Proceedings of the 42nd IEEE
Conference on Decision and Control
Maui, Hawaii USA, December 2003

A Logic-Based Hybrid Solver for Optimal Control of Hybrid Systems

A. Bemporad†, N. Giorgetti†
†Dip. Ingegneria dell'Informazione, University of Siena,
via Roma 56, 53100 Siena, Italy
bemporad,giorgetti@dii.unisi.it

Abstract

Combinatorial optimization over continuous and integer variables was proposed recently as an useful tool for solving complex optimal control problems for linear hybrid dynamical systems formulated in discrete-time. Current approaches are based on mixed-integer linear/quadratic programming (MIP), which provides the solution after solving a sequence of relaxed standard linear (or quadratic) programs (LP, QP). An MIP formulation has the drawback of requiring that the discrete/logic part of the hybrid problem needs to be converted to into mixed-integer inequalities. Although this operation can be done automatically, most of the original discrete structure of the problem is lost during the conversion. Moreover, the efficiency of the MIP solver only relies upon the tightness of the continuous LP/QP relaxations. In this paper we attempt at overcoming such difficulties by combining MIP and constraint programming (CP) techniques into a "hybrid" solver, taking advantage of CP for dealing efficiently with satisfiability of logic constraints. We detail how to model the hybrid dynamics so that the optimal control problem can be solved by the hybrid MIP+CP solver, and show on a case study that the achieved performance is superior to the one achieved by pure MIP solvers.

1 Introduction

Over the last few years we have witnessed a growing interest in the study of dynamical processes of a mixed continuous and discrete nature, denoted as hybrid systems, both in academia and in industry. Hybrid systems are characterized by the interaction of continuous models governed by differential or difference equations, and of logic rules, automata, and other discrete components (switches, selectors, etc.). Hybrid systems can switch between many operating modes where each mode is governed by its own characteristic continuous dynamical laws. Mode transitions may be triggered internally (variables crossing specific thresholds), or externally (discrete commands directly given to the system). The interest in hybrid systems is mainly motivated by the large variety of practical situations where physical processes interact with digital controllers, as for instance in embedded systems.

After the seminal work [1], several modelling frameworks for hybrid systems have appeared in the literature, we refer the interested reader to [2, 3] and references therein. Several authors focused on the problem of solving optimal control problems for hybrid systems. For continuous-time hybrid systems, most of the literature either studied necessary conditions for a trajectory to be optimal, or focused on the computation of optimal/suboptimal solutions by means of dynamic programming or the maximum principle [4, 5].

The hybrid optimal control problem becomes less complex when the dynamics is expressed in discrete-time, as the main source of complexity becomes the combinatorial (yet finite) number of possible switching sequences. In particular, in [6–8] the authors have solved optimal control problems for discrete-time hybrid systems by transforming the hybrid model into a set of linear equalities and inequalities involving both real and (0-1) variables, so that the optimal control problem can be solved by a mixed-integer programming (MIP) solver.

As a MIP solver provides the solution after solving a sequence of relaxed standard linear (or quadratic) programs (LP, QP), a potential drawback of MIP is (1) the need for converting the discrete/logic part of the hybrid problem into mixed-integer inequalities, therefore losing most of the original discrete structure, and (2) the fact that its efficiency relies upon the tightness of the continuous LP/QP relaxations.

At the light of the benefits and drawbacks of the previous work in [6–8] for solving control and stability/safety analysis problems for hybrid systems using MIP techniques, in this paper we follow a different route that uses a combined approach of MIP and CP techniques. We build up a new modeling approach directly tailored to the use of a "hybrid" MIP+CP solver, and show its computational advantages over pure MIP methods. This result is in accordance with other studies [9–11], that show that such mixed methods have a tremendous performance in solving mathematical programs with continuous (quantitative) and discrete (logical/symbolic) components, compared to MIP or CP individually.

The paper is organized as follows. Optimal control problems for hybrid systems are introduced in Section 2. In Section 3 the optimal control problem is reformulated in a suitable way for the combined approach of MIP and CP. In Section 4 is introduced a new solution algorithm based on a unifying framework for CP and MIP. In Section 5 is shown on an example the benefits of this technique, compared to pure MIP approaches [6, 8]. Finally, Section 6 contains some con-

0-7803-7924-1/03/$17.00 ©2003 IEEE

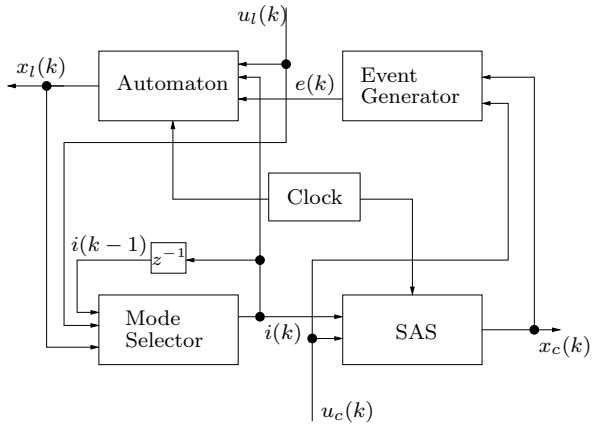

Figure 1: Discrete-time hybrid system

cluding remarks.

2 Optimal Control of Discrete-Time Hybrid Systems

Following the ideas in [8], a hybrid system can be modeled as the interconnection of an automaton (AUT) and a switched affine system (SAS) through an event generator (EG) and a mode selector (MS) (see Figure 1). The discrete-time hybrid dynamics is described as follows [8]:

$$\begin{aligned}
\text{(AUT)}\quad & x_l(k+1) = f_l(x_l(k), u_l(k), e(k)), \\
& y_l(k) = g_l(x_l(k), u_l(k), e(k)), && \text{(1a)}\\
\text{(SAS)}\quad & x_c(k+1) = A_{i(k)}x_c(k) + B_{i(k)}u_c(k) + f_{i(k)}, \\
& y_c(k) = C_{i(k)}x_c(k) + D_{i(k)}u_c(k) + g_{i(k)}, && \text{(1b)}\\
\text{(EG)}\quad & [e_j(k) = 1] \longleftrightarrow [a_j^T x_c(k) + b_j^T u(k) \le c_j] && \text{(1c)}\\
\text{(MS)}\quad & i(k) = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_s \end{bmatrix} = f_{\text{MS}}(x_l(k), u_l(k), i(k-1)) && \text{(1d)}
\end{aligned}$$

The automaton (or finite state machine) describes the logic dynamics of the hybrid system. We will only refer to "synchronous automata", where transitions are clocked and synchronous with the sampling time of the continuous dynamical equations. The dynamics of the automaton evolves according to the logic update functions (1a) where $k \in \mathbb{Z}^+$ is the time index, $x_l \in \mathcal{X}_l \subseteq \{0,1\}^{n_l}$ is the logic state, $u_l \in \mathcal{U}_l \subseteq \{0,1\}^{m_l}$ is the exogenous logic input, $y_l \in \mathcal{Y}_l \subseteq \{0,1\}^{p_l}$ is the logic output, $e \in \mathcal{E} \subseteq \{0,1\}^{n_e}$ is the endogenous input coming from the EG, and $f_l : \mathcal{X}_l \times \mathcal{U}_l \times \mathcal{E} \longrightarrow \mathcal{X}_l$, $g_l : \mathcal{X}_l \times \mathcal{U}_l \times \mathcal{E} \longrightarrow \mathcal{Y}_l$ are deterministic boolean functions.
The SAS describes the continuous dynamics and it is a collection of affine systems (1b) where $x_c \in \mathcal{X}_c \subseteq \mathbb{R}^{n_c}$ is the continuous state vector, $u_c \in \mathcal{U}_c \subseteq \mathbb{R}^{m_c}$ is the exogenous continuous input vector, $y_c \in \mathcal{Y}_c \subseteq \mathbb{R}^{p_c}$ is the continuous output vector, $i(k) \in \mathcal{I} \triangleq \left\{ \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \cdots, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \right\} \subseteq \{0,1\}^s$ is the "mode" in which the SAS is operating, $\sharp\mathcal{I} = s$ is the number of elements of $\mathcal{I}$, and $\{A_i, B_i, f_i, C_i, D_i, g_i\}_{i\in\mathcal{I}}$ is a collection of matrices of opportune dimensions. The mode $i(k)$ is generated by the mode selector, as described below. A SAS of the form (1b) preserves the value of the state when a switch occurs. Resets can be modeled in the present discrete-time setting as detailed in [8].
The event generator (EG) is a mathematical object that generates a boolean vector according to the satisfaction of a set of *threshold events* (1c) where $_j$ denotes the j-th component of the vector, and $a_j \in \mathbb{R}^{n_c}$, $b_j \in \mathbb{R}^{m_c}$, $c_j \in \mathbb{R}$ define the hyperplane in the space of continuous states and inputs.
The mode selector (MS) selects the dynamic mode $i(k) \in \mathcal{I} \subseteq \{0,1\}^s$, also called the *active mode*, of the SAS and it is described by the logic function (1d) where $f_{MS} : \mathcal{X}_l \times \mathcal{U}_l \times \mathcal{I} \longrightarrow \mathcal{I}$ is a boolean function of the logic state $x_l(k)$, of the logic input $u_l(k)$, and of the active mode $i(k-1)$ at the previous sampling instant. We say that a *mode switch* occurs at step k if $i(k) \neq i(k-1)$. Note that contrarily to continuous time hybrid models, where switches can occur at any time, in our discrete-time setting a mode switch can only occur at sampling instants.

A finite-time optimal control problem for the class of hybrid systems is formulated as follows:

$$\begin{aligned}
\min_{\{x(k),u(k)\}_{k=0}^{T-1}} \sum_{t=0}^{T-1} & \|Q_x(x(k) - x_e(k))\|_\infty + \\
& \|Q_u(u(k) - u_e(k))\|_\infty + \\
& \|Q_y(y(k) - y_e(k))\|_\infty && \text{(2a)}\\
\text{s.t. Dynamics (1)} & && \text{(2b)}\\
& h_D(k)(\{x,u,y,e,i\}_0^{T-1}) \le 0 && \text{(2c)}\\
& h_A(k)(\{x,u,y,e,i\}_0^{T-1}) \le 0 && \text{(2d)}
\end{aligned}$$

where T is the control horizon, Q_x, Q_u, Q_y are full-rank matrices with $n = n_c + n_l$, $m = m_c + m_l$, and $p = p_c + p_l$ rows, respectively, $\|\cdot\|_\infty$ denotes the infinity-norm ($\|Qx\|_\infty = \max_{j=1,\ldots,n} |Q^j x|$, where Q^j is the j-th row of Q). Vectors x_e, u_e, y_e are the given reference trajectories to be tracked by the state, input and output vectors, respectively.

The constraints of the optimal control problem can be classified in three distinct categories:

Dynamical constraints (2b). These constraints represent the discrete-time hybrid system. They may include other constraints such as saturation constraints on continuous input variables, that are embedded in the variable domain $\mathcal{U}_c$.

Design constraints (2c). These are artificial constraints imposed by the designer to fulfill the required specifications. Examples of such constraints may be state limits

$$x_{min}^{i(k)} \le x_c(k) \le x_{max}^{i(k)},\ k = 0, \ldots, T-1,$$

where x_{min}^i, x_{max}^i are (possibly mode-dependent) bounds that the designer wants to impose on the state vector.

Ancillary constraints (2d). These constraints provide an a priori additional and auxiliary information for determining the optimal solution. They do not change the solution itself, rather help the solver by restricting the set of feasible combinations, and therefore the size of the decision tree in a branch a bound strategy. For example, one may pre-compute all possible mode transitions of the SAS dynamics using reachability analysis, and impose *reachability constraints* of the form $[\delta_h(k) = 1] \longrightarrow [\delta_j(k+1) = 0]$ (or equivalently $\delta_h(k) + \delta_j(k+1) \leq 1$) for all $k = 0, \ldots, T-2$ whenever a transition from the h-th mode to the y-th mode is not possible.

3 Problem reformulation

In [7] the authors show how the problem (2) can be solved via MILP. In this paper following a different route we wish to solve the problem (2) by using a combined approach of MIP and CP techniques and taking advantage of CP for dealing with logic part of the problem. For this reason we need to reformulate the problem in a suitable way.

The automaton and mode selector parts of the hybrid system are described as a set of logic constraints so they need no transformation. The event generator (1c) can be equivalently expressed, by adopting the so-called "big-M" technique, as

$$(a_j^T x_c(k) + b_j^T u(k) - c_j) \leq M_j(1 - e_j(k)), \quad (3a)$$

$$(a_j^T x_c(k) + b_j^T u(k) - c_j) > m_j e_j(k), \quad (3b)$$

where $j = 1, \ldots, n_e$, M_j, m_j are upper and lower bounds, respectively, on $a_j^T x_c(k) + b_j^T u(k) - c_j$, and $e_j(k) \in \{0,1\}$. From a computational point of view, it may be convenient to have a set of inequalities without strict inequalities. In this case we will follow the common practice [12] to replace the strict inequality (3) as

$$(a_j^T x_c(k) + b_j^T u(k) - c_j) \geq \epsilon + (m_j - \epsilon) e_j(k), \quad (3c)$$

where ϵ is a small positive scalar, e.g., the machine precision, although the equivalence does not hold for $0 < (a_j^T x_c(k) + b_j^T u(k) - c_j) < \epsilon$ (i.e., for the numbers in the interval $(0, \epsilon)$ that cannot be represented in the machine). The continuous state update equation of the SAS dynamics (1b) can be equivalently written as the combination of linear terms and *if-then-else* rules:

$$w_i(k) = \begin{cases} A_i x_c(k) + B_i u_c(k) + f_i & \text{if } (\delta_i = 1) \\ 0 & \text{otherwise} \end{cases} \quad (4a)$$

$$x_c(k+1) = \sum_{i=1}^{s} w_i(k) \quad (4b)$$

where $w_i(k) \in \mathbb{R}^{n_c}$, $i = 1, \ldots, s$. The output y_c of the SAS dynamics admits a similar transformation. The SAS representation (4) can be also translated into a set of constraints by using the big-M technique [12]:

$$-M_i^j \delta_i(k) + w_i(k) \leq 0, \quad (5a)$$

$$m_i^j \delta_i(k) - w_i(k) \leq 0, \quad (5b)$$

$$m_i^j (1 - \delta_i(k)) + w_i(k) \leq A_i^j x_c(k) + B_i^j u_c(k) + f_i^j, \quad (5c)$$

$$-M_i^j (1 - \delta_i(k)) - w_i(k) \leq -A_i^j x_c(k) - B_i^j u_c(k) - f_i^j, \quad (5d)$$

where M_i^j, m_i^j are upper and lower bounds on $A_i^j x_c(k) + B_i^j u_c(k) + f_i^j$, $\delta_i(k) \in \{0,1\}$, $w_i(k) \in \mathbb{R}^{n_c}$, $x_c \in \mathbb{R}_c^n$, $u \in \mathbb{R}_c^m$, ^{j} denotes the jth component or row, $j = 1, \ldots, n_c$, $i = 1, \ldots, s$, and k is the time index. Note that the vector of (0-1) variables $i(k) = [\delta_1(k) \ \ldots \ \delta_s(k)]' \in \{0,1\}^s$ is subject to the exclusive or condition

$$\delta_1(k) \oplus \delta_2(k) \oplus \ldots \oplus \delta_s(k), \quad (6)$$

which can be described by the equality constraint

$$\delta_1(k) + \ldots + \delta_s(k) = 1. \quad (7)$$

By using the transformations into mixed integer inequalities described above, problem (2) can be recast as

$$\min_z \ f'z \quad (8a)$$

$$\text{s.t. } Gz \leq d, \ \overline{G}z = \overline{d} \quad (8b)$$

(Continuous constraints)

$$G'z + D'\mu \leq E', \ \overline{G'}z + \overline{D'}\mu = \overline{E'} \quad (8c)$$

(Mixed constraints)

$$g(\nu, \mu) \quad (8d)$$

(Logic constraints)

$$z \in \mathbb{R}^{n_z}, \ \nu \in \{0,1\}^{n_\nu}, \ \mu \in \{0,1\}^{n_\mu}$$

where z collects all continuous variables ($x_c(k)$, $u_c(k)$, $y_c(k)$, $k = 0, \ldots, T-1$, auxiliary variables needed for expressing the SAS dynamics, slack variables for upper bounding the cost function in (2) [7]), μ collects the integer variables that appear in mixed constraints ($e(k)$, $\delta_i(k)$, $k = 0, \ldots, T-1$, $i = 1, \ldots, s$), and ν collects the integer variables such as $x_l(k)$, $u_l(k)$, $y_l(k)$ that only appear in logic constraints. Constraints (8b), (8c) represent the EG and SAS parts (3), (5), (7) and the mixed constraints due to (2c), (2d), $g : \{0,1\}^{n_\nu \times n_\mu} \rightarrow \{0,1\}^{n_{CP}}$ represents the automaton and MS parts (1a), (1d). Constraint (8b) may contain additional constraints on the continuous part such as bounds imposed by (2c), (2d), while additional logic constraints (e.g. (6)) on the discrete part are included in (8d). Constraint (8c) contains other constraints needed for expressing the ∞-norm used in cost function (2a). Note that in general if the objective function is $f'z + r'\mu$ we can consider the new objective function $f'z + t$, $t \in \mathbb{R}$, and an additional constraint $r'\mu \leq t$ which is a mixed linear constraint and that could be included in (8c).

4 Logic-based Branch&Bound

CP consists of a set of techniques for solving finite domain constraint problems. A *finite domain constraint problem*, or (FD)CP, consists of a set of constraints over a set of integer finite domain variables. A *constraint* may be simply a logical relation, such as "X $\leq$ 4" or a more complex expression, such as "$X^2 = Y^3+4$".

A (FD)CP can always be solved with brute force search: all possible values of the variables are enumerated and each one is checked to see whether it is a solution. Except in very small problems, the number

of candidates is usually too large to enumerate them all. The efficiency of CP is based on two basic techniques: *constraint propagation* and *constraint distribution*. Constraint propagation is an inference rule for finite domain problems that narrows the domains of the variables. Constraint distribution splits a problem into complementary cases once constraint propagation cannot advance further. By iterating between propagation and distribution the solutions of the problem are finally determined. For more details, we refer the reader to the book [13].

In order to efficiently integrate MIP and CP we will use the special modeling framework (8) for "hybrid" optimization problems, where the word hybrid here means that some parts of the problem are described by logic constraints and finite domain variables, some others by linear constraints and continuous variables. We refer here to the basic framework introduced by Bockmayr and Kasper in [10] who did an interesting analysis of CP and MIP approaches by studying the so called *"branch and infer"* paradigm, which can be used to develop various integration strategies.

The basic ingredients for an integrated approach of MIP and CP are (1) a linear program (LP) obtained by relaxing a mixed integer linear programming (MILP) problem and (2) a CP feasibility problem, both of which can be solved efficiently. The relaxed MILP model is used to obtain a solution that satisfies the constraint sets (8b) and (8c) and optimizes the objective function (8a). The optimal solution of the relaxation may fix some (0-1) variables to an integer value. If all the (0-1) variables in the relaxed problem have been assigned the solution of the relaxation is also a feasible solution for the MILP problem. More often, however, some (0-1) variables are not assigned and in these cases further "branching" and solution of further relaxations is necessary. To accelerate the search of feasible solutions one may use the fixed (0-1) variables to "infer" new information on the other (0-1) variables by applying a constraint propagation phase on constraints (8d): when an integer solution of μ is found a CP(FD) then verifies whether this solution can be used to find an assignment of ν that satisfies (8d).

The basic branch&bound (B&B) algorithm for solving mixed integer problems can be extended to the present hybrid setting. In a B&B algorithm, the current best integer solution is updated whenever an integer solution with an even better value of the objective function is found. In the hybrid algorithm at hand an additional CP problem is solved to ensure that the integer solution obtained for the relaxed MILP problem is feasible for the constraints (8d) and to find an assignment for the other logic variables ν that appear in (8d). It is only in this case that the current best integer solution for the relaxed MILP problem is updated. If it cannot be extended, then the best current integer solution is not updated.

The B&B method requires the solution of a series of LP subproblems obtained by branching on integer variables. The non-integer variable to branch on is chosen with the constraint distribution by assigning an integer value which is consistent with the current solution of the MILP. The difference in the various LP subproblems is only in the upper and lower bounds for all integer variables updated by constraint propagation. The value of the objective function for any feasible solution of the problem is an upper bound (UB) of the objective function. When an integer feasible solution of the MILP relaxed problem is obtained a feasibility problem is solved via CP to complete the solution.

Let P denote the set of LP subproblems to be solved. The logic-based B&B method can be summarized as follows:

1. **Initialization**. $UB = \infty$, $P = \{p^0\}$. The LP subproblem p^0 is generated by using (8a),(8b),(8c) and possibly removing some of the (0-1) variables after a constraint propagation phase.
2. **Select a node**. Select and remove an LP problem p from the set P; if $P = \emptyset$ then go to 5. The criterion for selecting an LP is called *node selection rule*.
3. **Linear reasoning**. Solve the LP problem p, and:
   - If the LP is infeasible or the optimal value of the objective function is greater than UB then go to step 2.
   - If the solution is not integer feasible then go to step 4.
   - If the solution has integral values for all the integer variables then we solve the following (FD)CP problem to extend this partial solution: Find ν s.t. $g(\nu, \mu)$. If the (FD)CP problem is feasible then update UB; otherwise go to step 2.
4. **Branch on a variable**. Among all variables that have been assigned non-integral values, select one according to some specified *branching variable selection rule* (e.g., the variable with the largest fractional part). Let μ_i be the selected non-integer variable, and by using the constraint distribution define the constraint $\mu_i = 0$. Generate two LP subproblems $p_i^1 = p \cup \{\mu_i = 0\}$, $p_i^2 = p \cup \{\mu_i = 1\}$, apply constraint propagation to attempt to fix the other (0-1) variables and add p_i^1,p_i^2 to set P. Go to step 2.
5. **Termination**. If $UB = \infty$, then the problem is infeasible. Otherwise, the optimal solution corresponds to the current value UB.

Remark 1 The logic-based B&B method prunes the search tree with bounds that are obtained by solving the relaxed problems at each node. Because the relaxed problems contain fewer constraints, they may provide looser bounds that are less effective at pruning the tree. For this reason, it may be wise to include in the linear programs also the relaxations of the automaton and MS through (8c) by converting Boolean functions into linear inequalities [8]. It is still open whether such relaxations improve the performance of the overall algorithm.

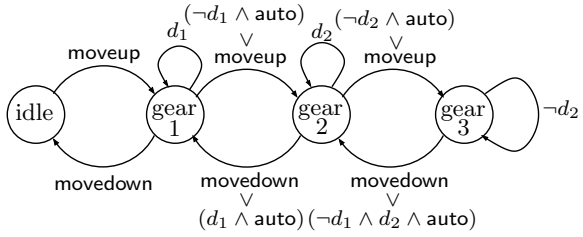

Figure 2: Gear automaton

5 Numerical Results

In this section we illustrate on a simple hybrid optimal control example that the hybrid solution technique described in the previous sections has a better performance compared to current MILP approaches [6, 8]. We consider a simplified hybrid model of a motorbike with three semi-automatic gears and solve an optimal control problem in order to track a desired speed.

5.1 Hybrid model

We consider only the speed v (Km·h^{-1}) and the engine speed ω (rpm) of the motorbike as continuous state variables. The continuous dynamics of the system depends on two continuous inputs that are the engine torque u_t (N·m) and the braking force u_b (N) and is described by the discrete-time update equations

$$v(k+1) = (1-\alpha_i)v(k) + \beta_i\omega(k) - cu_b \tag{9a}$$
$$\omega(k+1) = \omega(k) + du_t - fu_b, \tag{9b}$$

where v and w are dynamically decoupled (e.g. because of a loose mechanical coupling), c,d,f are suitable constants, and α_i, β_i, $i \in \{1,2,3\}$, are constants that depend on the gear.

Gear shifts may be commanded either automatically by a digital input auto or manually by two digital inputs, moveup and movedown. The automaton part of the system has an idle state and three states corresponding to each gear, as represented in Figure 2. d_1 and d_2 are logic variables defined as follows

$$[d_1(k) = 1] \longleftrightarrow [\omega(k) \le t_1], \tag{10a}$$
$$[d_2(k) = 1] \longleftrightarrow [\omega(k) \le t_2], \tag{10b}$$

where $t_1 \le t_2$ are constant thresholds. The gear automatically changes from gear_{i-1} to gear_i when the engine speed exceeds the threshold t_{i-1} and the auto command is active. If the auto command is inactive the gear can change only with moveup and movedown commands. The motorbike goes in idle state only when a movedown command from gear_1 is given. Conversely the motorbike passes from idle to gear_1 when a moveup command is given.

By introducing a (0-1) state for each logic state of the automaton, this can be described by the following relations:

$$id(k+1) = (g_1(k) \ \wedge \ \text{movedown}(k)); \tag{11a}$$
$$\begin{aligned} g_1(k+1) = {} & (g_1(k) \ \wedge d_1(k)) \ \vee \ (id(k) \ \wedge \ \text{moveup}(k)) \vee \\ & (g_2(k) \ \wedge \ \text{movedown}(k)) \vee \\ & (g_2(k) \ \wedge \ d_1(k) \ \wedge \ \text{auto}(k)); \end{aligned} \tag{11b}$$
$$\begin{aligned} g_2(k+1) = {} & (g_1(k) \ \wedge \ \text{moveup}(k)) \vee \\ & (g_1(k) \ \wedge \ \neg d_1(k) \ \wedge \ \text{auto}(k)) \ \vee \ (g_2(k) \ \wedge \ d_2(k)) \vee \\ & (g_3(k) \ \wedge \ \neg d_1(k) \ \wedge \ d_2(k) \ \wedge \ \text{auto}(k)) \vee \\ & (g_3(k) \ \wedge \ \text{movedown}(k)); \end{aligned} \tag{11c}$$
$$\begin{aligned} g_3(k+1) = {} & (g_2(k) \ \wedge \ \text{moveup}(k)) \vee \\ & (g_2(k) \ \wedge \ \neg d_2(k) \ \wedge \ \text{auto}(k)) \vee \\ & (g_3(k) \ \wedge \ \neg d_2(k)) \end{aligned} \tag{11d}$$
$$id(k) \oplus g_1(k) \oplus g_2(k) \oplus g_3(k). \tag{11e}$$

By following the notation of (1a), we have $x_l = [id \ g_1 \ g_2 \ g_3]' \in \{0,1\}^4$, $u_l = [\text{moveup movedown auto}]' \in \{0,1\}^3$ and $e(k) = [d_1 \ d_2]' \in \{0,1\}^2$. The relation (11e) corresponds to condition (6).

The mode selector function is simply

$$\delta(k) = x_l(k), \tag{12}$$

where $\delta(k) \in \{0,1\}^4$, along with $\sum_{j=1}^{4} \delta_j(k) = 1$.

The SAS dynamics (9), i.e., the continuous part of the hybrid system, is translated into a set of inequalities using (5), which provides the set of constraints

$$Ax_c(k) + Bu_c(k) + Cz(k) \le D\delta(k) + E, \tag{13}$$

where $x_c = [v \ \omega]'$, $u_c = [u_t \ u_b]'$, $z(k) \in \mathbb{R}^2$ and $\delta(k)$ as (12). Constraints (13) are obtained by employing the HYSDEL compiler [8].

Finally, the event generator is represented by (10a) and (10b). These are translated by HYSDEL into a set of linear inequalities using (3):

$$G'_x x_c(k) + G'_u u_c(k) + D'e(k) \le E', \tag{14}$$

where $e(k) = [d_1(k) \ d_2(k)]'$.

5.2 Optimal Control Problem

The goal is to design an optimal control profile for the continuous inputs u_t, u_b and the discrete inputs moveup, movedown and auto that minimize $\sum_{k=0}^{T} |v(k) - v_e|$ subject to the hybrid dynamics and the following additional constraints:

- Continuous constraints on torque and brake to avoid that they assume unacceptable values

$$0 \le u_t(k) \le 50 \tag{15a}$$
$$0 \le u_b(k) \le 50. \tag{15b}$$

  While lower-bound constraints are truly dynamical constraints, corresponding to restricting $\mathcal{U}_c$ to nonnegative real-valued vectors, the upper-bound constraints may be either interpreted as design constraints of the form (2c) or again as dynamical constraints due to physical limitations of the engine and brakes.

- A logic condition on torque and braking force

$$[d_t = 0] \longleftrightarrow [u_t(k) \leq 0] \tag{16a}$$
$$[d_b = 0] \longleftrightarrow [u_b(k) \leq 0] \tag{16b}$$
$$\neg(d_t \ \wedge \ d_b). \tag{16c}$$

  Thresholds events are translated in a set of constraints by using the big-M formulation (5) obtained by the HYSDEL compiler.

- A exclusive-or logic condition on the gear commands:

$$\mathsf{moveup}(k) \oplus \mathsf{movedown}(k). \tag{17}$$

The above dynamics and constraints are also modeled in HYSDEL [8] to obtain an MLD model of the hybrid system in order to compare the performance achieved by the hybrid solver with the one obtained by employing a pure MILP approach.

The optimal control problem is defined as:

$$\min_{\{x,u,z,\delta,\epsilon_v\}} \sum_{k=0}^{T-1} \epsilon_v(k) \tag{18a}$$
$$\text{s.t. } \epsilon_v(k) \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \geq \pm(v(k) - v_e), \tag{18b}$$
$$(1a),\ (12),\ (17) \tag{18c}$$
$$(13),\ (14) \tag{18d}$$
$$(15) \tag{18e}$$
$$(16) \tag{18f}$$

where $\{x, u, z, \delta, \epsilon_v\} = \{x(k), u(k), z(k), \delta(k), \ \epsilon_v(k)\}_{k=0}^{T-1}$, $\epsilon_v = [\epsilon_v(0) \ \ldots \ \epsilon_v(T-1)]' \in \mathbb{R}^T$.

Each part of the optimal control problem is managed by either the CP solver or the LP solver: the cost function (18a), the inequalities (18b), (18d), and the additional constraints (18e) are managed by the LP solver, the logic part (18c) and the additional logic constraints (18f) are managed by the CP solver. Big-M reformulations of (18f) are also managed by the LP solver. Relation (11e) is also included in the LP part as $\sum_{j=1}^{4} \delta_j(k) = 1$. In our simulations we have used, respectively, Ilog Solver for CP, CPLEX for LP, and the Hybrid tool, a plugin of ILOG used to share the bounds on the integer variables between the relaxed problem and the CP problem.

In all our simulations we have adopted depth first search as *node selection rule*, to reduce the amount of memory used during the search, and the variable whose fractional value is the farthest to an integer value as *branching variable selection rule* (i.e., in this case the variable closest to 0.5).

Table 1 shows the result obtained by varying the control horizon T. We can see that the performance of the logic-based B&B is always better than MILP up to 75%. The main reason is that the logic-based B&B solves a much smaller number of LPs than the MILP solver, see Table 1.

The results were simulated on a PC Pentium IV 1.8 GHz running CPLEX 8.1 and Solver 5.3.

T	Integer vars	MILP (s)	LbB&B (s)	MILP (LP)	LbB&B (LP)
10	110	0.221	0.12	309	6
20	220	8.091	1.992	30645	49
30	330	26.318	7.798	71070	126
40	440	69.961	20.128	146101	229

Table 1: Optimal control solution: comparison between MILP and Logic-based B&B (LbB&B)

6 Conclusions

In this paper we have proposed a new unifying framework for MIP and CP techniques for solving optimal control problems for discrete-time hybrid systems. The approach is based on the "branch and infer" paradigm of Bockmayr and Kasper [10] and consists of a logic-based branch and bound algorithm, whose performance in terms of computation time is superior in comparison to more standard mixed-integer programming techniques, as we have illustrated on an example.
At this stage the proposed approach is rather preliminary, and the ongoing research is devoted to improving the performance of the logic-based method by including relaxations of the automaton and MS parts of the hybrid system in the linear programming part, and alternative relaxations of the SAS dynamics that are tighter than the big-M method.

References

[1] M.S. Branicky. *Studies in hybrid systems: modeling, analysis, and control*. PhD thesis, LIDS-TH 2304, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[2] P.J. Antsaklis. A brief introduction to the theory and applications of hybrid systems. *Proc. IEEE, Special Issue on Hybrid Systems: Theory and Applications*, 88(7):879–886, July 2000.

[3] C.G. Cassandras, D.L. Pepyne, and Y.Wardi. Optimal control of a class of hybrid systems. *IEEE Trans. Automatic Control*, 46(3):3981–415, 2001.

[4] X. Xu and P.J. Antsaklis. An approach to switched systems optimal control based on parameterization of the switching instants. In *Proc. IFAC World Congress*, Barcelona, Spain, 2002.

[5] B. Lincoln and A. Rantzer. Optimizing linear system switching. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 2063–2068, 2001.

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

[7] A. Bemporad, F. Borrelli, and M. Morari. Piecewise linear optimal controllers for hybrid systems. In *Proc. American Contr. Conf.*, pages 1190–1194, Chicago, IL, June 2000.

[8] F.D. Torrisi and A. Bemporad. HYSDEL - A tool for generating computational hybrid models. *IEEE Transactions on Control Systems Technology*, 2003. To appear.

[9] J. Hooker. *Logic-based methods for Optimization*. Wiley-Interscience Series, 2000.

[10] A. Bockmayr and T. Kasper. Branch and infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing*, 10(3):287–300, Summer 1998.

[11] R. Rodosek, M. Wallace, , and M. Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Oper. Res.*, 86:63–87, 1997.

[12] H.P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, Third Edition, 1993.

[13] K. Marriot and P.J. Stuckey. *Programming with constraints: an introduction*. MIT Press, 1998.