S-Taliro: A Tool for Temporal Logic Falsification

for Hybrid Systems

by

Yashwanth Singh Rahul Annapureddy

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2011 by the
Graduate Supervisory Committee:

Georgios Fainekos, Chair
Yann-Hang Lee
Sandeep Gupta

ARIZONA STATE UNIVERSITY

December 2011

ABSTRACT

S-Taliro is a fully functional Matlab toolbox that searches for trajectories of minimal robustness in hybrid systems that are implemented as either m-functions or Simulink/State flow models. Trajectories with minimal robustness are found using automatic testing of hybrid systems against user specifications. In this work we use Metric Temporal Logic (MTL) to describe the user specifications for the hybrid systems. We then try to falsify the MTL specification using global minimization of robustness metric. Global minimization is carried out using stochastic optimization algorithms like Monte-Carlo (MC) and Extended Ant Colony Optimization (EACO) algorithms. Irrespective of the type of the model we provide as an input to S-Taliro, the user needs to specify the MTL specification, the initial conditions and the bounds on the inputs. S-Taliro then uses this information to generate test inputs which are used to simulate the system. The simulation trace is then provided as an input to Taliro which computes the robustness estimate of the MTL formula. Global minimization of this robustness metric is performed to generate new test inputs which again generate simulation traces which are closer to falsifying the MTL formula. Traces with negative robustness values indicate that the simulation trace falsified the MTL formula. Traces with positive robustness values are also of great importance because they indicate how robust the system is against the given specification. S-Taliro has been seamlessly integrated into the Matlab environment, which is extensively used for model-based development of control software. Moreover the toolbox has been developed in a modular fashion and therefore adding new optimization algorithms is easy and straightforward. In this work I present the architecture of S-Taliro and its working on a few benchmark problems.

DEDICATION

To my parents, sister and family

ACKNOWLEDGMENTS

First and foremost I offer my sincerest gratitude to my advisor, Prof. Georgios Fainekos, who has supported me throughout my thesis with his immense knowledge and excellence. His dedication and enthusiasm towards research is very inspirational. He has been an excellent mentor and guide throughout my thesis program.

I would like to extend my sincere thanks to Prof. Yann-Hang Lee and Prof. Sandeep K.S. Gupta for agreeing to be on my master's thesis committee and for all the valuable suggestions they made.

I would not have done my master's without my sister. She has truly been a role model throughout my life. Her continuous support and encouragement is unforgettable. Thank You for everything.

This acknowledgement would be incomplete without mentioning the following names. Siddhartha, Siddhant, Raviteja, Vamsy and Vivek, thank you so much for making my stay memorable. I am forever grateful to you guys.

I owe a lot to my parents. I cannot put into words the affection, love and blessing they show towards me. They always support me for everything I do. I would like to thank all my family members who supported me throughout my stay here.

This work would not have been possible without the support of the industry partners of the NSF I/UCRC for Embedded Systems at the Arizona State University and the NSF award CNS-1017074.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

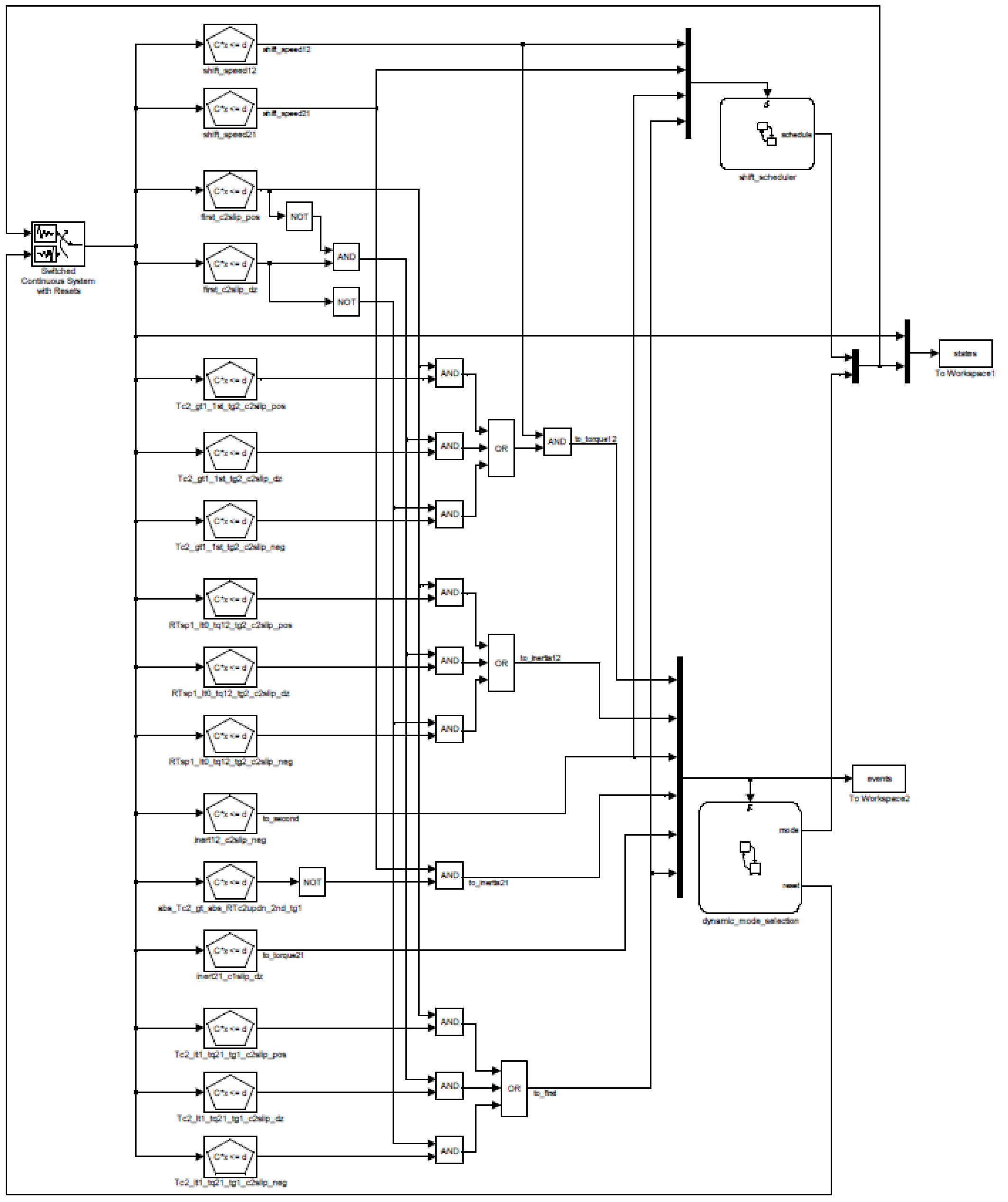
S-Taliro [1] is a Matlab toolbox that searches for falsifying trajectories of temporal logic properties of Simulink/State flow models. The toolbox can analyze arbitrary Simulink models or user-defined functions which are used to model the system. In this thesis I present the architecture of the toolbox, its usage and conclude with a few application examples.

1.1 Motivation

Hybrid systems are essentially dynamical systems which exhibit both discrete and continuous behaviors. These systems arise from a combination of continuous and discrete inputs, outputs, states or dynamics. In general, however hybrid systems arise whenever one inter-mixes logical decision making with the generation of continuous valued control laws. Typical examples of these hybrid systems include constrained robotic systems, biological systems [2], flight control and management systems and analog/digital circuit co-design and verification.

In Figure 1 I include the Simulink model for the Power Train System which clearly depicts the interaction between discrete and continuous behaviors for a hybrid system. In this model the Switched Continuous System with resets indicates the continuous dynamics for the hybrid system whereas the State Flow Charts indicate the discrete behavior for the hybrid system.

Analysis and safety verification of hybrid systems is generally considered to be critical and challenging because of the complex behaviors these systems can exhibit. Therefore practitioners are inclined to manual testing of individual components of the hybrid system as this gives them more control on the testing process. This process however is extremely time-consuming and hence various authors have come up with several methods for the safety verification [3] – [6] of hybrid systems. These methods take into consideration the complex behaviors that the hybrid systems can exhibit and also the user specifications.


C*x <= d
shift_speed12
shift_speed21
first_c2slip_pos
first_c2slip_dz
NOT
AND
OR
Switched Continuous System with Resets
Tc2_gt1_1st_tg2_c2slip_pos
Tc2_gt1_1st_tg2_c2slip_dz
Tc2_gt1_1st_tg2_c2slip_neg
RTsp1_lt0_tq12_tg2_c2slip_pos
RTsp1_lt0_tq12_tg2_c2slip_dz
RTsp1_lt0_tq12_tg2_c2slip_neg
inert12_c2slip_neg
to_second
abs_Tc2_gt_abs_RTc2updn_2nd_tg1
inert21_c1slip_dz
to_torque21
Tc2_lt1_tq21_tg1_c2slip_pos
Tc2_lt1_tq21_tg1_c2slip_dz
Tc2_lt1_tq21_tg1_c2slip_neg
to_torque12
to_inertia12
to_inertia21
to_first
schedule
shift_scheduler
states
To Workspace1
events
To Workspace2
mode
reset
dynamic_mode_selection
Figure 1: Power Train System in Simulink

Classic temporal logics have limitations on modeling real time systems as they cannot deal with quantitative temporal requirements. As a result Metric Temporal Logic which includes quantitative requirement on the elapse of time is used in this work. Metric Temporal Logic which was introduced by Koymans [7] is very useful for describing the user specifications/ real-time requirements of hybrid systems. MTL is essentially a formal language which closely resembles the natural language and hence can be used by practitioners easily to describe the specifications for hybrid systems.

MTL falsification of hybrid systems involves the ability to prove as well as falsify temporal logic properties of systems. Informally MTL falsification is the search through all the possible system trajectories to find a trajectory that does not satisfy the system specification. In this thesis I use the robust semantics of MTL as described in [8]. This semantics can be used to indicate if a system trajectory satisfies a given specification. Moreover it can be used to indicate how robustly it satisfies the specification which is of importance.

As an example, we describe a MTL specification for the Power Train System which captures a certain property that we are trying to falsify. Say the specification for the system is that there exists a gear transition "second-to-first-second" during the course of the simulation of the hybrid system. This is the specification that we try to falsify using S-Taliro.

1.2 Problem Formulation

In this section we describe the formal definition of the problem that we are trying to solve. We consider a system $\boldsymbol{\Sigma}$ which essentially maps a set of initial conditions X_0 and input signals U to output signals Y. Essentially we define the system as

$$\Sigma : X_0 \times U \rightarrow Y$$

i.e. given a point $x_0 \in X_0$ and an input signal $u \in U$, then

$$y = \Sigma(x_0, u)$$

Our goal in this case is to verify the correctness of the system by keeping track of the output signals for particular input signals and initial conditions. Essentially we are trying to find those system trajectories that falsify or are closest to falsifying a given specification for the system. By falsification, we intend to find the system trajectories with "negative robustness" with respect to the specification or trajectories with minimal positive robustness if a falsifying trajectory is not found.

In this context, robustness is informally defined as the bound on the perturbation that the system trajectory can handle such that the system trajectory satisfies the same specification. We define the robustness metric as:

$$G_\varphi(y)$$

where φ denotes the MTL specification we are trying to falsify and y is a finite duration test signal. The robustness metric essentially describes how robustly the test signal satisfies or falsifies the given specification for the system [10]. The satisfaction of a given specification takes a quantitative value rather than a Boolean value.

Because the satisfaction of a given specification is a quantitative value we can now convert the decision problem into an optimization problem given by:

Minimize $G_\varphi(y)$ for all possible system trajectories

Therefore, now we can use an optimization algorithm to search through all the possible system trajectories and, eventually, find a particular system trajectory that minimizes the robustness with respect to the system specification. The falsification problem searches for the system trajectory that yields a negative robustness value. However we use the above mentioned minimization problem as this could be used to find how robust the system is with respect to the specification, even if the system satisfies the specification for the system.

1.3 Contribution

As suggested earlier, an optimization algorithm can now be utilized to search for trajectories of minimal robustness for the MTL falsification problem. In this context I modify the Extended Ant Colony Optimization (EACO) algorithm so that it can be applied to the temporal logic falsification of hybrid systems problem. ACO is essentially a stochastic optimization technique which is inspired from the manner in which ants make and locate paths from their colonies to food sources. EACO is mainly an extension to the continuous domains of the more general discrete ACO algorithm. In order to search through the input space to find the system trajectories with minimal robustness the search space needs to be parameterized. In this context we use interpolating polynomials like cubic spline to parameterize the input space.

Cubic Spline curves are essentially piecewise polynomial functions which are very popular for their simplicity in construction and ease of evaluation. Therefore these curves have been utilized to parameterize the input space based on the range of the input signals for the MTL falsification problem. The EACO algorithm has been developed using C (Matlab executable), has been interfaced with Matlab code and has been packaged into fully functional Matlab toolbox which can be used for the temporal logic falsification of hybrid systems problem. Overall, my contribution in this thesis can be summarized as follows:

1) Modified the EACO Algorithm and applied it to the MTL falsification problem of hybrid systems.

2) Found optimal parameters for EACO by running the algorithm against couple of benchmark problems.

3) Have developed a fully functional Matlab toolbox named S-Taliro which incorporates the various optimization algorithms and solves the MTL falsification problem of hybrid systems.

4) Parameterize the input space using interpolating polynomials like cubic spline to solve the MTL falsification problem of hybrid systems.

In [9] I report my experience with ACO for Temporal Logic Falsification of Hybrid Systems.

1.4 Related Research

Testing, in general for hybrid system are difficult and time consuming because of the un-decidability in the manner in which the systems behave for extreme cases. As a result lot of research is being carried out to investigate testing approaches to the verification of hybrid systems (related research section in [10]). Two main approaches exist for the testing of hybrid systems. The first approach focusses on choosing the inputs in an orderly fashion to cover the entire state space while the second approach deals with robust simulations trajectories. However most of the research focusses on parameter estimation [11, 12]. Research in Temporal Logic Falsification of specifications describing the properties of the hybrid system is growing rapidly, though there are very few publicly available tools to solve this problem. Currently the only publicly available tool that supports the computation of robustness of temporal logic formulas is BREACH [12]. However this tool does not solve the problem of temporal logic falsification for Simulink/State flow models. MathWorks also provides System Test [13] and Simulink Design Verifier [14] along the commercial lines for this problem.

Lot of work is also being carried out in Ant Colony Optimization to extend it to the problem of falsification of hybrid systems. In [15] the authors talk about Fuzzy Ant Colony Optimization for Optimal Control using fuzzy partitioning of the state space system to parameterize the input space. In this thesis I use spline functions to parameterize the input space. In [16] authors use a different pheromone update rule in the form of noting a number of candidate solutions to update the pheromone based on Gaussian functions.

1.5 Organization of the Thesis

The reminder of this thesis is organized into 6 chapters.

Chapter 2 – Ant Colony Optimization: This chapter introduces the Ant Colony Optimization Algorithm and its application to the Travelling Salesman Problem and the MTL falsification problem.

Chapter 3 – The S-Taliro Tool: This chapter introduces the S-Taliro tool, describes the architecture of the toolbox and gives a detailed description of the interface to the tool.

Chapter 4 – Tuning of Parameters: This chapter describes the experimental analysis performed to find the optimal parameters of EACO by running the algorithm against few benchmark problems.

Chapter 5 – Results: This chapter shows a performance comparison between MC, UR and ACO by applying these algorithms to couple of bench mark problems.

Chapter 6 – Conclusion: This chapter comprises of the final summary of the research done and possible future work.

Chapter 2

ANT COLONY OPTIMIZATION

2.1 Related Background

Ant Colony Optimization is a Meta heuristic optimization algorithm inspired by the foraging behavior of ants. This algorithm was first proposed by Marco Dorigo and his colleagues [16] as a multi- agent approach for solving combinatorial optimization problems. Ant Colony Optimization is a essentially a member of the Ant Colony Algorithms family and the first algorithm was mainly aimed at finding an optimal path in a graph based on the manner in which ants find a path between their colonies and food locations. From the initial idea of finding an optimal path in a graph this algorithm has diversified to meet the requirements of many optimization problems and solve them effectively.

Common examples of combinatorial problems include scheduling, finding the minimum spanning tree, travelling salesman problem etc. These problems are in general difficult to solve and therefore heuristic methods (like ACO) are used to find solutions to these problems. Solving combinatorial problems involves finding optimal objects from a finite set of objects. This is done by operating on the input domains of the optimization problems to find the best solution. Initially this process was performed only for discrete solutions but later was applied to continuous ranges. The limitation for applying combinatorial optimization algorithms to these kind problems is that the continuous ranges have to be converted to sets of finite size which is difficult if the initial ranges are large.

A lot of work has been done to create an algorithm based on Ant Colony Optimization methods to solve continuous optimization problems. The first attempt in this direction was made in [17] which initially provided only local search capabilities. This is was at a later time extended to Continuous Ant Colony Optimization (CACO) algorithm [18] which solved the problem of local search but was not completely based on the ACO principles. In this work I utilize the concepts of the ACO methods to propose an algorithm for the MTL falsification of hybrid systems which is a continuous optimization problem.

2.2 Introduction to Ant Colony Optimization

The Ant Colony Optimization algorithm has been inspired from the manner in which ants effectively find food, lay pheromone (chemical secreted or excreted by ants) along the path to and from the food location and return to their colonies. Initially the ants move around randomly to search for food, but once they do find food at a certain location they keep laying pheromone in the path between the food locations and their colonies. Other ants have the capability to detect pheromone trails left by their counterparts and therefore are more likely to follow a path which is laid by pheromone than choose a path at random. Moreover these ants choose that particular path which is more densely laid by pheromone that any other path. If these ants do find food along this path then they return back to their colonies with the food along the same path thereby reinforcing (laying more pheromone) it so that other ants can detect and use this particular path to locate food sources.

Pheromone evaporation is extremely important for the ants and the algorithm in general as this has the advantage of avoiding convergence to a locally optimal solution or path in the case of the ants. If pheromone never evaporated, then the paths that were chosen by the first ants would have more pheromone on them and hence would be excessively attractive to the subsequent ones. This would lead to ants not exploring all the possible paths and hence result in a local optimal path between their colonies and food sources.

Briefly, the following is a simple model illustrating the behavior of ants to find food sources.

1) An ant randomly travels around the colony in search of food.

2) Once the ant finds food, it returns back to its colony depositing pheromone along the path.

3) Ants which are close to the pheromone deposited path get attracted to it.

4) They find food, and return back to their colonies along the same path thereby reinforcing the path.

5) Shorter paths to food sources from ant colonies will be the ones that are more travelled and hence will have greater deposition of pheromone.

6) This is will result in shorter paths becoming more attractive.

7) Because of pheromone evaporation the longer path loses its pheromone and eventually no ant chooses this path.

8) All the ants keep reinforcing the shorter path and follow only this path to move between their colonies and food sources.

2.3 ACO for the Travelling Salesman Problem

In this section we apply the Ant Colony Optimization algorithm to the Travelling Salesman Problem (TSP) to get a better understanding of the semantics of the algorithm. TSP has been chosen for this illustration because

1) TSP is NP hard problem.

2) TSP is considered a standard problem for performance evaluation of algorithms.

3) TSP is simple to understand and therefore the ACO's behavior to this problem can be seen clearly.

The TSP is the problem of a salesman who wants to find a shortest possible route from his home town through a given set of cities and return back to his hometown, without visiting a city twice. When ACO is applied to the problem of TSP, ants act as simple agents to make tours by moving from one city to another. These tours are guided by pheromone trails and local distance based decisions to find the shortest optimal tour.

At the beginning, each of the ants is placed on a city that is chosen randomly. At a particular city, an ant choses an unvisited city with a probability that is proportional to the pheromone deposit on that edge along with locally available heuristic information. This heuristic is a function of the distance between the connecting cities. This means that ants are more inclined to visiting cities which are closer in distance to their current city and have more pheromone deposition along the edge connecting the cities. All the ants make choices based on the above information and complete their tours.

Once all the tours are completed, the pheromones trails along each of the edges connecting neighboring cities are updated through evaporation and deposition. This pheromone update is done in a way such that shorter tours receive a higher amount of pheromone deposition so that they can be chosen with a greater probability in the next iterations. This means that shorter paths will tend to have more pheromone, which results in attracting a majority of the ants. This eventually leads to the convergence of the algorithm to an optimal shortest route, thereby providing a solution to the TSP.

2.4 ACO for the MTL Falsification Problem

In this section I modify EACO algorithm [19] to apply it to the problem of MTL falsification of hybrid systems. In the MTL falsification problem the user needs to specify the bounds on the initial conditions X_0 and input range U. Based on the number of control points that are assigned to each of the inputs, the regions which the ants have to visit are defined. These regions are essentially one dimensional set's which are bounded by the ranges of the initial conditions and the input ranges.

Each of these regions i is divided into finite sets of intervals N_i. These interval points are called stations which each ant can visit in each region. All these intervals are of the same length which can be set by the user while defining the options for the EACO algorithm. The manner in which an ant chooses to visit a particular station j in a region i depends on the probability factor which we define below. All the stations in each of the regions are initially laid with a certain amount of pheromone deposition $\boldsymbol{\tau_{ij}(c)}$ where c refers to the particular cycle during which the choice is being made by the ants. The probability with which the choice is made by each of the ants is given by:

$$\boldsymbol{p_c(j|i) = \frac{\tau_{ij}(c)}{\sum_{j=1}^{N_i} \tau_{ij}(c)}}$$

Each of the ants use this probability measure to visit the stations in each of the regions defined. For each ant k its exact location within each region at a particular cycle is denoted by $\Theta_{\boldsymbol{ij}}^{\boldsymbol{k}}(\boldsymbol{c})$. This choice is made at random based on uniform distribution. Based on the probabilities that are assigned to each of the stations, the exact location to which ant visits is decided by choosing a random number and comparing it with the probabilities assigned to the stations. We make this choice at random because this gives a greater state space exploration possibility.

Once all the ants have finished their tours in a particular cycle, the pheromone depositions at each of the stations in each region need to be updated. This must be done so that the ants converge to that particular tour that eventually falsifies the hybrid system. After the completion of each of the cycles we have sample trajectories $\{y^{(k)}\}$ for each of the ants. The robustness estimate of each of these trajectories is calculated and the ant which produced the least possible robustness value is used to update the pheromone deposits at each of the stations in each of the regions defined. In this thesis four different deposition techniques have been implemented and tested on various benchmark problems:

1) Deposition by the cycle best ant (the ant that found the best solution during the current cycle) and the global best ant (the ant that found the best solution during all the previous cycles).

2) Deposition by either the cycle best ant or the global best ant according to an option that is specified by the user which switches the deposition method after a given number of cycles (Auto Switch method)

3) Deposition by the cycle best ant (Cycle Least method)

4) Deposition by the global best ant (Global Least method)

Irrespective of the algorithm that is used pheromone values need to be updated at the end of each cycle. The pheromone deposition values in the next cycle are given by the following formula:

$$\boldsymbol{\tau_{ij}(c+1) = (1-\sigma)\tau_{ij}(c) + \sum_{k=1}^{M} \Delta\tau_{ij}^{k}(c)}$$

Where σ is the rate of evaporation and

$$\boldsymbol{\Delta\tau_{ij}^{k}(c)} = \frac{\boldsymbol{exp}(-\left\|\Theta_{ij}^{k}(\boldsymbol{c}) - \mu_{ij}\right\|^{2}/\boldsymbol{e}}{\boldsymbol{a} + [[\boldsymbol{\varphi}]](\boldsymbol{H}(\boldsymbol{y^{k}}))}$$

Where e is the parameter that is used to define the decay rate of the exponential distribution and a is a user defined constant.

Pheromone is updated in this manner at the end of each cycle and this process terminates either if a falsifying trajectory is found or if the maximum number of iterations are completed. In either case the trajectory with the least robustness along with the robustness values are returned to the user for analysis.

Chapter 3

THE S-TALIRO TOOL

3.1 About the Tool

In this thesis we present our tool S-Taliro for temporal logic falsification of hybrid systems. S-Taliro is a fully functional Matlab toolbox that essentially searches for counterexamples to MTL properties (which describe the specification for the hybrid system) through global minimization of robustness metric. The global minimization is performed using stochastic optimization techniques which are used to find a system trajectory with minimal robustness. These optimization algorithms essentially perform a random walk over the initial states and input ranges to find the minimal robustness trajectory. Currently the tool supports three optimization algorithms in the form of Uniform Random, MC and ACO.

To use this toolbox the user needs to specify the MTL specification, the initial conditions and the bounds on the input range. S-Taliro uses these inputs to generate test initial conditions and input ranges which are used to simulate the hybrid system. The simulation trace is then provided as an input to Taliro [20] which computes the robustness estimate of the MTL formula. Global minimization of this robustness value is carried out using optimization algorithms (ACO/MC/UR) to generate new test initial conditions and input ranges which are again used to generate simulation traces which are closer to falsifying the MTL formula. Traces with negative robustness values are falsifications of temporal logic properties. Traces with positive but low robustness values are also of great importance because they indicate how robust the system is against the given user specification. These system trajectories with small positive robustness values are closer in distance to falsifying traces using a mathematically well-defined notion of distance between trajectories and user specifications which are defined user temporal logic properties.

S-Taliro has been developed in a modular fashion in Matlab with respect to the optimization algorithms, and therefore other optimization algorithms can be added given a Matlab interface to their simulators. Though a basic understanding of using the MTL formulae is necessary to describe the specifications for the hybrid system, the entire design of the toolbox is extremely user friendly with extensive help documentation. It has a simple command line interface along with an in-built parser for the easy input of MTL formulae. These features make the tool easy to understand and solve the problem of temporal logic falsification of hybrid systems.

3.2 Overall Architecture

The figure below clearly depicts the overall architecture of the S-Taliro toolbox. The toolbox contains a temporal logic robustness analysis engine (Taliro) that works in tandem with a stochastic sampler. This stochastic sampler suggests an input array which comprises of the initial conditions, input ranges and any other parameters which are needed to execute the system implemented as an m-function or a Simulink/State flow model. The Simulink/State flow environment returns a simulation trace which is analyzed against the MTL formula for the hybrid system. For this purpose we have a robustness analyzer which eventually returns a robustness value for the system trace. This value is computed depending on the results of convex optimization problems which are used to compute signed distances. These signed distances are essentially distances between the trace and the MTL formula which we trying to falsify. Once we have the robustness value, this is used by the stochastic sampler to decide on the next inputs which can be used to generate another simulation trace. During this process if a falsifying trace is found i.e. a trajectory that does not satisfy the MTL formula, then it is returned to the user for examination. Moreover if the toolbox is unable to falsify the MTL formula then the trajectory with the least robustness value is returned along with the robustness value. This information is extremely valuable to know, how robust the system is with respect to the MTL user specification.

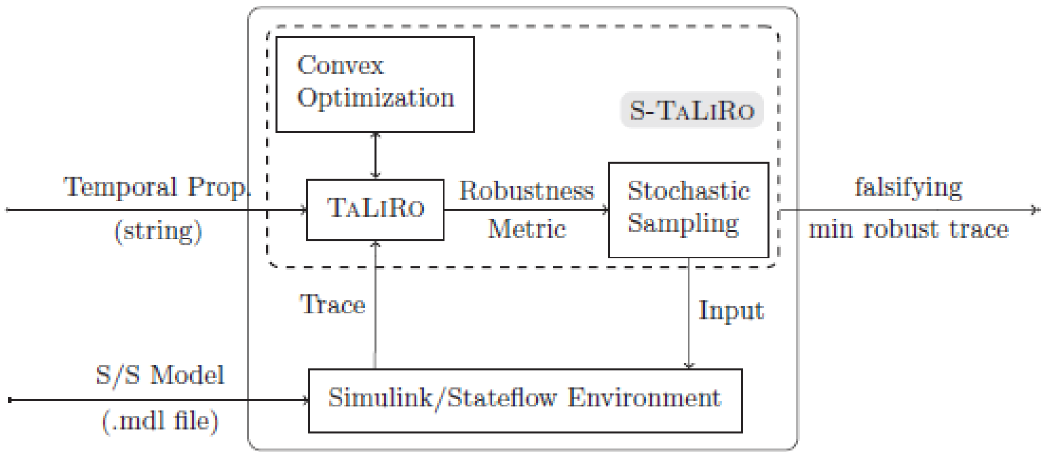


Figure 2: The Architecture of the S-Taliro Tool

3.3 S-Taliro User Guide

S-Taliro has been designed to be seamlessly integrated in the model based design process of Matlab/Simulink (TM). Currently, S-Taliro can be applied to three kinds of inputs:

1) Simulink Models

2) Matlab functions

3) Hybrid Automaton

In case of the Simulink models, the user specification can be defined either over the state space of the model or over the output space. This is an explicit option which is provided to S-Taliro to make the distinction. If the model does not have external inputs, then the search for the falsifying trajectory is performed only over the set of initial conditions. On the other hand, if the user would like to verify the MTL property over a Simulink model with external inputs, then the user has to make sure that the these inputs are defined as input ports to the Simulink model. The interface to the Simulink models is clearly explained in the description of input arguments for S-Taliro.

In case of Matlab functions, the input model must be provided as a function handle, where the function handle represents a pointer to the Matlab function. There is an options in S-Taliro called '*black_box*' which can be set/reset depending on whether the user wants his implementation of the hybrid system to be considered as a black box or not. Therefore if the model is a function pointer and the black box option is set to 0, then the function is passed to the ODE solver indicated by the option '*ode_solver*'. If the model is a function pointer and the black box options is set to 1, then it is assumed that the model will be given the time stamps, the initial conditions and input signals and it will output the time stamps, the state trajectory, the output trajectory and optionally the graph and guards depending on the option '*taliro_metric*'. On a higher level, with these settings S-Taliro will treat the input function as a black box. The options which are also inputs to the tool will be described in detail in the S-Taliro options section.

In case of hybrid automata, the input model must be an object of class '*hautomaton*' which has been defined in the S-Taliro toolbox. The user needs to understand the interface to the '*hautomaton*' class and define the initial continuous set, the dynamics in each location, the adjacency list, and guards for the transitions between locations, the unsafe set and the target set. A description of all these parameters is given in the input arguments section. Given below is a table that clearly illustrates the choices provided by S-Taliro for model inputs.

Input Model	**Input Type**	**Example**
Simulink model	String	Model =’sim_model’
Matlab function	Function handle	Model = @function_name
Hybrid automaton	Object of type hautomaton	Model = hautomaton object

Table 1: Choices provided by S-Taliro for model input

No matter what the type of the model we provide to S-Taliro, we also need to specify the set of initial conditions as well as the constraints on the input signals. In the current version of the toolbox both must be provided as hyper cubes. Note that if the user does not want to search for a falsifying trajectory over the set of initial conditions or if the hybrid system does not have any inputs, then that particular input needs to be set as an empty array. If the system does accept input signals, then in this case we need to parameterize the input function space using a finite set of points in time. For this particular reason the user needs to provide two more parameters: the type of interpolating function and the number of control points in time for each input signal. Currently S-Taliro supports all the interpolating functions provided in Matlab using the ‘*interp1*’ function and some more such as the piecewise constant and constant value functions.

Given below is the description of the interface to the tool. The user needs to provide the following parameters as inputs to the tool.

Syntax

[rob, rtime, nIter, samples] =

s_taliro (model, icond, irange, cparray, phi, pred, tt, opt);

In the following sections we describe the above parameters in detail. More information on declaring the parameters and using the S-Taliro tool can be obtained by typing '*help staliro*' at the Matlab command prompt.

3.4 Input Arguments

S-Taliro requires that the following parameters be defined before providing them as inputs to S-Taliro. The input arguments to S-Taliro are as follows:

1) *'model'* : As described earlier S-Taliro can currently handle three different kinds of inputs in the form of Simulink models, Matlab functions and Hybrid Automaton. Therefore the input argument must be either a string, function handle or an object of the '*hautomaton*' class as depicted in Table 1.

The only requirement for the Simulink input other than the ones discussed above is that the user must provide the Simulink model in the current directory. Moreover the input signals must be provided to the Simulink model through the input ports if they exist.

For the Matlab function case, as discussed earlier the user can model his system as a black box and set the appropriate option in S-Taliro. By doing so, black box computation is performed. In this case the user must provide the following interface to the Matlab function

$$[T, X, G] = \text{function}(X0, ET, TS, U)$$

Where:

X0: the initial conditions as a vector.

ET: the end time for the simulation. It is assumed that the start time is 0.

TS: the time stamps that correspond to the sampling instances for the input signals in *U*.

U: the input signals. This is an array where each column corresponds to a different signal and each row to time instance that corresponds to TS. This is optional if no input signals are required.

T: the new time stamps

X: the state/ output trajectory as an array where each column corresponds to a different state variable. If hybrid distances are used then the last column must be the trace (location trace) on the state machine of the system.

G: the graph that corresponds to the discrete transition graph of the system. This is required when hybrid distance metrics are used.

For the hybrid automata case, the user must understand the '*hautomaton*' class files to carefully define the following parameters to build an object of that particular class which will work as an input to S-Taliro. The interface for declaring this object is as follows:

object = hautomaton (*init*, *loc*, *adjList*, *guards*, *unsafe*, *target*)

The input parameters in this interface are as follows:

init: is the initial continuous set.

loc: holds the dynamics in each location which is a Matlab structure:

adjList: is the adjacency list for each location which is a Matlab cell array.

guards: is the guard set which contains the guards for the transitions between locations. This is also a Matlab structure.

unsafe: is essentially a conjunction of half spaces indicating regions which should not be reached by the system trajectory.

target: is the target set that needs to be reached by the system trajectory.

More details on using these parameters and defining the hybrid automaton can be found by typing '*help hautomaton*' at the Matlab command prompt.

2) *icond:* This input argument must be a hypercube which defines the set of initial conditions. It must be of type real. The input can be empty (indicating no initial conditions) or of dimension m by 2, where m is an integer and indicates the number of initial conditions. Each row in this input argument defines the minimum and maximum bounds on each of the initial condition.

For example:

icond = [3 6; 7 8]

Indicates two initial conditions where, the first initial condition is bounded between 3 and 6 and the second initial condition is bounded between 7 and 8.

For example:

icond = []

Indicates no initial conditions

3) *irange:* This input argument must also be defined as a hypercube defining the set of constraints on the input signals. It must be of type real. This input argument can be empty (indicating no inputs) or of dimension m by 2, where m is an integer and indicates the number of input signals to the hybrid system. Each row in this input argument defines the minimum and maximum bounds on each of the input signals to the hybrid system.

For example:

irange = [3 5; 6 8]

Indicates two input signals where, the first input signal is bounded between 3 and 5 and the second input signal is bounded between 6 and 8.

For example:

irange = []

Indicates no input signals to the hybrid system.

Defining the initial conditions and input signals as hyper cubes is not an inherent restriction of our method but rather we have observed that usually engineers do not provide constraints on the initial conditions that have variable dependencies. We plan to remove this constraint in the future versions of the toolbox.

4) *cparray:* This input argument contains the number of control points for the interpolating function associated with each input signal. This input argument must be of type real. This input can be empty (indicating no inputs to the hybrid system) or must be a 1 by n array (indicating n inputs to the hybrid system) where each of the n values refer to the number of control points associated with each input signal. Note that the number of rows in *irange* must be equal to the number of columns in *cparray* as each input signal needs to have a particular number of control points associated with it.

For example:

cparray = [10 14]

Indicates 10 control points for the first input signal and 14 control points for the second input signal

5) *phi:* This input argument must be a string in Matlab with the MTL formula that needs to be falsified. More information on defining MTL formulae can be found by typing '*help taliro*' at the Matlab command prompt.

6) *pred:* This input argument must be a Matlab structure with the atomic proposition mapping. More information on defining the predicates for an MTL formula can be found by typing '*help taliro*' at the Matlab command prompt.

7) *tt:* This input argument contains the total simulation time for the hybrid system. The input must be of type real.

8) *opt:* This input argument refers to the various options provided by S-Taliro. Table 2 indicates the default values for the options provided by the tool. The user can set the default options for the tool using the following command:

opt = staliro_options()

Given below is a description of each of the options provided by S-Taliro:

1) *optimization_solver:* This option indicates the optimization algorithm to be used by S-Taliro to solve the MTL falsification problem. This option can be set to any one of Ant Colony Optimization or Uniform Random or Monte Carlo optimization algorithms.

2) *ode_solver:* This options selects the ODE solver to be used by S-Taliro. The default option is 'ode45'. It is recommended that this option be set to 'default' for Simulink models. For Simulink models, the default option uses the default solver inside the Simulink model.

Property	Default Value
optimization_solver	'MonteCarlo'
ode_solver	'default'
interpolationtype	{'pchip'}
black_box	0
runs	100
n_tests	1000
ants_number	20
spec_space	'Y'
loc_traj	'none'
SampTime	.05
dispinfo	1
taliro_metric	'none'
map2line	1
rob_scale	100
taliro	'taliro'
sa_params	sa_parameters
hasim_params	[1 0 0 0]

Table 2: Default values to the options provided by S-Taliro

3) *interpolationtype:* This option must be defined as a Matlab structure where each one of its elements must be a string and refers to an interpolation type for each of the corresponding input signals. If the interpolation type is a structure variable containing only one element, then that particular interpolation type is applied to all

the input signals. The options for interpolating functions are the same as the options for interp1. The additional options are '*pconst*' for piecewise constant signals and '*const*' for constant signals. When the '*const*' interpolation type is used, only one control point must be provided for the corresponding input signal.

4) *black_box:* This option is set/reset when using function handles as the inputs to the tool. When this option is set S-Taliro will treat the input function as a black box. When this option is set to 0, the function is passed to the ODE solver indicated by the '*ode_solver*' option.

5) *runs:* This options sets the total number of iterations for which S-Taliro is executed against the MTL formula. It is recommended that this option be set to a large value (default is 100) as the optimization algorithms used are stochastic which would yield better overall results for a larger number of runs.

6) *n_tests:* This option indicates the maximum number of tests that the tool can make to find a falsifying trace in each run.

7) *ants_number:* This option indicates the number of ants to be used by the Ant Colony Optimization algorithm to solve the problem of MTL falsification.

8) *spec_space:* This option is set to 'X' when the MTL specification is over the trajectories of the state variables of the system. This option is set to 'Y' when the specification is over the output signals of the system.

9) *loc_traj:* This option is used to define which output signal corresponds to the location trace in case of hybrid system trajectories generated by a Simulink model. This option can be set to:

a) '*none*': the output signals do not contain a location trace. All the output signals are used for the temporal logic robustness computation.

b) '*end*': the location trace is in the last output port.

c) *integer*: If the location trace is outputted from another port number, then this option should contain the corresponding port number.

10) *SampTime:* This option indicates the sampling time to be used for the input signals for simulation.

11) *dispinfo:* This option is set when the user needs the run number to be displayed on the Matlab command prompt.

12) *taliro_metric:* This option indicates the type of the temporal logic metric to be used for the MTL falsification problem. This option can be set to the following values:

a) 'none': In this case only the continuous space is considered. Any location information on the predicates is ignored.

b) 'hybrid_inf': This metric considers the path distance between control locations and the Euclidean space distance.

c) 'hybrid': This metric considers also the distance to guards that enable a transition on the hybrid system.

13) *map2line:* This options is set to 1 for using a standard optimization algorithm with hybrid distance values. In this case we map the hybrid distances on the real line using the inverse logit function (see description for the option '*rob_scale*'). Setting this option to 0 will utilize more complicated algorithms that will attempt to directly minimize the hybrid metric.

14) *rob_scale:* For using a standard optimization algorithm with hybrid distance values, we are mapping the hybrid distances on the real line using the inverse logit function:

rob = h.dl + 2*(2*exp(h.ds/a)/(1+exp(h.ds/a))-1)

Where *a* is the scaling factor and h is the hybrid distance. If the scaling factor is not provided then, *a*=100. The scaling factor depends on the application and is important since a value of *h.ds* above 40 with *a* =1, already gives the upper bound 1 for the inverse logit function. This implies that a large range of robustness values might be mapped to the same number.

15) *taliro:* This option is set to '*taliro'* which is a tool that computes the robustness estimate of an MTL formula with respect to a finite times state sequence.

16) *sa_params:* This option is used to set the Simulated Annealing parameters with Monte Carlo Sampling.

17) *hasim_params:* This option holds a vector of values which are used by the hybrid automaton simulator. For more information on setting this option please type 'help hasimulator' at the Matlab command prompt.

To change the default values of the options to user specified values, first create the default options object by calling the '*staliro_options*' class as described above. Then use this object to change its default properties.

For example: to change the optimization solver to Ant Colony Optimization and to change the interpolation type to piecewise constant, follow the sequence of steps listed:

```
opt = s_taliro_options()
opt.optimization_solver = 'ACO'
opt.interpolationtype = {'pconst'}
```

3.5 Output Arguments

The following is a description of the output arguments that result after the MTL falsification problem.

1) *rob:* This output argument holds the minimum robustness value that was found at each run of the simulation.

2) *rtime:* This output argument holds the total running time until a falsifying trajectory is found or until the total number of stochastic tests is reached for each run of the simulation.

3) *nIter:* This output argument holds the total number of iterations until a falsifying trajectory is found or the total number of stochastic tests is reached for each run of the simulation.

4) *samples:* This output argument holds the vector containing the initial conditions and the inputs that produced a trajectory with minimum robustness for each run of the simulation.

Chapter 4

TUNING OF PARAMETERS

4.1 Optimal Parameters for EACO

In this section the parameters of the EACO are modified and tested against few benchmark problems. This is done to get a general idea on the parameters and the deposition techniques which work well for most of the benchmark problems. Along with the four deposition techniques mentioned in the previous section we have two other parameters which are varied to gauge the performance of the EACO on the benchmark problems.

1) Grid Size – denotes the number of stations/intervals in each region.

2) Evaporation Rate ($\boldsymbol{\sigma}$) - denotes the rate at which pheromone evaporates.

These parameters are varied over a certain range and for each one of those combinations the benchmark problems are executed against S-Taliro to find the trajectories that falsify a particular MTL user specification. The following sections illustrate the results of the various experiments that were executed to find the optimal parameters.

In the following tables:

1) Runs indicate the total number of independent simulations of the problem

2) #Falsifications indicate the times S-Taliro was able to falsify the formula.

3) Robustness indicates the minimum, average and maximum positive robustness.

4) Tests indicate the minimum, average and maximum number of tests.

We only list the positive robustness values as this would be useful to determine how robust the system is when compared to the specification.

Based on the experimental analysis that was performed on the benchmark problems in the following sections, it was observed that on an average the following parameters showed good results:

1) *Deposition by both the cycle best and global best ants.*

2) *Gridsize of 10*

3) *Evaporation rate of 0.5*

This result was based on calculating the average number of tests that each of the deposition technique required to falsify each of the benchmark problems. This value was the least for the case where deposition of pheromone was done by both the cycle best and global best ants.

4.2 EACO on the Delta – Sigma Modulator

The first benchmark against which the experiments are performed is the Delta – Sigma Modulator whose description can be found in [21]. The third order Delta – Sigma modulator has initital conditions in the range $[-0.1\ 0.1]^3$ and a one dimensional input signal that ranges between $[u_m\ u_M]$. For finding the optimal parameters for this model the input ranges have been chosen to be $[-0.45\ 0.45]$. The specification for this model that S-Taliro tries to falsify is that the state of the system should always remain in the set $[-1\ 1]^3$.

Deposition by the Global Best Ant has been used to generate the following tables.

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	94	.0040, .0109, .0190	20, 248.6, 1000
100	0.7	98	.0013, .0204, .0395	20, 160.4, 1000
100	0.5	97	.0027, .0150, .0383	20, 179.6, 1000
100	**0.3**	**99**	**.0341, .0341, .0341**	**20, 179.6, 1000**
100	0.1	97	1.4735e-4, .0095, .0177	20, 177.8, 1000

Table 3: Global Least Method for gridsize of 10 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	92	.0016, .0223, .0503	20, 300.8, 1000
100	0.7	92	3.9447e-4, .0215, .0530	20, 356.2, 1000
100	0.5	93	.0084, .0384, .0626	20, 324.6, 1000
100	0.3	89	.0015, .0239, .0406	20, 378.6, 1000
100	**0.1**	**94**	**.0024, .0171, .0594**	**20, 319.2, 1000**

Table 4: Global Least Method for gridsize of 32 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	91	.0035, .0269, .0575	20, 308, 1000
100	0.7	91	.0035, .0170, .0378	20, 348, 1000
100	**0.5**	**94**	**4.6015e-4, .0238, .0607**	**20, 301.8, 1000**
100	0.3	87	.0015, .0177, .0554	20, 380, 1000
100	0.1	92	.0021, .0208, .0510	20, 326.4, 1000

Table 5: Global Least Method for gridsize of 60 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	89	4.8651e-4, .0299, .0719	20, 381, 1000
100	0.7	93	1.5687e-4, .0201, .0504	20, 358.4, 1000
100	0.5	90	4.5552e-4, .0222, .0688	20, 348.4, 1000
100	**0.3**	**95**	**8.6144e-4, .0431, .0744**	**20, 348.2, 1000**
100	0.1	86	3.1873e-5, .0240, .0743	20, 399.8, 1000

Table 6: Global Least Method for gridsize of 90 on Delta-Sigma

Deposition by the Cycle Best Ant has been used to generate the following tables.

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	100	0, 0, 0	20, 156.6, 760
100	0.7	100	0, 0, 0	20, 156.8, 660
100	**0.5**	**100**	**0, 0, 0**	**20, 153.2, 460**
100	0.3	100	0, 0, 0	20, 160.6, 680
100	0.1	100	0, 0, 0	20, 183.2, 740

Table 7: Cycle Least Method for gridsize of 10 on on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	94	.0759, .0993, .1253	20, 321.6, 1000
100	**0.7**	**100**	**0, 0, 0**	**20, 274.8, 880**
100	0.5	99	.0966, .0966, .0966	20, 294.8, 1000
100	0.3	97	.0469, .0696, .1067	20, 291.4, 1000
100	0.1	98	.0886, .0929, .0973	20, 291.6, 1000

Table 8: Cycle Least Method for gridsize of 32 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	97	.0156, .0452, .0620	20, 318.8, 1000
100	0.7	96	.0099, .0870, .1331	20, 279.8, 1000
100	**0.5**	**98**	**.0573, .0632, .0690**	**20, 287.6, 1000**
100	0.3	94	.0086, .0616, .1398	20, 314, 1000
100	0.1	96	.0788, .1152, .1523	40, 275.2, 1000

Table 9: Cycle Least Method for gridsize of 60 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	91	.0150, .0843, .1462	20, 385.4, 1000
100	**0.7**	**99**	**.0579, .0579, .0579**	**20, 321.8, 1000**
100	0.5	95	.0110, .0644, .1352	20, 334.6, 1000
100	0.3	97	.0499, .0833, .1182	20, 379.4, 1000
100	0.1	98	.1132, .1225, .1319	20, 308.2, 1000

Table 10: Cycle Least Method for gridsize of 90 on Delta-Sigma

Auto-Switch between Cycle Best and Global Best Ants depositions after 25 cycles.

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	100	0, 0, 0	20, 178.8, 820
100	0.7	100	0, 0, 0	20, 157, 740
100	**0.5**	**100**	**0, 0, 0**	**20, 136, 640**
100	0.3	100	0, 0, 0	20, 175.2, 640
100	0.1	100	0, 0, 0	20, 179.2, 640

Table 11: Auto-Switch Method for gridsize of 10 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	96	.0012, .0166, .0526	20, 297.6, 1000
100	**0.7**	**100**	**0, 0, 0**	**20, 248.6, 1000**
100	0.5	98	.0020, .0082, .0145	20, 276.8, 1000
100	0.3	99	6.2061e-4, 6.2061e-4, 6.2061e-4	20, 284, 1000
100	0.1	96	.0042, .0147, .0253	20, 293.8, 1000

Table 12: Auto-Switch Method for gridsize of 32 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	97	.0026, .0196, .0379	20, 327.8, 1000
100	0.7	96	.0020, .0425, .0705	20, 284, 1000
100	0.5	97	.0029, .0118, .0182	20, 304.4, 1000
100	0.3	95	1.8723e-4, .0131, .0316	20, 328.8, 1000
100	**0.1**	**98**	**.0066, .0079, .0092**	**20, 352.2, 1000**

Table 13: Auto-Switch Method for gridsize of 60 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	96	.0022, .0246, .0507	20, 300.6, 1000
100	0.7	96	.0098, .0352, .0629	20, 309.4, 1000
100	0.5	95	.0066, .0335, .0650	20, 324.8, 1000
100	**0.3**	**97**	**.0044, .0170, .0268**	**20, 360.6, 1000**
100	0.1	94	.0012, .0102, .0251	20, 329.6, 1000

Table 14: Auto-Switch Method for gridsize of 90 on Delta-Sigma

Deposition by the Cycle Best and the Global Best Ants

Runs	σ	#Falsifications	Robustness	Tests
100	**0.9**	**100**	**0, 0, 0**	**20, 161.2, 960**
100	0.7	97	.0013, .0159, .0262	20, 204.2, 1000
100	0.5	100	0, 0, 0	20, 168.4, 700
100	0.3	99	.0182, .0182, .0182	20, 172.2, 1000
100	0.1	98	.0073, .0154, .0235	20, 192, 1000

Table 15: Cycle & Global Deposition for gridsize of 10 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	90	2.9453e-4, .0247, .0457	20, 326.8, 1000
100	0.7	94	.0049, .0152, .0248	20, 304.2, 1000
100	0.5	94	8.1405e-4, .0148, .0266	20, 310.4, 1000
100	**0.3**	**97**	**.0077, .0185, .0324**	**20, 274.8, 1000**
100	0.1	97	.0294, .0419, .0609	20, 293.8, 1000

Table 16: Cycle & Global Deposition for gridsize of 32 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	91	1.8335e-4, .0259, .0676	20, 394.4, 1000
100	0.7	88	3.3173e-4, .0299, .0669	20, 359.2, 1000
100	0.5	91	.0026, .0155, .0447	20, 325.8, 1000
100	**0.3**	**95**	**.0178, .0363, .0653**	**20, 297.6, 1000**
100	0.1	87	.0039, .0260, .0586	20, 373, 1000

Table 17: Cycle & Global Deposition for gridsize of 60 on Delta-Sigma

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	92	.0042, .0336, .0802	20, 360.8, 1000
100	0.7	93	.0179, .0336, .0553	20, 327, 1000
100	**0.5**	**98**	**.0036, .0037, .0039**	**20, 283.2, 1000**
100	0.3	92	.0010, .0171, .0519	20, 317.2, 1000
100	0.1	96	.0087, .0209, .0421	20, 303.8, 1000

Table 18: Cycle & Global Deposition for gridsize of 90 on Delta-Sigma

4.3 EACO on the Navigation Benchmark Problem

The second benchmark problem that we use to find the optimal parameters for EACO is the Navigation (NV) benchmark problem from [22]. This is a hybrid automaton benchmark with 4×4 discrete locations and 4 continuous variables x_1, x_2, x_3, x_4 . $[x_1\ x_2]^T$ and $[x_3\ x_4]^T$ refer to the position and velocity of the system respectively. The invariant set of every location is the box that constraints the position of the system. The guards in each location are the edges and the vertices that bound the location. The formulas against which S-Taliro is executed to find falsifying trajectories in the navigation benchmark problem have been described in the [24].

To find the optimal parameters for EACO we use S-Taliro to falsify the fifth formula in [24] for the NV benchmark problem.

Deposition by the Global Best Ant has been used to generate the following tables.

Runs	**σ**	#Falsifications	Robustness	Tests
100	0.9	61	1.0843e-7, 8.8661e-5, 1.9649e-4	40, 667, 1000
100	0.7	53	2.9471e-5, 8.8853e-5, 1.9733e-4	80, 700.4, 1000
100	0.5	62	1.2210e-5, 1.1245e-4, 2.0079e-4	40, 691.4, 1000
100	**0.3**	**64**	**1.1879e-7, 9.7389e-5, 1.9960e-4**	**40, 657.6, 1000**
100	0.1	58	8.0472e-6, 7.8040e-5, 1.9296e-4	20, 728.4, 1000

Table 19: Global Least Method for gridsize of 10 on Navigation benchmark

Runs	**σ**	#Falsifications	Robustness	Tests
100	0.9	47	3.0015e-6, 1.2148e-4, 2.0274e-4	60, 777.8, 1000
100	**0.7**	**48**	**3.1961e-6, 1.2192e-4, 1.9725e-4**	**40, 800.6, 1000**
100	0.5	46	1.9315e-5, 1.3334e-4, 1.9982e-4	60, 795.2, 1000
100	0.3	45	1.9069e-5, 1.2670e-4, 2.0534e-4	60, 790, 1000
100	0.1	38	2.3390e-6, 1.3442e-4, 2.0201e-4	20, 837, 1000

Table 20: Global Least Method for gridsize of 32 on Navigation benchmark

Runs	**σ**	#Falsifications	Robustness	Tests
100	0.9	41	3.4978e-5, 1.4302e-4, 2.0040e-4	20, 802, 1000
100	**0.7**	**45**	**3.2177e-5, 1.3841e-4, 2.0189e-4**	**80, 773.8, 1000**
100	0.5	33	4.2862e-7, 1.4419e-4, 2.0282e-4	60, 859, 1000
100	0.3	36	4.8758e-6, 1.3776e-4, 2.1236e-4	80, 827.4, 1000
100	0.1	39	3.4130e-5, 1.4559e-4, 2.0178e-4	100, 836.2, 1000

Table 21: Global Least Method for gridsize of 60 on Navigation benchmark

Runs	**σ**	#Falsifications	Robustness	Tests
100	0.9	31	5.6258e-7, 1.2861e-4, 2.0153e-4	40, 858, 1000
100	**0.7**	**46**	**7.4329e-6, 1.4064e-4, 2.0481e-4**	**20, 805, 1000**
100	0.5	40	1.8508e-5, 1.4216e-4, 1.9882e-4	60, 827.2, 1000
100	0.3	34	1.9961e-5, 1.4703e-4, 2.0091e-4	180, 872.4, 1000
100	0.1	39	2.5056e-6, 1.3171e-4, 2.0175e-4	160, 854.2, 1000

Table 22: Global Least Method for gridsize of 90 on Navigation benchmark

Deposition by the Cycle Best Ant has been used to generate the following tables.

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	44	3.1510e-5, 2.2777e-4, 7.7005e-4	20, 780, 1000
100	0.7	42	3.2003e-5, .0347, 2	60, 780, 1000
100	**0.5**	**45**	**3.7187e-5, 2.0166e-4, 6.2379e-4**	**40, 718.8, 1000**
100	0.3	42	3.0905e-5, .0347, 2	40, 798.6, 1000
100	0.1	43	3.3164e-5, 2.4495e-4, 6.1270e-4	40, 778.8, 1000

Table 23: Cycle Least Method for gridsize of 10 on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	**0.9**	**29**	**1.9543e-5, 3.2772e-4, 7.2911e-4**	**40, 842.8, 1000**
100	0.7	22	7.4857e-5, .0772, 2	80, 905.4, 1000
100	0.5	19	6.3969e-5, .0744, 2	20, 898.8, 1000
100	0.3	20	4.9701e-5, .0253, 2	120, 898.6, 1000
100	0.1	11	4.7541e-5, .1127, 2	60, 940.8, 1000

Table 24: Cycle Least Method for gridsize of 32 on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	12	9.2472e-5, .1140, 2	80, 941, 1000
100	0.7	16	5.0001e-5, .0241, 2	40, 926.6, 1000
100	**0.5**	**16**	**6.1007e-5, .0480, 2**	**100, 919.8, 1000**
100	0.3	10	5.9755e-5, .0448, 2	100, 972, 1000
100	0.1	14	8.3364e-5, .0469, 2	60, 934.6, 1000

Table 25: Cycle Least Method for gridsize of 60 on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	**0.9**	**13**	**8.5167e-5, .0463, 2**	**100, 919.2, 1000**
100	0.7	12	6.3932e-5, 3.5545e-4, 6.9061e-4	40, 940, 1000
100	0.5	9	4.3649e-5, .0443, 2	340, 967, 1000
100	0.3	12	6.9424e-5, .0686, 2	40, 927, 1000
100	0.1	7	5.0292e-5, 3.7626e-4, 6.9001e-4	40, 962, 1000

Table 26: Cycle Least Method for gridsize of 90 on Navigation benchmark

Auto-Switch between Cycle Best and Global Best Ants depositions after 25 cycles.

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	57	2.9978e-5, 1.1490e-4, 2.1197e-4	40, 724.2, 1000
100	0.7	53	1.0409e-5, 1.0392e-4, 2.0559e-4	60, 733.6, 1000
100	**0.5**	**58**	**2.9811e-5, 9.0305e-5, 2.0971e-4**	**40, 726.4, 1000**
100	0.3	52	2.2560e-5, 8.5693e-5, 1.9677e-4	20, 722.6, 1000
100	0.1	55	3.0438e-5, 9.7972e-5, 2.0313e-4	40, 709.8, 1000

Table 27: Auto-Switch Method for gridsize of 10 on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	33	8.8470e-6, 1.3923e-4, 2.1117e-4	20, 865.6, 1000
100	0.7	34	2.3765e-5, 1.4987e-4, 2.1416e-4	100, 886.8, 1000
100	0.5	31	2.6572e-5, 1.3519e-4, 2.1251e-4	120, 877, 1000
100	**0.3**	**35**	**1.3833e-5, 1.3969e-4, 2.1329e-4**	**100, 882.8, 1000**
100	0.1	19	3.4106e-5, 1.2139e-4, 2.1083e-4	60, 928.4, 1000

Table 28: Auto-Switch Method for gridsize of 32 on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	**0.9**	**28**	**3.7765e-5, 1.5181e-4, 2.1111e-4**	**40, 888.2, 1000**
100	0.7	28	4.9210e-7, 1.6060e-4, 2.1078e-4	60, 890.6, 1000
100	0.5	23	3.4208e-5, 1.3895e-4, 2.0727e-4	40, 917, 1000
100	0.3	18	3.7446e-6, 1.5026e-4, 2.1007e-4	20, 933.8, 1000
100	0.1	22	3.3875e-5, 1.3627e-4, 2.1041e-4	40, 925, 1000

Table 29: Auto-Switch Method for gridsize of 60 on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	**0.9**	**26**	**2.4225e-5, 1.5131e-4, 2.1444e-4**	**100, 898.4, 1000**
100	0.7	20	3.5574e-5, 1.5484e-4, 2.1325e-4	120, 912.6, 1000
100	0.5	26	2.3164e-5, 1.4312e-4, 2.1184e-4	160, 932, 1000
100	0.3	19	3.1702e-5, 1.5347e-4, 2.1357e-4	100, 955, 1000
100	0.1	16	7.6385e-6, 1.3620e-4, 2.1160e-4	100, 936, 1000

Table 30: Auto-Switch Method for gridsize of 90 on Navigation benchmark

Deposition by the Cycle Best and the Global Best Ants

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	61	7.6669e-6, 9.0180e-5, 2.0307e-4	20, 689.8, 1000
100	0.7	61	2.9697e-5, 1.0288e-4, 2.0469e-4	40, 653.8, 1000
100	**0.5**	**64**	**1.9084e-5, 8.6082e-5, 2.0035e-4**	**20, 642.4, 1000**
100	0.3	59	8.3212e-6, 8.8875e-5, 1.9925e-4	40, 666.2, 1000
100	0.1	55	8.2749e-7, 6.5438e-5, 1.9271e-4	20, 713, 1000

Table 31: Cycle & Global Deposition (gridsize 10) on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	43	3.1174e-6, 1.2027e-4, 2.0650e-4	20, 807, 1000
100	**0.7**	**47**	**4.8341e-6, 1.3059e-4, 2.1350e-4**	**140, 782.6, 1000**
100	0.5	39	3.1822e-6, 1.0807e-4, 2.0957e-4	60, 788.6, 1000
100	0.3	45	4.0260e-6, 1.2392e-4, 2.0250e-4	40, 803.8, 1000
100	0.1	31	3.5347e-5, 1.2037e-4, 2.0558e-4	140, 877.6, 1000

Table 32: Cycle & Global Deposition (gridsize 32) on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	39	2.6717e-5, 1.3852e-4, 2.0092e-4	160, 850.6, 1000
100	**0.7**	**46**	**2.6741e-5, 1.4144e-4, 2.0185e-4**	**140, 829.4, 1000**
100	0.5	39	3.3047e-5, 1.2794e-4, 2.0470e-4	80, 844.2, 1000
100	0.3	38	6.1733e-6, 1.3827e-4, 2.0510e-4	60, 839.2, 1000
100	0.1	34	7.0868e-6, 1.2344e-4, 2.0385e-4	140, 873, 1000

Table 33: Cycle & Global Deposition (gridsize 60) on Navigation benchmark

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	31	2.2009e-6, 1.2757e-4, 2.0737e-4	160, 870.6, 1000
100	**0.7**	**38**	**7.5714e-7, 1.4011e-4, 2.0051e-4**	**100, 863, 1000**
100	0.5	37	8.2471e-6, 1.3061e-4, 2.0638e-4	80, 830.4, 1000
100	0.3	35	2.0519e-5, 1.2926e-4, 2.0147e-4	80, 871.4, 1000
100	0.1	26	3.0209e-6, 1.3233e-4, 1.9908e-4	60, 896, 1000

Table 34: Cycle & Global Deposition (gridsize 90) on Navigation benchmark

4.4 EACO on the Automatic Transmission Model

The third benchmark problem that we use to find the optimal parameters for EACO is the Automatic Transmission Model(AT). This is a model of an automatic transmission controller which has one input and two outputs. The only input to the system is the throttle schedule, while the break schedule is set to 0 for the entire simulation duration. The time of simulation is 30s. The outputs to the system are the engine speed and the vehicle speed. A description of the model and the formulas against which S-Taliro is executed to find falsifying trajectories in the automatic transmission model have been described in [24].

To find the optimal parameters for EACO we use S-Taliro to falsify the fourth formula in [24] for the AT Model.

Deposition by the Global Best Ant has been used to generate the following tables.

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	41	.0023, 1.63, 8.24	60, 770.2, 1000
100	**0.7**	**49**	**.03, 1.42, 12.34**	**60, 720.4, 1000**
100	0.5	39	.0015, .88, 8.06	40, 761, 1000
100	0.3	39	.0044, 1.48, 7.28	40, 785.4, 1000
100	0.1	33	.013, 1.01, 9.88	60, 802.4, 1000

Table 35: Global Least Method for gridsize of 10 on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	0	.07, 4.3, 33.15	1000, 1000, 1000
100	**0.7**	**0**	**.08, 3.47, 19.85**	**1000, 1000, 1000**
100	0.5	0	.07, 3.92, 19.71	1000, 1000, 1000
100	0.3	0	.08, 3.75, 18.48	1000, 1000, 1000
100	0.1	0	.07, 3.72, 20.07	1000, 1000, 1000

Table 36: Global Least Method for gridsize of 32 on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	0	.22, 7.05, 26.84	1000, 1000, 1000
100	**0.7**	**0**	**.04, 6.69, 20.24**	**1000, 1000, 1000**
100	0.5	0	.18, 7.49, 29.31	1000, 1000, 1000
100	0.3	0	.17, 7.27, 25.81	1000, 1000, 1000
100	0.1	0	.26, 7.16, 29.08	1000, 1000, 1000

Table 37: Global Least Method for gridsize of 60 on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	**0.9**	**0**	**.16, 5.36, 21.97**	**1000, 1000, 1000**
100	0.7	0	.31, 7.28, 35.87	1000, 1000, 1000
100	0.5	0	.21, 6.52, 19.72	1000, 1000, 1000
100	0.3	0	.36, 8.08, 31.12	1000, 1000, 1000
100	0.1	0	.21, 7.5, 36.94	1000, 1000, 1000

Table 38: Global Least Method for gridsize of 90 on Automatic Transmission

Deposition by the Cycle Best Ant has been used to generate the following tables.

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	23	.13, 4.66, 40.72	80, 815.4, 1000
100	**0.7**	**29**	**.11, 4.16, 38.41**	**40, 758.2, 1000**
100	0.5	19	.098, 5.62, 57.84	40, 843, 1000
100	0.3	25	.12, 3.37, 21.5	20, 796, 1000
100	0.1	26	.14, 3.27, 27.64	80, 797.6, 1000

Table 39: Cycle Least Method for gridsize of 10 on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	0	.18, 9.61, 375.56	1000, 1000, 1000
100	0.7	0	.14, 6.22, 71.4	1000, 1000, 1000
100	**0.5**	**0**	**.10, 3.9, 23.73**	**1000, 1000, 1000**
100	0.3	0	.11, 6.14, 122.58	1000, 1000, 1000
100	0.1	0	.12, 7.48, 195.09	1000, 1000, 1000

Table 40: Cycle Least Method for gridsize of 32 on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	0	.43, 7.87, 40.44	1000, 1000, 1000
100	0.7	0	.43, 7.08, 33.67	1000, 1000, 1000
100	0.5	0	.42, 14.29, 542.26	1000, 1000, 1000
100	**0.3**	**0**	**.14, 7.04, 29.31**	**1000, 1000, 1000**
100	0.1	0	.28, 7.87, 22.4	1000, 1000, 1000

Table 41: Cycle Least Method for gridsize of 60 on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	**0.9**	**0**	**.18, 5.84, 20.36**	**1000, 1000, 1000**
100	0.7	0	.25, 7.89, 33.22	1000, 1000, 1000
100	0.5	0	.24, 8.07, 31.12	1000, 1000, 1000
100	0.3	0	.42, 8.04, 28.32	1000, 1000, 1000
100	0.1	0	.33, 8.73, 24.84	1000, 1000, 1000

Table 42: Cycle Least Method for gridsize of 90 on Automatic Transmission

Auto-Switch between Cycle Best and Global Best Ants depositions after 25 cycles.

Runs	**σ**	#Falsifications	Robustness	Tests
100	0.9	21	.0072, 6.89, 161.83	60, 823.2, 1000
100	0.7	27	.089, 2.11, 18.75	40, 796.8, 1000
100	0.5	27	.039, 3.34, 20.51	40, 776.6, 1000
100	0.3	26	.069, 3.73, 23.82	60, 799.6, 1000
100	**0.1**	**30**	**.094, 3.34, 22.86**	**60, 762, 1000**

Table 43: Auto-Switch Method for a grid size of 10 on Automatic Transmission

Runs	**σ**	#Falsifications	Robustness	Tests
100	0.9	0	.166, 5.24, 28.13	1000, 1000, 1000
100	**0.7**	**0**	**.079, 4.84, 100.61**	**1000, 1000, 1000**
100	0.5	0	.12, 5.15, 23.85	1000, 1000, 1000
100	0.3	0	.087, 5.86, 50.42	1000, 1000, 1000
100	0.1	0	.14, 4.87, 22.4	1000, 1000, 1000

Table 44: Auto-Switch Method for a grid size of 32 on Automatic Transmission

Runs	**σ**	#Falsifications	Robustness	Tests
100	0.9	0	.22, 7.61, 24.56	1000, 1000, 1000
100	0.7	0	.33, 7.34, 23.19	1000, 1000, 1000
100	**0.5**	**0**	**.31, 7.23, 24.95**	**1000, 1000, 1000**
100	0.3	0	.36, 7.74, 23.64	1000, 1000, 1000
100	0.1	0	.33, 9.08, 36.74	1000, 1000, 1000

Table 45: Auto-Switch Method for a grid size of 60 on Automatic Transmission

Runs	**σ**	#Falsifications	Robustness	Tests
100	**0.9**	**0**	**.27, 6.18, 29.73**	**1000, 1000, 1000**
100	0.7	0	.134, 6.76, 29.34	1000, 1000, 1000
100	0.5	0	.13, 7.28, 29.94	1000, 1000, 1000
100	0.3	0	.16, 6.5, 21.71	1000, 1000, 1000
100	0.1	0	.37, 7.85, 24.66	1000, 1000, 1000

Table 46: Auto-Switch Method for a grid size of 90 on Automatic Transmission

Deposition by the Cycle Best and the Global Best Ants

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	21	.02, 1.65, 10.47	60, 818, 1000
100	0.7	21	.069, 1.38, 11.48	40, 827.6, 1000
100	0.5	17	.0023, 1.57, 11.05	40, 859.8, 1000
100	0.3	15	.0051, 1.64, 11.08	120, 878.2, 1000
100	**0.1**	**32**	**.039, 1.51, 9.91**	**60, 734, 1000**

Table 47: Cycle & Global Deposition (grid size 10) on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	**0.9**	**0**	**.07, 3.69, 25.38**	**1000, 1000, 1000**
100	0.7	0	.07, 4.89, 19.71	1000, 1000, 1000
100	0.5	0	.138, 4.65, 22.79	1000, 1000, 1000
100	0.3	0	.072, 4.38, 24.42	1000, 1000, 1000
100	0.1	0	.088, 3.97, 21.96	1000, 1000, 1000

Table 48: Cycle & Global Deposition (grid size 32) on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	0	.21, 6.98, 33.78	1000, 1000, 1000
100	**0.7**	**0**	**.207, 6.28, 25.18**	**1000, 1000, 1000**
100	0.5	0	.16, 8.45, 28.07	1000, 1000, 1000
100	0.3	0	.25, 8.01, 27.63	1000, 1000, 1000
100	0.1	0	.277, 9.66, 25.87	1000, 1000, 1000

Table 49: Cycle & Global Deposition (grid size 60) on Automatic Transmission

Runs	σ	#Falsifications	Robustness	Tests
100	0.9	0	.18, 5.59, 23.49	1000, 1000, 1000
100	**0.7**	**0**	**.14, 5.58, 23.51**	**1000, 1000, 1000**
100	0.5	0	.15, 6.36, 21.39	1000, 1000, 1000
100	0.3	0	.13, 6.61, 22.7	1000, 1000, 1000
100	0.1	0	.31, 7.9, 26.36	1000, 1000, 1000

Table 50: Cycle & Global Deposition (grid size 90) on Automatic Transmission

Chapter 5

RESULTS

5.1 Comparison between Optimization Algorithms

In this section we make a comparison between the optimization algorithms provided by S-Taliro against the benchmark problems discussed in the previous section. The optimization algorithms provided by the tool are:

1) Uniform Random

2) Monte Carlo

3) ACO

For the Delta-Sigma modulator benchmark problem, the initial conditions are in the range $[-0.1, 0.1]^3$ and the following input ranges are considered:

1) $[-0.45, 0.45]$

2) $[-0.4, 0.4]$

3) $[-0.35, 0.35]$

The specification is that the state of the system must remain in the set $[-1, 1]$. Each of the three algorithms are run against this specification for each of the input ranges and a comparision is made on the performance of the algorithms.

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	82	.0019, .0087, .0186	5, 465.64, 1000	.1967, 17.7552, 38.3188
MC	78	.0032, .0394, .0756	10, 549.37, 1000	.3999, 21.3301, 38.9293
ACO	**96**	**3.2863e-4, .0086, .0200**	**20, 206.2, 1000**	**.7668, 7.9251, 38.4502**

Table 51: Comparison on Delta Sigma Modulator with input range [-.45, 45]

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	37	2.2727e-4, .0259, .0595	10, 811.91, 1000	.3923, 31.3805, 39.1409
MC	63	.0029, .0588, .0979	4, 650.33, 1000	.1634, 25.4940, 39.2649
ACO	**98**	**.0020, .0033, .0047**	**20, 252.4, 1000**	**.7630, 9.6045, 38.0263**

Table 52: Comparison on Delta Sigma Modulator with input range [-.4, 4]

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	0	.0141, .0656, .1199	1000, 1000, 1000	39.9770, 40.3558, 45.5991
MC	32	7.8402e-5, .08, .1482	165, 873.67, 1000	6.6135, 35.1957, 41.750
ACO	**82**	**.0030, .0225, .0809**	**20, 493, 1000**	**.7825, 19.3393, 39.3373**

Table 53: Comparison on Delta Sigma Modulator with input range [-.35, 35]

For the navigation benchmark problem similar comparison between the optimization algorithms is performed for each of the 5 formulas described in [24]. The following tables illustrate the comparison.

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	27	9.987e-7, 8.979e-5, 1.81e-4	54, 859.1, 1000	49.78, 787.5295, 926.61
MC	43	6.025e-7, 9.395e-5, 1.82e-4	5, 869.87, 1000	5.66, 803.1635, 975.97
ACO	**66**	**2.01e-6, 8.494e-5, 1.76e-4**	**20, 557, 1000**	**18.38, 544.388, 1.07e3**

Table 54: Comparison on Navigation benchmark for the first formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	100	0, 0, 0	2,53.58, 234	1.102, 26.914, 114.3076
MC	100	0, 0, 0	2, 47.36, 197	1.244, 26.488, 126.6534
ACO	**100**	**0, 0, 0**	**20, 35.6, 100**	**8.784, 19.675, 55.7264**

Table 55: Comparison on Navigation benchmark for the second formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	100	0, 0, 0	2,12.52, 162	.929, 6.342, 79.6583
MC	100	0, 0, 0	2, 24.66, 117	1.05, 12.1023, 53.2432
ACO	**100**	**0, 0, 0**	**20, 21.2, 60**	**8.6791, 10.8967, 26.951**

Table 56: Comparison on Navigation benchmark for the third formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	100	0, 0, 0	2, 67.07, 303	1.196, 35.4733, 159.843
MC	100	0, 0, 0	2, 56.49, 195	1.4964, 32.25, 102.263
ACO	**100**	**0, 0, 0**	**20, 45.2, 160**	**9.0459, 24.927, 87.384**

Table 57: Comparison on Navigation benchmark for the fourth formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	9	3.24e-5, 9.135e-5, 1.885e-4	23, 964.7, 1000	11.937, 490.043, 521.04
MC	46	3.701e-5, 1.225e-4, 1.92e-4	43, 767, 1000	32.616, 502.292, 688.31
ACO	**61**	**1.411e-5, 8.3e-5, 2.01e-4**	**20, 665, 1000**	**11.08, 383.302, 675.49**

Table 58: Comparison on Navigation benchmark for the fifth formula

For the Automatic Transmission Model similar comparison between the optimization algorithms is performed for the formulas described in [24]. The following table illustrates the comparison.

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	100	0, 0, 0	2, 32.8, 287	.2, 3.38, 29.5823
MC	98	5.1361, 11.68, 18.23	2, 131.81, 1000	.2, 13.59, 103.22
ACO	99	.0774, .0774, .0774	20, 41, 1000	2.0409, 4.2, 102.35

Table 59: Comparison on Automatic Transmission for the first formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	100	0, 0, 0	4, 81.57, 373	.48, 9.85, 44.84
MC	86	4.47, 8.84, 18.22	3, 271, 1000	.36, 32.89, 121.59
ACO	75	.137, 4.03, 12.02	20, 291.6, 1000	2.39, 35.25, 121.29

Table 60: Comparison on Automatic Transmission for the second formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	0	.114, .87, 3.11	1000, 1000, 1000	120.84, 121.14, 121.44
MC	82	.0043, 6.64, 23.23	75, 412.38, 1000	9.13, 50.27, 122.02
ACO	30	.015, 2.59, 13.82	20, 757.6, 1000	2.41, 91.87, 121.83

Table 61: Comparison on Automatic Transmission for the third formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	0	.207, .625, 2.89	1000, 1000, 1000	123.84, 124.31, 130.89
MC	0	.155, 1.56, 18.23	1000, 1000, 1000	124.58, 125.05, 125.52
ACO	19	.053, 1.57, 11.83	20, 846.2, 1000	2.47, 105.25, 124.75

Table 62: Comparison on Automatic Transmission for the fourth formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	0	5.89, 6.05, 6.37	1000, 1000, 1000	117.66, 118.08, 118.68
MC	0	5.86, 6.67, 18.22	1000, 1000, 1000	118.18, 118.42, 118.64
ACO	0	5.64, 6.17, 11.81	1000, 1000, 1000	117.84, 118.82, 119.08

Table 63: Comparison on Automatic Transmission for the fifth formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	100	0, 0, 0	4, 147.73, 510	.42, 15.47, 53.40
MC	100	0, 0, 0	2, 125.05, 406	.22, 13.17, 42.73
ACO	84	5.24e-4, .0079, .037	20, 230, 1000	2.09, 24.11, 105.01

Table 64: Comparison on Automatic Transmission for the sixth formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	97	1.11e-4, 2.96e-4, 6.18e-4	3, 379.03, 1000	.31, 39.7, 104.7
MC	100	0, 0, 0	3, 153.73, 476	.32, 16.18, 50.06
ACO	84	1.55e-4, .0083, .0278	20, 268.4, 1000	2.09, 28.14, 105.18

Table 65: Comparison on Automatic Transmission for the seventh formula

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	0	1.97e-5, .001, .0029	1000, 1000, 1000	104.52, 104.64, 104.82
MC	0	1.89e-7, 3.5e-4, 7.55e-4	1000, 1000, 1000	104.97, 105.18, 106.41
ACO	0	3.88e-7, .0018, .03	1000, 1000, 1000	104.53, 104.63, 104.95

Table 66: Comparison on Automatic Transmission for the eighth formula

5.2 Observations

In the case of the Delta-Sigma modulator benchmark problem, the performance of the algorithms vary depending on the ease/difficulty with which the specification is falsified. It is evident from the tablular results that ACO comes up with a greater number of falsifications in each of the three instances of the Delta-Sigma modulator. Moreover ACO comes up with lower average robustness values at lower average runtimes which indicate greater performance at a lower computational cost.

In the case of the Navigation benchmark problem, it is observed that the performance of all the algorithms is comparable in terms of the number of falsifications and computation time for the second, third and fourth formulas. On harder problem instances, however it is clearly eveident that ACO performs better than MC and UR in terms of both the computation time and the number of falsifications. Moreover the average number of tests needed to falsify also are lesser when ACO is used on the Navigation Benchmark problem.

In the case of the Automatic Transmission Model, it is observed that the performance of the algorithms vary depending on the ease/difficulty with which the specification is falsified.

Algo.	#Fals	Robustness	Tests	Time(sec)
UR	0	.207, .625, 2.89	1000, 1000, 1000	123.84, 124.31, 130.89
MC	0	.155, 1.56, 18.23	1000, 1000, 1000	124.58, 125.05, 125.52
ACO	47	.0086, 1.05, 11.96	80, 742.6, 1000	9.97, 93.26, 126.08

Table 67: Global Least Method (fourth formula) on Automatic Transmission

On easier problems performance of MC, UR and ACO were comparible, but on tougher problems (fourth formula) ACO was able to falsify 19 times where MC and UR could not find a falsifying trajectory. We then used the global least deposition technique for the fourth formula and the performace improved as the results indicate in Table 67. We remark that this may be a special case since ACO was optimized on this formula. This result seems to suggest that the problem may not have many minima and hence using the global best ant converges faster. Moreover the structure of the formula also seems to play a role in the rate at which an algorithm converges.

Following is a summary of the comparison made between MC and ACO for the various problem instances that have been discussed in the previous sections. All the formulas have been described in [24]. Table 68 has been generated by setting the *taliro_metric* = 'none' whereas Table 69 has been generated by setting the *taliro_metric* = 'hybrid_inf'.

Problem	Formula	Number of Falsifications	
		MC	ACO
AT	Φ_1^{AT}	98	99
AT	Φ_2^{AT}	86	75
AT	Φ_3^{AT}	82	30
AT	Φ_4^{AT}	0	19
AT	Φ_5^{AT}	0	0
$P_{[-0.45,0.45]}^{\boldsymbol{\Delta-\Sigma}}$	$\Phi_{\boldsymbol{\Delta-\Sigma}}$	78	96
$P_{[-0.4,0.4]}^{\boldsymbol{\Delta-\Sigma}}$	$\Phi_{\boldsymbol{\Delta-\Sigma}}$	63	98
$P_{[-0.35,0.35]}^{\boldsymbol{\Delta-\Sigma}}$	$\Phi_{\boldsymbol{\Delta-\Sigma}}$	32	82

Table 68: Comparison between MC & ACO with 'Euclidean metric'

Problem	Formula	Number of Falsifications	
		MC –H0	ACO
AT	Φ_6^{AT}	100	84
AT	Φ_7^{AT}	100	84
AT	Φ_8^{AT}	0	0
NV	Φ_1^{NV}	43	66
NV	Φ_2^{NV}	100	100
NV	Φ_3^{NV}	100	100
NV	Φ_4^{NV}	100	100
NV	Φ_5^{NV}	46	61

Table 69: Comparison between MC & ACO with 'hybrid distance metric'

Chapter 6

CONCLUSION

6.1 Conclusions

In this thesis we have applied the Extended Ant Colony Optimization (EACO) Algorithm to the problem of MTL falsification of hybrid systems. The algorithm was initially applied to the Delta-Sigma Modulator problem and the Navigation Benchmark problem to find the optimal parameters in EACO which best suit both the benchmark problems. The optimal parameters which worked well for the Delta-Sigma Modulator problem were different from the optimal parameters found for the Navigation Benchmark Problem. However, from the results it was evident that the algorithm, where the cycle best ant and the global best ant were used to update the pheromone, showed better results in the form of minimum number of tests needed to falsify the MTL specification for both the problems on an average. The performance was observed to be better with EACO parameters, grid size set to 10 and evaporation rate set to 0.5. These optimal parameters were set in the EACO algorithm and a comparison was made with Monte Carlo and Uniform Random Algorithms on various problem instances. The results of EACO on the problems that were easy to falsify were comparable with Monte Carlo and Uniform Random Algorithm. However on tougher problems the results indicate that EACO performed better that Monte Carlo and Uniform Random Algorithms in terms of the number of falsifications and the computational time.

The results obtained from the experimental analysis are very promising and we hope that the toolbox with all the inbuilt optimization algorithms will be very useful for test engineers of hybrid systems.. S-Taliro has been developed to handle the falsification problem of hybrid systems which are implemented as Simulink/State flow models or Matlab functions. The toolbox can be used not only to falsify MTL specifications of arbitrary systems but also to find how robust a hybrid system is with respect to a given specification. This can be especially useful in systems that have been proven to be correct.

6.2 Future work

In the experiments that were performed, four deposition techniques were used along with varying grid sizes and evaporation rates to find the optimal parameters for the Extended Ant Colony Optimization Algorithm which could be used on all the benchmark problems in general to find near optimal solutions. The rate at which this algorithm converges to a good solution varies based on the parameters being used and therefore additional tuning of the pheromone update rule could be done to better the rate at which the algorithm converges to a good solution. As future work we could look at:

1) Incorporating lower and upper bounds on the pheromone deposition.

2) Regular re-initialization of pheromone deposits on the regions.

3) Implement parallel ant colonies and gauge the performance of the algorithm on various benchmarks.

Moreover some of the results indicated that there might be some structure in the systems being considered or the formulas that makes one algorithm converge faster and perform better than others. This is a very interesting prospect for future research.

REFERENCES

[1] Yashwanth Annapureddy, Che Liu, Georgios Fainekos, Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and algorithms for the construction and analysis of systems,* volume 6605 of LNCS, pages 254-257. Springer, 2011.

[2] A. A. Julius, A. Halasz, V. Kumar, and G. J. Pappas, "Controlling biological systems: The lactose regulation system of escherichia coli," in *American Control Conference*, New York, July 11-13 2007.

[3] A. Bhatia and E. Frazzoli, "Incremental search methods for reachability analysis of continuous and hybrid systems," in *Hybrid Systems: Computation and Control*, ser. LNCS, vol. 2993. Springer, 2004, pp. 142–156.

[4] M. Branicky, M. Curtiss, J. Levine, and S. Morgan, "Sampling-based planning, control and verification of hybrid systems," *IEE Proc.- Control Theory Appl.*, vol. 153, no. 5, pp. 575–590, 2006.

[5] T. Nahhal and T. Dang, "Test coverage for continuous and hybrid systems," in *CAV*, ser. LNCS, vol. 4590. Springer, 2007, pp. 449–462.

[6] T. Dang, A. Donze, O. Maler, and N. Shalev, "Sensitive state-space exploration," in *Proc. of the 47th IEEE Conference on Decision and Control*, Dec. 2008, pp. 4049–4054.

[7] R. Koymans, "Specifying real-time properties with metric temporal logic." *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.

[8] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[9] Y.S.R. Annapureddy and G.Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *Proceedings of the 36th Annual Conference of* IEEE *Industrial Electronics,* 2010.

[10] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivan_ci_c, A. Gupta, and G. Pappas. Monte-Carlo techniques for the falsification of temporal properties of non-linear systems. In *Hybrid Systems*: *Computation and Control*, pages 211-220. ACM Press, 2010.

[11] A. Rizk, G. Batt, F. Fages, and S. Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *6th International Conference on Computational Methods in Systems Biology*, volume 5307 of LNCS, pages 251-268. Springer, 2008.

[12] A. Donze. BREACH, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, volume 6174 of LNCS, pages 167-170. Springer, 2010.

[13] http://www.mathworks.com/products/systemtest/

[14] http://www.mathworks.com/products/sldesignverifier/

[15] J. Van Ast, R. Babuˇska, and B. De Schutter, "Fuzzy ant colony optimization for optimal control," in *Proceedings of the 2009 conference on American Control Conference*. IEEE Press, 2009, pp. 1003-1008.

[16] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, pp. 1155–1173, 2008.

[17] Bilchev, G., and Parmee, I. C. (1995). The ant colony metaphor for searching continuous design spaces, In: T. Fogarty, ed. *Lecture Notes in Computer Science*, **993**, Springer Verlag, 25-39.

[18] Wodrich, M. and Bilchev, G. (1997). Cooperative distributed search: the ants' way, *Control and Cybernetic*s, Vol. **26** No. **3**, 413–445.

[19] G. E. Fainekos and K. C. Giannakoglou, "Inverse design of airfoils based on a novel formulation of the ant colony optimization method," *Inverse Problems in Engineering*, vol. 11, no. 1, pp. 21–38, 2003.

[20] G. E. Fainekos and G. J. Pappas. A user guide for Taliro. Technical report, Dept. of CIS, Univ. of Pennsylvania, 2008.

[21] T. Dang, A. Donz´e, and O. Maler, "Verification of analog and mixed-signal circuits using hybrid system techniques," in *5th International Conference on Formal Methods in Computer-Aided Design*, ser. LNCS, vol. 3312. Springer, 2004, pp. 21–36.

[22] A. Fehnker, F. Ivancic. Benchmarks for hybrid systems verification. In *HSCC*. LNCS Series, vol. 2993. Springer, 2004, pp. 326–341.

[23] ZHAO, Q., KROGH, B. H., AND HUBBARD, P. 2003. Generating test inputs for embedded control systems. *IEEE Control Systems Magazine Aug*., 49–57.

[24] Probabilistic Temporal Logic Falsification of Cyber-Physical Systems (unpublished)