{(1)}, ..., s_{\max}^{(v)})$ and

$\boldsymbol{s}_{\min} = (s_{\min}^{(1)}, ..., s_{\min}^{(w)})$. Note that, with some abuse of notation, some variables in $\boldsymbol{s}_{\max}, \boldsymbol{s}_{\min}$ in constraints (18e),(18f) may represent s_ξ and s_{new}, and some arguments of the $\overline{\min}$ function in (18f) are not affine predicates but the new variables.

Important properties of this program (18) are stated below.

Proposition 1: Let $\boldsymbol{z}' = (\boldsymbol{x}', \boldsymbol{u}', s_\xi', \boldsymbol{s}_{\max}', \boldsymbol{s}_{\min}')$ denote a feasible solution for program (18). Then, $\overline{\rho}_{\text{rev}}^\varphi(\boldsymbol{x}') \leq 0$ holds.

Proof: See Appendix B. ∎

Proposition 2: If $\boldsymbol{z}'$ is feasible for (18), $(\boldsymbol{x}', \boldsymbol{u}')$ is feasible for (6).

Proof: The inequality (6d) holds from Proposition 1. As (6b) and (6c) are the same as (18b) and (18c), $(\boldsymbol{x}', \boldsymbol{u}')$ satisfies all the constraints in (6). ∎

Using these two propositions, we state the next theorem.

Theorem 1: The global optimum of two programs, (6) and (18) are the same.

Proof: See Appendix C. ∎

Note that this theorem is not readily apparent (refer to the proof). It states that despite transforming program (6) into (18) for easier handling, the transformed program doesn't necessarily become more conservative.

IV. PROPERTIES OF THE SUBPROBLEM

After we obtain the transformed program (18), we apply the CCP method described in Subsection III-A. We majorize the $\overline{\min}$ operator in (18f) at the current point of each iteration. In general, the resulting program becomes a convex program such as a second-order cone program (SOCP) and CCP has to solve the convex program sequentially. However, the subproblem we solve at each iteration is a linear program or a quadratic program, and not the general convex program as stated in the following theorem:

Theorem 2: The program (18) after the majorization by CCP is a linear program (LP). If we add the quadratic cost function, the program can be written as a convex quadratic program (QP).

Proof: In the program (18), the cost function is s_ξ and the constraints including (18e) are all affine except for concave constraints (14). As CCP linearizes all the concave constraints, the program resulting from the majorization of (18) becomes an LP. ∎

Let $\mathcal{P}_{\text{LP}}$ denote this LP in the following. It is worth mentioning that inequalities in (18e) are all affine inequalities as we remove all the $\max$ operators in the end. As a corollary of the above theorem, our CCP-based method sequentially solves LP. If we add the quadratic cost function in (18a), our method becomes sequential quadratic programming (SQP).

Moreover, as all the constraints coming from the robustness decomposition framework in Section III do not have any equality constraints, the non-convex parts of (18) are only the inequality constraints (18f), which comes from $\min$ functions of the robustness function.

Proposition 3: The program (18) has no equality constraints except for the state equations (18b), which are affine constraints.

Hence, the CCP only approximates the *truly* concave portions of (18f) arising from the $\min$ functions in the robustness function. It is noteworthy that enforcing the constraints, such as (10e) and (17), as equality constraints would require the CCP to approximate them as described in Subsection III-A, thereby leading to undesirable approximations. In contrast, the proposed method minimizes the number of parts requiring approximation, which enhances the algorithm's performance.

Furthermore, the feasible region of the program $\mathcal{P}_{\text{LP}}$ is interior to the feasible region of the program (18).

Proposition 4: A feasible solution of the linear program $\mathcal{P}_{\text{LP}}$ is a feasible solution to the program (18).

Proof: This is because the first-order approximations of CCP at each step are global over-estimators; i.e., $f_i(\boldsymbol{z}) - g_i(\boldsymbol{z}) \leq f_i(\boldsymbol{z}) - g_i(\boldsymbol{z}_{(\cdot)}) + \nabla g_i(\boldsymbol{z}_{(\cdot)})^\top (\boldsymbol{z}_{(\cdot)} - \boldsymbol{z})$ where $\boldsymbol{z}_{(\cdot)}$ is the current point of variable $\boldsymbol{z}$. ∎

Finally, the next theorem provides the safety guarantee.

Theorem 3: Let $\boldsymbol{x}''$ denote a feasible trajectory of the program $\mathcal{P}_{\text{LP}}$. Then $\boldsymbol{x}''$ always satisfies the STL specification φ, i.e., $\boldsymbol{x}'' \vDash \varphi$ in the limit $k \to \infty$.

Proof: Based on Propositions 1 and 4, $\overline{\rho}_{\text{rev}}^\varphi(\boldsymbol{x}'') \leq 0$ holds. The limit as $k \to \infty$ results in the smoothed reversed-robustness $\overline{\rho}_{\text{rev}}^\varphi(\boldsymbol{x}'')$ being equivalent to the original reversed-robustness $\rho_{\text{rev}}^\varphi(\boldsymbol{x}'')$. Therefore, $\rho_{\text{rev}}^\varphi(\boldsymbol{x}'') \leq 0$ as $k \to \infty$, which proves the statement. ∎

Therefore, if the initial guess is feasible, the solution obtained by our CCP-based SQP method can always satisfy the specification. In practice, we select a sufficiently large k to minimize the approximation error.

V. NUMERICAL EXPERIMENTS

We illustrate the advantages of our proposed method through the two-target scenarios with varying time horizons ranging from $T = 50$ to 140. In the subsequent analysis, we set the smoothing parameter k to 10. All experiments were conducted on a 2020 MacBook Air with an Apple M1 processor and 8 GB of RAM[1]. The state x_t and input u_t are defined as $x_t = [p_x, p_y, \dot{p}_x, \dot{p}_y]^\top, u_t = [\ddot{p}_x, \ddot{p}_y]^\top$, where p_x is the horizontal position of the robot and p_y is the vertical position. As for the system dynamics, we use a double integrator, i.e., a system (1) with the matrix

$$A = \begin{bmatrix} I_2 & I_2 \\ 0_{2\times 2} & I_2 \end{bmatrix}, \quad B = \begin{bmatrix} 0_{2\times 2} \\ I_2 \end{bmatrix}. \quad (19)$$

We implemented our algorithm in Python using CVXPY [17] as the interface to the optimizer. We chose the GUROBI(ver. 10)'s QP-solver with the default option. We took the penalty CCP [7] to solve the problem, which can admit the violation of constraints during the optimization procedures by penalizing the violation with variables. We introduced this penalty variable only for the concave constraints because the CCP only approximates such concave parts. Additionally, we added a small quadratic cost function. The weights on the penalty variables and the quadratic cost

---

[1]The source code and experimental data results of this paper are available at `https://github.com/yotakayama/STLCCP`.

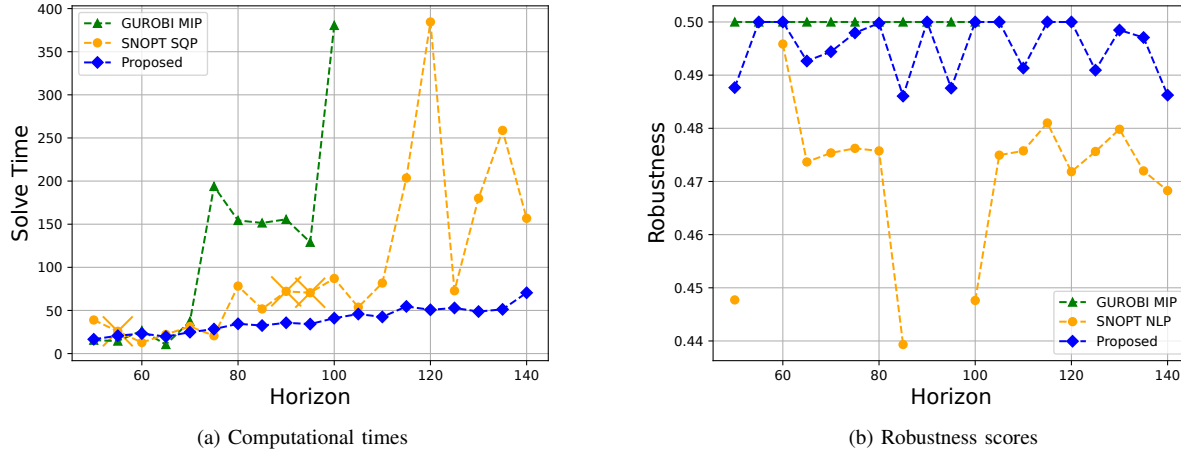(a) Computational times   (b) Robustness scores

Fig. 3: Computation time and robustness score of the three methods over every 5 horizons from $T = 50$ to 140.

functions were 50.0 and 0.001, respectively. The resulting cost function (18a) is represented as:

$$s_\xi + 50.0 \sum_{i=0}^{w}(s_i) + 0.001 \sum_{t=0}^{T}(x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t), \quad (20)$$

where $Q = \mathrm{diag}(0, 0, 1, 1), R = I_2$ are positive semidefinite symmetric matrices, and s_i for $i \in \{1, ..., w\}$ (w is the number of constraints in (18f)) is the penalty variable added to the r.h.s. of each concave constraints (18f) satisfying $s_i \leq 0$. The initial state is fixed as $x_0 = [2.0, 2.0, 0, 0]$, and the bounds on the state and input variables are $\mathcal{X} = [0.0, 0.0, -1.0, -1.0]^\mathsf{T} < x_t < [10.0, 10.0, 1.0, 1.0]^\mathsf{T}$ and $\mathcal{U} = [-0.2, -0.2]^\mathsf{T} < u_t < [0.2, 0.2]^\mathsf{T}$, respectively.

We compared our method with a popular MIP-based method and a state-of-the-art NLP-based method. For the MIP-based approach, we formulated the problem as a MIP using the encoding framework in [11] and solved the problem with the GUROBI solver (GUROBI is often the fastest MIP solver). On the other hand, for the NLP-based method, we used the naive sequential quadratic programming (SQP) approach [18] using the smoothing method proposed in [5]. All the parameters of the two methods were the default settings.

Figure 3 compares convergence time and robustness score of the three methods from $T = 50$ to 140. Note that all the robustness scores are of the original semantics ρ_{orig} [11, Definition 2]. The GUROBI-MIP method (green) is truncated at the horizon $T = 100$ due to the computation time 7371.3 s, which significantly exceeds the range of the plotted region. Subsequent results for longer horizons took even more computational time, so they are not displayed for the sake of visibility. The plots of our method are the averages of different trials varying the initial values of variables, whereas the other two plots are the results of one trial. All the scores in the right plot are calculated using the original robustness function.

In summary, our proposed method outperformed the state-of-the-art SNOPT-NLP method with respect to both the

robustness score and the computation time. Specifically, as shown in the left plot, the computation time of our proposed method remained at a consistently low level even when the horizon increased. In contrast, the other two methods displayed volatility or were infeasibly slow, particularly above $T = 100$. Furthermore, the right plot demonstrates that our proposed method did not sacrifice the robustness score to shorten the computation time. On the contrary, our method consistently achieved a higher robustness score than the SNOPT-NLP method. Moreover, our method achieved a score of 0.5 at certain horizons, which matches the global optimum achieved by the MIP-based method. As the plot of our method represents the average of trials, this implies that all the trials at these horizons are global optima.

Regarding the success rate of these two methods, the proposed method failed in 12 out of a total of 190 trials (with 10 trials conducted at each horizon), indicating a success rate of approximately 93.7%. In contrast, the SNOPT-NLP method failed at three instances ($T = 55, 90, 95$) out of the 19 horizons (with a success rate of approximately 84.2%), all of which resulted in robustness scores of $-\infty$. These instances are indicated by orange cross markers ($\times$) in the left plot, and their robustness scores are not displayed in the right plot due to being $-\infty$. These results suggest that our method does not exhibit a higher failure rate compared to the SNOPT-NLP method.

Lastly, Figure 4 shows the trajectories generated by the proposed method when $T = 50$, where the weight on the cost function is set to 0.01 for visibility. Our method produced a range of satisfactory trajectories, depending on the initial values of the system's variables, while every trajectory represents a global optimum, i.e., $\rho_{\mathrm{orig}} = 0.5$.

VI. CONCLUSION

In this study, we have introduced the CCP-based SQP algorithm for control problems involving STL specifications leveraging the inherent structures of STL. The subproblem is an efficient quadratic program, obtained by approximating
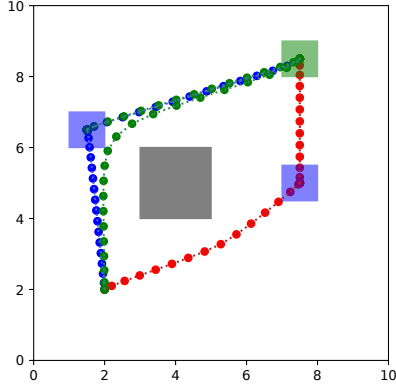
Fig. 4: Three trajectories generated by the proposed method with random initial values for $T = 50$. Every trajectory represents a global optimum.

solely the concave constraints of the program that correspond to the disjunctive nodes of the STL tree.

Future work includes extending the proposed method to non-affine predicates, nonlinear systems, and non-smooth cases. While this paper focused on linear systems to establish an efficient SQP approach, the same methodology can be applied to tackle more complex cases.

REFERENCES

[1] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[2] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," *in IEEE Conference on Decision and Control*, pp. 3953–3958, 2008.

[3] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," *in IEEE Conference on Decision and Control*, pp. 81–87, 2014.

[4] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," *in IEEE Conference on Control Technology and Applications*, pp. 1235–1240, 2017.

[5] Y. Gilpin, V. Kurtz, and H. Lin, "A smooth robustness measure of signal temporal logic for symbolic control," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 241–246, 2021.

[6] T. Lipp and S. Boyd, "Variations and extension of the convex–concave procedure," *Optimization and Engineering*, vol. 17, no. 2, pp. 263–287, 2016.

[7] X. Shen, S. Diamond, Y. Gu, and S. Boyd, "Disciplined convex-concave programming," *in IEEE Conference on Decision and Control (CDC)*, pp. 1009–1014, 2016.

[8] S. Sadraddini and C. Belta, "Formal synthesis of control strategies for positive monotone systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 2, pp. 480–495, 2019.

[9] ——, "Robust temporal logic model predictive control," *in Annual Allerton Conference on Communication, Control, and Computing*, pp. 772–779, 2015.

[10] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[11] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 115–140, 2019.

[12] V. Kurtz and H. Lin, "Mixed-integer programming for signal temporal logic with fewer binary variables," *IEEE Control Systems Letters*, vol. 6, pp. 2635–2640, 2022. Their source code is available at https://stlpy.readthedocs.io.

[13] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *IEEE Robotics and Automation Letters*, 2022.

[14] B. K. Sriperumbudur and G. R. G. Lanckriet, "On the convergence of the concave-convex procedure," *in Advances in Neural Information Processing Systems*, vol. 22, 2009.

[15] L. Lindemann and D. V. Dimarogonas, "Robust control for signal temporal logic specifications using discrete average space robustness," *Automatica*, vol. 101, pp. 377–387, 2019.

[16] G. C. Calafiore and L. El Ghaoui, *Optimization Models*. Cambridge University Press, 2014.

[17] S. Diamond and S. Boyd, "CVXPY: A python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 1, p. 2909–2913, 2016.

[18] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Review*, vol. 47, pp. 99–131, 2005.

APPENDIX

A. Equivalence between the two programs: (6) and (10)

We prove the following equivalence.

Proposition 5: Both formulations (6) and (10) are equivalent in the sense that the global optimum is the same.

Proof: We show in the following that if $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ is optimal for (6), then $(\boldsymbol{x}^*, \boldsymbol{u}^*, s_\xi^* = \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x}^*))$ is optimal for (10). Firstly, note that for any feasible solution $(\boldsymbol{x}, \boldsymbol{u})$ of (6), $(\boldsymbol{x}, \boldsymbol{u}, s_\xi = \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x}))$ is feasible for (10) as (10e) becomes an identity by the substitution $s_\xi = \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x})$. Thus, $(\boldsymbol{x}^*, \boldsymbol{u}^*, s_\xi^* = \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x}^*))$ is also feasible for (10).

Suppose this solution $(\boldsymbol{x}^*, \boldsymbol{u}^*, s_\xi^* = \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x}^*))$ is not optimal for (10). Then, there exists another feasible solution $(\mathsf{x}^*, \mathsf{u}^*, \mathsf{s}_\xi^*)$ with a better objective, i.e., such that $\mathsf{s}_\xi^* < s_\xi^* = \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x}^*)$. Since this feasible solution also satisfies (10e), that is, $\overline{\rho}_{\mathrm{rev}}^\varphi(\mathsf{x}^*) \leq \mathsf{s}_\xi^*$, the inequality $\overline{\rho}_{\mathrm{rev}}^\varphi(\mathsf{x}^*) < \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x}^*)$ holds. Considering that $(\mathsf{x}^*, \mathsf{u}^*)$ is feasible for (6) (as one of the inequality conditions in (10e) must become the equality condition when the cost function s_ξ takes a local minimum), the inequality $\overline{\rho}_{\mathrm{rev}}^\varphi(\mathsf{x}^*) < \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x}^*)$ contradicts the optimality of $\boldsymbol{x}^*$ for (6). Therefore, $(\boldsymbol{x}^*, \boldsymbol{u}^*, s_\xi^* = \overline{\rho}_{\mathrm{rev}}^\varphi(\boldsymbol{x}^*))$ is optimal for (10), and the global optimum is the same. ∎

B. Proof of Proposition 1

Proof: We present a proof sketch. Let us consider the initial replacement of the robustness function $\overline{\rho}_{\mathrm{rev}}^{\Psi_i}(\boldsymbol{x}')$ with a new variable s_{new} that converts inequality (14) into two inequalities, (16) and (17). We first focus on this initial replacement without considering the similar replacements that follow after this one. Note that (17) represents the inequality

$$\overline{\rho}_{\mathrm{rev}}^{\Psi_i}(\boldsymbol{x}') \leq s_{\mathrm{new}}, \qquad (21)$$

although this inequality sign should be an equality if s_{new} were a substituting variable. Nevertheless, due to the fact that $\overline{\min}$ is an increasing function with respect to each argument, we have the inequality

$$\overline{\min}(\overline{\rho}_{\mathrm{rev}}^{\Psi_1}, ..., \overline{\rho}_{\mathrm{rev}}^{\Psi_i}, ..., \overline{\rho}_{\mathrm{rev}}^{\Psi_{y_{\min}}}) \leq \overline{\min}(\overline{\rho}_{\mathrm{rev}}^{\Psi_1}, ..., s_{\mathrm{new}}, ..., \overline{\rho}_{\mathrm{rev}}^{\Psi_{y_{\min}}}),$$

Combining this inequality with (16), we can conclude that if the transformed constraints (16) and (17) holds, inequality (14) holds. Similar analogies can be used for all subsequent transformations of non-convex robustness functions with new variables that follow the same pattern. Thus, ultimately, we can say that if inequalities (18f) and (18e) hold, the original

inequalities in (10e) hold. By this inequality and (18d), we have $\overline{\rho}_{\mathrm{rev}}^{\varphi}(\boldsymbol{x}') \leq 0$ for (18). ∎

From the tree perspective, this proof demonstrates that by replacing the robustness of any subformula with a new variable that is greater than or equal to the original value as in (21), the parent node's robustness score, due to the monotonicity of the max and min functions, also becomes greater than or equal to the original value as in (14). This property extends further to the values of the grandparents and subsequent nodes, ultimately resulting in the cost function s_{new}, representing the top-node value, being greater than the robustness value of φ. This tree perspective also becomes important in the proof of Theorem 1.

C. Proof of Theorem 1

We first prove the following proposition, which states the converse of Proposition 2:

Proposition 6: Let $\lambda_{\mathrm{parent}}^{(i)}$ denotes the parent node of formulas $\lambda_1^{(i)}, ..., \lambda_{y_{\mathrm{max}}}^{(i)}$ for $i \in \{1, ..., v\}$, and $\mu_{\mathrm{parent}}^{(j)}$ denotes the parent node of formulas $\mu_1^{(j)}, ..., \mu_{y_{\mathrm{min}}}^{(j)}$ for $j \in \{1, ..., w\}$. If $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ is a feasible solution of (6), the following solution $\boldsymbol{z}^*$ is feasible for (18).

$$\boldsymbol{z}^* := (\boldsymbol{x}^*, \boldsymbol{u}^*, s_\xi^* = \overline{\rho}_{\mathrm{rev}}^{\varphi}(\boldsymbol{x}^*), \boldsymbol{s}_{\mathrm{max}}^* = (\overline{\rho}_{\mathrm{rev}}^{\lambda_{\mathrm{parent}}^{(1)}}(\boldsymbol{x}^*),$$

$$..., \overline{\rho}_{\mathrm{rev}}^{\lambda_{\mathrm{parent}}^{(v)}}(\boldsymbol{x}^*)), \boldsymbol{s}_{\mathrm{min}}^* = (\overline{\rho}_{\mathrm{rev}}^{\mu_{\mathrm{parent}}^{(1)}}(\boldsymbol{x}^*), ..., \overline{\rho}_{\mathrm{rev}}^{\mu_{\mathrm{parent}}^{(w)}}(\boldsymbol{x}^*))) \quad (22)$$

Proof: When all the inequalities in (18e) and (18f) becoming identity equations, the feasible region of (18) becomes identical to that of (6), i.e., $S = \mathrm{Proj}(S_{\mathrm{DC}})$ where S and S_{DC} denote the feasible region of (6) and (18) respectively, and the operator Proj denotes the projection mapping by the substitution $\boldsymbol{z}^*(\cdot)$ from $(\boldsymbol{x}, \boldsymbol{u}, s_\xi, \boldsymbol{s}_{\mathrm{max}}, \boldsymbol{s}_{\mathrm{min}})$ to $(\boldsymbol{x}, \boldsymbol{u})$. Therefore, the feasible region of (6) is included in that of (18). ∎

We now show the following optimality equivalence:

Proposition 7: If a feasible solution $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ is optimal for (6), then $\boldsymbol{z}^*$ in (22) is optimal for (18).

Proof: From Proposition 6, if $\boldsymbol{z}^*$ is not the optimal solution for (18), there exists another feasible solution $(\boldsymbol{x}', \boldsymbol{u}', s_\xi', \boldsymbol{s}_{\mathrm{max}}', \boldsymbol{s}_{\mathrm{min}}')$, with a better objective value, i.e., such that $s_\xi' < s_\xi^* = \overline{\rho}_{\mathrm{rev}}^{\varphi}(\boldsymbol{x}^*)$. As in the proof of Proposition 1, $\overline{\rho}_{\mathrm{rev}}^{\varphi}(\boldsymbol{x}') \leq s_\xi'$. Therefore, $\overline{\rho}_{\mathrm{rev}}^{\varphi}(\boldsymbol{x}') < \overline{\rho}_{\mathrm{rev}}^{\varphi}(\boldsymbol{x}^*)$. However, as Proposition 2 shows that $\boldsymbol{x}'$ is also feasible for (6), this inequality contradicts the fact that $\boldsymbol{x}^*$ is optimal for (6). Therefore, $\boldsymbol{z}^*$ is optimal for (18). ∎

From this proposition, the global optimum of the two programs (6) and (18) is the same, that is, $\overline{\rho}_{\mathrm{rev}}^{\varphi}(\boldsymbol{x}^*)$. Moreover, we can also prove analogously that if a feasible solution $\boldsymbol{z}'$ is optimal for (18), then $(\boldsymbol{x}', \boldsymbol{u}')$ is optimal for (6).