DYNAMIC OPTIMIZATION OF PROCESSING SYSTEMS WITH MIXED DEGREES OF FREEDOM

Olaf Stursberg

Process Control Laboratory (BCI-AST)
University of Dortmund, D-44221 Dortmund, Germany
Email: olaf.stursberg@uni-dortmund.de

Abstract: Procedures like start-up, product change-over, or shutdown of processing systems usually involve the manipulation of continuous and discrete controls. To optimize such procedures, the use of hybrid models is often appropriate to account for the change of dynamics during the transition. Some recently developed approaches for hybrid system optimization are based on mixed-integer linear programming. Applications reveal that the complexity inherent to these approaches considerably limits the applicability to industrial-size problems. This paper suggests as an alternative a tailor-made algorithm that optimizes the discrete and continuous degrees of freedom in a two-stage procedure. While the discrete controls are selected by a graph search algorithm, the continuous controls are obtained by embedded nonlinear programming. The method is illustrated for the start-up of a CSTR.

Keywords: Automata, Hybrid Systems, Integer Programming, Nonlinear Programming, Dynamic Optimization.

1. INTRODUCTION

It is certainly the majority of processing systems for which the operation is determined by the combination of continuous and discrete controls. While the continuous inputs are usually set by standard feedback control loops, discrete controls (like on-off actors occurring in form of valves, pumps, heaters, etc.) are determined predominantly by logic controllers. Traditionally, the control tasks that involve logical decisions and those which establish continuous regulation are separated and not designed in conjunction. For operations like start-up, shutdown, product change-over, or also the sequential operation of batch processes, the simultaneous consideration of both types of controls seems necessary. This is true in particular for the optimization of above-named procedures, since a separated design can not account for potentially opposing effects of both control components. This paper thus advocates a simultaneous optimization of both types of controls.

An important effect in model-based optimization of the considered procedures is that the process dynamics usually changes significantly. Is is often not possible to obtain one model that is sufficiently accurate for the complete procedure, but a set of models is required to represent the variation of the dynamics. Hybrid dynamic models have been identified in recent years as a suitable means to account for dynamic-dependent transitions between different models and for the combination of continuous as well as discrete (switching) inputs, see e. g. (Lynch *et al.*, 2003).

The task addressed in this paper is to model the process dynamics as a hybrid dynamic system, to formulate the transition procedure as an optimization problem, and – most importantly – to develop an algorithm to efficiently compute the (optimal) control inputs. The cost to be minimized can be the transition time in the simplest case, or more complex cost functions which depend on the state and input variables. Different approaches to the

optimization of hybrid systems have been published in recent years, ranging from rather generic formulations to tailor-made methods for certain subtypes of hybrid systems, see e.g. (Branicky *et al.*, 1998; Avraam *et al.*, 1998; Broucke *et al.*, 2000; Gokbayrak and Cassandras, 2000; Hedlund and Rantzer, 2002; Barton and Lee, 2002; Shaikh and Caines, 2003; Oldenburg *et al.*, 2003). One branch of methods follows the idea of transforming the hybrid dynamics into a set of algebraic (in-)equalities that serve as constraints for a mixed-integer program (Bemporad and Morari, 1999; Stursberg *et al.*, 2002). If all constraints are written in linear form, mixed-integer linear (or quadratic) programming (MILP, MIQP) can be used for solution, i.e., standard solvers that employ branch-and-bound strategies, where bounds are obtained from linear relaxations, can be used. In (Till *et al.*, 2003) it has been shown exemplarily for the approach in (Stursberg *et al.*, 2002) that a drawback is the somewhat limited applicability when it comes to efficient solution for large-scale systems. The investigation allows the conclusion that one reason for the complexity is the fact that the MILP method does not restrict the search for the optimum to the true degrees of freedom (the model inputs) but has to cope with a large number of auxiliary integer variables and constraints introduced to map logical into algebraic constraints.

As an alternative, this paper suggest a method with the following characteristics: (a) the discrete degrees of freedom are determined by a tree search algorithm with tailor-made heuristics to determine the optimal discrete control sequence with low effort, (b) the continuous degrees of freedom are obtained from solving embedded (nonlinear) programming problems, (c) the cost function is evaluated by hybrid simulation which takes care of the state-dependent structural changes of the model.

2. HYBRID SYSTEM OPTIMIZATION

The type of model used to represent the dynamics to be considered for the transition procedure is first defined precisely. As argued in (Stursberg and Engell, 2002), the following type of hybrid automaton is suited for this purpose since it accounts for different nonlinear dynamics assigned to certain regions of the continuous state space (or different modes of operation), and for the combination of continuous and discrete inputs. The latter are useful to model, e.g., actuators with an 'on/off'-type of behavior.

Definition 2.1. Hybrid Automaton A
The hybrid automaton $A = (X, U, V, Z, inv, T, g, r, f)$ consists of the following components:

- the continuous state space $X \subseteq \mathbb{R}^{n_x}$ on which the *state vector* x is defined [1];
- the continuous input space $U = [u_1^-, u_1^+] \times \ldots [u_{n_u}^-, u_{n_u}^+]$ with $u_j^-, u_j^+ \in \mathbb{R}$ and the *continuous inputs* $u \in U$;
- the finite discrete input space $V = \{v_1, \ldots, v_{nd}\}$ with *discrete inputs* $v \in V$ for which $v_j \in \mathbb{R}^{n_v}$;
- the finite set of *locations* $Z = \{z_1, \ldots, z_{n_z}\}$;
- a mapping $inv : Z \rightarrow 2^X$ which assigns a non-empty *invariant* of the form $inv(z_j) = \{x \mid \exists\ n_{p_j} \in \mathbb{N}, C_j \in \mathbb{R}^{n_{p_j} \times n_x}, d_j \in \mathbb{R}^{n_{p_j}}, x \in X : C_j \cdot x \leq d_j\}$ to each location $z_j \in Z$;
- the set of *transitions* $\Theta \subseteq Z \times Z$, where a transition from $z_1 \in Z$ into $z_2 \in Z$ is denoted by (z_1, z_2);
- a mapping $g : \Theta \rightarrow 2^X$ that associates a *guard* $g((z_1, z_2)) \subseteq X$ with each $(z_1, z_2) \in \Theta$ such that: $g((z_1, z_2)_j) = \{x \mid \exists\ n_{g_j} \in \mathbb{N}, C_j \in \mathbb{R}^{n_{g_j} \times n_x}, d_j \in \mathbb{R}^{n_{g_j}} : C_j \cdot x \leq d_j\}$. Given a state $z_1 \in Z$, it is required for all pairs of transitions originating from z_1 that the corresponding guards are disjoint;
- a *reset function* $r : \Theta \times X \rightarrow X$ which assigns an updated state $x' \in X$ to each $(z_1, z_2) \in \Theta$ and $x \in g((z_1, z_2))$;
- a *flow function* $f : Z \times X \times U \times V \rightarrow \mathbb{R}^n$ that defines a continuous vector field $\dot{x} = f(z, x, u, v)$ for each location $z \in Z$.

Semantics: Let $T = \{t_0, t_1, t_2, \ldots\}$ be an ordered set of time points $t_k \in \mathbb{R}^{\geq 0}$ which contains the initial time t_0, and all points of time at which an input change or a transition occurs. The states $x_k := x(t_k)$, $z_k := z(t_k)$ and the inputs $u_k = u(t_k)$, $v_k = v(t_k)$ are defined on T, i.e. their values are piecewise constant on intervals $[t_k, t_{k+1}[$, $k \in \mathbb{N} \cup \{0\}$. For given input sequences $u(t_k)$ and $v(t_k)$, a *feasible run* ϕ_σ of A is then defined as a sequence: $\phi_\sigma = (\sigma(t_0), \sigma(t_1), \sigma(t_2), \ldots)$ of hybrid states $\sigma_k := \sigma(t_k) = (z_k, x_k)$ such that:

- *Initialization*: $\sigma(t_0)$ is initialized to a given $z_0 = z(t_0) \in Z$ and $x_0 = x(t_0) \in X$ with $x_0 \in inv(z_0)$, $x_0 \notin g((z_0, \bullet))$ for any $(z_0, \bullet) \in \Theta$.
- *Progress*: $\sigma(t_{k+1})$ results from $\sigma(t_k)$ by:
  - (i) continuous evolution: $\chi : [0, \tau] \rightarrow X$, $\tau \in \mathbb{R}^{>0}$ where $\chi(t_0) = x_k$, $\dot{\chi}(t) = f(z_k, \chi(t), u_k, v_k)$ with an existing unique solution for $t \in [0, \tau]$, and $\chi(t) \in inv(z_k)$ but for all $g(z_k, \bullet) \in \Theta$ and $t \in [0, \tau[$: $\chi(t) \notin g(z_k, \bullet)$;
  - (ii) followed by a transition: $(z_k, z_{k+1}) \in \Theta$, $\chi(\tau) \in g(z_k, z_{k+1})$, and $x_{k+1} = r((z_k, z_{k+1}), \chi(\tau)) \in inv(z_{k+1})$. □

For the procedure developed in the next section it is important to note that a run of A is deterministic for given input trajectories. In particular, a transition is taken immediately if the correspond-

[1] For all parameters $n_\bullet$ in Def. 2.1 applies $n_\bullet \in \mathbb{N}$.

ing unique guard condition is met. Compared to the model used before in this context (Stursberg and Engell, 2002), the automaton in Def. 2.1 comprises two extensions: (a) the invariants $inv(z)$ do not necessarily establish a partition of the state space X, but can possibly overlap, such that different dynamics can be valid for the same region of X if different operating conditions apply; (b) if a transition is taken, the continuous state can be updated by the reset function r.

The objective of optimizing transition procedures of processing systems is now cast into a minimization problem employing the model according to Def. 2.1 as constraints. To be optimized is the procedure of driving the system from an initial state $\sigma(t_0)$ into a set of hybrid target states, defined as $\Sigma_{tar} = (z_{tar}, X_{tar})$ with $z_{tar} \in Z$ and $X_{tar} \subset inv(z_{tar})$. An additional requirement for the transition procedure is that a set of forbidden regions $F = \{F_1, \ldots, F_{n_F}\}$ is never entered, where each region $F_j = (z_{F,j}, X_{F,j})$ is a hybrid state set with $X_{F,j} \subset inv(z_{F,j})$. These regions are suitable to model the exclusion of undesired operating conditions, e.g., the concentration ranges of explosive mixtures.

The task is then to find the input trajectories that minimize a given cost functional Ω for the transition procedure. For simplicity of notation, $\phi_u = (u_0, u_1, \ldots, u_f)$ is written for a continuous input trajectory on a finite time set $T = \{t_0, \ldots, t_f\}$ and Φ_u for the set of all input trajectories of this length; likewise, $\phi_v = (v_0, v_1, \ldots, v_f)$ is a discrete input trajectory and Φ_v contains all ϕ_v. Given the sets Φ_u and Φ_v, the set of all corresponding feasible runs [2] is denoted by Φ_σ. The objective is to determine the input trajectories ϕ_u^*, ϕ_v^* that lead to a feasible run ϕ_σ^* which minimizes the cost function according to:

$$\min_{\phi_u \in \Phi_u, \phi_v \in \Phi_v} \Omega(t_f, \phi_\sigma, \phi_u, \phi_v) \tag{1}$$

$$\begin{aligned} s.t. \quad & \phi_\sigma = (\sigma_0, \ldots, \sigma_f) \text{ with: } \sigma_0 = (z_0, x_0), \\ & \sigma_f := (z(t_f), x(t_f)) \in \Sigma_{tar}, \text{ and for } \phi_\sigma \\ & \text{applies in each phase of cont. evolution} \\ & \text{(Def. 2.1)} : \chi(t) \notin F_j \ \forall \ F_j \in F, \ \forall \ t \in [0, \tau]. \end{aligned}$$

A slightly simplified version of this problem has been tackled in (Stursberg and Engell, 2002; Stursberg *et al.*, 2002) by an approach comprising the following steps: (a) the flow functions are approximated by discrete-time and linear models, (b) all logical parts of the model are reformulated in algebraic linear form such that a mixed-integer linear program results, (c) the optimization problem is solved in an MPC-like fashion where mixed-integer linear programming (MILP) is applied on a limited look-ahead horizon in each step.

[2] Of course, some combinations of ϕ_u and ϕ_v do not lead to feasible runs.

It has been identified as a drawback of this technique that the transformation into algebraic form requires a large set of auxiliary integer variables (and auxiliary constraints). When applying branch-and-bound techniques for MILP, the search tree encoding the discrete alternatives thus is considerably larger than one would expect from the true set of discrete degrees of freedom (i.e., the cardinality of V for each time point in T). As a consequence for larger systems, the method could be applied only for rather short prediction horizons. This is the motivation for an alternative procedure that does not use a completely algebraic formulation of the transition dynamics of A.

3. A GRAPH SEARCH ALGORITHM

The key idea is to separate the optimization of the continuous and discrete degrees of freedom in the following sense: The discrete choices (i. e., the input trajectories ϕ_v) are determined by a graph search algorithm resembling the well-known principle of shortest-path search. For each node contained in the search graph, an embedded optimization for the continuous degrees of freedom (and optionally for relaxed discrete degrees of freedom for future steps) is carried out. Within this embedded nonlinear programming, numerical simulation is employed to evaluate the hybrid dynamics of A, leading to a cost evaluation for the referring system evolution. These costs are used in the graph search to apply a branch-and-bound strategy, i.e., upper (and lower) bounds on the optimal costs for the transition procedure are iteratively computed to prune branches of the search tree as soon as possible.

In detail, the algorithm operates on a directed acyclic graph in which the nodes represent a system state at a time t_k for the case that a discrete input $v \in V$ has been applied in the previous step. As illustrated in Fig. 1, the graph can be understood as a representation for the hybrid states reachable over the time set T if all possible discrete input trajectories are applied. Clearly, the size of the graph grows exponentially with the number of discrete decisions, i.e. according to $|V|^{|T|-1}$ – hence, an objective of the optimization algorithm is to prune branches as quickly as possible. The information stored with each node is a structure $n = (\phi_\sigma, \phi_u, \phi_v, c_a, c_p)$, i.e. it contains the hybrid state trajectory ϕ_σ up to the current state (z_k, x_k), the input trajectories ϕ_u and ϕ_v that led to this state, as well as the costs c_a accumulated on the path into (z_k, x_k), and (optionally) a prediction c_p for the costs encountered on the remaining path into the target.

For a state trajectory ϕ_σ that leads from the initial state to the target, a *solution* is defined as a tuple

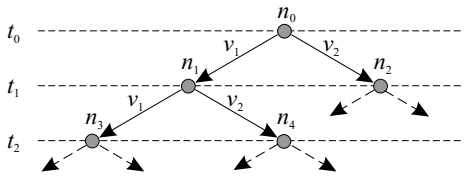

Fig. 1. Graph encoding the discrete choices for $V = \{v_1, v_2\}$.

$\phi_{sol} = (\phi_\sigma, \phi_u, \phi_v, c_a)$, and Φ_{sol} as the set of all found solutions. If, furthermore, the initial node is specified as $n_0 = (\sigma_0, -, -, 0, \infty)$ and n_{max} as the number of nodes to be explored maximally, the algorithm can be written as shown in Fig. 3: The set N denotes the set of all feasible nodes explored during the algorithm, L is the set of *live nodes* starting from which the system evolution still has to be investigated further, G is the set of nodes generated in the last iteration, and S are the nodes selected from L or G in the current iteration. This selection is realized by a function *select* which returns S depending on a chosen search mode. In an implementation of the algorithm the modes *breadth-first* ($S := L$), *depth-first* ($S := \{n_{best}\}$, where n_{best} is the node from G with minimum cost), and *best-first* ($S := \{n_{best}\}$ with n_{best} as the best node from L) can be chosen. In experiments

$N = L = G := \{n_0\}$;
$S := \emptyset$; $\Phi_{sol} := \emptyset$;
WHILE $(L \neq \emptyset) \ \wedge \ (|N| < n_{max})$ DO {
  $S := select(L, G)$;
  $G := \emptyset$;
  FOR ALL $(n \in S)$ DO {
    IF $(n.c_a \leq ub)$:
      FOR ALL $(v \in V)$ DO {
        $(\sigma, u, \hat{\phi}_x, \hat{\phi}_u, \hat{\phi}_v) = extend(n, v)$;
        $\phi_\sigma := (\phi_\sigma, \sigma)$;
        $\phi_u := (\phi_u, u)$;
        $\phi_v := (\phi_v, v)$;
        $c_a = history(\phi_\sigma, \phi_u, \phi_v)$;
        $c_p = future(\hat{\phi}_\sigma, \hat{\phi}_u, \hat{\phi}_v)$;
        $n_{new} = (\phi_\sigma, \phi_u, \phi_v, c_a, c_p)$;
        IF $(n_{new} \notin N) \ \wedge \ (feasible(\sigma))$:
          IF $(\sigma \in \Sigma_{tar})$:
            $N := N \cup \{n_{new}\}$;
            $\Phi_{sol} := \Phi_{sol} \cup \{(\phi_\sigma, \phi_u, \phi_v, c_a)\}$;
            IF $(c_a < ub)$: $ub := c_a$; END
          ELSE:
            IF $(c_a < ub)$:
              $N := N \cup \{n_{new}\}$; $G := G \cup \{n_{new}\}$;
            END
          END
        END }
    END }
  $L := (L \cup G) \setminus S$; }
$\phi^*_{sol} = \min_{\phi_{sol} \in \Phi_{sol}} \phi_{sol}.c_a$

Fig. 2. The optimization algorithm.

for different examples the following strategy has been found to be efficient: (a) the depth-first mode is applied first to find a feasible solution quickly (and thus start the pruning over upper bounds, see below), and then the procedure continues with the best-first or breadth-first mode; (b) the selection of n_{best} in both modes is based on the notion of a minimum distance of the current state σ_k to Σ_{tar}. These costs include, e.g., a Euclidean distance in X and a distance in Z defined as the number of transitions in Θ to be taken in order to reach z_{tar}.

The algorithm proceeds with computing successors for each $v \in V$ for all those nodes n in S, for which the costs $n.c_a$ accumulated along the path from n_0 to n do not exceed an upper bound ub for the costs for the complete path from n_0 to the target. Branches with $n.c_a > ub$ are pruned at this point. The computation of successors is accomplished by the function *extend*. It performs for the node n (reached at time t_k) and the corresponding input v_k the following optimization for an ordered time set $T' = \{t_k, \ldots, t_p\}$ with $p > k$:

$$\min_{\hat{\phi}_u, \hat{\phi}_v} \Omega(t_p, \hat{\phi}_\sigma, \hat{\phi}_u, \hat{\phi}_v) \tag{2}$$

where $\hat{\phi}_u = (u_k, u_{k+1}, \ldots, u_{p-1})$ and $\hat{\phi}_v = (v_{k+1}, \ldots, v_{p-1})$. It is important for the components of the trajectory $\hat{\phi}_v$ that they are not restricted to the set V, but each component of the vector v is relaxed to the range between the minimum and maximum value that occurs in V for the respective component. The optimization in Eq. 2 does then not contain any discrete degrees of freedom anymore, and can be solved by standard techniques for nonlinear programming. During this optimization, the solver computes the state trajectory $\hat{\phi}_x = (x_k, x_{k+1}, \ldots, x_p)$ that belongs to a guess for $\hat{\phi}_u$ and $\hat{\phi}_v$. This task can be solved by numerical simulation of the corresponding run of A and involves the following steps: (a) evaluating the continuous evolution, (b) detecting that a guard condition is satisfied, and (c) executing the transition with the associated reset function. The values of $\hat{\phi}_x$, $\hat{\phi}_u$, and $\hat{\phi}_v$ determine the costs $\Omega(t_p, \hat{\phi}_\sigma, \hat{\phi}_u, \hat{\phi}_v)$. As a result of the optimization, the function *extend* returns a hybrid successor state σ_{k+1}, the continuous input u_k to get there, and the predicted trajectories over the complete prediction horizon T'.

The function *history* determines the accumulated costs c_a from the initial state to the new hybrid state. A function $future$, on the other hand, computes future costs c_p over the predicted trajectories $\hat{\phi}_x$, $\hat{\phi}_u$, and $\hat{\phi}_v$. In the simplest case, c_p can just encode the distance of the last state in $\hat{\phi}_x$ to the target, i.e., it represents an estimation of how close the system can get to the target within the horizon T'. c_p is used by *select* to steer the exploration of the graph. The values c_p and

c_a together with the extended state and input trajectories determine the new node n_{new}.

If n_{new} was not contained in N before and if it is feasible, i.e., σ_{k+1} is within X but not in any forbidden region, it is considered for inclusion in N. If, in addition, σ_{k+1} is in the target, a new solution has been found and is included in Φ_{sol}. If the accumulated cost of n_{new} is below the current value of ub, this bound is updated. If on the other hand $\sigma_{k+1} \notin \Sigma_{tar}$ and the accumulated costs $n_{new}.c_a$ are still below ub, the node is included in N and G. If $n_{new}.c_a \geq ub$, this path is not further considered. At the end of an iteration the set L of live nodes is updated. The loop continues until L is empty, or the maximum number of nodes n_{max} is exceeded. If, at this stage, the set Φ_{sol} is non-empty, the element ϕ^*_{sol} with the smallest accumulated costs is the best solution found.

Two extensions of the algorithm are currently investigated: (i) A second important criterion for efficiently pruning the search graph is the use of lower bounds lb for the costs of the complete transition procedure: If the predicted state trajectory $\hat{\phi}_\sigma$ ends in Σ_{tar}, the associated costs c_p are a lower bound for the costs of the remaining path into Σ_{tar} (since the discrete degrees of freedom are relaxed to continuous ones). Hence, if the sum of c_a and c_p for a node is greater than the current value of ub, the node can not be an element of the optimal path, and the branch of the graph can be pruned. For certain cost functions (as, e. g., time-optimality of the transition procedure) this pruning can be applied even if $\hat{\phi}_\sigma$ does not end in Σ_{tar}. This means that the criterion is applicable also if the embedded nonlinear programming leads to solutions only for rather short prediction horizons T'. (ii) Another use of the result of the function *extend* is to take $\hat{\phi}_v$ as a heuristics for further exploring the subgraph emerging from the current node. As an obvious choice, one would first follow that sequence of discrete inputs ϕ_v in which the components are nearest to the relaxed input values of $\hat{\phi}_v$.

4. APPLICATION TO A CSTR START-UP

The method is illustrated by using the example of the start-up of a continuous stirred tank reactor (CSTR), as described in (Stursberg and Panek, 2002). The system consists of a tank equipped with two inlets, a heating coil, a cooling jacket, a stirrer, and one outlet (see Fig. 4). The inlets feed the reactor with two dissolved substances A and B which react exothermically to form a product D. The inlet flows F_1 and F_2 (with temperatures T_1 and T_2) can be switched discretely between two values each. The outlet flow F_3 is controlled continuously. In order to heat

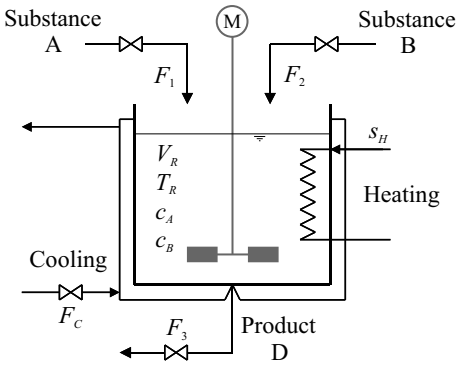

Fig. 3. Scheme of the CSTR.

up the reaction mixture to a desired temperature range with a high reaction rate, the heating can be switched on (denoted by a discrete variable $s_H \in \{0, 1\}$). The continuously controlled cooling flow F_C serves as a means to remove an excess of heat once the reaction has started. It is the objective for this system to determine the input trajectories that drive the initially empty reactor into a status of desired operation in which the liquid volume V_R, the temperature T_R, and the concentrations c_A and c_B have reached nominal ranges. Additionally, regions with $T_R \geq 360$ and $V_R \geq 1.6$ are forbidden.

To model the system, the state vector is defined as $x := (V_R, T_R, c_A, c_B)^\mathrm{T}$, the continuous input vector as $u := (F_3, F_C)^\mathrm{T}$, and the discrete input vector as $v := (F_1, F_2, s_H)^\mathrm{T}$. Depending on the continuous state, the system dynamics can be written as $\dot{x} = f(z, x, u, v)$ where:

- for z_1 with $V_R \in [0.1, 0.8]$:

$$\boldsymbol{f}^I = \begin{pmatrix} F_1 + F_2 - F_3 \\ (F_1(T_1 - T_R) + F_2(T_2 - T_R))/V_R \\ \quad + F_C k_1 (T_C - T_R)(k_2/V_R + k_3) - k_4 q \\ (F_1 c_{A,1} - c_A(F_1 + F_2))/V_R + k_9 q \\ (F_2 c_{B,2} - c_B(F_1 + F_2))/V_R + k_{10} q \end{pmatrix}$$

- for z_2 with $V_R \in]0.8, 2.2]$:

$$\boldsymbol{f}^{II} = \left(f_1^I,\ f_2^I + s_H k_6 (T_H - T_R)(k_7 - \frac{k_8}{V_R}),\ f_3^I,\ f_4^I \right)^\mathrm{T}$$

and $q = c_A c_B^2 \exp(-k_5/T_R)$. The separation in two V_R-regions accounts for the fact that the heating is only effective above $V_R = 0.8$. The initial state is $x_0 = (0.1, 300, 0, 0)^\mathrm{T}$ and the target is given by z_2 and a hyper-ball with radius 0.1 around the continuous state $x_{tar} = (1.5, 345, 0.4, 0.2)^\mathrm{T}$. The optimization is run with the cost criterion that the transition time for the startup procedure is minimized. The strategy is chosen that depth-first search is used until a solution is found, then a breadth-first strategy is applied. For selecting the nodes from L and G, the criterion of minimum distance to the target is

selected. Figure 4 shows the best solution obtained for a search with $n_{max} = 400$ nodes and with $|T'| = 2$. The result has been obtained within 125 seconds and is qualitatively comparable to the one obtained in 13 minutes with the method in (Stursberg *et al.*, 2002).

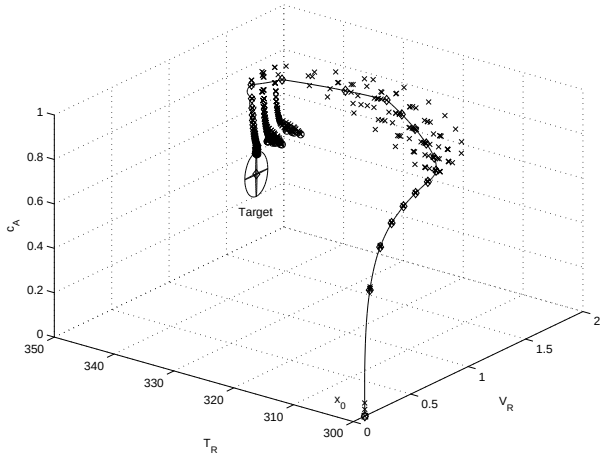

Fig. 4. CSTR: The optimal x-trajectory (solid line) is projected into the (V_R, T_R, c_a)-space. Explored nodes are marked by crosses.

5. CONCLUSIONS

Since space limitations exclude a detailed assessment of the proposed algorithm (including a comparison to other techniques), only the main differences to the approach in (Stursberg and Engell, 2002) are summarized briefly: (a) the method introduced in this paper evaluates directly the original hybrid model A and does not operate with linearized dynamics; (b) the hybrid model is extended with respect to the location invariants and resets (whereas not relevant for the example in Sec. 4); (c) the search is restricted to the true degrees of freedom and does not have to cope with auxiliary variables and constraints; (d) experiments also for a scalable example treated in (Till *et al.*, 2003) have shown that the computation times are by an order of magnitude smaller for the majority of tested configurations, (e) unlike the MPC-like approach in (Stursberg and Engell, 2002), which discards all but the best trajectory in each iteration, the graph search stores alternative paths in the set of live nodes for later exploration.

The focus of the current work is to introduce suitable heuristics to steer the graph search, to develop a scheme for adapting the time steps (rather than using constant sampling intervals), and to use tailor-made modifications for the embedded nonlinear programming to improve its performance.

6. REFERENCES

Avraam, M. P., R. Shah and C. C. Pantelides (1998). Modelling and optimisation of general hybrid systems in the cont. time domain. *Comp. Chem. Eng.* **22 (Suppl.)**, 221–228.

Barton, P.I. and C.K. Lee (2002). Modeling, simulation, sensitivity analysis and optimization of hybrid systems. *ACM Trans. Modeling and Comp. Simulation* **12**(4), 256–289.

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

Branicky, M. S., V. S. Borkar and S. K. Mitter (1998). A unified framework for hybrid control: Model and optimal control theory. *IEEE Trans. Automatic Control* **43**(1), 31–45.

Broucke, M., M.D. Di Benedetto, S. Di Gennaro and A. Sangiovanni-Vincentelli (2000). Theory of optimal control using bisimulations. In: *Hybrid Systems: Comp. and Control*. Vol. 1790 of *LNCS*. Springer. pp. 89–102.

Gokbayrak, K. and C. G. Cassandras (2000). Hybrid controllers for hierarchically decomposed systems. In: *Hybrid Systems*. Vol. 1790 of *LNCS*. Springer. pp. 117–129.

Hedlund, S. and A. Rantzer (2002). Convex dynamic programming for hybrid systems. *IEEE TAC* **47**(9), 1539–1540.

Lynch, N., R. Segala and F. Vaandrager (2003). Hybrid i/o automata. *Information and Computation* **185**(1), 105–157.

Oldenburg, J., W. Marquardt, D. Heinz and D. Leineweber (2003). Mixed logic dynamic optimization applied to batch distillation process design. *AICHE Journ.* **49**, 2900–2917.

Shaikh, M.S. and P.E. Caines (2003). On the optimal control of hybrid systems. In: *Hybrid Systems: Comp. and Control*. Vol. 2623 of *LNCS*. Springer. pp. 466–481.

Stursberg, O. and S. Engell (2002). Optimal control of switched continuous systems using mixed-integer programming. In: *Proc.* 15^{th} *IFAC World Congr. on Automatic Control*. Vol. Th-A06-4.

Stursberg, O. and S. Panek (2002). Control of switched hybrid systems based on disjunctive formulations. In: *Hybrid Systems*. Vol. 2289 of *LNCS*. Springer. pp. 421–435.

Stursberg, O., S. Panek, J. Till and S. Engell (2002). Generation of optimal control policies for systems with switched hybrid dynamics. In: *Modelling, Analysis, and Design of Hybrid Systems*. Vol. 279 of *LNCIS*. Springer. pp. 337–352.

Till, J., S. Engell, S. Panek and O. Stursberg (2003). Empirical complexity analysis of a milp-approach for optimization of hybrid systems. In: *IFAC Conf. Analysis and Design of Hybrid Systems*. pp. 159–164.