Journal of Nonlinear Analysis
Special Issue on Hybrid Systems
pp. 1453-1465, Vol. 62, 2005.

On Time Optimal Control of Integrator Switched Systems with State Constraints

Xuping Xu[1] and Panos J. Antsaklis[2]

[1] Corresponding Author. Department of Electrical and Computer Engineering
Penn State Erie, Erie, PA 16563, USA. E-mail: Xuping-Xu@psu.edu

[2] Department of Electrical Engineering, University of Notre Dame
Notre Dame, IN 46556, USA. E-mail: antsaklis.1@nd.edu

Abstract: In this paper, time optimal control problems of a class of integrator switched systems with polyhedral state constraint subsets are studied. We first develop a directed graph representation of the system discrete structure. Based on the graph representation, we generate candidate solution paths and propose an algorithm for seeking the optimal solution. A linear programming method for finding the optimal timing information for each path is then proposed. Finally, we report preliminary results on some sufficient conditions and techniques which help reduce the number of candidate paths.

Key Words: Switched Systems; Hybrid Systems; Integrator Systems; Time Optimal Control; Linear Programming

1 Introduction

Recently, optimal control problems of switched and hybrid systems have attracted many researchers from various fields in science and engineering, due to their theoretical challenges and importance in real-world applications (see, e.g., [2, 4, 10, 15] and the references therein). Due to the diversity and complexity of such problems, there has been no general solution technique either theoretically or numerically. However, many approaches have been developed to solve special classes of such problems (e.g., [3, 5, 7, 9, 11, 13]).

In this paper, we study time optimal control problems of switched systems consisting of integrator subsystems with polyhedral state constraint subsets. Such problems are worth studying due to the followings. First, many real-world processes such as chemical batch processes can be modelled as integrator switched systems [12, 14] and time optimality is a common control criterion. Second, such problems are among the few classes which we can develop a theoretical framework for. Third, the study of such problems will shed light on general problems and on many important research topics such as controllability of integrator switched systems [14].

Such time optimal control problems, though simple in appearance, actually present difficulties due to the involvement of the discrete dynamics. Even though the subsystem dynamics are simple, the overall system behavior is no longer linear. In this paper, we pay attention to not only the continuous dynamics but also the discrete dynamics. The main results of the paper concern both the discrete structure and the continuous evolution of the system and are as follows. We first develop a directed graph representation of the system discrete structure. Based on it, we generate candidate solution paths and propose an algorithm which seeks the optimal solution among all candidate paths. A linear programming method for finding the optimal timing information for each path is then proposed. Besides these results, we also report a sufficient condition for eliminating infeasible paths, a sufficient condition for reducing the looping times, and techniques for dealing with zigzagging loops. These conditions and techniques can help reduce the number of candidate paths.

Note that, in the computer science community, linear hybrid systems similar to integrator switched systems have also been studied [1]. However, our focus here is on control synthesis as opposed to verification in [1]. With different objectives in mind, our approach, which utilizes the discrete structure of the systems and uses linear programming extensively, is different from the method in [1]. We believe that our result contributes to controller design of hybrid systems and is a first step towards a more general and more efficient solution of such time optimal control problems.

2 Problem Formulation

A *switched system* is a particular kind of hybrid system that consists of subsystems[*]

$$\dot{x} = f_i(x), \quad f_i : X_i \to \mathbb{R}^n, \ X_i \subseteq \mathbb{R}^n, \ i \in I = \{1, 2, \cdots, M\} \tag{1}$$

where f_i is the vector field and X_i is the state constraint set for the i-th subsystem, and a switching law orchestrating the active subsystem at each instant. The trajectory of a switched system is determined by the initial state and the timed sequence of active subsystems. A *switching sequence* defined as follows regulates the timed sequence of active subsystems.

Definition 1 (Switching Sequence) *A switching sequence σ in $[t_0, t_f]$ is defined as*

$$\sigma = \big((t_0, i_0), (t_1, i_1), \cdots, (t_K, i_K)\big) \tag{2}$$

where $0 \le K < \infty$, $t_0 \le t_1 \le \cdots \le t_K \le t_f$, $i_k \in I$ for $k = 0, 1, \cdots, K$. □

[*]The term 'subsystem' is widely used in switched systems literatures (e.g., [8]), although 'mode' might be better than 'subsystem' here as the system dynamics can be different but the state variables remain the same.

σ defined in (2) indicates that the system starts with subsystem i_0 at t_0, and switches to subsystem i_k from i_{k-1} at instant t_k for $1 \le k \le K$. We only consider **nonZeno** sequences which switch at most a finite number of times in any finite interval $[t_0, t_f]$. Given an initial $x(t_0)$, we call a switching sequence σ in $[t_0, t_f]$ *admissible* if it is nonZeno and the system under it generates a state trajectory $x(t)$ for $t \in [t_0, t_f]$ satisfying the conditions $x(t_k) \in X_{i_{k-1}} \cap X_{i_k}$ and $x(t) \in X_{i_k}$ for $t \in [t_k, t_{k+1})$. Finally, we note that the continuous state of a switched system does not exhibit jumps at switching instants.

2.1 Integrator Switched Systems with State Constraints

Before we define the class of switched systems we will study in the sequel, let us first introduce the following notion.

Definition 2 (Polyhedral Subset) *A polyhedral subset P of $\mathbb{R}^n$ is a set of form*

$$P = \{x \in \mathbb{R}^n | Ax \le b,\ A \in \mathbb{R}^{r\times n}, b \in \mathbb{R}^r\}. \tag{3}$$

□

Remark 1 In (3), the inequality is in the componentwise sense, i.e., P is the intersection of the closed halfspaces $\{x \in \mathbb{R}^n | a_j^T x \le b_j\}$, $j = 1, \cdots, r$, where a_j^T is the j-th row of the matrix A and b_j is the j-th component of the column vector b. In view of this, a polyhedral subset is a subset defined by (finitely many) linear inequalities. Also note that a polyhedral subset is a closed, convex subset of $\mathbb{R}^n$. □

In this paper, we are particularly interested in a class of integrator switched systems whose subsystems have the form

$$\dot{x} = \alpha_i,\ x \in P_i \tag{4}$$

where each P_i is a polyhedral subset of $\mathbb{R}^n$.

Example 1 The following switched system in $\mathbb{R}^2$ is an example of an integrator switched system. It consists of 4 subsystems: subsystem 1: $\dot{x} = [-1,\ 0.25]^T$, P_1=$\{[x_1, x_2]^T \in \mathbb{R}^2 | 0 \le x_1 \le 3,\ 0 \le x_2 \le 1\}$; subsystem 2: $\dot{x} = [0.25,\ 1]^T$, P_2=$\{[x_1, x_2]^T \in \mathbb{R}^2 | 0 \le x_1 \le 1.5,\ 0 \le x_2 \le 3\}$; subsystem 3: $\dot{x} = [1,\ -0.25]^T$, P_3=$\{[x_1, x_2]^T \in \mathbb{R}^2 | 0 \le x_1 \le 3,\ 2 \le x_2 \le 3\}$; subsystem 4: $\dot{x} = [-0.25,\ -1]^T$, P_4=$\{[x_1, x_2]^T \in \mathbb{R}^2 | 1.5 \le x_1 \le 3,\ 0 \le x_2 \le 2\}$. Figure 1 shows the polyhedral subsets and the corresponding vector fields. □

Note that many chemical batch processes can be modelled as such integrator systems [14]. In batch processes, each combination of valve positions (on or off) corresponds to an integrator subsystem. Due to certain logical constraints which specify the conditions under which each combination is allowable, each subsystem will have certain polyhedral state constraint subsets. The system in Example 1 can model a two tank system with the continuous states corresponding to water levels and each subsystem corresponding to a valve position combination.

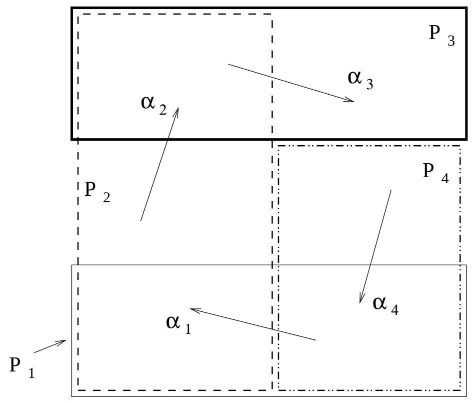

Figure 1: The polyhedral subsets and vector fields for Example 1.

2.2 Time Optimal Control Problem

Problem 1 (Time Optimal Control Problem) *Consider an integrator switched system (4) with polyhedral state constraint subsets P_i, $i \in I$. For any $x_0,\ x_f \in X \overset{\triangle}{=} \bigcup_{i\in I} P_i$, find an admissible switching sequence which transfers the continuous state from x_0 to x_f in minimum amount of time.* □

Remark 2 In [11], time optimal control problems of switched systems without state constraints are studied and a closed-loop control strategy is proposed. The idea in [11] is to partition the state space into regions corresponding to subsystems to be activated. However, up to now, there has been no general approach for problems with state constraints. □

Example 2 We will study the following example problem in the sequel. Consider the system in Example 1. Given $x_0 = [2,\ 0.5]^T$ and $x_f = [2,\ 2.5]^T$, find an admissible switching sequence which drives the continuous state from x_0 to x_f in minimum amount of time. □

In the sequel, we call Problem 1 *feasible* if there exists at least one switching sequence that leads the system trajectory from x_0 to x_f.

3 A Linear Programming Approach

Now we propose a linear programming approach for Problem 1. Our approach consists of several steps as follows where each step may utilize linear programming.

3.1 Directed Graph Representation of System Discrete Structure

An important question regarding the discrete structure of a switched system is whether the system can switch from one given subsystem to another. To characterize such discrete structural information, here we introduce a directed graph representation. In order to obtain such a representation, we need the following notions.

Definition 3 (Adjacent Polyhedral Subsets) *Two polyhedral subsets P_{i_1} and P_{i_2} are said to be adjacent if $P_{i_1} \cap P_{i_2} \neq \emptyset$.* □

The following linear programming feasibility problem can be solved to determine the adjacency of two polyhedral subsets $P_{i_1} = \{x | A_{i_1} x \leq b_{i_1}\}$ and $P_{i_2} = \{x | A_{i_2} x \leq b_{i_2}\}$.

$$\exists x \text{ such that } \begin{cases} A_{i_1} x \leq b_{i_1} \\ A_{i_2} x \leq b_{i_2} \end{cases} ? \tag{5}$$

The two subsets are adjacent if and only if there exists a feasible solution to problem (5).

Definition 4 (Legal Successor) *Subsystem i_2 is said to be a legal successor of i_1 if*
(a). P_{i_1} and P_{i_2} are adjacent, and
(b). there exists $x \in P_{i_1} \cap P_{i_2}$ and $\tau > 0$ s.t. $x + \tau \alpha_{i_2} \in P_{i_2}$. □

Remark 3 An alternative interpretation of subsystem i_2 being a legal successor of i_1 is that there exists a point in the intersection of the two polyhedral subsets from where subsystem i_2 can be activated for a nonzero amount of time. □

The following linear programming problem can be solved to determine whether subsystem i_2 is a legal successor of subsystem i_1 (assuming P_{i_2} is adjacent to P_{i_1}).

$$\max_{\tau \in \mathbb{R}, x \in \mathbb{R}^n} \tau + \mathbf{0}^T x \tag{6}$$

$$\text{s.t.} \begin{cases} A_{i_1} x \leq b_{i_1} \text{ and } A_{i_2} x \leq b_{i_2} \\ A_{i_2}(x + \tau \alpha_{i_2}) \leq b_{i_2} \\ \tau \geq 0 \end{cases} \tag{7}$$

Subsystem i_2 is a legal successor of i_1 if and only if the optimal solution τ to (6)-(7) is greater than 0.

We can now depict the system's discrete structure using a directed graph representation in which each node corresponds to a subsystem i and a directed branch exists from node i_1 to i_2 if and only if subsystem i_2 is a legal successor of subsystem i_1. The following is an example.

Example 3 By the methods above for pairs of subsystems, we construct the directed graph representation for the system in Example 1 in Figure 2. In Figure 2, P_1 and P_3 are not adjacent; P_3 and P_4 are adjacent, however, only subsystem 4 is a legal successor of 3 but not vice versa. Although P_2 and P_4 are adjacent, neither subsystem is a legal successor of the other. □

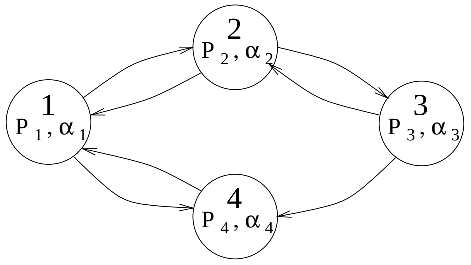

Figure 2: Example 3: a directed graph representation.

Remark 4 The directed graph representation can clearly lay out the system discrete structure and help eliminate impossible discrete node sequences. Note that in Chapter 2 of [14], Tittus also proposed a directed graph representation for hybrid systems; in [1], hybrid automata for linear hybrid systems were also modelled as a similar transition graph. However, the graphs in [1, 14] are obtained from the mode transition functions instead of being determined from the continuous constraint subset information as we did above. Our construction of the directed graph representation actually shows how to obtain the possible mode transition functions. □

Having the directed graph representation, we can determine all possible node sequences (i.e., sequences of active subsystems) starting from a given node i_0 and ending at a given node i_f by enumerating all possible paths from i_0 to i_f.

Definition 5 (Path) *A path from node i_0 to i_f ($i_0, i_f \in I$) is a sequence*

$$\pi = (i_0, i_1, i_2, \cdots, i_{K-1}, i_f) \tag{8}$$

in which each node i_k is a legal successor of i_{k-1}, $1 \leq k \leq K$ (we denote i_f also as i_K). □

Definition 6 (Loop) *A path $\lambda = (i_0, i_1, \cdots, i_m, i_{m+1})$ is called a loop if $i_0 = i_{m+1}$ ($m \geq 1$). A loop is elementary if i_0 and i_{m+1} are the only pair of repeated nodes. If more than one pair of repeated nodes can be found in a loop, the loop is said to be nonelementary.* □

The presence of loops makes Problem 1 difficult because the number of possible paths from one node to another might be infinite if loops are allowed in paths. A path is said to contain a loop if one and the same node appear more than once in it. A path is called an *elementary path* if it contains no loop, otherwise it is called a *nonelementary path*.

Any nonelementary path can be obtained by adding certain number of loops to some elementary path. Given each elementary path $(i_0, i_1, \cdots, i_{K-1}, i_f)$ from node i_0 to i_f, we can add an elementary loop $(i_j, i_{l_1}, i_{l_2}, \cdots, i_{l_m}, i_j)$ to obtain the generic expression of a nonelementary path $\pi = \big(i_0, i_1, \cdots, i_j, (i_{l_1}, i_{l_2}, \cdots, i_{l_m}, i_j)^*, i_{j+1}, \cdots, i_{K-1}, i_f\big)$ where the operator $(\cdot)^*$ means that the sequence inside the parenthesis can be repeated an arbitrary number of times. Furthermore, we can add more elementary loops to π to form more complicated nonelementary paths containing several loops or nested loops.

Example 4 Consider Figure 2, there is one elementary path $\pi = (1, 2, 3)$ from node 1 to 3. There are 4 elementary loops $\lambda_1 = (1, 4, 1)$, $\lambda_2 = (1, 2, 1)$, $\lambda_3 = (2, 3, 2)$, $\lambda_4 = (1, 2, 3, 4, 1)$. We can construct all possible path expressions by adding elementary loops or combination of them to the elementary paths to obtain infinite path expressions from node 1 to 3, e.g.,
$\pi_1 = \big(1, (4, 1)^*, (2, 1)^*, 2, (3, 2)^*, 3\big)$,
$\pi_2 = \Big(1, (4, 1)^*, \big(2, (3, 2)^*, 1\big)^*, 2, (3, 2)^*, 3\Big)$, $\pi_3 = \Big(1, (4, 1)^*, 2, \big(1, (4, 1)^*, 2\big)^*, (3, 2)^*, 3\Big)$,
$\pi_4 = \Big(1, \big((4, 1)^*, (2, 1)^*, 2, (3, 2)^*, 3, 4, 1\big)^*, (4, 1)^*, (2, 1)^*, 2, (3, 2)^*, 3\Big), \cdots$. □

3.2 An Algorithm

Given x_0 and x_f, we can determine the possible initial subsystem i_0 that satisfies $x_0 \in P_{i_0}$ and similarly the possible final subsystem i_f (there may be several possible i_0's and i_f's). Possible candidate paths that take the system from subsystem i_0 to i_f can then be obtained from the directed graph representation. There might be more than one such possible paths and some paths might also contain loops. If x_0 and/or x_f belong to multiple polyhedral subsets, then multiple paths with different starting and/or ending nodes should be considered. Once we have the candidate paths, the following algorithm can be applied to solve Problem 1.

Algorithm 1

(1). Construct the directed graph representation.
(2). Based on the directed graph representation, select a candidate path $(i_0, i_1, \cdots, i_{K-1}, i_f)$, where $x_0 \in P_{i_0}$ and $x_f \in P_{i_f}$, find the optimal timing information in this case.
(3). Vary the candidate path and repeat step (2) so as to find the global optimal solution.
□

Candidate paths can be obtained from generic path expressions by substituting each $*$ with a specific integer number (including 0). Due to the possibility of infinitely many candidate paths, step (3) poses difficulty because it may require the repetition of step (2) infinitely many times. Such difficulty may hinder us from finding the global minimum. In practice, we usually enforce certain upper bound on the number of loops allowed or on the times a loop may repeat. In so doing, Algorithm 1 can terminate after finite steps and provide a suboptimal solution. We will present some preliminary results on reducing the number of candidate paths in Section 4.

3.3 A Linear Programming Method for Finding Optimal Timing

Now we propose a linear programming method to address step (2) in Algorithm 1. Given a candidate path $(i_0, i_1, \cdots, i_{K-1}, i_K)$, we can solve the following linear programming problem to determine the optimal timing at each node i_k, $0 \leq k \leq K$.

$$\min_{\tau_0,\tau_1,\cdots,\tau_K} \tau_0 + \tau_1 + \cdots + \tau_K \tag{9}$$

$$\text{s.t.} \begin{cases} \tau_0\alpha_{i_0} + \cdots + \tau_K\alpha_{i_K} = x_f - x_0 \\ A_{i_0}(x_0 + \tau_0\alpha_{i_0}) \leq b_{i_0} \\ A_{i_1}(x_0 + \tau_0\alpha_{i_0}) \leq b_{i_1} \\ \vdots \\ A_{i_{k-1}}(x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_{k-1}\alpha_{i_{k-1}}) \leq b_{i_{k-1}} \\ A_{i_k}(x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_{k-1}\alpha_{i_{k-1}}) \leq b_{i_k} \\ \vdots \\ A_{i_{K-1}}(x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_{K-1}\alpha_{i_{K-1}}) \leq b_{i_{K-1}} \\ A_{i_K}(x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_{K-1}\alpha_{i_{K-1}}) \leq b_{i_K} \\ \tau_0, \tau_1, \cdots, \tau_K \geq 0 \end{cases} \tag{10}$$

In (9)-(10), τ_k is the time duration at node i_k. The first equality in (10) specifies that the state trajectory goes from x_0 to x_f. The second to the second to the last inequalities are constraints for the state at switching instants $t_1, \cdots, t_K$. To show the meaning of these inequalities, we look at the general terms in the middle two inequalities. In these inequalities, $x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_{k-1}\alpha_{i_{k-1}}$ is the state x at t_k, and the two inequalities can be rewritten as $A_{i_{k-1}}x(t_k) \leq b_{i_{k-1}}$ and $A_{i_k}x(t_k) \leq b_{i_k}$, which guarantee $x(t_k)$ being in $P_{i_{k-1}} \cap P_{i_k}$. Here we only need to be concerned with the state x at the switching instant t_k's, since any $x(t)$ for $t \in [t_k, t_{k+1})$ will always be in P_{i_k} as long as $x(t_k)$ and $x(t_{k+1})$ are in P_{i_k} due to the convexity of the polyhedral subsets.

Software packages such as Matlab or Maple can be used to solve (9)-(10). The given path is infeasible if no feasible solution exists. If a solution exists, then it will provide the optimal time duration τ_k at each node i_k. With this information, we can determine the optimal switching instants $t_k = \sum_{j=0}^{k-1}\tau_j$ for $1 \leq k \leq K$ and the optimal switching sequence $\sigma = \big((t_0, i_0), (t_1, i_1), \cdots, (t_K, i_K)\big)$ (usually we regard $t_0 = 0$). The minimum total time along the given path can also be determined to be $\sum_{k=0}^{K}\tau_k$.

Example 5 Consider the problem in Example 2. Given a candidate path (i_0, i_1, i_2) with $i_0 = 1$, $i_1 = 2$, $i_2 = 3$, by solving (9)-(10), we can find the optimal $\tau_0 = 0.9706$, $\tau_1 = 1.8824$, $\tau_2 = 0.5000$. The optimal switching sequence in this case is $\sigma = \big((0, 1), (0.9706, 2), (2.8529, 3)\big)$ and the minimum time to bring x from x_0 to x_f is 3.3529. □

4 Some Discussion on Paths with Loop

Steps (1) and (2) of Algorithm 1 can be addressed using linear programming methods as detailed in Sections 3.1 and 3.3. However, step (3), which may require the repetition of step (2) infinite times, hinders us from finding the global minimum. Now we propose some preliminary sufficient conditions and techniques which help relieve such difficulty.

4.1 Elimination of Infeasible Nonelementary Loops

The following lemma provides us with a sufficient condition that can help eliminate paths containing certain infeasible nonelementary loops from the candidate path list.

Lemma 1 (Infeasible Nonelementary Loop) *Consider a nonelementary loop* $\lambda = \big(i_0, i_1, \cdots, i_{j-1}, i_j, (i_{j_1}, i_{j_2}, \cdots, i_{j_p}, i_j)^*, i_{j+1}, \cdots, i_m, i_0\big)$ *(*$j \neq 0$*) which is the combination of two elementary loops* $\lambda_1 = (i_0, i_1, \cdots, i_{j-1}, i_j, i_{j+1}, \cdots, i_m, i_0)$ *and* $\lambda_2 = (i_j, i_{j_1}, i_{j_2}, \cdots, i_{j_p}, i_j)$ *(*λ_2 *repeats at least once so that* λ *is nonelementary).* λ *is infeasible if* $\exists$ *a hyperplane* $H = \{x \in \mathbb{R}^n | a^T x = b\}$ *s.t.* $S_1 \triangleq (P_{i_{j-1}} \cap P_{i_j}) \bigcup (P_{i_{j+1}} \cap P_{i_j})$ *and* $S_2 \triangleq (P_{i_{j_1}} \cap P_{i_j}) \bigcup (P_{i_{j_p}} \cap P_{i_j})$ *are in the two different open halfspaces formed by* H.

Proof: Consider the dynamics $\dot{x} = \alpha_{i_j}$ of subsystem i_j. If α_{i_j} is in parallel with H, such dynamics cannot drive the state trajectory from any open halfspace to the other. If α_{i_j} is not in parallel with H, such dynamics can drive the trajectory from one open halfspace to the other, but not vice versa. Therefore, for either case of α_{i_j}, it is impossible to drive the trajectory back and forth between the two open halfspaces. While under the lemma's condition, in order for the loop to be feasible, the trajectory must be able to travel back and forth between S_1 and S_2 which are in two different open halfspaces. This is impossible due to our previous argument. □

Remark 5 Lemma 1 is useful and provides a way to eliminate candidate paths containing infeasible nonelementary loops. Although the verification of the condition in the lemma requires techniques in computational geometry, for some special case we can actually verify the condition using linear programming. For example, consider the simple case where $\lambda_1 = (i_0, i_1, i_0)$ and $\lambda_2 = (i_1, i_2, i_1)$ are two elementary loops. If $(P_{i_0} \cap P_{i_1}) \bigcap (P_{i_2} \cap P_{i_1}) = \emptyset$, then by using Lemma 1 we can prove that any loop in the form of $\big(i_0, i_1, (i_2, i_1)^*, i_0\big)$ (λ_2 repeats at least once) is infeasible. Moreover, if neither of P_{i_0} and P_{i_2} is a legal successor of the other, then the loop $\big(i_0, i_1, (i_2, i_1)^*, i_0\big)$ (λ_2 repeats at least once) is infeasible. □

4.2 A Sufficient Condition for Finiteness of Looping Times

In practice, we often content ourselves by suboptimal solutions resulted from enforcing an upper bound on the number of times any loop is allowed to repeat and then solving finitely many times step (2). In many cases, such an upper bound is justifiable and a global optimal solution can be obtained by considering only finitely many candidate paths.

Lemma 2 (A Sufficient Condition for Finiteness of Looping Times) *Consider a given feasible problem 1. Assume that a loop $\lambda = (i_0, i_1, \cdots, i_m, i_{m+1})$ with $i_0 = i_{m+1}$ is given. If $\exists\ 0 \leq j_1 < j_2 \leq m$ s.t. $(P_{i_{j_1}} \cap P_{i_{j_1+1}}) \bigcap (P_{i_{j_2}} \cap P_{i_{j_2+1}}) = \emptyset$, then $\exists$ an L s.t., in order to search for the global optimal solution, we only need to consider candidate paths in which the loop λ repeats no more than L times.*

Proof: Since the problem is feasible, $\exists$ at least one feasible trajectory from x_0 to x_f with total time duration T. Let d be the minimum distance between the sets $(P_{i_{j_1}} \cap P_{i_{j_1+1}})$ and $(P_{i_{j_2}} \cap P_{i_{j_2+1}})$ ($d > 0$ because $(P_{i_{j_1}} \cap P_{i_{j_1+1}}) \bigcap (P_{i_{j_2}} \cap P_{i_{j_2+1}}) = \emptyset$). In order for the loop λ to repeat once, the trajectory must travel at least once from $(P_{i_{j_1}} \cap P_{i_{j_1+1}})$ to $(P_{i_{j_2}} \cap P_{i_{j_2+1}})$, which takes no less than the time duration $t_{\min} = \min_{j=0,1,\cdots,m}\{\frac{d}{\|\alpha_{i_j}\|}\}$. An upper bound L can be chosen to be the greatest integer no greater than $\frac{T}{t_{\min}}$. Any path in which λ repeats more than L times will have the total time duration greater than $\frac{T}{t_{\min}} t_{\min} = T$ and hence should not be considered as a candidate for the global optimal solution. $\square$

Remark 6 If $P_{j_1} \cap P_{j_2} = \emptyset$ for some $0 \leq j_1 < j_2 \leq m$, the condition in Lemma 2 must be satisfied. This provides a quick way to verify the sufficient condition in the lemma. $\square$

4.3 Zigzagging Loops

Loops in the form of $\lambda_z = \big(i_j, (i_{j'}, i_j)^*\big)$ are commonly encountered when constructing candidate paths. Whenever i_j and $i_{j'}$ are legal successors of each other, such a loop can appear in candidate paths. Here we propose a technique to simplify the computation along paths containing such loops.

Figure 3(a) shows the general behavior of the portion of the state trajectory corresponding to the loop (repeating at least once). First, subsystem i_j is active. The state trajectory follows its dynamics and reaches $x_a \in P_{i_j} \cap P_{i_{j'}}$. At x_a, the system switches to subsystem $i_{j'}$. After evolving at $i_{j'}$ for some time, the system switches back to i_j (at the switching instant, the state must also be in $P_{i_j} \cap P_{i_{j'}}$). The system then continues to switch between subsystems i_j and $i_{j'}$ for finitely many times. At last, the system switches back to i_j and the state is driven to the next portion of the overall trajectory (corresponding to the last line segment in Figure 3(a)).

If we choose a point x_b on the intersection of the last line segment and $P_{i_{j'}}$ (see Figure 3(a)), we observe that the trajectory portion between x_a and x_b then exhibits zigzagging behavior and is totally inside $P_{i_j} \cap P_{i_{j'}}$. Therefore we call λ_z a *zigzagging loop*. The total time duration for each subsystem is important for zigzagging loops. However, the zigzagging patterns are of minor importance because multiple patterns leading the

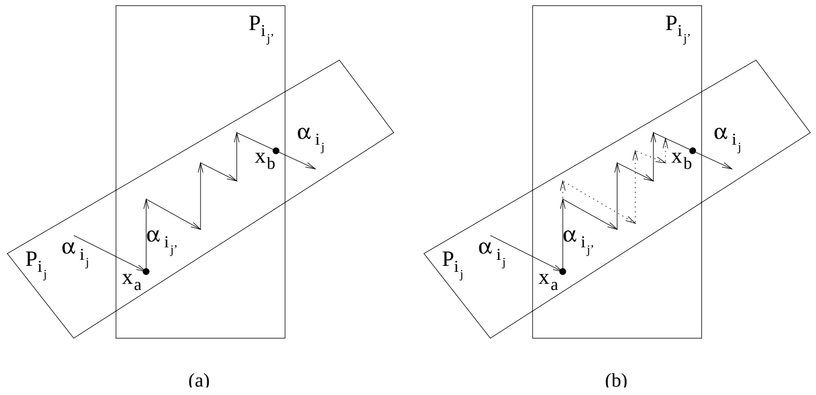

Figure 3: (a) Portion of the system state trajectory corresponding to the loop. (b) Multiple zigzagging patterns may exhibit the same total time duration for each subsystem.

state from x_a to x_b may exhibit the same total time duration for each subsystem, e.g., the trajectories denoted by the solid line and the dotted line in Figure 3(b) have different zigzagging patterns but the same total time duration for subsystems $i_{j'}$ and i_j, respectively. The above discussion therefore reveals the following way to simplify the computation for zigzagging loops. In between x_a and x_b, we compute the total time durations for $i_{j'}$ and i_j but are not concerned with the details of the zigzagging pattern, i.e., write

$$x_b = x_a + \tau_{j'}^{\text{tot}} \alpha_{j'} + \tau_j^{\text{tot}} \alpha_j. \tag{11}$$

With the total time durations $\tau_{j'}^{\text{tot}}$ and τ_j^{tot}, we can implement the state trajectory between x_a and x_b as follows.

Algorithm 2 (An Implementation of Zigzagging Loops)

(1). At first, when the state trajectory reaches x_a, set the active subsystem to be $i_{j'}$.

(2). Let the system evolve according to the currently active subsystem until either

 (i). the total time duration τ^{tot} for the current active subsystem is exhausted, or

 (ii). the trajectory intersects the boundary of $P_{i_j} \cap P_{i_{j'}}$.

(3). Set the active subsystem to be the other subsystem and repeat (2) until x_b is reached.

□

An illustration of Algorithm 2 is shown in Figure 4. In Figure 4(a), we let subsystem $i_{j'}$ evolve for time $\tau_{j'}^{\text{tot}}$ and then let subsystem i_j evolve for time τ_j^{tot}. However, such a brute force implementation results in a trajectory violating the constraint subset $P_{i_j} \cap P_{i_{j'}}$. Figure 4(b) shows the implementation using Algorithm 2, which results in a valid trajectory.

Now let us consider a general path expression $\pi = \big(i_0, i_1, \cdots, i_j, (i_{j'}, i_j)^*, i_{j+1}, \cdots, i_m\big)$ that contains a zigzagging loop $\lambda_z = \big(i_j, (i_{j'}, i_j)^*\big)$. To find the optimal timing information for such a path, we propose the following technique which is based on two possible cases.

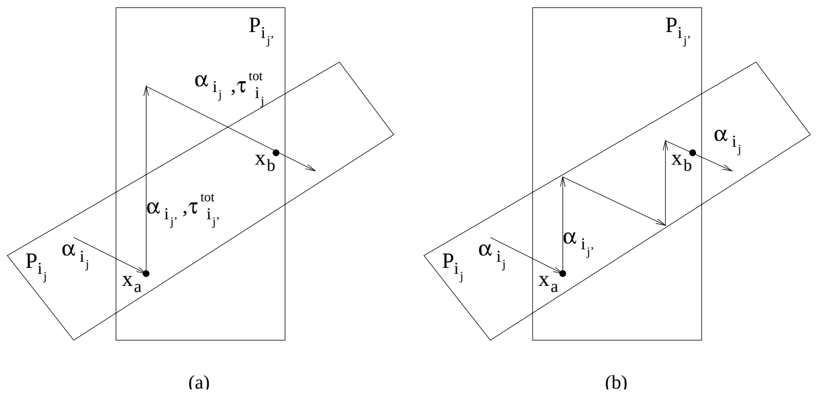

Figure 4: (a) A brute force implementation. (b) Zigzagging trajectory implementation using Algorithm 2.

Case 1: λ_z Repeats 0 Times

In this case, the path is reduced to $\pi_1 = (i_0, i_1, \cdots, i_j, i_{j+1}, \cdots, i_m)$. The optimal timing for such a path can be directly obtained by solving (9)-(10).

Case 2: λ_z Repeats At Least Once

In this case, we can rewrite the path as $\pi_2 = (i_0, i_1, \cdots, i_j, (i_{j'}, i_j)^?, i_j, i_{j+1}, \cdots, i_m)$ in which the expression $(i_{j'}, i_j)^?$ indicates that it corresponds to a zigzagging loop. For this loop, we only need to find the total time durations for subsystems $i_{j'}$ and i_j. And we duplicate an i_j before i_{j+1} so that the portion after x_b on the last line segment (see Figure 4(b)) can be taken care of before the system switches to i_{j+1}. To find the optimal timing for path π_2, (9)-(10) can be similarly applied except for some modifications on the portion of constraints corresponding to the part of trajectory when x is evolving from x_a to x_b. This portion of constraints are now posed as follows

$$A_{i_j}(x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_j\alpha_{i_j}) \leq b_{i_j} \quad (12)$$

$$A_{i_{j'}}(x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_j\alpha_{i_j}) \leq b_{i_{j'}} \quad (13)$$

$$A_{i_j}(x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_j\alpha_{i_j} + \tau_{j'}^{\text{tot}}\alpha_{j'} + \tau_j^{\text{tot}}\alpha_j) \leq b_{i_j} \quad (14)$$

$$A_{i_{j'}}(x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_j\alpha_{i_j} + \tau_{j'}^{\text{tot}}\alpha_{j'} + \tau_j^{\text{tot}}\alpha_j) \leq b_{i_{j'}} \quad (15)$$

Note (12)-(13) correspond to the constraint $x_a \in P_{i_j} \cap P_{i_{j'}}$ and (14)-(15) correspond to $x_b \in P_{i_j} \cap P_{i_{j'}}$. We do not pose any constraints for $x_0 + \tau_0\alpha_{i_0} + \cdots + \tau_j\alpha_{i_j} + \tau_{j'}^{\text{tot}}\alpha_{j'}$ since Algorithm 2 will help generate a valid zigzagging trajectory.

Example 6 There are infinitely many paths in Example 4. We can apply the sufficient condition in Section 4.1 to paths containing loops $\big(1, 2, (3, 2)^*, 1\big)$ and $\big(2, 1, (4, 1)^*, 2\big)$ to conclude that they are infeasible (see Remark 5). The number of path expressions can then be greatly reduced, e.g., π_2 and π_3 in Example 4 do not need to be considered.

If we apply the sufficient condition in Section 4.2 to path expressions containing loops $(1, 2, 3, 4, 1)$ or loops containing this loop, we can determine an upper bound L for

the number of times that these loops can repeat. If we combine this observation with the above-mentioned elimination of infeasible paths, we only need to consider the path expression
$\pi = \Big(1, \big((4,1)^*, (2,1)^*, 2, (3,2)^*, 3, 4, 1\big)^{<L>}, (4,1)^*, (2,1)^*, 2, (3,2)^*, 3\Big)$ for this example, where $< L >$ indicates that the loop can repeat at most L times. By utilizing the total time obtained in Example 5 and noting that the distance between P_1 and P_3 is 1, we can choose $L = 3$ (see Proof of Lemma 2). Similarly, we only need to consider the path expression $\hat{\pi} = \Big(4, 1, \big((4,1)^*, (2,1)^*, 2, (3,2)^*, 3, 4, 1\big)^{<L>}, (4,1)^*, (2,1)^*, 2, (3,2)^*, 3\Big)$ for paths starting at node 4 (since x_0 also belongs to P_4). Considering π and $\hat{\pi}$ and using the techniques for zigzagging loops, we only need to compute optimal timing information for finitely many possible paths. After computing along all possible paths, we confirm that the solution $\sigma = \big((0,1), (0.9706, 2), (2.8529, 3)\big)$ we obtained in Example 5 is a global minimum solution. □

Remark 7 It can be seen from [1] that in general reachability problem is undecidable for linear hybrid systems[†]. Such a result can shed light on the computational complexity of our time optimal control problem. In general, the time optimal control problem is also undecidable since it is by itself also a reachability problem. Therefore, it is possible that there may be infinitely many candidate paths to look into in order to find the optimal solution. However, we point out that the approach we propose in this paper utilizes the structural information of the problem and can significantly reduce the number of candidate paths (or even reduce to finitely many paths as in Example 6). This could significantly reduce computation and is the first step towards more efficient methods. In the case that there are still infinitely many paths after reduction, suboptimal solutions by restricting the maximum number of paths can usually be obtained to solve practical problems. □

5 Conclusion

This paper has reported some results on time optimal control of integrator switched systems with polyhedral state constraints. We first developed a directed graph representation which helps generate candidate paths. A linear programming method for finding the optimal timing for each path was then proposed. An algorithm which seeks global optimal solution among all candidate paths was also proposed. Since there may be infinitely many candidate paths, an upper bound on looping times will usually be enforced in practice in order to obtain a suboptimal solution. How to efficiently reduce the number of candidate paths is still a largely open problem. We have proposed preliminary conditions and techniques for reducing the number of candidate paths. In our future research, we will seek

[†]It is known that initialized rectangular automata are on the boundary of decidability. Slight generalizations of them lead to undecidability [6].

more conditions under which a finite number of candidate paths can be guaranteed.

References

[1] R. Alur, C. Coucoubetis, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3-34, 1995.

[2] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407-427, 1999.

[3] A. Bemporad, F. Borrelli, and M. Morari. On the optimal control law for linear discrete time hybrid systems. In *Hybrid Systems: Computation and Control 2002, Lecture Notes in Computer Science 2289*, pp. 105-119, Springer, 2002.

[4] S.C. Bengea and R.A. DeCarlo. Optimal control of switching systems. *Automatica*, 41: 11-27, 2005.

[5] B. De Schutter. Optimal control of a class of linear hybrid systems with saturations. *SIAM Journal on Control and Optimization*, 39(3):835-851, 2000.

[6] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94-124, 1998.

[7] A. Giua, C. Seatzu, and C. Van Der Mee. Optimal control of switched autonomous linear systems. In *Proceedings of the 40th IEEE Conference on Decision and Control*, pp. 2472-2477, 2001.

[8] D. Liberzon and A.S. Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19:59-70, October 1999.

[9] B. Lincoln and B. Bernhardsson. LQR optimization of linear system switching. *IEEE Transactions on Automatic Control*, 47(10):1701-1705, 2002.

[10] D.L. Pepyne and C.G. Cassandras. Optimal control of hybrid systems in manufacturing. *Proceedings of the IEEE*, 88(7):1108-1123, 2000.

[11] S. Pettersson and B. Lennartson. Time-optimal control and disturbance compensation for a class of hybrid systems. In *Proceedings of the 13th IFAC World Congress*, Vol. J, pp. 281-286, 1996.

[12] S. Pettersson. Analysis and Design of Hybrid Systems. Ph.D. Thesis, Chalmers Univ. of Tech., Sweden, 1999.

[13] P. Riedinger, C. Zanne, and F. Kratz. Time optimal control of hybrid systems. In *Proceedings of the 1999 American Control Conference*, pp. 2466-2470, 1999.

[14] M. Tittus. Control Synthesis for Batch Processes. Ph.D. Thesis, Chalmers Univ. of Tech., Sweden, 1995.

[15] X. Xu and P.J. Antsaklis. Results and perspectives on computational methods for optimal control of switched systems. In *Hybrid Systems: Computation and Control 2003, Lecture Notes in Computer Science 2623*, pp. 540-555, Springer, 2003.