Computing Maximizer Trajectories of Affine Dynamics for Reachability

Goran Frehse[1]

Abstract— **Computing an overapproximation of the reachable set of states of a continuous or hybrid system is a challenging problem. The use of overapproximations based on a set of template directions has led to scalable algorithms. When the approximation is too conservative, it can be refined by adding more template directions. Synthesizing suitable directions is possible but costly without tight underapproximations to guide the refinement. Suitable underapproximations can be constructed from the trajectories of states that are maximal in the template directions, which we call maximizers. In this paper, we propose algorithms to compute maximizer trajectories for dynamical systems with affine dynamics and nondeterministic inputs. Our computations are based solely on solving ODEs and, assuming the initial condition is a polytope, solving linear programs. Highly optimized commercial tools are available for both tasks, with corresponding performance and numerical robustness, which may help pave the way towards industrial applications of reachability analysis. Since maximizer trajectories represent actual executions of the system, or parts thereof, they can be used as counterexamples and provide additional feedback and insight to the user.**

I. Introduction

We consider the problem of computing – in an approximative form – the reachable states of a hybrid system. Starting from a convex set of initial states, the states evolve over time according to a set of affine ordinary differential equations (ODEs). The set of these states as a function of time is called a flowpipe. Under these assumptions, this set is convex at every instant of time, but nonconvex overall. The flowpipe is generally difficult to compute, so that the challenge lies in finding an overapproximation that is both precise and cheap.

A compact, i.e., closed and bounded, convex set can be represented by its support function, which describes the set by attributing to every direction the signed distance between the origin and the farthest point of the set in that direction. Such a point is called a *maximizer* (support vector) of the given direction and for many types of sets, e.g., polytopes, it can be computed efficiently. Choosing a set of template directions and computing the corresponding values of the support function, one can obtain an overapproximation in the form of a polyhedron. In addition, the convex hull of the maximizers is an underapproximation of the set.

Support functions have been used to overapproximate flowpipes with a series of template polyhedra, leading to very efficient and scalable algorithms [1]. However, the accuracy depends on the chosen template directions. To decide the reachability of a target set or to obtain an arbitrarily exact approximation of the reachable set requires refining the result by adding more directions. It is possible to find suitable directions using support functions only, but the computational cost is substantial and a large number of directions are needed for higher-dimensional systems [2].

In this paper, we present algorithms to compute the evolution of the maximizing points over time. Our computation of maximizer trajectories is based solely on solving ODEs and, assuming the initial set is a polytope, solving linear programs. Highly optimized commercial tools are available for both tasks, and using them may help dealing with numerical problems that arise quickly in practice, as we have witnessed in working on an industrial case study [3]. Indeed, our entire approach can be implemented by piloting an external ODE solver, as long as linear programs (LPs) can be provided as root functions. One could therefore envision the integration in existing tools, in particular in an industrial context.

The main interest of maximizer trajectories is that they provide a tight underapproximation of the flowpipe. This opens up new possiblities for refinement on several levels. To give just two examples, it allows one to use optimal approximation algorithms such as the *Mutually Converging Polytopes* (MCP) by Kamenev [4], or separation algorithm such as the *Gilbert-Johnson-Keerthi (GJK) algorithm* [5] for showing that a target or guard set is not reachable. Computing maximizer trajectories is more expensive than simply using support functions, but our experiments in [2], which use coarse underapproximations derived from support functions, indicate that having tight underapproximations can save a lot of effort in the refinement process. This is particularly relevant for hybrid systems, in which the intersection of the flowpipe with a guard set is used as the initial states in the next discrete state. Overapproximation may lead to spurious transitions, which in turn can lead to a state explosion that renders standard reachability algorithms useless even for simple systems [2]. Underapproximations are evidently useful for creating counterexamples, which are difficult to obtain for hybrid systems and essential for falsification and refinement schemes such as CEGAR [6]. A positive side effect of maximizer trajectories is that they represent actual executions of the system (or parts thereof), which provides additional feedback and insight to the user.

Related work. The use of optimal control and ODEs to derive bounds on the reachable set has been proposed in [7], [4], [8] and the principle has first been applied to set-based reachability in [9]. But to the best of our knowledge, algorithmically

*This work was partly supported by the European Commission under grant 643921 (UnCoVerCPS) and by the Institut Carnot–Logiciels et Systèmes Intelligents.

[1]Goran Frehse is with the Department of Computer Science, Mathematics, and Applied Mathematics, Univ. Grenoble Alpes, Verimag, 38610 Gières, France `goran.frehse@imag.fr`

solving the corresponding continuous-time ODEs has never been addressed in detail. The central problem is the construction of a sequence of input vectors over time, such that a linear cost function (the position in the template direction) is maximized. This is a classic optimal control problem [10], but our case is somewhat particular. The number of switching alternatives can be very large (the number of vertices of the input set), so an explicit enumeration is to be avoided. The number of switching times is unknown and may vary widely depending on the time horizon and the set of inputs. The cost function is particularly simple, which allows us to reformulate the problem as detecting zero-crossings of a given root function. We can therefore delegate the problem of finding the optimal switching times to an ODE solver that is capable of precisely detecting zero-crossings.

Both fixed-direction and dynamic-direction templates have been used for set-based reachability based on time discretization. Applying fixed-direction templates to continuous-time systems, [8], [1], [11], [12] discretize time and derive bounds in continuous time from a first-order Taylor-series approximation. To obtain bounds that are conservative over a continuous time interval, these approaches use compensation terms like $e^{\|A\|\delta}$, where δ is the time step. To meet a given error bound thus may require extremely small time steps, which may stall progress or lead to numerical problems. The computation of maximizer trajectories completely avoids this problem by relying on the capabilities of the ODE solver. Higher-order Taylor series approximations are used in [13]. Dynamic-direction templates have been used for discrete-time systems in [14], [15], where refinement heuristics are included to detect additional directions that improve the accuracy.

In the next section, we recall the basics of flowpipe approximation using support functions and maximizers. In Sect. III, we present our algorithm for computing trajectories in a given, fixed direction. In Sect. IV, we show that the same techniques can be applied to computing trajectories in directions that evolve with the system. The proofs have been omitted for lack of space, and can be found in [16].

II. Set-Based Reachability

We consider the reachability of a continuous dynamic system of the form

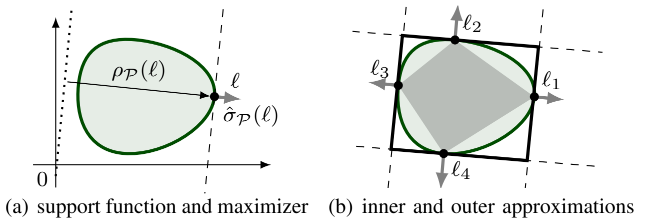
$$\dot{x}(t) = Ax(t) + u(t), \qquad u(t) \in \mathcal{U}, \tag{1}$$

where $x(t) \in \mathbb{R}^n$ is an n-dimensional vector, A is a real matrix, and $\mathcal{U} \subseteq \mathbb{R}^n$ a compact convex set. The states *reachable* at time t from *initial states* $\mathcal{X}_0 \subseteq \mathbb{R}^n$ are

$$\begin{aligned}\mathcal{X}_t(A, \mathcal{X}_0, \mathcal{U}) = \{&x(t) \mid x(0) \in \mathcal{X}_0,\\ &\forall 0 \le \tau \le t\ \exists u(\tau) \in \mathcal{U} : \dot{x}(\tau) = Ax(\tau) + u(\tau)\}.\end{aligned} \tag{2}$$

In this paper, we consider $\mathcal{X}_0$ and $\mathcal{U}$ to be polyhedral. We simply write $\mathcal{X}_t$ if A, $\mathcal{X}_0$, and $\mathcal{U}$ are clear from the context. The *flowpipe* over a time interval $[0, T]$ is

$$\mathcal{X}_{[0,T]} = \bigcup\nolimits_{t\in[0,T]} \mathcal{X}_t(A, \mathcal{X}_0, \mathcal{U}).$$



$\rho_{\mathcal{P}}(\ell)$
ℓ
$\hat{\sigma}_{\mathcal{P}}(\ell)$
0
ℓ_2
ℓ_3
ℓ_1
ℓ_4
(a) support function and maximizer
(b) inner and outer approximations

Fig. 1. The values of the support function in a given set of directions define a polyhedral outer approximation, and the corresponding maximizers define an inner approximation

Sometimes we are interested in whether a given *target set* $\mathcal{G} \subseteq \mathbb{R}^n$ is reachable. In a hybrid setting, the target can be the guard of a transition and for reachability one needs to compute the intersection of the reachable states with the guard set. The goal may also be to show that the target set is actually not reachable. E.g., when $\mathcal{G}$ is a set of unsafe states or the guard set of a transition that leads to unsafe states.

A. Convex Sets and Support Functions

Different types of sets can be used to represent $\mathcal{X}_t$ and $\mathcal{X}_{[0,T]}$ exactly or approximately, and the choice of representation has a large impact on the computational complexity. Using support functions, one can derive such approximations very efficiently from maximizer trajectories. We recall the basics.

The *support function* of a nonempty compact convex set $\mathcal{S} \subseteq \mathbb{R}^n$ with respect to a *direction* $\ell \in \mathbb{R}^n$ is

$$\rho_{\mathcal{S}}(\ell) = \max\nolimits_{x\in\mathcal{S}} \ell^{\mathsf{T}} x.$$

The support function of a compact convex set $\mathcal{S}$ is an exact representation of the set, since $\mathcal{S}$ can be constructed from the function values using

$$\mathcal{S} = \bigcap\nolimits_{\ell\in\mathbb{R}^n} \{x \mid \ell^{\mathsf{T}} x \le \rho_{\mathcal{S}}(\ell)\}.$$

A point $x \in \mathcal{S}$ is called a *support vector* or *maximizer* of $\mathcal{S}$ for direction ℓ if $\ell^{\mathsf{T}} x = \rho_{\mathcal{S}}(\ell)$. The set of maximizers of $\mathcal{S}$ for direction ℓ is denoted by $\sigma_{\mathcal{S}}(\ell)$. Let $\hat{\sigma}_{\mathcal{S}}(\ell)$ be a function that returns just one of the maximizers in $\sigma_{\mathcal{S}}(\ell)$, see Fig. 1(a) for an illustration.

We now recall several geometric operations that can be carried out efficiently using support functions. Let $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2 \subseteq \mathbb{R}^n$ be compact convex sets. The *convex hull* of a set $\mathcal{V} \subseteq \mathbb{R}^n$ is

$$\mathrm{CH}(\mathcal{V}) = \Big\{ \sum\nolimits_{i=1}^{m} \lambda_i v_i \;\Big|\; v_i \in \mathcal{V}, \lambda_i \ge 0 : \sum\nolimits_{i=1}^{m} \lambda_i = 1 \Big\}.$$

The image of a linear map (matrix) $M \in \mathbb{R}^m \times \mathbb{R}^n$ is the set $M\mathcal{S} = \{Ms \mid s \in \mathcal{S}\}$. The *Minkowski sum* is $\mathcal{S}_1 \oplus \mathcal{S}_2 = \{s_1 + s_2 \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\}$. The following rules apply to support functions:

$$\begin{aligned}\rho_{M\mathcal{S}}(\ell) &= \rho_{\mathcal{S}}(M^{\mathsf{T}}\ell),\\ \rho_{\mathcal{S}_1\oplus\mathcal{S}_2}(\ell) &= \rho_{\mathcal{S}_1}(\ell) + \rho_{\mathcal{S}_2}(\ell),\\ \rho_{\mathrm{CH}(\mathcal{S}_1\cup\mathcal{S}_2)}(\ell) &= \max\big(\rho_{\mathcal{S}_1}(\ell), \rho_{\mathcal{S}_2}(\ell)\big).\end{aligned} \tag{3}$$

Similar rules apply to maximizers:

$$\begin{aligned}\sigma_{M\mathcal{S}}(\ell) &= M\sigma_{\mathcal{S}}(M^{\mathsf{T}}\ell),\\ \sigma_{\mathcal{S}_1\oplus\mathcal{S}_2}(\ell) &= \sigma_{\mathcal{S}_1}(\ell)\oplus\sigma_{\mathcal{S}_2}(\ell),\\ \sigma_{\mathrm{CH}(\mathcal{S}_1\cup\mathcal{S}_2)}(\ell) &= \sigma_{\mathrm{CH}(\sigma_{\mathcal{S}_1}(\ell)\cup\sigma_{\mathcal{S}_2}(\ell))}(\ell).\end{aligned} \tag{4}$$

Polyhedra. A *halfspace* $\mathcal{H}\subseteq\mathbb{R}^n$ is the set of points satisfying a linear constraint, $\mathcal{H}=\{x \mid a^{\mathsf{T}}x\le b\}$, where $a=(a_1\cdots a_n)\in\mathbb{R}^n$ and $b\in\mathbb{R}$. A *polyhedron* $\mathcal{P}$ is the intersection of a finite set of halfspaces and called a *polytope* if it is bounded. Using vector-matrix notation,

$$\mathcal{P}=\{x \mid Ax\le b\},\quad A=\begin{pmatrix}a_1^{\mathsf{T}}\\ \vdots\\ a_n^{\mathsf{T}}\end{pmatrix},\quad b=\begin{pmatrix}b_1\\ \vdots\\ b_n\end{pmatrix}.$$

A linear constraint is *active* (touching) in x if $a^{\mathsf{T}}x=b$.

Normal Cones. The *cone* of a set $\mathcal{V}\subseteq\mathbb{R}^n$ is

$$\mathrm{Cone}(\mathcal{V})=\left\{\sum_{i=1}^{m}\lambda_i v_i \,\middle|\, v_i\in\mathcal{V},\lambda_i\ge 0\right\}.$$

A cone is *simplical* of dimension m if it is the cone of m linearly independent vectors $v_1,\ldots,v_m$. A simplical cone of dimension n is a polyhedron with n constraints:

$$\mathrm{Cone}(v_1,\ldots,v_n)=\{x \mid -(v_1\cdots v_n)^{-1}x\le 0\}. \tag{5}$$

The *normal cone* of any point $x\in\mathcal{P}$ is spanned by the normal vectors of the constraints active in x.

$$\mathcal{N}_{\mathcal{P}}(x)=\mathrm{Cone}\left(\left\{a_i \,\middle|\, a_i^{\mathsf{T}}x=b_i, i=1,\ldots,m\right\}\right).$$

The normal cone of x is also the set of directions for which x is a maximizer:

Lemma 2.1 (folklore): A point x in a polyhedron $\mathcal{P}$ is a maximizer in direction $\ell\neq 0$ iff $\ell\in\mathcal{N}_{\mathcal{P}}(x)$.

B. Flowpipe Approximation with Maximizers

Let $x_1,\ldots,x_K$ be maximizers of a compact convex set $\mathcal{S}$ in directions $\ell_1,\ldots,\ell_K$. Then $\mathcal{S}$ is bounded by the inner and outer approximations [7],

$$\mathrm{CH}(x_1,\ldots,x_K)\subseteq\mathcal{S}\subseteq\bigcap_k\{\ell_k^{\mathsf{T}}x\le\ell_k^{\mathsf{T}}x_k\}, \tag{6}$$

see Fig. 1(b) for an illustration. A *maximizer trajectory* $(x^*(\cdot),\ell^*(\cdot))$ attributes to every $t\ge 0$ a state $x^*(t)\in\mathcal{X}_t$ and a direction $\ell^*(t)\in\mathbb{R}^n$ for which $x^*(t)$ is a maximizer. Applying (6) at every time instant t, a set of maximizer trajectories $(x_1^*,\ell_1^*),\ldots,(x_K^*,\ell_K^*)$ defines an approximation of the flowpipe [7]:

$$\mathrm{CH}\big(x_1^*(t),\ldots,x_K^*(t)\big)\subseteq\mathcal{X}_t\subseteq\bigcap_k\Big\{\ell_k^*(t)^{\mathsf{T}}x\le\ell_k^*(t)^{\mathsf{T}}x_k^*(t)\Big\}. \tag{7}$$

We are interested in computing the values of $x^*(t')$ and $\ell^*(t')$ for a given $t'\ge 0$. Ignoring numerical errors, we assume to have an *ideal ODE solver* at our disposal, i.e., a function

$$\mathrm{ODE}_A(x_0,u_0,t_f)$$

that takes as arguments an initial state x_0, a constant input u_0, a final time t_f, and returns the state $x(t_f)$, where $x(t)$ is given by $x(0)=x_0$, $\dot{x}(t)=Ax(t)+u_0$. We also assume an *ideal root solver*

$$\mathrm{ODE}_A(x_0,u_0,t_0,T,g),$$

which is an ideal ODE solver that takes as additional arguments a start time t_0 and a vector g of root functions $g_i:\mathbb{R}^n\to\mathbb{R}$. If for any t' in the time interval $(t_0,T]$ one of the root functions satisfies $g(x(t'))=0$, the solver returns $(x(t'),t')$ for the smallest such t'. Otherwise, it returns $(x(T),T)$.

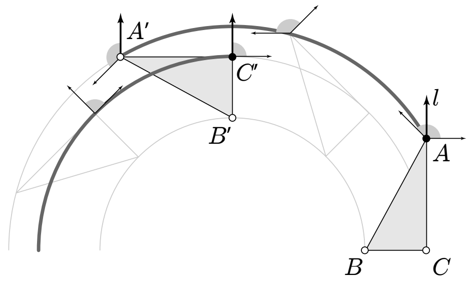


Fig. 2. A fixed-direction maximizer trajectory $x^*(t)$ (bold grey) for fixed-direction ℓ. The trajectory $x_A(t)$ with $x_A(0)=A$ is valid until t'. At t', both $x_A(t')=A'$ and $x_C(t')=C'$ are maximizers. However, ℓ leaves the normal cone of $x_A(t)$ (light grey) as time goes on, while it enters that of $x_C(t)$. Therefore, $x^*(t)$ jumps from $x_A(t)$ to $x_C(t)$ at t' and remains with it for the remainder of the shown trajectory

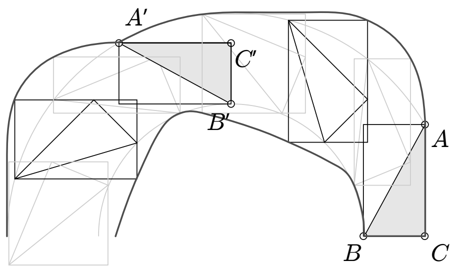


Fig. 3. A flowpipe approximation constructed from fixed-direction maximizer trajectories in the positive and negative axis directions. At every time instant t, the approximation is the bounding box of the reachable set $\mathcal{X}_t$. The nonsmoothness in the outline at A' corresponds to the switch from the point A' as maximizer to the point C' as maximizer as illustrated in Fig. 2 – the vertical derivative at C' is zero while at A' it is not

In this paper, we consider two particular instances of maximizer trajectories: In the first case, the direction remains constant and we change the state when necessary to obtain a maximizer for that direction. We call this a *fixed-direction maximizer trajectory*. In the second case, we track the evolution of the same state and let the direction evolve such that the state remains its maximizer. We call this a *dynamic-direction maximizer trajectory*. Note that in the first case, the trajectory is generally not continuous. As we will see, both types are closely related.

III. Fixed-Direction Maximizer Trajectories

In this section, we discuss how to compute a trajectory (states over time) that are maximal points of the flowpipe with respect to a given, fixed, direction. We consider a system with affine dynamics

$$\dot{x}(t) = Ax(t) + u(t), \qquad x(0) \in \mathcal{X}_0, u(t) \in \mathcal{U}. \tag{8}$$

Given a direction $\ell \in \mathbb{R}^n$, our goal is to compute a solution $x^*(t)$ such that $(x^*(t), \ell^*(t))$ is a maximizer trajectory, where $\ell^*(t) = \ell = \text{const}$. An example maximizer trajectory is shown in Fig. 2, and a flowpipe approximation constructed using the axis directions is shown in Fig. 3. Note that the flowpipe approximation is equivalent to the time-discretization approach from [1] with infinitesimally small time steps.

According to the superposition principle, a solution of (8) is the sum of a solution for $\mathcal{U} = 0$ (autonomous solution) and a solution for $\mathcal{X}_0 = 0$ (input solution). Let $x_{\text{aut}}(t)$ be a maximizer trajectory for $\mathcal{X}_t(\mathcal{X}_0, 0)$ and $x_{\text{inp}}(t)$ a maximizer trajectory for $\mathcal{X}_t(0, \mathcal{U})$, both for the same direction ℓ. Their sum is a maximizer trajectory of (8),

$$x^*(t) = x_{\text{aut}}(t) + x_{\text{inp}}(t).$$

To compute $x_{\text{aut}}(t)$, we need to consider linear dynamics

$$\dot{x}_{\text{aut}}(t) = Ax_{\text{aut}}(t), \qquad x_{\text{aut}}(0) \in \mathcal{X}_0 \tag{9}$$

The flowpipe for these dynamics is $\mathcal{X}_t = e^{At}\mathcal{X}_0$. Given a direction $\ell \in \mathbb{R}^n$, the goal is to find a function $x^*_{\text{aut}}(t)$ such that for every $t \geq 0$, $x^*_{\text{aut}}(t) \in \sigma_{\mathcal{X}_t}(\ell)$. To find a solution, we can change to a moving coordinate system, such that the set is fixed and the direction is moving. This is equivalent since with (4),

$$\sigma_{\mathcal{X}_t}(\ell) = \sigma_{e^{At}\mathcal{X}_0}(\ell) = e^{At}\sigma_{\mathcal{X}_0}(e^{A^\mathsf{T}t}\ell), \tag{10}$$

where on the left we have the maximizers of $\mathcal{X}_t$ for a fixed direction ℓ and on the right the maximizers of $\mathcal{X}_0$ for the *dynamic direction*

$$\ell_t = e^{A^\mathsf{T}t}\ell_0, \tag{11}$$

with $\ell_0 = \ell$. Let $y^* \in \sigma_{\mathcal{X}_0}(\ell_t)$, then $x^*_{\text{aut}}(t) = e^{At}y^*$. The value of $x^*_{\text{aut}}(t)$ can be computed from y^* using an ODE solver, but this is not incremental. For subsequent values of t it will be quicker to compute the matrix exponential incrementally by solving n initial value problems

$$\dot{e}_i(t) = Ae_i(t), \tag{12}$$

where $e_i(0)$ is a unit vector with the i-th coordinate being 1 and the others being zero. Then $e^{At} = \big(e_1(t) \cdots e_n(t)\big)$.

The trajectory $x_{\text{inp}}(t)$ can be obtained by solving [8]

$$\dot{x}_{\text{inp}}(t) = Ax_{\text{inp}}(t) + u^*(t), \qquad x_{\text{inp}}(0) = 0, \tag{13}$$

$$\dot{\ell}^*(t) = A^\mathsf{T}\ell^*(t), \qquad \ell^*(0) = \ell, \tag{14}$$

$$u^*(t) \in \sigma_{\mathcal{U}}(\ell^*(t)). \tag{15}$$

The difficulty in solving (13)–(15) lies with the last equation: We need to find $u^*(t)$ that satisfies (15). With (14) and (11),

$$\ell^*(t) = e^{A^\mathsf{T}t}\ell = \ell_t.$$

We must therefore find for each $\tau \in [0, t]$ a point $u^*(\tau)$ in $\mathcal{U}$ that is a maximizer for ℓ_τ. This will be discussed in the next section. The final algorithm for computing a fixed-direction maximizer will be presented in Sect. III-B.

A. Fixed-Direction Maximizer Sequences

The goal is to compute maximizers of a compact convex set $\mathcal{P}$ as a function of time t and direction $\ell_t = e^{A^\mathsf{T}t}\ell$. The result is a function $y^*(t)$ such that for every $t \geq 0$, $y^*(t) \in \sigma_{\mathcal{P}}(\ell_t)$. We reduce the problem to attributing to each t a normal cone of $\mathcal{P}$ that contains ℓ_t. With Lemma 2.1, we have:

Lemma 3.1: Let (y_k, t_k) for $k \geq 0$ be a sequence such that $t_0 = 0$, $y_k \in \mathcal{P}$, and for all τ, $t_k \leq \tau \leq t_{k+1}$, $\ell_\tau \in \mathcal{N}_{\mathcal{P}}(y_k)$. Then $y^*(t)$ defined by $y^*(t) = y_k$ for $t_k \leq t < t_{k+1}$ satisfies $y^*(t) \in \sigma_{\mathcal{P}}(\ell_t)$.

Such a sequence of normal cones exists, since the union of all normal cones of $\mathcal{P}$ is $\mathbb{R}^n$ [17]. We refer to (y_k, t_k) as the *fixed-direction maximizer sequence*. Each y_k is the sequence where $t_k < t_{k+1}$ not only is a maximizer at t_k, but remains a maximizer for some positive time span. We call any such y_k a *forward maximizer* of ℓ_t. As an example, consider the set $A'B'C'$ in Fig. 2. Both A' and C' are maximizers at time t', but only C' is a forward maximizer. In the following, we discuss different criteria to identify a suitable maximizer y_k, and to check whether it remains a maximizer at future points in time.

1) Pointwise finding a maximizer: We start with a basic algorithm for finding a maximizer sequence and then discuss alternatives for different steps. Initially, $k = 0$ and we compute a maximizer y_k for ℓ_k. Then we use the ODE solver with a root function to compute ℓ_t and detect t_{k+1}, i.e., the end of the time interval over which y_0 is a maximizer. The process is the repeated from t_{k+1}. Let $\lambda(\ell)$ be the distance of the current point y_k to the supporting hyperplane of $\mathcal{P}$,

$$\lambda(\ell) = (\hat{\sigma}_{\mathcal{P}}(\ell) - y_k)^\mathsf{T}\ell/\|\ell\|.$$

The value of $\lambda(\ell)$ is zero if y_k is a maximizer and strictly positive otherwise. According to our definition, an ideal ODE root solver stops at the first root, while here we would need it to stop at the last t such that $\lambda(\ell) = 0$. We solve this problem by using the root function $\lambda(\ell) - \varepsilon$ to create a zero crossing that can be detected and then let the solver backtrack to the last zero of $\lambda(\ell)$. Going backwards, we can use as root function $\lambda(\ell)$, or $\gamma(\ell) = (y^* - y_k)^\mathsf{T}\ell/\|\ell\|$, where y^* is the maximizer at the position where a nonzero value of $\lambda(\ell)$ was detected. The use of $\gamma(\ell)$ may be advantageous in practice, because it is a smooth function and can be computed faster. A large value of ε increases the risk of missing a maximizer change (see discussion below), so we add a maximum time step δ for safety. An alternative root function will be discussed in Sect. III-A.3.

In detail, the algorithm proceeds as follows. Given a threshold $\varepsilon > 0$, a maximum time step $\delta > 0$ and a time horizon T, the following algorithm $\text{MSEQ}_A(\mathcal{P}, t_0, T, \varepsilon, \delta)$ computes a maximizer sequence (y_k, t_k) of $\mathcal{P}$ for the time interval $[t_0, T]$.

1) Let $k = 0$, $\ell' = \ell$, $t' = t_0$, and $y_0 = \hat{\sigma}_{\mathcal{P}}(\ell)$.
2) Let $t_f = \min(T, t' + \delta)$ and compute

$$(\ell', t') = \text{ODE}_{A^\mathsf{T}}(\ell', 0, t', t_f, \lambda - \varepsilon).$$

3) If $\lambda(\ell') > 0$, let $y^* = \hat{\sigma}_{\mathcal{P}}(\ell^*)$, $\ell^- = \ell'$, $t^- = t'$ and backtrack to the first zero crossing after t_k using the root function

$$\gamma(\ell) = (y^* - y_k)^{\mathsf{T}} \ell / \|\ell\|.$$

   a) Let $t^* = t^-$, $\ell^* = \ell^-$.
   b) Compute (backwards in time from t^-)

$$(\ell^-, \Delta^-) = \mathrm{ODE}_{-A^{\mathsf{T}}}(\ell^-, 0, 0, t^- - t_k, \gamma).$$

   c) Let $t^- := t^- - \Delta^-$.
   d) If $t^- > t_k$ and $\Delta^- > 0$, go to step 3a. (search for earlier zero until t_k is reached)
   e) If $t^* = t'$, let $y_k = y^*$ (no zero after t_k, so change current maximizer).
   f) Otherwise, let $t_{k+1} = t^*$, $y_{k+1} = y^*$, $t' = t_{k+1}$, $\ell' = \ell^*$, $k := k+1$ (adding a new maximizer).

4) If $t' = T$, stop. Otherwise, go to step 2.

Example: A small numerical example shall illustrate the algorithm. Let $A^{\mathsf{T}} = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$, so that ℓ_t describes a counter-clockwise circular trajectory around the origin. Let $\mathcal{P}$ be the box $\mathcal{P} = \{0 \le x_1 \le 1, -1 \le x_2 \le 0.5\}$, so the normal cones of $\mathcal{P}$ are simply the quadrants of the plane. Starting from $\ell_0 = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$, a fixed-direction maximizer sequence up to $T = \pi$ is given by

$$\left(\left(\begin{smallmatrix} 1 \\ 0.5 \end{smallmatrix}\right), 0\right), \left(\left(\begin{smallmatrix} 0 \\ 0.5 \end{smallmatrix}\right), \tfrac{\pi}{2}\right), \left(\left(\begin{smallmatrix} 0 \\ -1 \end{smallmatrix}\right), \pi\right), \left(\left(\begin{smallmatrix} 1 \\ -1 \end{smallmatrix}\right), \tfrac{3\pi}{2}\right). \quad (16)$$

We run the MSEQ algorithm implemented in C++ on a standard laptop using double precision. A number comparisons are to a relative error of 10^{-12} and an absolute error of 10^{-14}, relaxing equalities and tightening strict inequalities. As ODE solver we use CVODE [18] with a relative tolerance of 10^{-9} and an absolute tolerance of 10^{-13}. The computation of the support vectors is carried out by the linear programming library GLPK [19]. We use $\varepsilon = 10^{-6}$ and $\delta = t_f$.

Initially, $y_0 = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$, which is a maximizer only for $t = 0$, i.e., not a forward maximizer. In step 2 the solver stops at $t' = 2 \cdot 10^{-6}$. In step 3, $\lambda(\ell')$ evaluates to almost exactly $1 \cdot 10^{-6}$, and we obtain the maximizer $y^* = \left(\begin{smallmatrix} 1 \\ 0.5 \end{smallmatrix}\right)$. The corresponding time stamp t^* takes the value t' if no zero is found during backtracking, or the time of the earliest zero. It's initial value is t'. Step 3b backtracks to $t^- = 7 \cdot 10^{-18}$, and since t^* still has the value t', step 3e sets $y_0 = \left(\begin{smallmatrix} 1 \\ 0.5 \end{smallmatrix}\right)$. The backtracking has turned y_0 into a forward maximizer. Continuing with step 2, the solver finds a root at t' slightly past $\frac{\pi}{2}$, and step 3 obtains $y^* = \left(\begin{smallmatrix} 0 \\ 0.5 \end{smallmatrix}\right)$. Step 3b backtracks to almost exactly $t^- = \frac{\pi}{2}$, which in step 3d is greater than t_0 so the backtracking is continued to check for earlier roots. In step 3a, $t^* = \frac{\pi}{2}$, which is the time stamp of the earliest zero found in backtracking. Step 3b backtracks to t^- being almost exactly t_0, which ends the backtracking. In step 3f, we have $t^* < t'$, so a new maximizer $y_1 = \left(\begin{smallmatrix} 0 \\ 0.5 \end{smallmatrix}\right)$ with time stamp $t_1 = \frac{\pi}{2}$ is added to the sequence. The remaining maximizers in the sequence are detected analogously to y_1.

The output of this run approximates the maximizer sequence (16) with a relative error of $8 \cdot 10^{-10}$ on the switching times, with the maximizers being exact up to machine precision. The support vector computation of $\mathcal{P}$ is called 432 times in total.

As an alternative, backtracking with λ instead of γ increases the relative error to $7 \cdot 10^{-6}$ and requires an additional 17 support vector computations for backtracking. As another alternative, going forward with λ, i.e., $\varepsilon = 0$, we need to replace values of the root function where $\lambda(\ell) = 0$ with $\lambda(\ell) = -1$ since otherwise CVODE will not detect any root in step 2. Here, we skip the backtracking steps 3a–3e and rely on the internal backtracking of the ODE solver to find the earliest t with $\lambda(\ell_t) > 0$. In step 2 the solver stops at $t' = 1.3 \cdot 10^{-8}$. Since no backtracking is performed, the output sequence is $\left(\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right), 0\right), \left(\left(\begin{smallmatrix} 1 \\ 0.5 \end{smallmatrix}\right), 1.3 \cdot 10^{-8}\right), \ldots$ We therefore obtain a false result for the first time interval, since $\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ is not a forward maximizer. By coincidence, the remaining elements in the sequence are forward maximizers and are therefore correct up to a relative error of $8 \cdot 10^{-10}$. A total of 603 support vectors were computed.

The computational cost of the MSEQ algorithm is dominated by the fact that every evaluation of the root function requires solving a linear program. In our experience with SpaceEx, a stateful LP solver can solve an LP with (slowly) changing cost function very quickly if the constraints remain the same. Indeed, solving an LP of this form is the most frequently executed operation in the support function reachability algorithm of SpaceEx.

The algorithm may overlook changes in maximizers if y_k is a maximizer at t and $t + \delta$, and the difference in the root function is smaller than ε. The likelihood of overlooking a maximizer can be reduced by choosing smaller δ and ε. A sufficient criterion for detecting an overlooked change will be discussed in Sect. III-A.3. The algorithm is sure to terminate: If $\mathcal{P}$ is a polytope, the changes in the maximizer function $y^*(t)$ are generated from zero crossings of affine functions of the trajectories of the vertices of $\mathcal{P}$. So over a bounded time horizon T, there is a maximizer sequence with a finite number of elements, which also implies that there is a minimum time step that separates the t_k.

The above algorithm may require several backtracking steps (no more than vertices in $\mathcal{P}$) to stabilize on the maximizer y_k. The following section presents a criterion to identify a maximizer that will be valid for some positive time interval, thus rendering the test in step 3e unnecessary.

2) Finding a forward maximizer: The set of forward maximizers can be constructed according to the following lemma, whose proof can be found in [16].

Lemma 3.2: Let $\mathcal{P}$ be a nonempty polytope, $\mathcal{P}_0 = \mathcal{P}$, and for $i = 0, \ldots, n-1$,

$$\mathcal{P}_{i+1} = \mathcal{P}_i \cap \left\{ y \;\middle|\; \ell^{\mathsf{T}} A^i y = \rho_{\mathcal{P}_i}(A^{i\top} \ell) \right\}.$$

Then y is a forward maximizer in direction ℓ iff $y \in \mathcal{P}_n$. Furthermore, $\mathcal{P}_n$ is nonempty.

Lemma 3.2 provides a constructive way of finding a forward maximizer. Naively applied to a polytope $\mathcal{P}$, it requires solving n linear programs, each with n variables and up to $m+n$ constraints, where m is the number of constraints of $\mathcal{P}$. For a more efficient approach, one could exploit that the constraint added to $\mathcal{P}_i$ reduces the search space to a $n-i$ dimensional face of $\mathcal{P}$. Instead of adding constraints, a modified simplex algorithm could fix the basis constraints accordingly.

3) Using normal cones: Let $\mathcal{P}$ be a polytope

$$\mathcal{P} = \Big\{ y \;\Big|\; \bigwedge_{i=1}^{m} p_i^\mathsf{T} y \le q_i \Big\},$$

so we can construct the normal cones explicitly and use them for root functions.

If y_k is a nondegenerate vertex, i.e., only n constraints are active in y_k, the normal cone of y_k is spanned by the n linearly independent normal vectors of the active constraints, say $p_1, \ldots, p_n$. This is a simplical cone, so with (5) we have $\ell_t \in \mathcal{N}_\mathcal{P}(y_k)$ iff

$$(p_1 \cdots p_n)^{-1} \ell_t \ge 0. \tag{17}$$

Figures 2 and 5 show the normal cones of $\mathcal{X}_t$, and illustrate how the direction identifies the maximizers for $\ell^*(t) = \ell$, respectively $\ell^*(t) = \ell_t$.

Let $\lambda(\ell) \in \mathbb{R}^n$ be $\lambda(\ell) = (p_1 \cdots p_n)^{-1}\ell$. Then λ is a vector of root functions that can replace the root function in Sect. III-A.1. The cost is $O(n^3)$ for the matrix inversion (necessary each time we choose a new maximizer) and $O(n^2)$ for the matrix-vector product (necessary at each time step).

If y_k is a degenerate vertex, then $m_a > n$ constraints are active in y_k, say $p_1, \ldots, p_{m_a}$. Then $\ell_t \in \mathcal{N}_\mathcal{P}(y_k)$ iff there exists $\lambda_1, \ldots, \lambda_{m_a} \ge 0$ such that

$$\sum_{i=1}^{m_a} \lambda_i p_i = \ell_t. \tag{18}$$

This can be solved as a (m_a, n)-LP at every iteration of the root solver. With Farkas' Lemma, it is equivalent to

$$\min \Big\{ \ell_t^\mathsf{T} y \;\Big|\; \bigwedge_{i=1}^{m_a} p_i^\mathsf{T} y \ge 0 \Big\} \ge 0. \tag{19}$$

One possibility is therefore to use the left side of (19) as a root function, which involves solving a (n, m_a)-LP at every iteration of the root solver. The difference between (18) and (19) is that the changing parameter is part of the constraints in one and part of the cost function in the other. As noted before, a stateful LP solver typically performs well on a changing cost function if the constraints remain the same, so we expect (19) to perform better.

A potential problem with using normal cones as root functions is that ℓ_t may graze tangentially to a facet of the normal cone. In this case, the root function is identically zero over a time interval. We can use the same remedy as in Sect. III-A.1 and use $\lambda(\ell) + \varepsilon$ followed by backtracking if a crossing has been detected (note that the sign of $\lambda(\ell)$ is opposite compared to Sect. III-A.1).

In terms of performance, the difference between the root function in Sect. III-A.1 and (19) is that the former requires solving an LP with all constraints of $\mathcal{X}_0$, while the latter only involves the subset of constraints that are active at the current maximizer. If the maximizer is a nondegenerate vertex, using (17) can be significantly faster since at every time step it is only $O(n^2)$, and $O(n^3)$ only when the maximizer changes.

Constructing nomal cones can also help to detect if a third maximizer lies between y_k and y_{k+1}. The normal cones of two successive maximizers overlap, so if $\mathcal{N}_\mathcal{P}(y_k) \cap \mathcal{N}_\mathcal{P}(y_{k+1}) = \emptyset$, a search for another maximizer between t_k and t_{k+1} is required. The check can be carried out by solving an LP with n variables and $2m_a$ constraints. Problems similar to (18) and (19) have been studied in the literature on parametric linear programming. An algorithm for finding a forward maximizing normal cone (called compatible basis) is sketched out in [20].

B. Fixed-Direction Maximizer Algorithm

We now combine the maximizer sequence with ODE solving to obtain the final algorithm for computing a fixed-direction maximizer at a given point in time. The following algorithm $\mathrm{ODEMFD}_A(\mathcal{X}_0, t_f, \ell, \varepsilon, \delta)$ takes as arguments a set of initial states $\mathcal{X}_0$, a final time t_f, a direction ℓ, a threshold $\varepsilon > 0$, a maximum time step $\delta > 0$ and returns the fixed-direction maximizer $x^*_{fd} = x^*(t_f)$:

1) matrix exponential:
   $e_i(t_f) = \mathrm{ODE}_A(e_i(0), 0, t_f)$ for $i = 1, \ldots, n$;
   $M = \big(e_1(t_f), \ldots, e_n(t_f)\big)$
2) autonomous maximizer:
   $\ell' = M^\mathsf{T}\ell$, $x^*_0 = \hat{\sigma}_{\mathcal{X}_0}(\ell')$, $x_{\mathrm{aut}} = M x^*_0$.
3) maximizing input sequence:
   $(u_k, t_k)_{k=0,\ldots,K} = \mathrm{MSEQ}_A(\mathcal{U}, 0, t_f, \varepsilon, \delta)$
4) nonautonomous maximizer:
   $x_{\mathrm{inp}} := 0$. Let $t_{K+1} = t_f$. For $k = 0, \ldots, K$, let
   $x_{\mathrm{inp}} := \mathrm{ODE}_A(x_{\mathrm{inp}}, u_k, t_{k+1})$.
5) $x^*_{fd} = x_{\mathrm{aut}} + x_{\mathrm{inp}}$.

All computations in the above procedure can be made incremental, i.e., having computed $x^*(t')$, we may compute $x^*(t'')$, $t'' > t'$, by starting from $t = t'$ instead of $t = 0$. The techniques from Sect. III-A (forward maximizers, normal cones) can be used to reduce redundant computations of $\hat{\sigma}_{\mathcal{X}_0}(\ell')$, and to identify the time points at which the maximizer of $\mathcal{X}_0$ changes.

In practice, the ODE solver might not return the exact switching times t_k for the input sequence u_k. The resulting approximate, i.e., suboptimal, sequence (u_k, t_k) voids the use of $x^*(t)$ for creating an overapproximation of $\mathcal{X}_t$ in (7), but $x^*(t)$ is nonetheless a valid underapproximation of $\mathcal{X}_t$.

Example: We illustrate the ODEMFD algorithm with a helicopter model, taken from [21], 8-dim. flight dynamics, an 20-dim. H_∞-controller, and a clock variable. The goal of the controller is to attenuate wind disturbances, which we model with $\mathcal{U}$ as $[-0.01, 0.01]$ in the dimensions that represent longitude, latitude, and altitude, and 0 in all other dimensions except the clock. The initial states of the helicopter are $[0, 0.1]$ in all dimensions, while the controller and clock variables are zero. We construct the maximizer trajectories in the positive and negative axis directions, see Fig. 4, running

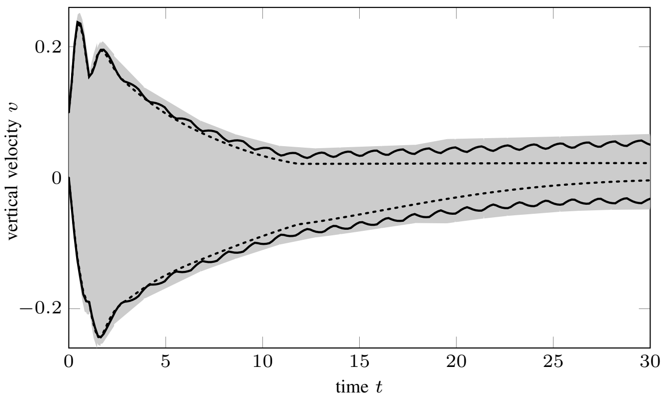


Fig. 4. Maximizer trajectories for a 29-dim. helicopter model, shown in the directions of positive and negative vertical speed for a nondeterministic continuous wind disturbance (solid) and without the disturbance (dotted). A reachable set approximation obtained with the same number of support function evaluations is shown for comparison (shaded)

ODEMFD incrementally with a time step of $h = 0.15$. The computation takes 21 s for all 58 trajectories, with 291,877 evaluations of the support vectors of the initial states and the input set that account for 17% of the runtime. The maximizer sequences have a length between 1 and 47.

For comparison, Fig. 4 also shows a conservative approximation of the reachable states computed using the approach in [11] for the same template directions. It consists of the outer polyhedral approximations of a piecewise linear approximation of the support function. An error bound $\epsilon = 0.05$ in each template direction was chosen to give the same number of support evaluations, and the construction takes 14 s for approximating the support functions and 3 s for constructing the convex polyhedra.

C. Maximizers for Mapped Sets

Assume we wish to compute a maximizer trajectory in direction $\ell(t)$ for a set $\mathcal{P}$ that is the result of a linear map, i.e., $\mathcal{P} = M\mathcal{Q}$. If M is not invertible, computing $\mathcal{P}$ involves a potentially costly projection operation, which may in addition produce degenerate vertices. We can avoid explicitly mapping $\mathcal{Q}$ to $\mathcal{P}$ by first finding a maximizer trajectory $z^*(t)$ of $\mathcal{Q}$ in direction $M^\mathsf{T}\ell(t)$, and then mapping the sequence with (4) to $y^*(t) = Mz^*(t)$. This may be useful, e.g., when the dynamics are of the form $\dot{x} = Ax + B\mathcal{V}$, such that in (8) we have $\mathcal{U} = B\mathcal{V}$.

More generally, if we have a set $\mathcal{P}$ that can be obtained through any combination of linear map, Minkowski sum, and convex hull from sets $\mathcal{Q}_i$, we can compute maximizer trajectories for the $\mathcal{Q}_i$ and then derive the maximizer trajectory for $\mathcal{P}$ by pointwise applying the rules in (4). As an example, a zonotope is the affine image of a hypercube, for which computing with (17) is particularly simple. Applying the affine map we obtain a maximizer sequence for the zonotope. As another example, let $\mathcal{P}$ be symmetric to y_c, i.e., $-\mathcal{P} \oplus \{y_c\} = \mathcal{P} \oplus \{-y_c\}$, and let $y^*(t)$ maximize $\mathcal{P}$ in direction $\ell(t)$. Then $z^*(t) = 2y_c - y^*(t)$ maximizes $\mathcal{P}$ in direction $-\ell(t)$.

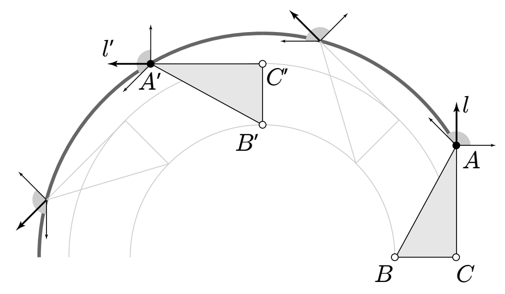


Fig. 5. A given direction ℓ with maximizer $x(0)$ defines a direction $\ell(t)$ with maximizer $x(t)$. The trajectory of state $x(t)$ with $x(0) = A$ (bold grey) is a maximizer of ABC in the evolving direction $\ell(t)$. E.g., at time t', $x(t') = A'$ and $\ell(t') = \ell'$. The normal cone $N_\mathcal{P}(x(t))$ (shown in light grey) is the set of all directions for which $x(t)$ is a maximizer

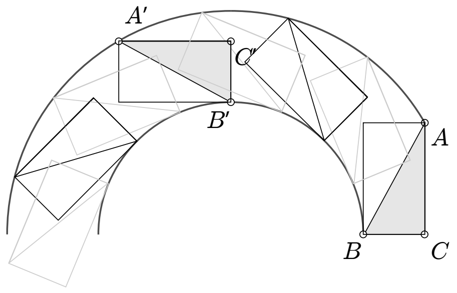


Fig. 6. A flowpipe approximation constructed from dynamic-direction maximizer trajectories starting with the axis directions. At every time instant t, the approximation is the bounding box of the reachable set $\mathcal{X}_t$, rotated to match the rotation of $\mathcal{X}_t$. The maximizer trajectories are $x(t)$ with $x(0) = A$ and $x(t)$ with $x(0) = B$ as illustrated in Fig. 5

IV. Dynamic-Direction Maximizer Trajectories

A dynamic-direction maximizer trajectory for a direction ℓ and any $x_0 \in \sigma_{\mathcal{X}_0}(\ell)$ is given by [7], [4]:

$$\dot{x}^*(t) = Ax^*(t) + u^*(t), \qquad x^*(0) = x_0, \tag{20}$$

$$\dot{\ell}^*(t) = -A^\mathsf{T}\ell^*(t), \qquad \ell^*(0) = \ell, \tag{21}$$

$$u^*(t) \in \sigma_\mathcal{U}(\ell^*(t)). \tag{22}$$

An example maximizer trajectory is shown in Fig. 5. Figure 6 shows a flowpipe approximation constructed from dynamic-direction maximizers, starting at $t = 0$ with the axis directions (bounding box). Note that for a given direction ℓ one is free to choose any maximizer of ℓ as x_0. The difficulty in solving (20)–(22) lies with the last equation: We need to find a function $u^*(t)$ that satisfies (22). With (21), we have

$$\ell^*(t) = e^{-A^\mathsf{T}t}\ell_0 = \ell_{-t}.$$

We must therefore find for each t a point $u^*(t)$ in a polyhedron $\mathcal{U}$ that is a maximizer in direction ℓ_{-t}. Again, this is the problem discussed in Sect. III-A, but going backwards in time, or, equivalently, substituting A by $-A$. This points towards a potential problem: If A is stable, then $-A$ is unstable, and computing ℓ_{-t} may be numerically challenging. Since the length of ℓ_{-t} is irrelevant for computing $\sigma_\mathcal{U}(\ell_{-t})$, we could instead use $\nu = \ell_{-t}/\|\ell_{-t}\|$. It can be obtained by solving the nonlinear ODE,

$$\dot{\nu} = -A^\mathsf{T}\nu + (\nu^\mathsf{T}A^\mathsf{T}\nu)\nu, \quad \nu(0) = \ell/\|\ell\|. \tag{23}$$

As with fixed-direction maximizers, a suboptimal function $u^*(t)$ voids the use of $x^*(t)$ for constructing an overapproximation of $\mathcal{X}_t$, but it is nonetheless a valid underapproximation. Contrary to a fixed-direction maximizer, the maximizer trajectory $x^*(t)$ is continuous and an actual behavior of the system.

A. Dynamic-Direction Maximizer Algorithm

The following algorithm $\mathrm{ODEMDD}_A(\mathcal{X}_0, t_f, \ell, \varepsilon, \delta)$ takes as arguments a set of initial states $\mathcal{X}_0$, a final time t_f, a direction ℓ, a threshold $\varepsilon > 0$, a maximum time step $\delta > 0$ and returns the dynamic-direction maximizer $x^*_{dd} = x^*(t_f)$:

1) autonomous maximizer: $x_0 = \hat{\sigma}_{\mathcal{X}_0}(\ell)$, $x_{\mathrm{aut}} = \mathrm{ODE}_A(x_0, 0, t_f)$.
2) maximizing input sequence: $(u_k, t_k)_{k=0,\ldots,K} = \mathrm{MSEQ}_{-A}(\mathcal{U}, 0, t_f, \varepsilon, \delta)$
3) nonautonomous maximizer: $x_{\mathrm{inp}} := 0$. Let $t_{K+1} = t_f$. For $k = 0, \ldots, K$, let $x_{\mathrm{inp}} := \mathrm{ODE}_A(x_{\mathrm{inp}}, u_k, t_{k+1})$.
4) $x^*_{dd} = x_{\mathrm{aut}} + x_{\mathrm{inp}}$.

As in Sect. III-B, the procedure can be made incremental. For the autonomous part, the computation is easier, since for a given ℓ we need to find $x_0 \in \sigma_{\mathcal{X}_0}(\ell)$ only once, not for every value of t_f.

In the special case where $\mathcal{X}_0$ is a polytope with constraints $\bigwedge_{i=1}^m p_i^\mathsf{T} x \leq q_i$ and $\mathcal{U}$ is a singleton set $\mathcal{U} = \{u_0\}$, computing dynamic-direction maximizers in directions $\ell^1 = p_1, \ldots, \ell^m = p_m$ produces the exact reachable set $\mathcal{X}_t$:

$$\mathcal{X}_t = \bigcap_{i=1}^m \left\{ x \;\middle|\; {\ell_t^i}^\mathsf{T} x \leq {\ell_t^i}^\mathsf{T} x^*(t) \right\}.$$

In this case, each x_0 is a witness for at least n directions (the normal vectors of constraints active in x_0), so at most m/n maximizer trajectories need to be computed. If $\mathcal{U}$ is not a singleton, the dynamic-direction maximizers define a tight outer approximation of the reachable set, just like fixed-direction maximizers.

B. Maximizers and Backwards Reachability

Starting from a set of of initial states $\mathcal{X}_0$, the *backwards reachable set* consist of the solutions

$$\dot{x}(t) = -Ax(t) - u(t), \qquad x(0) \in \mathcal{X}_0, u(t) \in \mathcal{U}. \tag{24}$$

To perform backwards instead of forwards reachability, it therefore suffices to substitute $-A$ for A and $-\mathcal{U}$ for $\mathcal{U}$. Applying this substitution to (20)–(22), we obtain a connection between forward and backward reachability: Having computed a maximizing input function $u^*(t)$ for a fixed-direction maximizer trajectory in direction ℓ, a dynamic-direction maximizer trajectory going backwards in time can be obtained using the same $u^*(t)$ by solving

$$\dot{x}^*(t) = -Ax^*(t) - u^*(t).$$

Similarly, a maximizing input for a dynamic-direction maximizer can be used to obtain a fixed-direction maximizer trajectory going backwards in time.

References

[1] C. Le Guernic and A. Girard, "Reachability analysis of linear systems using support functions," *Nonlinear Analysis: Hybrid Systems*, vol. 4, no. 2, pp. 250 – 262, 2010, iFAC World Congress 2008.

[2] G. Frehse, S. Bogomolov, M. Greitschus, T. Strump, and A. Podelski, "Eliminating spurious transitions in reachability with support functions," in *HSCC'15*. ACM, 2015, pp. 149–158.

[3] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle, "Formal analysis of timing effects on closed-loop properties of control software," in *Real-Time Systems Symposium (RTSS)*. IEEE, 2014, pp. 53–62.

[4] A. V. Lotov, V. A. Bushenkov, and G. K. Kamenev, *Interactive Decision Maps*, ser. Applied Optimization. Kluwer Academic Publishers, Boston, 2004, vol. 89.

[5] G. Van Den Bergen, *Collision detection in interactive 3D environments*. Elsevier, 2004.

[6] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald, "Abstraction and counterexample-guided refinement in model checking of hybrid systems," *International Journal of Foundations of Computer Science*, vol. 14, no. 04, pp. 583–604, 2003.

[7] P. Varaiya, "Reach set computation using optimal control," in *Proc. KIT Workshop*, 1997, pp. 377–383.

[8] A. Girard and C. Le Guernic, "Efficient reachability analysis for linear systems using support functions," in *IFAC World Congress*, 2008.

[9] E. Asarin, O. Bournez, T. Dang, and O. Maler, "Approximate reachability analysis of piecewise-linear dynamical systems," in *Hybrid Systems: Computation and Control*. Springer, 2000, pp. 20–31.

[10] M. Egerstedt, Y. Wardi, and F. Delmotte, "Optimal control of switching times in switched dynamical systems," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 3. IEEE, 2003, pp. 2138–2143.

[11] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *CAV*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 379–395.

[12] G. Frehse, R. Kateja, and C. Le Guernic, "Flowpipe approximation and clustering in space-time," in *HSCC'13*. ACM, 2013, pp. 203–212.

[13] S. Sankaranarayanan, T. Dang, and F. Ivančić, "Symbolic model checking of hybrid systems using template polyhedra," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 188–202.

[14] E. Asarin, T. Dang, O. Maler, and R. Testylier, "Using redundant constraints for refinement," in *Automated Technology for Verification and Analysis*. Springer, 2010, pp. 37–51.

[15] M. A. B. Sassi, R. Testylier, T. Dang, and A. Girard, "Reachability analysis of polynomial systems using linear programming relaxations," in *Automated Technology for Verification and Analysis*. Springer, 2012, pp. 137–151.

[16] G. Frehse, "Computing maximizer trajectories of affine dynamics for reachability," Verimag, Tech. Rep. TR-2015-10, September 2015.

[17] N. T. B. Kim and D. T. Luc, "Normal cones to a polyhedral convex set and generating efficient faces in linear multiobjective programming," *Acta Mathematica Vietnamica*, vol. 25, pp. 101–124, 2000.

[18] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, "Sundials: Suite of nonlinear and differential/algebraic equation solvers," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 363–396, 2005.

[19] A. Makhorin, "GNU Linear Programming Kit, v.4.37," 2009, http://www.gnu.org/software/glpk.

[20] A. Holder, "Parametric LP analysis," in *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, 2010.

[21] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, 2005.