Model-Based Software Testing via Incremental Treatment Learning

Dustin Geletko, Tim Menzies
Lane Department of Computer Science and Electrical Engineering
West Virginia University, USA
dgeletko@mix.wvu.edu, tim@menzies.us

Abstract

Model-based software has become quite popular in recent years, making its way into a broad range of areas, including the aerospace industry. The models provide an easy graphical interface to develop systems, which can generate the sometimes tedious code that follows. While there are many tools available to assess standard procedural code, there are limits to the testing of model-based systems. A major problem with the models are that their internals often contain gray areas of unknown system behavior. These possible behaviors form what is known as a data cloud*, which is an overwhelming range of possibilities of a system that can overload analysts [3]. With large* data clouds*, it is hard to demonstrate which particular decision leads to a particular outcome. Even if definite decisions can't be made, it is possible to reduce the variance of and condense the clouds [3]. This paper presents two case studies; one with a simple illustrative model and another with a more complex application. The TAR3 treatment learning tool summarizes the particular attribute ranges that selects for particular behaviors of interest, reducing the* data clouds*.*

1 Introduction

Our thesis is that, when debating about complex systems with uncertain behavior, humans are slower than machine learners at identifying key decisions that give the most leverage, saving both time and money, and allowing humans to devote more time on important decisions and less time on irrelevancies [9]. Benefits are then maximized, while costs are minimized. The TAR3 treatment learner further benefits the scenario by automatically finding the best and worst possible situation within a model's domain [9]. Our bottom line is "time = money", which is why automatic methods are preferred.

In this paper, we test incremental treatment learning using software models such as those simulated in Matlab © Simulink for NASA applications; e.g. (*STEREO, GLAST*). These models contain many complex control variables that produce various outputs. Although a certain range of values that the inputs to a system normally contain are known, a particular value at any certain point in time could be random, and checking every scenario for each value could prove to be very time consuming. Knowledge of the range of values or what particular value tends to steer a system towards interesting points such as success or failure could be critical and help evaluate the clouds of uncertainty in complex systems.

In developing our tool to search for attributes that led to particular model outputs, a few assumptions were necessary. First, the output values of interest were known and divided into predetermined classes. That is, users could act as an oracle to assign some utility to the outputs of the model. This is the key to the assessment of the model. There must be some notion of a *desired* class in order to find mitigations that lead to that class.

A second assumption was that *narrow funnels* existed within the system. There can be numerous inputs or control variables in any particular system. The *narrow funnel* assumption states that only a few critical variables control the overall behavior of the entire system. In other words, only a subset of all of the system controllables can be used to adequately predict the overall system behavior [10].

Another important assumption was that the operational profile of the system, or the normal operating input value ranges, was known. In order to test a system, parameters of the system must be known. In other words, ranges of the possible values that the inputs can hold must be known so a Monte Carlo analysis can be performed between those ranges and the funnels that lead to certain predetermined output classes can be utilized. Otherwise, there is an infinite space to be sampled when conducting the analysis.

The rest of the paper applies these assumptions to two models. The first model is a simple model, mainly used for illustrative purposes. The second model utilized the method on a complex real-world model, illustrating scalability on more practical applications. The second model also illustrates the generality of this method, being applicable to any

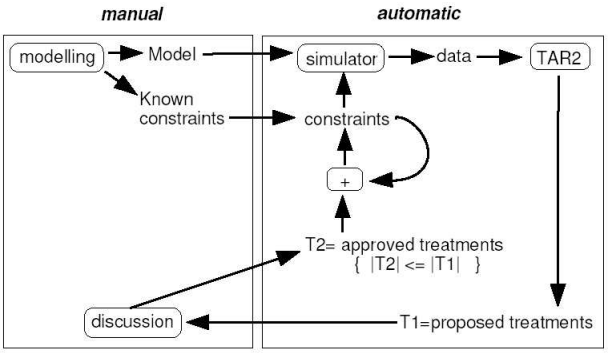

Figure 1. Incremental Treatment Learning

model-based software package.

2 Incremental Treatment Learning

"Understanding" models refers to the act of finding a model's strengths and flaws, and possibly mitigations that ultimately reduce those flaws [10]. Our proposed model understanding tool is the process of incremental treatment learning (See Figure 1).

To utilize incremental treatment learning, a controller was developed to test for funnels in the software models. The basic concept was to continuously generate random inputs to the model within the known range and simulate the model numerous times. This Monte Carlo analysis generated a large data file. The file was fed through the treatment learner (TAR3), which summarized the inputs along with the ranges of those inputs, called treatments, that most predicted for a predetermined desired state. The inputs evaluated could either be system inputs or inputs to a subsystem. This process was repeated until no new information was learned. The output's worth, or the amount that the original distribution was moved towards the desired outcome, was used as a measure of the information learned.

Repeating this process forces the output of the system to shift towards the desired distribution by continually constraining the inputs according to the newly discovered treatments. The process will stop when either TAR3 finds no new treatments or there are no new treatments that will drive the output towards the desired state, and thereby increase the worth. Decisions can then be made about the treatment variables, which override decisions about variables not given in the treatments because they simply add redundancy to the outcome. Minimal strategies can then be defined to decrease uncertainty in the model [3]. The usefulness of this system can be visualized if it is found to provide significant results and help to better understand the model.

The core of incremental treatment learning is the TAR3 treatment learner. In summary, TAR3 is a tool for performing automatic sensitivity analysis of large data files looking for constraints to parameter ranges that can most *improve* or *degrade* the performance of a system [7, 8]. The tool performs a random search over the data and is capable of discovering treatments not easily found by hand.

Another way to characterize TAR3 is to call it a *data mining* tool. For a comparison of TAR2, the predecessor of TAR3, with standard data miners, see [6]. For more on standard data mining tools, see [14]. The advantage of TAR3 over classic machine learners is its ability to give short concise treatments of interest rather than complicated and sometimes very large decision trees. The quick and easy to read treatments dubs it the name "data mining for very busy people" [8] because most practitioners are too busy or have too many time constraints to deal with long complicated reports.

TAR3 assumes the *small treatment effect*; i.e. that within a model, a very small number of variables control most of the other variables. This small treatment effect has been reported in many domains, albeit under different names. So much so that Menzies and Cukic [4, 5] and Menzies and Singh [11] speculated that small treatments are an emergent property that will appear in most models. This property, known as *Funnel Theory*, is described in detail in [3]. The theory has been extensively tested in the discrete domain, but research in the numeric domain has been limited.

TAR3 is a machine learning technique that orders classes by a scheme of weights, with the desired class having the highest weight. A calculation, called lift, then finds a subset of the data, called treatments, which are those attributes that shift the baseline class distribution to a distribution containing the most occurrences of the desired class. These attributes given as treatments are the funnels of the system. These treatments can then predict the attribute ranges that lead to the desired class. For more details on treatment learning, see [1]. For a short overview, see [8].

3 Feasibility

This method may not be feasible for certain models. Decisions must be made about the advantages of the method, especially for critical applications, since TAR3 does not implement an exhaustive search method. This method could also require performing an overwhelming number of simulations in the Monte Carlo analysis, which could be unfeasible for large-scale models whose simulations require unusually long run-times [10]. Determining a sufficient number of random samples in the Monte Carlo analysis that adequately represent the input space, and therefore converge the treatments, could also be very time consuming.

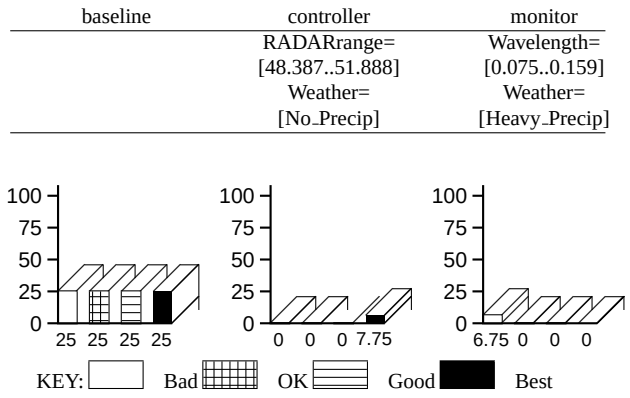

Figure 2. RADAR
Output class distribution percentages of the original treatments provided

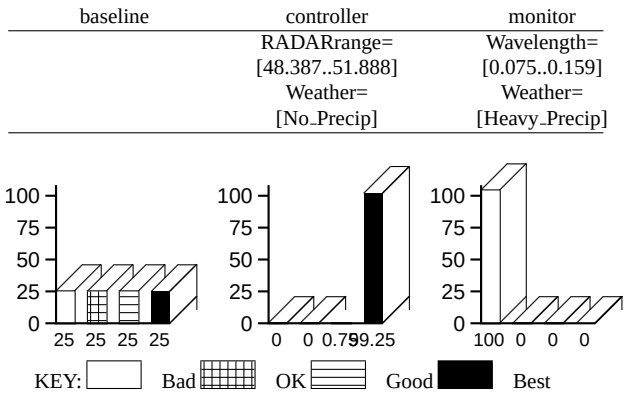

Figure 3. RADAR
Output class distribution percentages after testing the accuracy of the treatments predicted

4 Case Studies

4.1 RADAR Design

4.1.1 Background

The first study illustrates the process of incremental treatment learning on a simple, small-scale model. A more complex and large-scale model follows. In performing the research, the built-in Simulink aerospace 'Air Traffic Control Radar Design' demo was used. This model supplied a GUI in which certain radar design parameters could be chosen to observe the various effects on the performance of the system. The goal of this case study was to perform the process of incremental treatment learning twice, once to find the controllers of the system, and the second time to find the monitors of the system. The controllers are the variables that lead the system to the *desired* state, and the monitors are the variables that lead the system to the *undesirable* state, in which preventive measures need to be implemented [1]. The *desired* state in this case was detecting the aircraft at higher altitudes, and the *undesirable* state was detecting the aircraft at lower altitudes, corresponding to poor system performance.

4.1.2 Process

The process of testing the system involved numerous automated steps. The first step was writing a parameter file that determined the types of inputs to the system. In the experiments performed, possible inputs included normal, gamma, beta, or discrete value distributions. The range and function parameters such as mean, standard deviation, gamma value, and beta values also had to be specified. The input distributions were written to a file and fed through the software model to simulate the system.

All of the model input parameters to design the radar were programmed through a GUI. The GUI was modified to allow ranges of values instead of a single value to be set for each of the system's attributes. Beta distributions with a beta value of 0.5 (which is equivalent to uniform random sampling) were then generated according to the values set in the GUI and fed into the model's inputs. The aircraft's motion was simulated using a sinusoidal wave. For each set of random input values, the simulation was run and the maximum altitude that the aircraft was detected was recorded as the system output. The altitude was then discretized into 4 equal categories: *Bad*, *OK*, *Good*, *Best*, with *Best* corresponding to detecting the plane at the highest altitudes and *Bad* not picking up the aircraft until it was flying at low altitudes.

Once TAR3 found the treatments of certain inputs, these values, whether a continuous range or a dicrete subset, acted as a constraint and replaced the original input distributions. The parameter file was rewritten according to the treatments and the new range of values were fed through the system, producing a *funnel* and narrowing the input range further and further. These treated inputs were then found to lead to the desired classification for both the case of finding the system controllers and monitors.

4.1.3 Results

The process was terminated after either the treatment learner found no new treatments or the worth of the treat-

ments remained static, which occurred after two runs with satisfying results for both the case of finding system controllers and monitors. The outputs were discretized into the four equal classes, so that the baseline distribution was even. In the case of the controller, TAR3 found a 7.75% subset of the original data, in which the inputs were constrained by the treatments, that predicted for the desired class all of the time. Listed below are the input attributes that were given by the treatments in the case of finding a system controller:

1. *RADAR range = [48.387..51.888]*
2. *Weather = No Precipitation*

In the case of the monitor, TAR3 found a 6.75% subset of the original data, in which the inputs were constrained by the treatments, that also predicted for the desired class all of the time. Listed below are the input attributes that were given by the treatments in the case of finding a system monitor:

1. *Wavelength = [0.075..0.159]*
2. *Weather = Heavy Precipitation*

TAR3's final treatments and output class distributions for each scenario are shown in Figure 2.

4.1.4 Validation

In order to check the accuracy of the predictions given by TAR3, the inputs were then constrained according to the treatments in both cases and simulated the same number of times as performed in the Monte Carlo analysis to check the percentage of cases in which the output fell within the desired range when the treatments were applied. The distributions of the simulation outputs are shown in Figure 3 after the treatments were fed into the system. These distributions illustrate that TAR3 correctly predicted the treatments in both scenarios. For the case of the system controller, 99.25% of the output values fell within the desired range. In the case of the system monitor, TAR3 found treatments such that 100% of the output values fell within the interesting class, which was poor system performance in this scenario.

Figure 4 illustrates the improvement along with the degradation in radar altitude detection performance. In this figure, the average altitude of all the simulations in the Monte Carlo analysis is plotted. The error bars indicate the standard deviation of the average detection altitude. Run number 1 corresponds to the initial analysis, and run number 2 corresponds to the analysis after the inputs were constrained according to the treatments. The system controller altitudes are given in the graph on the left, while the system monitor altitudes are given in the graph on the right. This plot demonstrates that when the inputs were constricted to

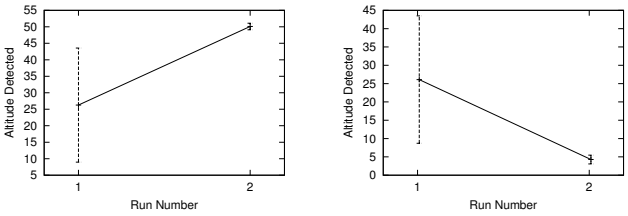

Figure 4. Average Altitude Detected by RADAR for *Desirable* and *Undesirable* Classes

the values given by the treatments, nearly 100% of the outputs fell within the desired range while simultaneously reducing the variance significantly, providing a more confident result.

4.1.5 Discussion

The treatments predicted may sound trivial, for instance, *Heavy Precipitation* leading to low altitude detection and therefore poor performance, but we speculate that the method could prove to be very insightful on more complex models, providing useful information. The method succeded in this experiment, appropriately finding predictors for both the *desired* and *undesirable* class and reducing system variance and uncertainty.

4.2 Word Model

4.2.1 Background

In order to explore the generality of the previous study, we chose a more complex model, which is described below. In 1972, a team of system scientists and computer modelers studied the effects of the world's exponentially growing population and economy. A model was developed of the world, and it predicted *Doom!* for the future, as shown in Figure 6.

This model of global economics was complex, containing approximately 295 variables and over 100 nodes. Figure 5 illustrates the complexity of the revised model, a *Vensim*© model entitled *World3-91*. Such complex models require skilled analysts to perform assessments. Qualified analysts with knowledge about the internals of the model are scarce and expensive.

The analysts spent a long time assessing the model and determining possible solutions to prevent this predicted disaster. It would prove beneficial to have an automatic interpretation of the model and mitigations to improve the *state of the world*. After applying incremental treatment learning, we arrived at the same conclusions as the scientists with our automated method in a matter of about 30 minutes. Our method also produced confident results because

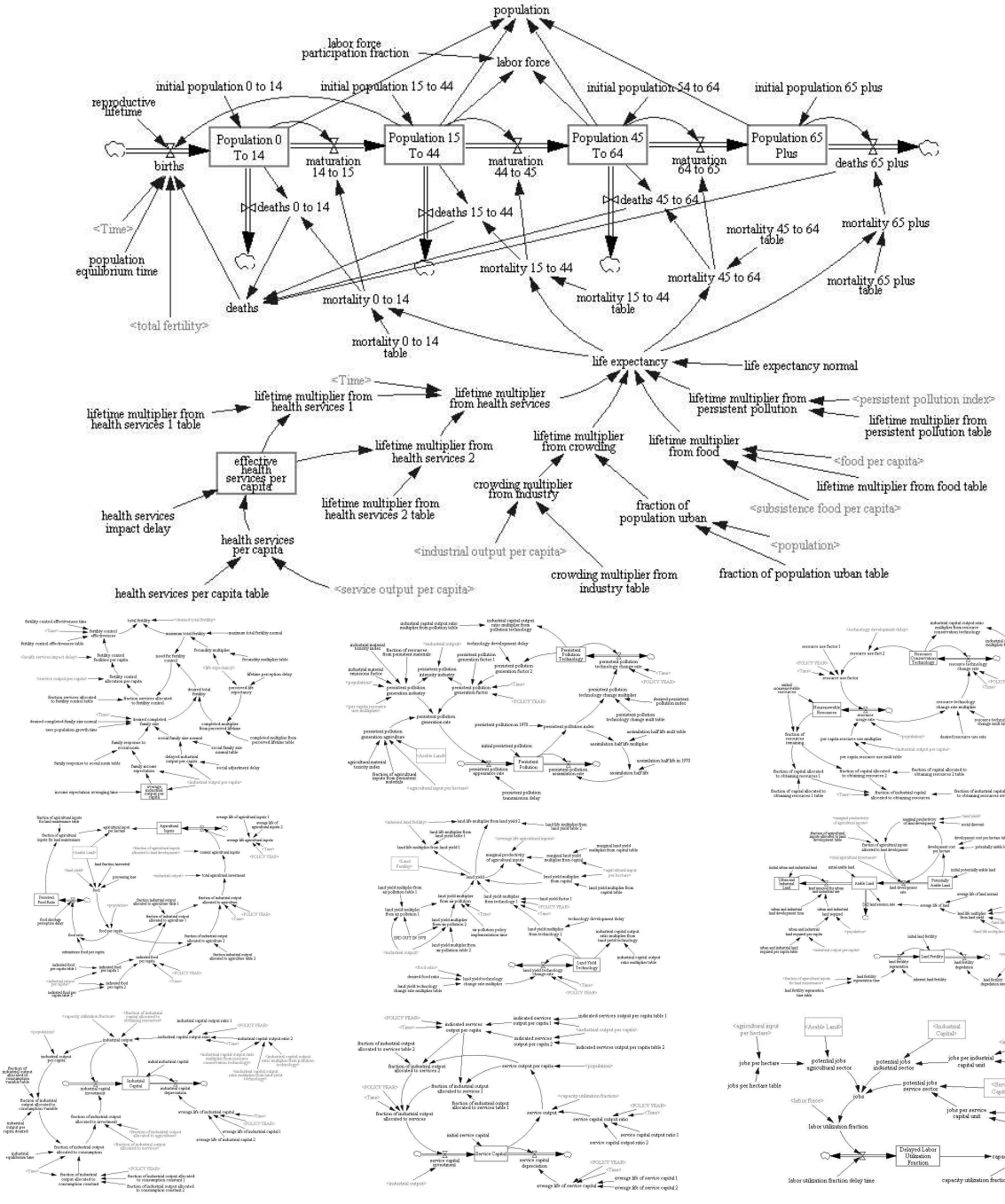

Figure 5. The main part of the *Vensim*© *World3-91* model is shown at the top along with all of the supporting models shown below

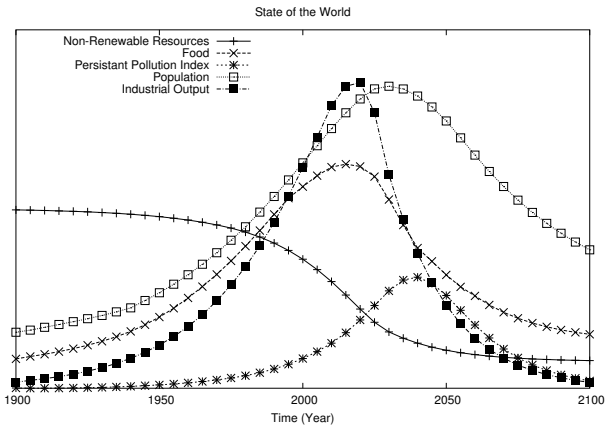

Figure 6. Default "state of the world" plot illustrating overshoot and collapse

the machine learner could quickly look at a wider range of scenarios.

Figure 7 shows the results of the experiment given by the C4.5 classifier, which further illustrates the usefulness of the small treatments given by TAR3. These complex decision trees are standard practice in current machine learning. Clearly, this complex and opaque tree is too complicated and time consuming for practitioners to read and less insightful as the small compact key variables given as treatments by the TAR3 treatment learner.

The default output plots of the revised model consists of the classes of world population, nonrenewable resources, food, industrial output, and persistent pollution index from the year range 1900 to 2100. The model is rather complex, consisting of hundreds of variables, comprised of the five main sectors of persistent pollution, nonrenewable resources, population, agriculture(food production, land fertility, and land development and loss), and economy(industrial output, services output, and jobs) [2].

All of the plots illustrate the continued exponential growth for the next couple of decades and then the "limits" of the earth and its resources are reached. In assessing the model, the projected year of complete collapse of 2100 (Shown in Figure 6) was used as the output class to try to learn the model controllers in order to prevent this scenario.

4.2.2 Process

The model constants were first divided into controllables and uncontrollables, because we needed variables that could be controlled in order to find mitigations to lead us to a more desirable state. We chose to use 44 model variables in our analysis that appeared feasible to control, whether by people's choices, political organizations, or laws. They

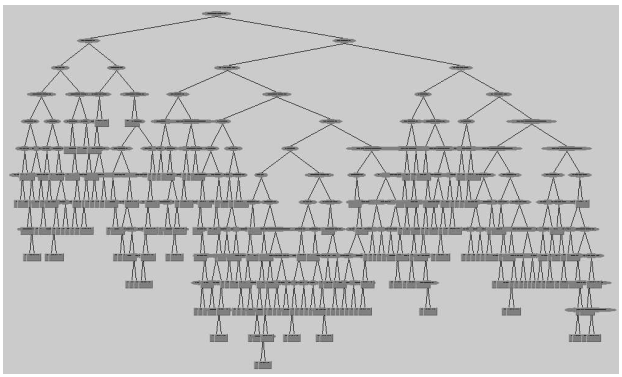

Figure 7. Decision Tree produced by C4.5

were chosen to perform the Monte Carlo analysis on and fed into the learner as the input attributes. The interesting class variables wished to be changed were an increased and steady life expectancy, food, nonrenewable resources, industrial output, and decreased pollution.

The input values and output of every class at the final time step was first saved for every simulation of the Monte Carlo analysis. Each class was then individually sorted, and normalized between 0 and 1. Because the TAR3 treatment learner is limited to predicting a single class, a utility function had to be implemented to combine all five of the interesting classes.

The utility function was a scheme of weights multiplied with each class and then summed to obtain a single numeric value as follows:

$$w_1 * C_1 + w_2 * C_2 + \cdots + w_n * C_n$$

where $w = weight$ and $C = Class$. This allowed important output classes to carry more or less weight, depending on their importance in the model.

The main question was how to appropriately develop the utility function. We decided to allow TAR3 to automatically choose the optimal weights. We developed these weights by randomly choosing five numbers between 0 and 1, one for each interesting class, multiplying the number by the corresponding class, and adding them together to obtain a single class value. Each of the weights were added to the data file as model attributes, along with the final class number after the utility function was applied. This method of developing the function was used to determine a number that combined all of the classes and was utilized throughout the experiment.

Because the TAR3 treatment learner is only capable of handling nominal classes, the final class was sorted numerically and discretized into four equally sized groups and assigned class names *(Bad,Ok,Good,Best)*, with *Best* corresponding to the highest value and *Bad* corresponding to the lowest value indicating worst case scenarios.

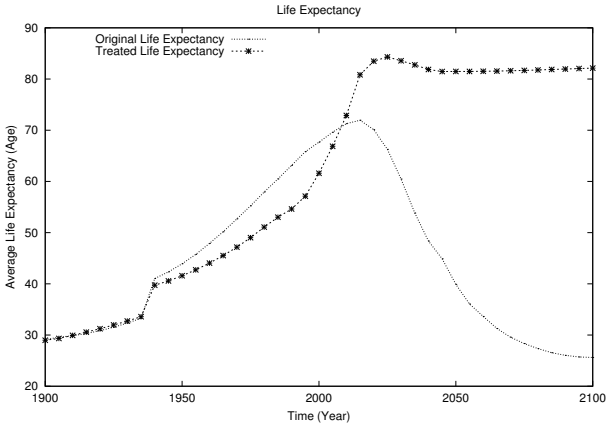

Figure 8. Life expectancy before and after incremental treatment learning

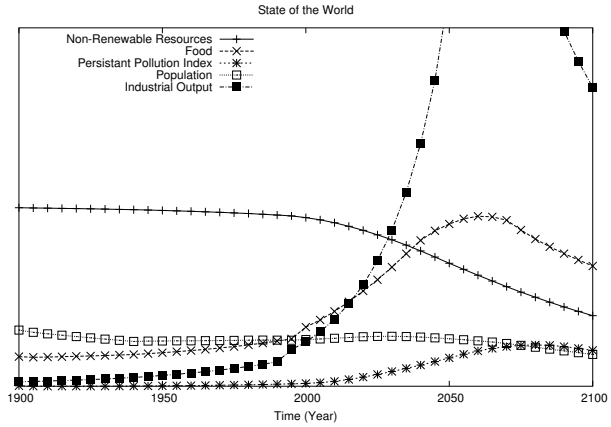

Figure 9. *State of the world* **after implementing treatments found by the Tar3 treatment learner**

This final file was fed into TAR3 to find treatments that led to the desired class. If any of the weights were given as treatments, the random numbers were regenerated and multiplied according to the ranges given in the treatments. This was repeated until a model attribute was given as a treatment, and then incremental treatment learning was utilized.

In this model, all treatments that increased the likelihood of the desirable class were discovered after just 2 cycles of the process. The two attributes given as the final treatments for the classes of interest were:

1. *desired completed family size normal = [0..2]*

2. *Industrial Capital Output Ratio 1 = [3..5]*

4.2.3 Validation

The above treatments, *desired completed family size normal* and *Industrial capital output ratio 1*, were then fed back into the model to observe the output and check the accuracy of the predictions given by TAR3. These attributes not only controlled the classes given in the *state of the world* default plot, but also raised and stabilized life expectancy in years, as illustrated in Figure 8. The plots are shown in Figure 6 for the original *state of the world* and in Figure 9 for the stabilized *state of the world*. The final plot (Figure 9), which shows the outcome of the *state of the world* when the model values were changed according to the treatments, indeed improved the chosen classes. Although they never reach as high quantities as in the original scenario, the values, in general, remain constant throughout the time frame. The major gain was in life expectancy, as was shown in Figure 8, in which the average life expectancy was increased to over 80 years of age and remained constant.

4.2.4 Discussion

The book that summarizes the findings of the *World3-91* model, *Beyond the Limits*, states that one way to transition to a sustainable system is to develop better birth control methods and *having at most two children*. This shows that the main path found by both TAR3 and the authors was to decrease and limit family sizes, with our method observing these conclusions quickly and competently, taking into account thousands of possible scenarios.

Further, we have removed one potential criticism of this analysis only capable of handling a single output class by incorporating a utility function, in which multiple classes are optimally combined to achieve an improvement in the overall result.

5 External Validity

Some questions may arise about the external validity of this method to finding certain funnels in the system that lead to a desired output. The first is the question about the accuracy of the random sampling of the inputs. How many values must be sampled and continued to be simulated in order for the machine learner to accurately predict treatments that led to the output? This question is not an easy one to answer and is probably system dependent. If the input can contain a large range of values, more samples are needed to ensure that every extreme is randomly chosen.

If the process correctly predicts the treatments that lead to the desired class, the following results could be observed. First of all, the solutions will eventually converge. For instance, if thousands of random samples are an adequate estimation of the system performance, then sampling millions

of values will deliver similar results. So when the final treatments that the program delivers converge to a particular value within a certain precision, sampling more and more input values is unnecessary. The value of convergence could be determined and verified by trial and error. If an insufficient number of samples from the input ranges are taken, TAR3 could give false or misleading treatments. In the case of the RADAR model, 400 random samples were sufficient to converge the treatments. In the more complex *World3-91* model, 10,000 random samples were used to converge the treatments.

Finally, if the final treatments are correct and lead to a particular output value of interest, those values can be fed back into the system and yield similar outputs every time. This data could provide important information regarding any system. Large complex systems sometimes contain cloudy areas that are not fully understood. This method treats the system as a *black box*, with the controllable inputs and outputs as the only interests. The inside operation is not important, just the final results and outputs. Measures could then be taken to ensure fault tolerance to handle failures in particular outputs when the inputs reach the given values or the inputs could be steered towards a value that leads to desired outputs. In both case studies, the treatments given were verified and proved to lead the system to the *desired* state.

6 Related Work

Traditional control theory methods such as State-Space analysis or Root Locus methods are more accurate methods of determining the manner in which the open-loop poles and zeros should be modified in order for the system response to meet the required specifications [12]. Qualitative reasoning is another approach to this problem. Sensitivity analysis could also be performed on the system to observe how changes in input produced variations in the system output [13]. In critical applications, more time should be focused on these methods. The disadvantage of performing these tedious mathematical methods is that much has to be known about the model to develop the system equations. The main benefit of using data mining methods to perform this crucial role of controlling the model is that knowledge about the model's internals is unnecessary, so we can treat it like a *black box*.

7 Problems to Address

There are numerous obstacles that must be overcome when utilizing incremental treatment learning. One major problem that must be addressed is developing the oracle that determines the operational input value ranges and provides an assessment criteria for the *desired* output. Different subsystems and portions of the model are designed and reused by many different organizations, so obtaining a specification sheet on input ranges and optimum output values from each contributor to the system could be costly and time-consuming. If the operational profile is unknown, there is an infinite input sample space. Also, *desirable* output values must be known in order to find controllers, and *undesirable* outputs must be known to find monitors by the TAR3 treatment learner.

Another major problem, especially with the NASA *STEREO* model, is that the model is comprised of the actual model and an environment simulator. All of the data from the environment entering the model is strictly numbers of different data types. There is no real meaning in assessing the model unless something is known about these numeric values. For instance, if the machine learner gives a treatment that includes some of the environment attributes, which are simply numbers of different data types, how would a practitioner assess the numbers to understand the problem in order to utilize the mitigations given by the TAR3 treatment learner without a specification sheet?

In conclusion, knowledge of the domain space is the major issue when utilizing incremental treatment learning to test model-based software. It is difficult and sometimes impossible to obtain all of the data needed to assess the input and output spaces, yet it is crucial to the analysis.

8 Conclusion

These experiments suggest that machine learning can be a valuable and time-saving technique for quickly finding treatments that move the output distributions of complex models towards a desired class, replicating the actions of highly skilled practitioners in a fraction of the time, significantly reducing costs.

The possibilities of incremental treatment learning are illustrated by these case studies. Suppose little is known about a system and limited formal testing tools are available for software models. If the output values that lead a system to failure are known along with a constricted range of possible input values, analysis of all of these inputs via treatment learning could illustrate the important ranges that lead the model to the desired state.

If the learner finds possible attribute ranges that can potentially lead to failure, appropriate actions could be taken to ensure the proper handling to either tolerate the faults or to avoid them. This improvement of the plots illustrates the success of treating the model as a *black box* and using data mininig techniques, in particular the TAR3 treatment learner, to summarize the results.

This analysis could prove useful in any kind of model in which the internals are unknown but need to be controlled.

This experiment may also lead practitioners to believe that *narrow funnels* are not limited to the discrete world. They may also be a property that many numeric models and data possess, expanding the applications of incremental treatment learning to evaluate models.

Our future direction is clear. More case studies must be conducted on larger scale NASA models. The next model that will test the value of this method will be the *STEREO* model. Once a constricted range of input values are known and a desired output range is known, the model will be tested. Either the entire system or an important subsystem of the *STEREO* model will be used due to lengthy simulation run-times. If the studies are successful and significant results are found, other models will possibly be tested. Hopefully significant issues can be determined quickly that are not detected using traditional verification methods.

Acknowledgements

This research was conducted at West Virginia University under NASA contract NCC2-0979 and NCC5-685. The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

[1] Y. Hu. *Treatment Learning: Implementation and Application*. May 2003. Available from `http://menzies.us/pdf/03hu.pdf`.

[2] D. H. Meadows, D. L. Meadows, and J. Randers. *Beyond the Limits*. Chelsea Green Publishing Company, Post Mills, Vermont, 1992.

[3] T. Menzies, E. Chiang, M. Feather, Y. Hu, and J. Kiper. Condensing uncertainty via incremental treatment learning. In T. M. Khoshgoftaar, editor, *Software Engineering with Computational Intelligence*. Kluwer, 2003. Available from `http://menzies.us/pdf/02itar2.pdf`.

[4] T. Menzies and B. Cukic. Adequacy of limited testing for knowledge based systems. *International Journal on Artificial Intelligence Tools (IJAIT)*, June 2000. Available from `http://menzies.us/pdf/00ijait.pdf`.

[5] T. Menzies and B. Cukic. When to test less. *IEEE Software*, 17(5):107–112, 2000. Available from `http://menzies.us/pdf/00iesoft.pdf`.

[6] T. Menzies and Y. Hu. Just enough learning (of association rules): The tar2 treatment learner. In *Journal of Data and Knowledge Engineering (submitted)*, 2002. Available from `http://menzies.us/pdf/02tar2.pdf`.

[7] T. Menzies and Y. Hu. The TAR2 treatment learner, 2002. Available from `http://www.ece.ubc.ca/twiki/pub/Softeng/TreatmentLearner/intro.pdf`.

[8] T. Menzies and Y. Hu. Data mining for busy people. In *IEEE Computer*, November 2003. Available from `http://menzies.us/pdf/03tar2.pdf`.

[9] T. Menzies, J. Kiper, and M. Feather. Improved software engineering decision support through automatic argument reduction tools. In *SEDECS: The 2nd International Workshop on Software Engineering Decision Support (part of SEKE2003)*, June 2003. Available from `http://menzies.us/pdf/03star1.pdf`.

[10] T. Menzies, D. Raffo, S. on Setamanit, Y. Hu, and S. Tootoonian. Model-based tests of truisms. In *Proceedings of IEEE ASE 2002*, 2002. Available from `http://menzies.us/pdf/02truisms.pdf`.

[11] T. Menzies and H. Singh. Many maybes mean (mostly) the same thing. In *2nd International Workshop on Soft Computing applied to Software Engineering (Netherlands), February*, 2001. Available from `http://menzies.us/pdf/00maybe.pdf`.

[12] K. Ogata. *Modern Control Engineering: 4th Edition*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 2002.

[13] A. Saltelli, K. Chan, and E. Scott. *Sensitivity Analysis*. Wiley Series in Probability and Statistics, 2000.

[14] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.