dtcontrol

Release 1.0.0rc5

Mathias Jackermeier, Pranav Ashok, Maximilian Weininger

Mar 04, 2020

TABLE OF CONTENTS

CHAPTER

ONE

INTRODUCTION

dtControl is a tool for compressing memoryless controllers arising out of automatic controller synthesis of cyber-physical systems (CPS). dtControl takes as input a controller synthesised by various formal verification tools and represents them in the form of decision trees. In the process, the size of the controller is reduced greatly, and at the same time, it becomes more explainable. While in principle, memoryless strategies in any format can be handled by dtControl, currently it supports controllers output by two tools: SCOTS (link) Uppaal Stratego (link)

Moreover, it also supports a CSV-based format which allows the user to quickly experiment with the techniques provided by dtControl.

We provide a *User Manual*, which gives information necessary to use dtControl and run the various decision tree learning algorithms implemented in it, as described in the paper "dtControl: Decision Tree Learning Algorithms for Controller Representation" appearing at the 23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2020).

An additional *Developer Manual* is made available for those who are interested in interfacing their own controller synthesis tools with dtControl and/or those interested in implementing their own strategy representation algorithms.

1.1 User Manual

This document equips the user with the information necessary to use dtControl and run the various decision tree learning algorithms implemented in it.

1.1.1 Capabilities

dtControl achieves best results on non-deterministic or permissive controllers. The following algorithms are supported for such controllers:

Non-determinism preserving

1. CART
2. CART with predicates from Linear SVM (LinSVM)
3. CART with predicates from Logistic Regression (LogReg)
4. CART with predicates from OC1 (OC1)

Determinizing

1. CART + maximal frequencies (MaxFreq)
2. LinSVM/LogReg + maximal frequencies (MaxFreqLC)
3. CART + minimal norm (MinNorm)

4. LinSMV/LogReg + minimal norm (MinNormLC)

dtControl can also be used with deterministic controllers, however empirical results with such controllers are not as significant as with nondeterministic controllers.

1.1.2 Quick Start with Command-line Interface

Once the dtControl package is installed, the command line interface can be accessed from your favourite shell using the dtcontrol command. Get started by running:

```
$ dtcontrol -h
```

If your installation has run successfully, you will now see the help page detailing the usage and arguments.:

```
usage: dtcontrol [-h] [-v] [--input INPUT [INPUT ...]] [--output OUTPUT]
                 [--method METHOD [METHOD ...]]
                 [--determinize DETSTRATEGY [DETSTRATEGY ...]]
                 [--timeout TIMEOUT] [--no-multiprocessing] [--artifact]
                 [--benchmark-file FILENAME] [--rerun]

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  --input INPUT [INPUT ...], -i INPUT [INPUT ...]
                        The input switch takes in one or more space separated
                        file names or a folder name which contains valid
                        controllers (.scs, .dump or .csv)
  --output OUTPUT, -o OUTPUT
.
.
.
.
```

First Run

The examples folder available in the RE package contains all the SCOTS (*.scs sparse-matrix format) and UPPAAL (*.dump format) examples against which dtControl has been benchmarked, as shown in Table 1 of the paper.

To run the CART algorithm on, say, the *cartpole* example (`cartpole.scs`), use the following command:

```
$ dtcontrol --input examples/cartpole.scs --method cart
```

This will produce some new files and folders in the current folder:

```
decision_trees
|-- CART
|   `-- cartpole
|       |-- CART.c
|       |-- CART.dot
benchmark.json
benchmark.html
```

where

- `CART.c` contains the C-code of the decision tree

- `CART.dot` contains the DOT source code which can be compiled, for example, using the dot -Tpdf CART.dot -o CART.pdf command
- `benchmark.json` contains a JSON file containing some statistics (tree size, bandwidth, construction time and other metadata)
- `benchmark.html` summarizes the experiments whose results are stored in benchmark.json

Determinization

Now let us see another example where our best determinizing technique, MaxFreq, is used along with CART.:

```
$ dtcontrol -i examples/cartpole.scs -m cart -d maxfreq
```

(Note the use of the `-i`, `-m` and `-d` switches instead of their full versions: `--input`, `--method` and `--determinize`.)

Output

After dtControl finishes execution, the directory structure should look like this::

```
decision_trees
|-- CART
|   `-- cartpole
|       |-- CART.c
|       |-- CART.dot
|-- MaxFreqDT
|   `-- cartpole
|       |-- MaxFreqDT.c
|       |-- MaxFreqDT.dot
benchmark.json
benchmark.html
```

The `benchmark.html` file is updated after every run. View a tabular summary of all the results stored in `benchmark.json` by opening `benchmark.html` in your favourite browser.

Multiple input controllers and methods

dtControl can run multiple methods on multiple controllers. For example, if one desires to evaluate *LinSVM* and *LogReg* methods along with the *MinNorm* determinizing strategy on *cartpole* and *10rooms*, the following command may be used.:

```
$ dtcontrol \
  --input examples/cartpole.scs examples/10rooms.scs \
  --method linsvm logreg \
  --determinize minnorm
```

Both the `--method` and `--determinize` flags support the *all* shorthand. For example, the following command will run all methods with all determinization strategies on *cartpole*.:

```
$ dtcontrol -i examples/cartpole.scs -m all -d all
```

where `-m` all is a shorthand for `-m cart linsvm logreg oc1` and `-d` all is a shorthand for `-d none minnorm maxfreq`

dtControl can also take whole folders or wildcards as input. In case the `-i` switch gets a valid folder as argument, dtControl tries to read all `*.scs`, `*.dump` and `*.csv` as inputs for its methods. Wildcards behave as you would expect in any shell.:

```
$ dtcontrol -i examples -m cart
$ dtcontrol -i examples/*.dump -m cart -d maxfreq
```

A list of methods supported by dtControl and their respective command-line switches is given in the table below.

Table 1: List of methods

Method	Switch
CART	`-m cart -d none`
Linear SVM	`-m linsvm -d none`
Logistic Regression	`-m logreg -d none`
OC1	`-m oc1 -d none`
CART + maximal frequencies (MaxFreq)	`-m cart -d maxfreq`
LogReg + maximal frequencies (MaxFreqLC)	`-m logreg -d maxfreq`
CART + minimal norm (MinNorm)	`-m cart -d minnorm`
LinSMV/LogReg + minimal norm (MinNormLC)	`-m logreg -d minnorm`

1.1.3 Advanced Features

Output location

By default, the decision trees are stored in the decision_trees folder and the statistics are stored in the benchmark.json and benchmark.html files. This can however be customized with the help of the `--output/-o` and the `--benchmark-file/-b` switches. For example,:

```
$ dtcontrol -i examples/cartpole.scs -m cart \
  --output cartpole_trees \
  --benchmark-file cartpole_stats
```

Will produce the following files and directories:

```
cartpole_trees
|-- CART
|   `-- cartpole
|       |-- CART.c
|       |-- CART.dot
cartpole_stats.json
cartpole_stats.html
```

Timeout

Another useful feature is timeout which can be set with the `--timeout/-t` switch. For example,:

```
$ dtcontrol -i examples/truck_trailer.scs -m cart -t 3m
```

will run CART on the *truck_trailer* example, and time out if it is taking longer than 3 minutes to finish. The `--timeout/-t` switch can accept timeout in seconds, minutes and hours (`-t 42s` or `-t 30m` or `-t 1h`). The timeouts are applied for each method individually, and not for the whole set of experiments.

Re-run

By default, new results are appended to `benchmark.json` (or the file passed to the `--benchmark-file` switch) and experiments are not re-run if results already exist. In case you want to re-run a method and overwrite existing results, use the `--rerun` flag.:

```
$ dtcontrol -i examples/truck_trailer.scs -m cart -t 3m --rerun
```

1.1.4 Quick Start with Python Interface

More advanced users can use dtControl programmatically using Python. Here is a sample code.:

```
# imports
# user might have to import additional classifiers

from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from benchmark_suite import BenchmarkSuite
from dtcontrol.classifiers.cart_custom_dt import CartDT
from dtcontrol.classifiers.linear_classifier_dt import LinearClassifierDT
from dtcontrol.classifiers.max_freq_dt import MaxFreqDT
from dtcontrol.classifiers.max_freq_linear_classifier_dt import␣
↪MaxFreqLinearClassifierDT
from dtcontrol.classifiers.norm_dt import NormDT
from dtcontrol.classifiers.norm_linear_classifier_dt import NormLinearClassifierDT
from dtcontrol.classifiers.oc1_wrapper import OC1Wrapper

# instantiate the benchmark suite with a timeout of 2 hours
# rest of the parameters behave like in CLI

suite = BenchmarkSuite(timeout=60 * 60 * 2,
                       output_folder='decision_trees',
                       benchmark_file='benchmark',
                       is_artifact=True,
                       rerun=False)

# Add the 'examples' directory as the base where
# the different controllers will be searched for

suite.add_datasets(
    # names of the folder containing scs/dump/csv files
    ['examples'],
    # names of the scs/dump/csv files without the extension
    include=["cartpole", "10rooms"],
)

# select the methods which would be applied on the datasets
classifiers = [
    # CART
    CartDT(),
    # LinSVM
    LinearClassifierDT(LinearSVC, max_iter=5000),
    # LogReg
    LinearClassifierDT(LogisticRegression, solver='lbfgs', penalty='none'),
    # OC1
    OC1Wrapper(num_restarts=20, num_jumps=5),
```

(continues on next page)

(continued from previous page)

```
    # MaxFreq
    MaxFreqDT(),
    # MaxFreqLC
    MaxFreqLinearClassifierDT(LogisticRegression, solver='lbfgs', penalty='none'),
    # MinNorm
    NormDT(min),
    # MinNormLC
    NormLinearClassifierDT(min, LogisticRegression, solver='lbfgs', penalty='none'),
]

# finally, execute the benchmarks
suite.benchmark(classifiers)
```

1.2 Developer Manual

This document aims to provide the reader with the necessary information to be able to extend or customize dtControl. We first briefly describe how new decision tree algorithms can be added to the tool. Subsequently, we outline how new file formats can be supported.

dtControl is written entirely in Python and makes use of both the numpy and scikit-learn packages for data representation and manipulation. A basic familiarity with this programming environment is assumed throughout this manual. More information on dependencies can be found in the provided readme and setup.py files.

1.2.1 Extending dtControl with new algorithms

dtControl already supports a wide variety of decision tree construction algorithms. Furthermore, the tool can readily be extended with new algorithms, as we will see in this section.

The general decision tree structure is provided in the abstract base class `CustomDT`. While it is not necessary for new classifiers to extend this class, it is highly recommended, since it already satisfies the interface that dtControl expects. This includes the following attributes and methods:

- `name`: the name of the algorithm, as it will be displayed in the benchmark results.
- `fit(dataset)`: constructs the decision tree for a `dataset`.
- `predict(dataset)`: returns a list of control inputs predicted for the dataset.
- `get_stats()`: returns the statistics to be displayed in the benchmark results as a dictionary. This will mainly include the number of nodes and potentially some algorithm-specific statistics.
- `is_applicable(dataset)`: some algorithms might be restricted to either single- or multi-output datasets, in which case this method can be used to indicate that an algorithm is not applicable to a `dataset`.
- `save()`: saves a representation of the class that can be used for debugging.
- `export_dot()`: saves a representation of the decision tree in the DOT format.
- `export_c()`: exports the decision tree to a C-file as a chain of if-else statements.

A `CustomDT` object also contains a reference to the root node of the decision tree (which is `None` before `predict()` is first called). The abstract base class `Node` provides the actual tree data structure and various methods that can be overridden to customize its behavior.

To implement a new algorithm, you thus need to provide two classes: One represents the actual decision tree and should extend `CustomDT`, while the other represents nodes in the decision tree and extends the `Node` class.

The `fit()` method is given a dataset object, which is used to construct the decision tree. The two most important attributes of datasets are `X_train`, a numpy array containing all states, and `Y_train`, containing the actions that can be performed in those states. Depending on whether the dataset is single- or multi-output, the format of `Y_train` differs:

- In the case of single-output datasets, `Y_train` is a two-dimensional array, where each row contains all (non-deterministic) actions that can be performed at the corresponding row of `X_train`. Instead of the actual floating point values, we use integer indices representing those values throughout the code; the mapping of indices to the actual values can be found in `dataset.index_to_value`. Since numpy usually cannot deal with rows of different sizes, but we have varying numbers of possible actions, some rows have to be filled with `-1` s. These `-1` s have to be ignored during tree construction.
- In the case of multi-output datasets, `Y_train` is a three-dimensional array whose first dimension (or axis) corresponds to the different control inputs. Thus, there is a two-dimensional array for each control-input, which exactly matches the structure outlined above. To get the possible (multi-input) actions for a specific state, the arrays for the different control inputs have to be "stacked" in order to get the list of action tuples that can be performed.

The dataset class provides various methods to convert the format of `Y_train` to a more convenient representation to be used in decision tree construction. For example, `get_unique_labels()` maps all non-deterministic actions to a single index and thus returns simply a list of indices, which can directly be used as labels for any decision tree algorithm. After the tree has been constructed, its labels can be mapped back to the original non-deterministic actions using the `set_labels()` method provided in the Node class.

For examples of how new algorithms are implemented, it could be instructive to look at the `LinearClassifierDT` and `MaxFreqDT` classes, which implement tree construction using predicates from linear classifiers and the MaxFreq determinization procedure, respectively.

1.2.2 Supporting new file formats

dtControl currently supports the file formats generated by the the tools SCOTS and Uppaal Stratego. There are two ways to make the tool work with other formats, as described in the following.

The CSV format

The first option is to convert the new file format to a custom CSV format that dtControl also supports. We now describe the specification of the custom CSV format.

The first two lines of the file are reserved for metadata. The first line must always reflect whether the controller is permissive (non-deterministic) or non-permissive (deterministic). This is done using either of the following lines:

```
#PERMISSIVE
```

or

```
#NON-PERMISSIVE
```

The second line must reflect the number of state variables (or the state dimension) and the number of control input variables (or the input dimension). This line looks as follows::

```
#BEGIN N M
```

where *N* is the state dimension and *M* is the input dimension.

Every line after the 2nd line lists the state action/input pairs as a comma separated list::

```
x1,x2,...,xN,y1,y2,...,yM
```

if the controller prescribes the action `(y1,y2,...,yM)` for the state `(x1,x2,...,xN)`. If the state allows more actions, for example, `(y1',y2',...,yM')`, then this should be described on a new line::

```
x1,x2,...,xN,y1,y2,...,yM
x1,x2,...,xN,y1',y2',...,yM'
```

An excerpt of the `10rooms.scs` controller written in this CSV format would look as follows::

```
#PERMISSIVE
#BEGIN 10 2
18.75,20.0,18.75,18.75,20.0,18.75,18.75,18.75,18.75,18.75,1.0,1.0
20.0,20.0,18.75,18.75,20.0,18.75,18.75,18.75,18.75,18.75,1.0,1.0
21.25,20.0,18.75,18.75,20.0,18.75,18.75,18.75,18.75,18.75,1.0,1.0
18.75,21.25,18.75,18.75,20.0,18.75,18.75,18.75,18.75,18.75,0.0,1.0
18.75,21.25,18.75,18.75,20.0,18.75,18.75,18.75,18.75,18.75,0.5,1.0
18.75,21.25,18.75,18.75,20.0,18.75,18.75,18.75,18.75,18.75,1.0,1.0
20.0,21.25,18.75,18.75,20.0,18.75,18.75,18.75,18.75,18.75,0.0,1.0
```

dtControl will automatically look for files with a .csv extension and parse them with the assumption that they follow this format.

Implementing a new dataset loader

Additionally, it is also possible to integrate the new file format natively into dtControl by providing a dataset loader. This should be a class that sub-classes the `DatasetLoader` class and provides exactly one method: `_load_dataset()` parses a file in the new format and returns a tuple with the following elements:

- `X_train`: the data array as outlined above.
- `X_metadata`: is a dictionary containing various information about the dataset, such as the names of the columns in `X_train` and the minimum and maximum values for each column.
- `Y_train`: the label array as outlined above.
- `Y_metadata`: is a dictionary containing information about `Y_train`.
- `index_to_value`: maps from integer indices to the actual floating point values used as control inputs.

The new dataset loader can be registered in the `extension_to_loader` attribute of the `Dataset` class. Now, if dtControl encounters a file with an extension of the new file format, it will attempt to load it using the registered loader.

Examples of such dataset loaders can be found in the `ScotsDatasetLoader` and `UppaalDatasetLoader` classes, however, they are very specific to the file formats used by the two tools.