A Box-Based Distance between Regions for Guiding the Reachability Analysis of SpaceEx

Sergiy Bogomolov[1], Goran Frehse[2], Radu Grosu[3], Hamed Ladan[1], Andreas Podelski[1], and Martin Wehrle[1,4]

[1] University of Freiburg, Germany
{bogom,ladanh,podelski}@informatik.uni-freiburg.de
[2] Université Joseph Fourier Grenoble 1 – Verimag, France
goran.frehse@imag.fr
[3] Vienna University of Technology, Austria
radu.grosu@tuwien.ac.at
[4] University of Basel, Switzerland
martin.wehrle@unibas.ch

Abstract. A recent technique used in falsification methods for hybrid systems relies on distance-based heuristics for guiding the search towards a goal state. The question is whether the technique can be carried over to reachability analyses that use regions as their basic data structure. In this paper, we introduce a box-based distance measure between regions. We present an algorithm that, given two regions, efficiently computes the box-based distance between them. We have implemented the algorithm in SpaceEx and use it for guiding the region-based reachability analysis of SpaceEx. We illustrate the practical potential of our approach in a case study for the navigation benchmark.

1 Introduction

The theory of hybrid systems provides a rich and popular framework for the representation of systems which incorporate both continuous and discrete behavior [2, 29]. This framework has been utilized for the purpose of modeling and analyzing a large range of practically relevant systems. In particular, hybrid systems have been used for the analysis of automotive controllers [4], real-time circuits [30], and biological systems [5, 1, 28, 35, 6, 18, 7, 21, 20].

An important problem in the context of hybrid systems is the problem to determine whether a given set of bad states can be reached from an initial state. If such a set cannot be reached, the hybrid system is called safe. Unfortunately, this reachability analysis problem is decidable only for a restricted class of hybrid systems [2, 22]. In order to be able to prove safety for richer classes of hybrid systems, state-of-the-art reachability tools such as SpaceEx [16] make use of over-approximations [34, 10, 16]. Furthermore, a lot of attention has recently been devoted to *falsification* based on testing techniques. Testing techniques are tuned to find unsafe behaviors rather than to prove safety [9, 8, 19, 11, 33,

31]. In general, falsification techniques are of great interest in industrial applications especially in the early phases of system development. The majority of these techniques are inspired from motion planning, and construct in a numeric execution-based way a rapidly exploring random tree (RRT). If the tree ends up in an unsafe state, then one has found an unsafe behavior.

In this work, we propose a best-first symbolic-reachability analysis algorithm (GBFS) which combines safety analysis and falsification. This algorithm has been added as an alternative to the current DFS algorithm of SpaceEx. GBFS produces the same result, in similar amount of time as DFS, if the bad states of the system cannot be reached from its initial states, despite SpaceEx inherent over-approximations. However, GBFS is much faster than DFS on our benchmarks in producing a symbolic counterexample if the system is potentially unsafe. The heuristic used to guide the search is based on an appropriate cost measure for hybrid systems based on dwell times.

For a given state s, the cost measure estimates the search effort to reach an error state from s. The search preferably explores states with smaller estimated costs, and thus avoids exploring unnecessary states. Obviously, to obtain an overall efficient model checking approach, cost measures are both supposed to guide the search accurately and to be efficiently computable. Guided search has recently been successfully applied in the context of discrete and timed systems [13, 32, 25, 24, 27, 12, 36, 26]. In those contexts, the costs of a state s have been defined as the smallest number of transitions in the state space to reach a nearest error state from s. Overall, guided search has shown to be able to significantly improve the efficiency of model checking for these classes of systems.

While measuring the costs of states in terms of transitions in the state space is an appropriate method for discrete and timed systems, the situation becomes slightly more complex for the more general class of hybrid systems. In the context of hybrid systems, the overall model checking time specifically depends on the operations to compute the continuous post of states because this operation is most expensive during the exploration of the state space. Moreover, in contrast to discrete and timed systems, the costs of the post operation depend on the *dwell time*, i. e., the amount of time spent in the corresponding location. This dependence occurs, at least in practice, because common tools like SpaceEx compute the post operation based on iteratively computing the continuous image of the region in intermediate points. Therefore, in the context of hybrid systems, it is desirable to explore traces with low accumulated dwell time.

We theoretically show that, for a certain class of hybrid systems, our cost measure provides desired search behavior. Moreover, although hard to compute exactly, the representation of our cost measure lends itself to an accurate (box-based) cost measure as an approximation. This approximation turns out to be efficiently computable and, although surprisingly simple, to accurately guide the search in the state space. Our experiments with SpaceEx show the practical potential on challenging benchmarks.

The paper is organized as follows. In Sec. 2, we introduce the preliminaries for the paper. Sec. 3 introduces the main contribution of this work based on a

trajectory-based cost measure. This cost measure is experimentally evaluated in Sec. 4. Finally, we conclude the paper in Sec. 5.

2 Preliminaries

In this section, we give the preliminaries for this paper. In Sec. 2.1, we introduce our computational model. In Sec. 2.2, we present a basic reachability algorithm for the state space exploration. Based on the reachability algorithm, guided search and cost measures are introduced in Sec. 2.3.

2.1 Notation

In this paper, we consider models that can be represented by hybrid systems.

Definition 1 (Hybrid System). *A* hybrid system *is formally a tuple* $\mathcal{H} = (Loc, Var, Init, Flow, Trans, Inv)$ *defining*

- *the finite set of locations* Loc,
- *the set of continuous variables* $Var = \{x_1, \ldots, x_n\}$ *from* $\mathbb{R}^n$,
- *the initial condition, given by the constraint* $Init(\ell)$ *for each location* ℓ,
- *the continuous transition relation, given by the expression* $Flow(\ell)(v)$ *for each continuous variable* v *and each location* ℓ. *We assume* $Flow(\ell)$ *to be of the form*
$$\dot{x}(t) = Ax(t) + u(t), u(t) \in \mathcal{U},$$
where $x(t) \in \mathbb{R}^n$, A *is a real-valued* $n \times n$ *matrix and* $\mathcal{U} \subseteq \mathbb{R}^n$ *is a closed and bounded convex set.*
- *the discrete transition relation, given by a set* $Trans$ *of discrete transitions; a discrete transition is formally a tuple* (ℓ, g, ξ, ℓ') *defining*
  - *the source location* ℓ *and the target location* ℓ',
  - *the guard, given by a linear constraint* g,
  - *the update, given by a affine mapping* ξ,
- *the invariant, given by the linear constraint* $Inv(\ell)$ *for each location* ℓ.

A *state* of the hybrid system $\mathcal{H}$ is a tuple $(\ell, \mathbf{x})$ consisting of a location $\ell \in Loc$ and a point $\mathbf{x} \in \mathbb{R}^n$, i. e., $\mathbf{x}$ is an evaluation of the continuous variables in Var.

The semantics of a hybrid system is defined in terms of its *trajectories*. Let $\mathcal{T} = [0, \Delta]$ for $\Delta \geq 0$. A trajectory of a hybrid system $\mathcal{H}$ from state $s = (\ell, \mathbf{x})$ to state $s' = (\ell', \mathbf{x}')$ is defined by a tuple $\rho = (L, \mathbf{X})$, where $L : \mathcal{T} \to Loc$ and $\mathbf{X} : \mathcal{T} \to \mathbb{R}^n$ are functions that define for each time point in $\mathcal{T}$ the location and values of the continuous variables, respectively. For a given trajectory ρ, we define a sequence of time points where location switches happen by $(\tau_i)_{i=0\ldots k} \in \mathcal{T}^{k+1}$. In such case we say that the trajectory ρ has discrete length $|\tau| = k$. Trajectories $\rho = (L, \mathbf{X})$ (and the corresponding sequence $(\tau_i)_{i=0\ldots k}$) have to satisfy the following conditions:

- $\tau_0 = 0$, $\tau_i < \tau_{i+1}$, and $\tau_k = \Delta$ – the sequence of switching points increases, starts with 0 and ends with Δ

- $L(0) = \ell$, $\mathbf{X}(0) = \mathbf{x}$, $L(\Delta) = \ell'$, $\mathbf{X}(\Delta) = \mathbf{x}'$ – the trajectory starts in $s = (\ell, \mathbf{x})$ and ends in $s' = (\ell', \mathbf{x}')$
- $\forall i\ \forall t \in [\tau_i, \tau_{i+1}) : L(t) = L(\tau_i)$ – the location is not changed during the continuous evolution
- $\forall i\ \forall t \in [\tau_i, \tau_{i+1}) :\ \mathbf{X}(t) \in \mathit{Flow}(L(\tau_i))$, i.e. $\dot{\mathbf{X}}(t) = A\mathbf{X}(t) + u(t)$ holds and thus the continuous evolution is consistent with the differential equations of the corresponding location
- $\forall i\ \forall t \in [\tau_i, \tau_{i+1}) :\ \mathbf{X}(t) \in \mathit{Inv}(L(\tau_i))$ – the continuous evolution is consistent with the corresponding invariants
- $\forall i\ \exists (L(\tau_i), g, \xi, L(\tau_{i+1})) \in \mathit{Trans} : \mathbf{X}_{end}(i) = \lim_{\tau \to \tau_{i+1}^-} \mathbf{X}(\tau) \wedge \mathbf{X}_{end}(i) \in g \wedge \mathbf{X}(\tau_{i+1}) = \xi(\mathbf{X}_{end}(i))$ – every continuous transition is followed by a discrete one, $\mathbf{X}_{end}(i)$ defines the values of continuous variables right before the discrete transition at the time moment τ_{i+1} whereas $\mathbf{X}_{start}(i) = \mathbf{X}(\tau_i)$ denotes the values of continuous variables right after the switch at the time moment τ_i.

We say that s' is *reachable* from s if a trajectory from s to s' exists. The reachable state space $\mathcal{R}(\mathcal{H})$ of $\mathcal{H}$ is defined as the set of states such that a state s is contained in $\mathcal{R}(\mathcal{H})$ iff s is reachable from S_{init}. In this paper, we also refer to *symbolic states*. A symbolic state $s = (\ell, R)$ is defined as a tuple, where $\ell \in \mathit{Loc}$, and R is a convex and bounded set consisting of points $\mathbf{x} \in \mathbb{R}^n$. The symbolic part of a symbolic state is also called *region*. The symbolic state space of $\mathcal{H}$ is called the *region space*. The initial set of states S_{init} of $\mathcal{H}$ is defined as $\bigcup_\ell (\ell, \mathit{Init}(\ell))$.

In this paper, we assume there is a given set of *error states* that violate a given property. Our goal is to find a trajectory from S_{init} to an error state. A trajectory that starts in a state s and leads to an error state is called an *error trajectory* $\rho_e(s)$. As already outlined in the introduction, the time to compute an error trajectory $\rho_e(s)$ can significantly depend on the accumulated *dwell times* of the locations in $\rho_e(s)$. Therefore, we define the costs of a state s as

$$\mathit{cost}(s) := \min_{\rho_e(s)} \sum_{i=0}^{|\tau|-1} \delta_i,$$

i. e., as the minimal sum of dwell times ranged over the error traces that start in s, where $\delta_i = \tau_{i+1} - \tau_i$. Error trajectories can be found by searching in the state space. We will give a short introduction to this search-based approach in the next section.

2.2 Finding Error States Through Search

We introduce a standard basic reachability algorithm along the lines of the reachability algorithm used by SpaceEx. It works on the region space of a given hybrid system. The algorithm checks if a symbolic error state in a given set S_{error} is reachable from a given set of symbolic initial states S_{init}. We define a symbolic state s in the region space of $\mathcal{H}$ to be an error state if there is a symbolic state

$s_e \in S_{error}$ such that s and s_e agree on their discrete part, and the intersection of the regions of s and s_e is not empty.

Algorithm 1 Basic reachability algorithm

Input: Set of initial symbolic states S_{init}, set of error states S_{error}
Output: Can a symbolic state in S_{error} be reached from a symbolic state in S_{init} ?

```
1: PUSH (L_passed, S_init)
2: PUSH (L_waiting, S_init)
3: i ← 0
4: while (L_waiting ≠ ∅ ∧ i < i_max) do
5:    s_curr = GETNEXT (L_waiting)
6:    i ← i + 1
7:    if s_curr ∈ S_error then
8:       return "Error state reached"
9:    end if
10:   S' ← COMPUTESUCCESSORS(s_curr)
11:   for all s' ∈ S' do
12:      if s' ∉ L_passed then
13:         PUSH (L_passed, s')
14:         PUSH (L_waiting, s')
15:      end if
16:   end for
17: end while
18: return "Error state not reachable"
```

Starting with the set of initial symbolic states from S_{init}, the algorithm explores the region space of a given hybrid system by iteratively computing symbolic successor states until an error state is found, no more states remain to be considered, or a (given) maximum number of iterations i_{max} is reached.

In the following, we explain Alg. 1 in more detail. Symbolic states for which the successor states have been computed are called *explored*, whereas symbolic states that have been computed but not yet explored are called *visited*. Both visited and explored states are stored in a dedicated data structure $\mathcal{L}_{passed}$. Symbolic states in $\mathcal{L}_{passed}$ are used to detect cycles in the region space (see below). Moreover, there is a data structure $\mathcal{L}_{waiting}$ that contains visited states that have not necessarily been explored yet. An iteration of the algorithm consists of several steps. First, a symbolic state s is taken from $\mathcal{L}_{waiting}$ and checked if s is an error state. If this is the case, the algorithm terminates. If s is not an error state, the symbolic successor states of s are computed (which in turn is a 2-step operation consisting of the computation of the discrete and continuous post state; we omit a more detailed description here). To avoid exploring cycles in the region space, symbolic successor states that are already contained in $\mathcal{L}_{passed}$ are not considered again; the others are stored in $\mathcal{L}_{waiting}$.

The order in which the region space is explored by Alg. 1 depends on the implementation of $\mathcal{L}_{waiting}$ (e. g., a queue-based implementation corresponds to

breadth-first search). Specifically, *guided search* preferably explores states that appear to be more promising according to a cost measure. We will describe this approach in more detail in the next section.

2.3 Guided Search

Guided search is an instantiation of the basic reachability algorithm that has been introduced in the previous section. As a first characteristic of guided search algorithms, $\mathcal{L}_{waiting}$ is implemented as a priority queue. Therefore, the PUSH function additionally requires a priority (cost) value for the pushed state, and the GETNEXT function (line 5 in Alg. 1) returns a state with best priority according to the cost measure. In the following, we discuss a desirable property of cost measures in the context of guided search. As already outlined, we intend to design a cost measure that guides the search well in the region space. To achieve good guidance, the relative error of a cost measure h to the *cost* function as defined in the previous section is not necessarily correlated to the accuracy of h. In other words, h may accurately guide the search although the relative error of h's cost estimations is high. This is because it suffices for h to always select the "right" state to be explored next.[1] Based on this observation, we give the definition of *order-preserving*.

Definition 2 (Order-Preserving). *Let $\mathcal{H}$ be a hybrid system. A cost measure h is* order-preserving *if for all states s and s' with $cost(s) < cost(s')$, then also $h(s) < h(s')$.*

Cost measures that are order-preserving lead to perfect search behavior with respect to the *cost* function. Therefore, it is desirable to have cost measures that satisfy this property. We will come back to this point in the next section.

3 The Box-Based Distance Measure

In this section, we present the main contribution of this work. In Sec. 3.1, we provide a conceptual description of an idealized distance measure based on the length of trajectories. This idealized distance measure is used as the basis for our box-based distance measure which is presented in Sec. 3.2.

3.1 A Trajectory-Based Distance Measure

In this section, we formulate a distance measure *dist* that can be expressed in terms of the length of trajectories (see below for a justification of the name). For states s and s', the distance measure $dist(s, s')$ is defined as the minimal length

[1] As a simple example, consider two states s and s' with real costs 100 and 200, respectively. Furthermore, consider a cost measure that estimates the costs of these states as 1 and 2, respectively. We observe that the relative error is high, but the better state is determined nevertheless.

of a trajectory ρ that is obtained from the continuous flow and discrete switches of trajectories that lead from s to s'. To define this more formally, we denote the set of trajectories that lead from s to s' with $\mathcal{T}(s, s')$. Moreover, $dist_{eq}(\mathbf{x}, \mathbf{x}')$ denotes the Euclidean distance between points $\mathbf{x}$ and $\mathbf{x}'$. Using this notation, we give the definition of our trajectory-based distance measure.

Definition 3 (Trajectory-Based Distance Measure). *Let $\mathcal{H}$ be a hybrid system, let s and s' be states of $\mathcal{H}$. We define the distance measure*

$$dist(s, s') := \min_{\rho \in \mathcal{T}(s,s')} \sum_{i=0}^{|\tau|-1} \left(\int_{\tau_i}^{\tau_{i+1}} \sqrt{\dot{x}_1^2(t) + \cdots + \dot{x}_n^2(t)}\ dt + dist_{eq}(i, i+1) \right),$$

where $\rho = (L, \mathbf{X})$, $\mathbf{X}(t) = (x_1(t), \ldots, x_n(t))$, and $dist_{eq}(i, i+1)$ is a short-hand for $dist_{eq}(\mathbf{X}_{end}(i), \mathbf{X}_{start}(i+1))$.

Informally speaking, the distance between states s and s' is defined as the length of a shortest trajectory ρ from s to s' induced by the differential equations and discrete updates of the visited locations $L(\tau_i)$ in ρ. Obviously, the trajectory-based distance measure can be applied to error states in a straightforward way by setting s' to an error state. We call the trajectory-based error distance measure $dist_E(s) := \min_{s_e} dist(s, s_e)$, where s_e ranges over the set of given error states of $\mathcal{H}$.

In the following, we show that for a certain class of hybrid systems $\mathcal{H}$, $dist(s)$ is indeed correlated to the costs of s for all states s of $\mathcal{H}$. In fact, this correlation can be established for hybrid systems such that

1. all differential equations in $\mathcal{H}$ are of the form $\dot{x}_i(t) = \pm c_i$ for every continuous variable $x_i \in Var$ and a constant $c_i \in \mathbb{N}$, and
2. all guards in $\mathcal{H}$ do not contain discrete updates.

We call hybrid systems that satisfy the above requirements *restricted systems*. Specifically, we observe that a necessary condition for hybrid system $\mathcal{H}$ to be a restricted system is that for every continuous variable x_i in $\mathcal{H}$, there is a global constant $c_i \in \mathbb{N}$ such that *all* differential equations in $\mathcal{H}$ that talk about x_i only differ in the sign. It is not difficult to see that for the class of restricted systems, the length of the obtained flow is linearly correlated with the time. Therefore, the error distance measure $dist_E$ is order-preserving.

Proposition 1. *For restricted systems $\mathcal{H}$, $dist_E$ is order-preserving.*

Proof. We show that from $cost(s) < cost(s')$, it follows that $dist_E(s) < dist_E(s')$. As $\mathcal{H}$ is a restricted system, the square root of $\dot{x}_1^2(t) + \cdots + \dot{x}_n^2(t)$ is constant and $dist_{eq}(i, i+1)$ is equal to zero. Therefore, $dist_E(s) = \min_{s_e} \min_{\rho \in \mathcal{T}(s,s_e)} c \sum \delta_i$, which is equal to $c \cdot cost(s)$. This proves the claim.

Prop. 1 leads to an interesting and important observation. Roughly speaking, we have reduced the problem of computing (dwell time) costs in the state

space to the problem of computing "shortest" flows between regions. Therefore, Prop. 1 shows that under certain circumstances, we can choose between *cost* and *dist* without loosing precision. However, although still hard to compute, the representation of *dist* based on lengths of flows lends itself to an approximation based on *estimated* flow lengths. This approximation is presented in the next section.

3.2 The Box-Based Approximation

In the following, we propose an effective approximation of the *dist* function that we have derived in the last section. While the *dist* measure has been defined for concrete states, our box-based approximation is defined for symbolic states. The approximation is based on the following two ingredients.

1. Instead of computing the exact length of trajectories between two points $\mathbf{x}$ and $\mathbf{x}'$ (as required in Def. 3), we use the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$.
2. As we are working in the region space, we approximate a given region R with the smallest box $\mathcal{B}$ such that R is contained in $\mathcal{B}$. This corresponds to the well-known principle of Cartesian abstraction.

In the following, we will discuss these ideas and make them precise. As stated, we define the estimated distance between points $\mathbf{x}$ and $\mathbf{x}'$ as the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$. Unfortunately, the Euclidean distance is not order-preserving for restricted systems, but only for even more restricted systems that allow even less behavior. This is formalized in the following proposition. For a state $s = (\ell, \mathbf{x})$, we define $dist_E^{eq}(s) := \min_{s_e} dist_{eq}(\mathbf{x}, \mathbf{x}_e)$, where $s_e = (\ell_e, \mathbf{x}_e)$ ranges over the error states, and $dist_{eq}$ is the Euclidean distance function as introduced earlier.

Proposition 2. *For restricted systems $\mathcal{H}$ with $\dot{x}_i(t) = c_i$, i. e., for restricted systems where all locations have the same continuous behavior, $dist_E^{eq}$ is order-preserving.*

Proof. We show that from $cost(s) < cost(s')$, it follows that $dist_E^{eq}(s) < dist_E^{eq}(s')$. By assumption, $\mathcal{H}$ is a restricted system where every location has the same continuous dynamics. Therefore, the Euclidean distance $dist_{eq}(s, s_e)$ is equal to $\int_0^{\tau_k} \sqrt{\dot{x}_1^2(t) + \cdots + \dot{x}_n^2(t)}\, dt$, where τ_k is equal to the accumulated dwell time of the trajectory from s to s_e. Furthermore, the square root of $\dot{x}_1^2(t) + \cdots + \dot{x}_n^2(t)$ is some constant c. Thus $dist_E^{eq}(s) = \min_{s_e} dist_{eq}(s, s_e) = \min_{s_e} c \cdot \tau_k = c \cdot \min_{s_e} \tau_k$ which in turn is equal to $c \cdot cost(s)$.

The above proposition reflects that the Euclidean distance is a coarse approximation of the trajectory-based distance measure because it is effectively only order-preserving for systems that allow behavior that corresponds to systems with only one location. Indeed, it is the coarsest approximation one can think of on the one hand. However, on the other hand, we have shown that there *exist*

systems for which it *is* order-preserving, which suggests (together with Prop. 1) that the Euclidean distance could be a good heuristic to estimate distances also for richer classes of hybrid systems. Moreover, it is efficiently computable which is particularly important for distance heuristics that are computed on-the-fly during the state space exploration. (Obviously, one can think of arbitrary more precise approximations based on piecewise linear functions; however, such approximations also become more expensive to compute. We will come back to this point in the conclusions.)

For our distance heuristic, we approximate a given symbolic state $s = (\ell, R)$ with the smallest box $\mathcal{B}(s)$ that contains R. Formally, this corresponds to the requirement

$$R \subseteq \mathcal{B}(s) = [x_1, x_1'] \times \ldots [x_n, x_n'] \subseteq \mathbb{R}^n \wedge \forall \mathcal{B}' \neq \mathcal{B}(s) : R \subseteq \mathcal{B}' \Rightarrow \mathcal{B}(s) \subseteq \mathcal{B}'.$$

In order to be efficiently computable, it is essential that tight over-approximating boxes can be computed efficiently. This can be achieved using linear programming techniques. Our distance heuristic h^{eq} is defined as the Euclidean distance between the center of two boxes. Formally, for a symbolic state $s = (\ell, R)$, we define

$$h^{eq}(s) := min_{s_e}\, dist_{eq}(\textsc{Center}(\mathcal{B}(R)), \textsc{Center}(\mathcal{B}(R_e))),$$

where $s_e = (\ell_e, R_e)$ ranges over the set of error states of $\mathcal{H}$, $dist_{eq}$ is the Euclidean distance metric, and $\textsc{Center}(B)$ denotes the central point of box B. Obviously, central points of boxes can be computed efficiently as the arithmetic average of its lower and upper bounds for every dimension.

Overall, our distance heuristic h^{eq} determines distance estimations for symbolic states $s = (\ell, R)$ by first over-approximating R with the smallest box $\mathcal{B}$ that contains R, and then computing the minimal Euclidean distance between $\mathcal{B}$'s center and the center of an error state. This procedure is summed up by Alg. 2.

Algorithm 2 Compute Distance Heuristic h^{eq}

Input: State $s = (\ell, R)$
Output: Estimated distance to a closest error state in S_{error}

```
1: d_min ← ∞
2: B ← B(R)
3: for all s' = (ℓ', R') ∈ S_error do
4:    B' ← B(R')
5:    d_curr ← dist_eq(Center(B), Center(B'))
6:    if d_curr < d_min then
7:       d_min ← d_curr
8:    end if
9: end for
10: return d_min
```

4 Experiments

We have implemented our box-based distance heuristic h^{eq} in SpaceEx and compare the resulting guided search algorithm to the standard depth-first search (DFS) algorithm of SpaceEx. The experiments have been performed on a machine with an Intel Core i3 2.40GHz processor and with 4 GB of memory. For both search settings (i. e., for guided search as well as for uninformed DFS), SpaceEx has been run with the same parameters (see Sec. 4.2).

In the following, we first introduce our benchmark problems in Sec. 4.1. Afterwards, the experimental results are presented and discussed in Sec. 4.2.

4.1 Case Studies

For the evaluation of our approach, we used several problem instances of the two case studies *Navigation benchmark* and *System of Tanks*.

Navigation benchmark As a first case study, we apply the *navigation benchmark* that has been proposed in the literature [14]. In the scope of this benchmark, we consider an object moving in the plane. The plane is divided into the grid of squares where some initial state is given (i. e., region, velocity in direction x and velocity in direction y). Furthermore, for each square, some differential equations are defined which govern the system in the considered square. Finally, some square A which should be reached and some square B which is to be avoided are defined. We will look for a path from the initial state to square A. We consider different problem instances of this benchmark with different sizes of the grid.

System of tanks As a second benchmark which is similar to the one presented by Frehse and Maler [17], we consider a network of tanks which are connected by channels. Tanks have a fixed capacity, channels are characterized by their throughputs. We have several kinds of tanks with different functionality, namely *production*, *buffer*, *delivery buffer*, and *reactor*. The functionality of these tanks is as follows. Liquid can be delivered from the production to one of the buffers B_1, B_2 or B_3. In particular, liquid can be stored in those buffers for some time. Furthermore, liquid is forwarded to reactors R_1, R_2 and R_3. Liquid can be transferred only from B_i to R_i. Finally, liquid is transferred from the reactors to the delivery buffer B_D from which it is directly delivered to the customer.

For a given control strategy, it must be ensured that the delivery buffer B_D never gets empty according to this strategy (i. e., requirements of the customer are satisfied). For a fixed $i \in \{1, 2, 3\}$, we have equal throughput v_i from production to B_i, from B_i to R_i, and from R_i to B_D. Finally, the transmission rate from B_D to the customer is defined by v_{out}.

We will investigate the behavior of the following controller. The controller has the following phases which are characterized by constant m defining the length of the time period when liquid is transferred between tanks:

1. $time = 0$: $open(Production, B_i)$ - transfer of liquid from $Production$ to B_i starts
2. $time \in [0, m]$: $Production \rightarrow B_i$ - buffer B_i is filled
3. $time = m$: $close(Production, B_i)$ - stop filling B_i
4. $time = m$: $open(B_i, R_i)$ - start transfer to R_i
5. $time \in [m, 2m]$: $B_i \rightarrow R_i$ - liquid is transferred from the buffer B_i to the reactor R_i
6. $time = 2m$: $close(B_i, R_i)$ - stop transferring to R_i
7. $time = 2m$: $open(R_i, B_D)$ - start transfer to B_D
8. $2m$ sec - $3m$ sec: $R_i \rightarrow B_D$ - liquid is transferred from the reactor R_i to the delivery buffer B_D, i.e. delivery buffer is refilled.
9. $time = 3m$: $close(R_i, B_D)$ - stop refilling B_D. After this phase the controller goes back to the phase 1.

The controller can non-deterministically choose which buffer (and reactor) to use in each iteration. Thus the number of trajectories grows exponentially with time. We assume the consumption to be constant. Thus it is essential to transfer to the delivery buffer enough liquid so it does not get empty within phases in which the level of liquid in the delivery buffer sinks.

Our goal is to check whether the presented controlled may fail, i. e., may lead to the drainage of the production buffer, and to discover the possible failure as soon as possible. For our test, we set v_1, v_2, v_3 such that $v_3 > v_1 > v_2$, and v_{out} such that the delivery buffer will get empty no matter which buffer is chosen by the controller. However, choosing B_2 with the smallest throughput (and thus refillment) rate v_2 will obviously lead to the faster drainage of the delivery buffer and therefore to the faster discovery of the controller failure. Contrarily, the unfortunate choice of v_3 will lead to the delay in the controller failure discovery.

4.2 Experimental Results

The experimental results for the largest problem instances in the navigation benchmark (NAV25, ..., NAV30) are provided in Table 1. The results have been obtained using the LGG support function scenario of SpaceEx. Template polyhedra are represented using 32 uniform directions. Furthermore, the maximal number of iterations is set to 200, and the continuous sampling time is set to 0.1 seconds. Finally, the local time horizon for the continuous post operation is set to 40. We compare the number of iterations, the search time, as well as the overall accumulated dwell time during the exploration of the state space of SpaceEx. The accumulated dwell time serves as an additional measure to compare the "quality" of the search guidance because, as argued in the previous sections, this time is correlated with overall search effort, and our distance heuristic tries to minimize it.

First, we observe that in all these problem instances, the guided search algorithm with our box-based distance heuristic could significantly improve the overall performance of the model checking process. Specifically, we observe that the overall accumulated dwell time is reduced when guiding the search, which

Table 1. Experimental results of SpaceEx for the navigation benchmark with uninformed depth-first search and guided search. Abbreviations: DFS: depth-first search, DTime: overall accumulated dwell time for all explored states, time in s: overall search time of SpaceEx in seconds

Benchmark instance	DFS			Guided search		
	#iterations	DTime	time in s	#iterations	DTime	time in s
NAV25	200	245.7	160.157	43	111.1	44.891
NAV26	200	391.5	327.66	44	110.2	57.95
NAV27	200	539.6	366.621	49	121.4	59.212
NAV28	47	96.3	63.176	34	99.8	50.528
NAV29	162	410.5	217.521	42	133.1	66.479
NAV30	174	308.6	176.457	40	129.4	69.779

apparently results in a lower search effort. Furthermore, the number of iterations of SpaceEx reduces. As a side remark, for NAV25, NAV26 and NAV27, the standard depth-first search did not find a solution after the maximal number of 200 iterations.

Let us consider the results for the largest navigation benchmark problem, NAV30, in more detail. Fig. 1 and Fig. 2 graphically compare the way of the object in NAV30 while moving over the 25×25 grid and searching for the target state. The initial region of the object is on the left above, the goal region is on the right below. We observe that, using depth-first-search as shown in Fig. 1, the object reaches the target state on a trace with a considerable (circle-shaped) detour on the one hand. On the other hand, using guided search with the h^{eq} distance heuristic as shown in Fig. 2, the way of the object apparently becomes more straight. As a consequence, with uninformed depth-first search, SpaceEx needs 174 iterations and over 176 seconds to reach the target state. In contrast, with guided search, SpaceEx finds the target state within only 40 iterations, resulting in a remarkable speed-up considering the overall search time.

Considering the experimental results for the benchmark problems based on the system of tanks, we first report that our h^{eq} distance heuristic does not provide further guidance information for the state space exploration in the classical search setting. More precisely, for a symbolic state s of that system, all successor states of s are equally evaluated by h^{eq}. This shows that, unsurprisingly, h^{eq} does not fire appropriately in all problem domains. However, h^{eq} can still give benefits compared to uninformed depth-first search in the context of *look-aheads* in the state space. For a given state s, a look-ahead works by not only considering heuristic values of the direct successor states of s, but also by considering heuristic values of successor states in a fixed depth greater than one. In Table 2, we report experimental results within this search setting. We have used the PHAVer scenario of SpaceEx. In addition, the maximal number of iterations is set to 250000, and the continuous sampling time is set to 1 second. Finally, the local time horizon for the continuous post operation is set to 200. The problems TANK01, ..., TANK04 are benchmark instances of increasing complexity that differ in the initial level of liquid in the delivery buffer.

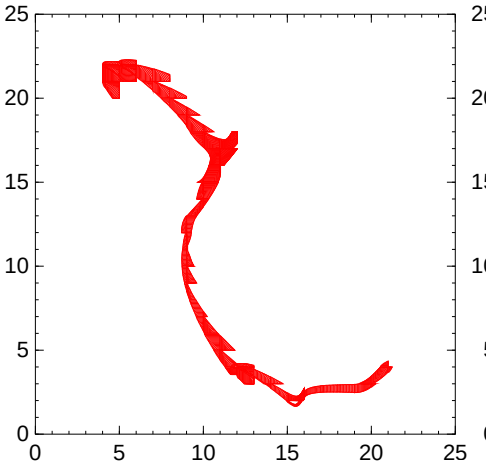

Fig. 1. Depth-first search

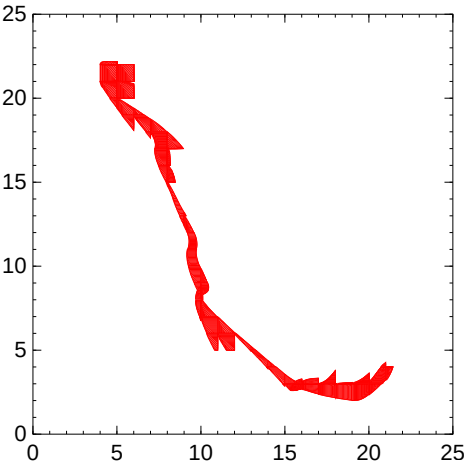

Fig. 2. Guided search with h^{eq}

Table 2. Experimental results of SpaceEx for the tank benchmark with uninformed depth-first search and guided search with look-ahead. Abbreviations: DFS: depth-first search, DTime: overall accumulated dwell time for all explored states, time in s: overall search time of SpaceEx in seconds

Benchmark instance	DFS			Guided search		
	#iterations	DTime	time in s	#iterations	DTime	time in s
TANK01	1396	34900	7.356	207	3850	3.962
TANK02	3712	92800	42.179	526	9850	8.952
TANK03	6034	150850	105.22	846	15850	14.876
TANK04	8349	208750	196.69	1166	21850	21.311

Considering these results, we again observe that guided search can significantly outperform uninformed depth-first search of SpaceEx. In particular, the number of iterations and the overall search time could be considerably reduced.

We conclude the section with a short discussion for which kind of systems our approach is suited best. First, we observe that our heuristic is especially accurate when the system dynamics depends only on the continuous state. In particular, this is the case for the navigation benchmark. In general, systems of that kind occur in practice when complex dynamics is approximated with a simpler one through state space partitioning: For example, the phase portrait approximations [23], the approximation techniques employed by PHAVer [15] and hybridization techniques [3] fall into this category. Additionally considering look-aheads (i. e., considering not only heuristic values of direct successor states, but also of states in a fixed depth greater than one) is useful when crucial changes of a system state may arise after performing *several* steps (as it is the case in the system of tanks).

5 Conclusions

In this paper we have introduced a best-first symbolic-reachability analysis algorithm (GBFS) for a particular class of hybrid systems, the ones that have a linear behavior (in the control-theoretic sense) in each mode. The algorithm has been added as an additional reachability-analysis engine to SpaceEx, the state-of-the-art reachability-analysis tool for this class of systems [16].

GBFS takes advantage of the symbolic-computation routines of SpaceEx, and in particular of its efficient computation of the smallest box enclosing a symbolic region. As a consequence, the algorithm has a similar time-complexity for reachability analysis as the depth-first search algorithm (DFS) of this tool.

GBFS is tuned for efficient falsification, where it considerably outperforms DFS on our benchmarks. The improved efficiency is achieved by choosing the successor region which has the smallest Euclidean distance between the center of its enclosing box and the center of the box enclosing the bad-region. We have shown that for a particular class of hybrid systems, this metric is an appropriate approximation of an idealized trajectory metric. Our experimental evaluation additionally shows that this metric can serve as an informed cost heuristic even for richer classes of hybrid systems.

For the future, it will be interesting to further refine our box-based distance metric. In this paper, we have chosen the most canonical way as an approximation; however, we also have already outlined that arbitrary more precise approximations based on piecewise linear functions are possible. In this context, an important topic for future research will be the question how much precision can be gained while still being efficiently computable.

Acknowledgments

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, http://www.avacs.org/).

References

1. R. Alur, C. Belta, F. Ivancic, V. Kumar, M. Mintz, G. Pappas, H. Rubin, and J. Schug. Hybrid modeling and simulation of biomolecular networks. In *Hybrid Systems: Computation and Control*, pages 19–32, 2001.
2. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.H. Ho, X. Nicolin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
3. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.
4. A. Balluchi, L. Benvenuti, M. D. Di Benedetto, C. Pinello, and A. L. Sangiovanni-Vincentelli. Automotive engine control and hybrid systems: challenges and opportunities. *Proceedings of the IEEE*, 88(7):888–912, July 2000.

5. P. Barbano, M. Spivak, J. Feng, M. Antoniotti, and B. Misra. A coherent framework for multi-resolution analysis of biological networks with memory: Ras pathway, cell cycle and immune system. In *National Academy of Science*, pages 6245–6250, 2005.
6. G. Batt, C. Belta, and R. Weiss. Model checking genetic regulatory networks with parameter uncertainty. In *Hybrid Systems: Computation and Control*, pages 61–75, 2007.
7. C. Belta, P. Finin, L. Habets, A. Halasz, M. Imielinski, V. Kumar, and H. Rubin. Understanding the bacterial stringent response using reachability analysis of hybrid systems. In *Hybrid Systems: Computation and Control*, pages 111–126, 2004.
8. A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *Hybrid Systems: Computation and Control*, pages 451–471, 2004.
9. M.S. Branicky and M.M. Curtiss. Nonlinear and hybrid control via RRTs. In *Symp. on Mathematical Theory of Networks and Systems*, 2002.
10. C. Chutinan and B.H. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, 2003.
11. A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. *Hybrid Systems: Computation and Control*, pages 174–189, 2007.
12. K. Dräger, B. Finkbeiner, and A. Podelski. Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer*, 11(1):27–37, 2009.
13. S. Edelkamp, S. Leue, and A. Lluch-Lafuente. Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology Transfer*, 5(2):247–267, 2004.
14. A. Fehnker and F. Ivančić. Benchmarks for hybrid systems verification. In *Hybrid Systems: Computation and Control*, pages 381–397, 2004.
15. G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *Hybrid Systems: Computation and Control*, pages 258–273. Springer, 2005.
16. G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395, 2011.
17. G. Frehse and O. Maler. Reachability analysis of a switched buffer network. In *Hybrid Systems: Computation and Control*, pages 698–701, 2007.
18. R. Ghosh and C.J. Tomlin. Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modeling: Delta-notch protein signaling. *IEEE Transactions on Systems Biology*, 1(1):170–183, 2004.
19. A. Girard and G. Pappas. Verification using simulation. In *Hybrid Systems: Computation and Control*, pages 272–286, 2006.
20. R. Grosu, G. Batt, F. Fenton, J. Glimm, C. Le Guernic, S.A. Smolka, and E. Bartocci. From cardiac cells to genetic regulatory networks. In *Computer Aided Verification*, pages 396–411, 2011.
21. R. Grosu, S.A. Smolka, F. Corradini, A. Wasilewska, E. Entcheva, and E. Bartocci. Learning and detecting emergent behavior in networks of cardiac myocytes. *Communications of the ACM (CACM)*, 52(3):1–10, March 2009.
22. T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *ACM Symposium on Theory of Computing*, pages 373–382, 1995.
23. T. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. *Hybrid Systems III*, pages 377–388, 1996.

24. S. Kupferschmid, K. Dräger, J. Hoffmann, B. Finkbeiner, H. Dierks, A. Podelski, and G. Behrmann. Uppaal/DMC – abstraction-based heuristics for directed model checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 679–682, 2007.
25. S. Kupferschmid, J. Hoffmann, H. Dierks, and G. Behrmann. Adapting an AI planning heuristic for directed model checking. In *International SPIN Workshop*, pages 35–52, 2006.
26. S. Kupferschmid and M. Wehrle. Abstractions and pattern databases: The quest for succinctness and accuracy. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 276–290, 2011.
27. S. Kupferschmid, M. Wehrle, B. Nebel, and A. Podelski. Faster than Uppaal? In *Computer Aided Verification*, pages 552–555, 2008.
28. P. Lincoln and A. Tiwari. Symbolic systems biology: Hybrid modeling and analysis of biological networks. In *Hybrid Systems: Computation and Control*, pages 660–672, 2004.
29. N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Inf. and Comp.*, 185(1):103–157, 2003.
30. O. Maler and S. Yovine. Hardware timing verification using kronos. In *Israeli Conference on Computer Systems and Software Engineering*, 1996.
31. E. Plaku, L. Kavraki, and M. Vardi. Hybrid systems: From verification to falsification. In *Computer Aided Verification*, pages 463–476, 2007.
32. K. Qian and A. Nymeyer. Guided invariant model checking based on abstraction and symbolic pattern databases. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 497–511, 2004.
33. S. Ratschan and J.G. Smaus. Verification-Integrated falsification of Non-Deterministic hybrid systems. In *Analysis and Design of Hybrid Systems*, 2006.
34. B. Silva, O. Stursberg, B. Krogh, and S. Engell. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *IEEE Conf. on Decision and Control*, pages 2867—2874, 2001.
35. A. Singh and J. Hespanha. Models for generegulatory networks using polynomial stochastic hybrid systems. In *CDC05*, 2005.
36. M. Wehrle and M. Helmert. The causal graph revisited for directed model checking. In *Symposium on Static Analysis*, pages 86–101, 2009.