Exploiting Causal Domain Knowledge for Learning to Control Dynamic Systems

Achim G. Hoffmann

School of Computer Science and Engineering, University of New South Wales
P.O.Box 1, Sydney 2052, Australia, Email: achim@cs.unsw.oz.au

Abstract. This paper introduces a simple yet effective method for using causal domain knowledge for learning to control dynamic systems. Elementary qualitative causal dependencies of the domain are exploited in order to dramatically speed up the learning of reliable control strategies from a simulation model of the system. The reliability of the obtained control strategies is strengthened as well. The effectiveness of the method has experimentally been studied at the problem of learning to balance a pole.

1 Introduction

The automatic generation of controllers for dynamic systems from a model of the system is of great practical importance. For complexity reasons, classical control theory deals basically with linear models of systems. This limitation is perceived as an important issue because many practical systems are non-linear when described in a 'natural way'. An example is the classical pole balancing problem. To overcome this shortcoming, different methods for coping with nonlinear systems have been proposed. Among them are Fuzzy Control approaches, e.g. [4], symbolic Machine Learning approaches, e.g. [5] as well as Neural Networks [1], and Genetic Algorithms [11]. All these approaches face the problem of credit assignment: A control strategy can only be evaluated as a whole. If a strategy proves unsatisfactory, it is not clear how to alter the strategy to obtain improved performance. Although systems using various approaches have been designed, e.g. [5, 7, 1, 9], which manage to learn to balance a pole, it is unclear how well these approaches work for other and in particular for more complex systems. The mentioned learning approaches basically consider the system to be controlled as a black box. This appears in many cases unnecessarily restrictive. The more complex a system and the longer a sequence of control actions which has to be evaluated as a whole, the harder is the credit assignment problem. Generally speaking, the following two factors, which determine essentially the success or failure of the learning process, can be distinguished:

- The *set of strategies* which are potentially used as candidates for a solution.
- The *order* in which candidates are selected for testing.

If these factors are tailored for a learning task, the learning process may very quickly achieve its goal. On the other hand, if these factors are supposed to suit arbitrary problems, only relatively poor performance can be expected in average. Of course, tailoring the mentioned factors for each particular application requires engineering effort, which is actually tried to be avoided by learning approaches. However, in many domains tailoring can be done with little effort, while its effect on the learning speed as well as on the reliability of the learning result may be substantial.

This paper proposes to use coarse qualitative knowledge about the effect of control actions on the system being controlled to substantially improve the performance of learning. Recently, the problem of forming the class of considered control strategies by qualitative knowledge has been addressed, e.g. [10, 2, 11]. In these investigations, not exclusively but mainly qualitative models *about the system* to be controlled were used. In the following, qualitative knowledge *about possible control strategies* is exploited without stating anything about the physics of the system. But more importantly, this paper introduces the systematic use of *causal domain knowledge* for determining the order in which candidate strategies are tested. It should be emphasized, that a proper order of testing control strategies is absolutely crucial, since the number of possible strategies is in most cases excessive.

The paper is organized as follows. The next section reviews shortly other approaches to learn to control dynamic systems. Section 3 discusses qualitative domain knowledge that may be easy to provide. Section 4 presents a simple algorithm which exploits qualitative and causal domain knowledge for substantially improving the effectiveness of learning. This is demonstrated using the pole balancing problem. The conclusions are given in section 5.

2 Learning Control Strategies

The learning of strategies for controlling dynamic systems has attracted a large deal of investigations. In classical control theory, a profound body of analysis has been developed which is, however, basically restricted to *linear* systems. The pole balancing problem is a popular domain for case studies on controlling nonlinear dynamic systems. It is not only an attractive benchmark. It shows also similarities with control problems of practical importance, such as satellite altitude control [8]. A recent survey on approaches for learning to balance a pole can be found in [11].

The pole balancing system, see figure 1, consists of a pole which is hinged on a cart. The pole can swing in the verti-

cal plane. The cart can be moved to the right and left on a bounded track. The control task is to move the cart in a way that the pole stays balanced. The possible control actions are to apply a fixed force to the cart either to the left or to the right. This simple setting is also called *bang-bang regime*. The control decision can be made in small time intervals. Usually time intervals of a $\frac{1}{50}$ sec. are considered. A system state is
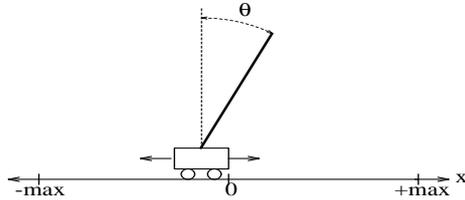

Figure 1. The pole is hinged on the cart. The cart is moving and the control actions are to expose the cart to a fixed amount of force either to the left or to the right.

described by the current position x of the cart on the track, its velocity $\dot{x}$ along the track, the angle θ of the pole and the pole's angle velocity $\dot{\theta}$.

Most of the learning approaches to control such a system determine a control strategy and test the strategy on the system over a specific time span. If the pole falls down, usually the time it has been successfully balanced, is considered as a quality measure for the tested control strategy. Subsequently, usually the single control decisions are analyzed, e.g. how often they have been taken, how long it took until the pole fell down after a particular control decision, etc. Upon that data, the tested control strategy is altered in order to generate the next candidate being tested. This idea has been implemented in a number of systems, e.g. in [3, 1] and many others. BOXES [5] is one of the very early approaches of that kind. BOXES learns a state-action table which specifies a control action for each possible system state. For that purpose, the possible system states are discretized. The four system parameters span a space of 4-dimensional boxes. Each of the boxes contains one of the two possible control actions. A control strategy is described by the complete set of control decisions for each of the boxes.

A typical discretization of the system states considers 3 values for each of x, $\dot{x}$ and $\dot{\theta}$ and 6 values for θ. Performance of BOXES is usually measured by the number of trials needed for balancing the pole for a given initial system state. Average numbers lie between 75 [6] and 557 [5] depending on the exact strategy for altering the control decision in a particular box.

Although these approaches proved to work for the pole balancing problem, it is not clear whether and how far they represent methods for coping with the credit assignment problem in general. For example, one problem that has to be solved for each application individually, is the meaningful division of system parameter ranges into a small number of intervals.

3 Using domain knowledge

The credit assignment problem is prevailing in learning to control dynamic systems from a simulation model. Normally, only a sequence of control actions can be evaluated as a whole, to

meet the success criterion to a certain degree. In this section, we consider helpful domain knowledge which can be divided into two kinds:

- The set of admissible control strategies is constrained.
- Credit assignment is guided by domain knowledge on the causes for failure in controlling the system.

3.1 Admissible control strategies

In many domains, helpful restrictions of the set of control strategies can be provided with little effort. Often, there are many strategies which are obviously absurd. The idea is to exclude those strategies by characterizations which are easy to provide. For instance, in controlling a car on a curving road, the following monotonicity relation may be obvious: *If reducing speed is appropriate for taking a curve of radius r, then reducing speed is at least as appropriate for taking a curve of radius r', if $r' \leq r$.* In fact, it would be absurd attempting to take a curve of radius r', if it is already clear, that a curve of radius r is too narrow for the current speed. It is important to note, that such knowledge can be provided, even when neither the physics of a moving car are properly understood nor the successful driving behavior of an experienced driver.

In the extreme case, the constraints on the class of admissible control strategies are so strict that only a single control strategy is left. This amounts to the classical engineering approach to system control. Since that approach may require considerable effort, the emphasis lies on restrictions that can be provided with *little effort*.

To treat the mentioned kind of monotonicity restrictions more formally, we consider a control strategy for the bang-bang regime as represented by a monotonic boolean function on a range of integers $\{0, ..., k-1\}$, where k is the number of distinguished intervals which divide the numerical range of the system parameter. I.e. we can formalize this monotonicity constraint by defining the set of monotonic boolean functions F_{mon} on $\{0, ..., k-1\}$ as follows: $F_{mon} = \{f | f : \{0, ..., k-1\} \rightarrow \{0, 1\} \;\wedge\; \forall n_{\in \{0,...,k-2\}} \;\; (f(n) = 1 \rightarrow f(n+1) = 1)\}$. This simplifies the learning task significantly as is illustrated in figure 2. For controlling systems, described by more than one
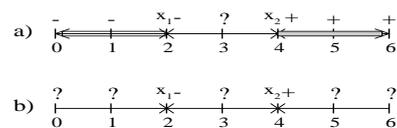

Figure 2. In case a) monotonicity is assumed: For any n, it holds $f(n) = 1 \rightarrow f(n+1) = 1$ and $f(n) = 0 \rightarrow f(n-1) = 0$. This contrasts the unconstrained case in b), where the two examples do not tell anything about other function values.

parameter, this simple monotonicity may not be applicable. E.g. not only the speed of the car may be important but also the slope of the road for determining whether a curve of radius r can be taken safely. For such purposes, multi-dimensional monotonicity constraints can be defined as follows:

Definition 1 *We denote by $F_{mon,m}$ the set of functions f on m parameters of the form $f : \{0, ..., k-1\}^m \rightarrow \{0, 1\}$ which are monotonic in all its parameters. For all functions*

$f \in F_{mon,m}$ *holds the following for all parameters ranging in their respective domain:*

$$f(n_1, ..., n_m) = 1 \rightarrow f(n_1 + 1, n_2, ..., n_m) = 1 \text{ and}$$
$$f(n_1, ..., n_m) = 1 \rightarrow f(n_1, n_2 + 1, ..., n_m) = 1 \text{ ... and ...}$$
$$f(n_1, ..., n_m) = 1 \rightarrow f(n_1, n_2, ..., n_m + 1) = 1$$

Intuitively speaking, this generalization amounts to saying: *Everything else being equal (e.g. the road slope and surface, condition of car's tires and suspenders etc.), the smaller the radius, the slower the car has to go.*
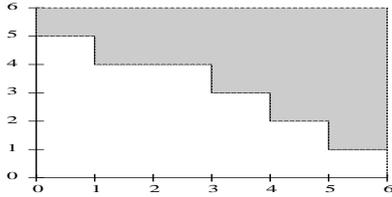

Figure 3. The monotonicity constraint in the 2-dimensional parameter space. The white area indicates the function value '0', while the shaded area represents '1'. The borderline is represented by a descending step function. The monotonicity is geometrically expressed by the fact that the shaded area extends uninterruptedly to the top and to the right.

3.2 Guiding credit assignment by causal domain knowledge

For guiding the credit (or blame) assignment, when a given control strategy turns out to be insufficient, obvious knowledge may be available, which accelerates the learning process significantly.

In the car example such domain knowledge may be as follows: *If the car is beginning to glide, then the speed was too high for the respective conditions of the car, the road, and the angle of the steering wheel.* Such knowledge may be very helpful in that it tells, that a change of the control strategy resulting in even higher speed in the same situation is certainly no improvement.

More generally speaking, such knowledge may relate a class of possible states of failure to a class of control actions which may have caused the failure. E.g. if the car is beginning to glide, some of the actions taken, failed to slow down the car. The idea is, to provide knowledge which says for a particular state of failure that at least one of the used control actions should be altered in a certain way. E.g. an applied force should be increased (decreased). In the simplest case, the boolean bang-bang regime, such information would say that some action '0' (e.g. *apply force to the left*) has to be altered to '1' (e.g. *apply force to the right*) or vice versa.

3.3 The reliability of control strategies

For many applications the issue of reliability of the learned control strategy will be even more important than the speed of learning: A control strategy learned from a simulation model of the system will have to rely on a set of initial sample states. The learned control strategy will have been proved to work for these initial sample states and possibly for some more initial

test states. However, if the system encounters a new initial state, it cannot be guaranteed that the learned control strategy will still be working. *By using sound domain knowledge the reliability can be improved significantly.* In the car example, the monotonicity constraint for instance, would guarantee that the speed is still reduced, even if the speed of the car is higher or the curve narrower than in any training example. A control strategy obtained from general purpose algorithms, could not guarantee that the action taken would not even further accelerate the car instead of reducing its speed !

4   A case study using a simple learning algorithm

Consider the multi-dimensional monotonicity constraint as expressed by definition 1. A very simple data structure for representing a control strategy is a large table containing one entry for each element of the Cartesian product of the system parameter ranges (n-dimensional boxes)[1] The monotonicity constraints are easily maintained by the following procedure, shown in figure 4. For each new entry into the table, all entries in all dimensions according to the monotonicity constraints are updated. I.e. nested loops for each parameter, either counting upwards or downwards, ensure the consistency of the table entries with the multi-dimensional monotonicity constraints.

The next problem is to determine, when a control action in a particular system state is assumed to be adequate. As discussed in section 3.2 domain knowledge may be used to find suitable alterations of a given control strategy to improve the performance. In the simplest case of only two possible control actions the domain knowledge could relate the kind of failure in controlling the system to one of the two control actions as follows: If failure of kind f occurred, then there is at least one control action a in the employed sequence of control actions that should be changed. This reduces the number of possible actions which might be responsible roughly from n to $\frac{n}{2}$.

An experimental case study

The new approach has been tested on the problem of learning to balance a pole. See figure 1. The system being controlled is described by the four system parameters x, $\dot{x}$, θ, and $\dot{\theta}$. The simulator used the equations as in Anderson [1]. The following domain knowledge has been used to support the learning process:

- **Causal knowledge:**

  - If the pole falls to the left side, then at least one control action which applied the force to accelerate the cart to the right has to be replaced by accelerating the cart to the left and vice versa.
  - If the cart exceeded the track to the left, then at least one control action which applied the force to accelerate the cart to the left has to be replaced by accelerating the cart to the right and vice versa.

- **Qualitative constraints:** If applying the force to the left is appropriate in a system state $S_0 = \langle x_0, \dot{x}_0, \theta_0, \dot{\theta}_0 \rangle$ *to bal-*

---

[1] This is basically the same data structure as in BOXES [5], although the number of distinguished intervals for each system parameter may be much larger.

```
begin
  if (action EQU positive) then
    for (i0=p0; i0 < MAX_I0; i0++)
      for (i1=p1; i1 < MAX_I1; i1++)
        for (i2=p2; i2 > 0; i2-)
          for (i3=p3-1; i3 > 0; i3-)
            if (control_table[i0][i1][i2][i3] EQU action) then break;
            control_table[i0][i1][i2][i3] = action;
          endfor
          if (control_table[i0][i1][i2][p3] EQU action) then break;
          if (i2 != p2) then control_table[i0][i1][i2][p3] = action;
        endfor
        if (control_table[i0][i1][p2][p3] EQU action) then break;
        if (i1 != p1) then control_table[i0][i1][p2][p3] = action;
      endfor
      if (control_table[i0][p1][p2][p3] EQU action) then break;
      if (i0 != p0) then control_table[i0][p1][p2][p3] = action;
    endfor
    control_table[p0][p1][p2][p3] = action;
end
```

Figure 4. A fraction of the algorithm which maintains the monotonicity constraint on the current strategy represented in a 4-dimensional array.

ance the pole, then it is also appropriate in each of the following four system states:

1. $S_1 = \langle x_0, \dot{x}_0, \theta_0, \dot{\theta}_1 \rangle$, where $\dot{\theta}_1 < \dot{\theta}_0$, i.e. where the pole is even faster swinging to the left.

2. $S_2 = \langle x_0, \dot{x}_0, \theta_2, \dot{\theta}_0 \rangle$, where $\theta_2 < \theta_0$, i.e. where the pole is even more inclined to the left.

3. $S_3 = \langle x_0, \dot{x}_3, \theta_0, \dot{\theta}_0 \rangle$, where $\dot{x}_3 < \dot{x}_0$, i.e. where the cart is moving even faster to the left. (and therefore reaches earlier the left boundary of the track. Thus, it is even more important to move to the left, since a delayed movement would potentially exceed the boundary of the track.)

4. $S_4 = \langle x_4, \dot{x}_0, \theta_0, \dot{\theta}_0 \rangle$, where $x_4 < x_0$, i.e. where the cart is even closer to the left boundary of the track. (Reason as above.)

If applying the force to the left is appropriate in a system state $S_0 = \langle x_0, \dot{x}_0, \theta_0, \dot{\theta}_0 \rangle$ *to keep the cart on the track*, then it is also appropriate in each of the following two system states:

1. $S_1 = \langle x_0, \dot{x}_1, \theta_0, \dot{\theta}_0 \rangle$, where $\dot{x}_1 > \dot{x}_0$, i.e. where the cart is moving even faster to the right.

2. $S_2 = \langle x_2, \dot{x}_0, \theta_0, \dot{\theta}_0 \rangle$, where $x_2 > x_0$, i.e. where the cart is even closer to the right track boundary.

Note that keeping the cart on the track and balancing the pole may require contradicting actions. Keeping the cart on the track has priority and overwrites the actions determined by the rules for balancing the pole.

The system was set up with a division of the numerical ranges into 25 equally large intervals for each of the four parameters. Reason for that was just to pick some larger number which ensures that a sufficiently fine division of the state space is provided. Initially the control action for all $25^4 = 390,625$ possible system states was set to 'undecided'. This was interpreted as no particular action, i.e. in alternating order as *apply force to the right* and *apply force to the left*. The system was initialized with a randomly chosen system state within the following intervals: $-1.2m \le x \le 1.2m$, $-0.5m/s \le \dot{x} \le 0.5m/s$,

$-6^\circ \le \theta \le 6^\circ$, and $-25^\circ/s \le \dot{\theta} \le 25^\circ/s$. The first simulation of the system resulted, as expected, in a pole fallen to either side. The last action a taken before the system indicated failure, which applied the force to the wrong side, was identified. a was tentatively altered including the control actions for all monotonically depending system states according to the given qualitative constraints.

After the tentative alterations, the system's simulation was continued applying the altered control action in the respective system state to test whether the alterations may suffice to avoid the same failure. Two cases are to be distinguished:

a) Pole fell down: The pole's falling is usually delayed by the respective action. However, for giving the pole a longer lasting 'push' the 'new action' was considered appropriate for the next encountered system state. Hence, the possibly necessary changes were performed including the actions for the monotonically depending system states. b) Cart exceeds track: This has invariably to be avoided; therefore respective control actions 'overwrite' control actions which try to balance the pole but cause the cart to exceed the track.

If the pole fell still to the same side, no alteration was implemented, instead the next action before the altered action which applied the force to the wrong side was tentatively altered including all monotonically dependent actions. This process was continued until the pole fell to the other side. This completed a single learning step. See figure 5.

1. Find the last inappropriate action before the system failure.
2. If the system state can be recovered, change respective action. Change strategy consistently with qualitative constraints. Return altered strategy.
3. Trace back the control sequence to find previous inappropriate action. Goto step 2.

Figure 5. The procedure for a single learning step.

After each single learning step, the system was simulated again with the same initial state. This learning and test cycle was repeated until the pole was successfully balanced for a prespecified number of $100,000$ simulation steps ($2,000$ seconds). On average, the system needed for the first initial state about 50–80 improvements using 25 intervals per parameter. Balancing the system starting from a further initial state, required a quickly decreasing number of further learning steps, as shown in figure 6. This indicates already the tendency of the system to develop a generally valid control strategy as opposed to control strategies which work only for the given initial system state. In fact, after balancing some 25 different initial states successfully, mostly no failures for new initial states were observed. Moreover, it was observed that the number of required learning steps is apparently directly depending on the number of divisions of the parameter value ranges (figure 6).

Direct comparisons with other approaches are difficult, since either no explicit or different background knowledge has been used. Moreover, the number of 'learning steps' counted is somewhat arbitrary, since a group of learning steps could be encapsulated into one 'bigger step', as it is done in the presented approach to a certain degree. E.g. approaches like BOXES as reported in [6] do not use explicit domain knowledge. However, they rely on a meaningful division of the pa-
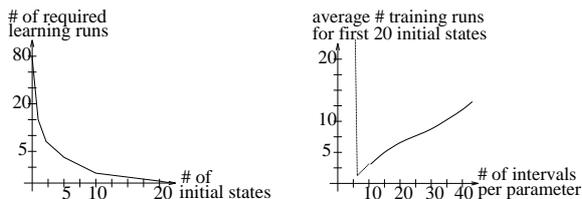
Figure 6. The left chart shows the number of unsuccessful balancing trials before a working control strategy is found. The right chart shows the dependency of the number of required learning steps on the 'coarseness' of the division of parameter value ranges. The 'infinite' indicates the case, where the division is too coarse and no successful control strategy can be found.

rameter ranges into a small number of intervals and use implicitly the 'knowledge' that a division of parameters into 3 respectively 6 intervals is sufficient. The performance reported in [6] is quite impressive (75 trials in average for obtaining a working control strategy for a single initial system state, though the reliability that the control strategy is working for new initial states as well, however, appeared somewhat 'fragile'. The experiments reported in [11] rely on the same 'knowledge' that a division into small number of intervals is sufficient. In addition, they used a qualitative model of the pole-and-cart system and a genetic algorithm for optimizing the control strategy. It may be mentioned that the reported number of simulation steps needed in order to obtain a viable control strategy using genetic algorithms is much higher than required in the presented approach. But this number may possibly reduce considerably by improving the use of the genetic algorithm. The results using Neural Networks reported in [1] show very slow learning behavior, which is not competitive in terms of learning speed. The work on Fuzzy Control in [4] seems to deal with pole balancing on an unbounded track which is a much easier task.

One may argue that the success of the presented approach in the experiments is due to the fact that the difficult part of the problem being learned by other approaches has been provided as domain knowledge. This, however, is the philosophy behind the presented approach: It is assumed that no universal learning algorithm will be found that suits all problems. Hence, any algorithm needs some tailoring to the problem at hand. To allow easy handling, the tailoring should rather use 'declarative bias' as opposed to bias that is 'hidden' in the program code or in the tuning of parameters which are hard to interpret.

5 Conclusions and Future Work

The general goal of this research is to develop learning techniques which allow to learn faster and more reliably by taking domain-specific background knowledge into account. Causal knowledge about the reasons for control failure has been used to significantly speed up learning. Qualitative knowledge, like monotonicity constraints has been exploited as well. The used knowledge improved not only learning speed but ensured more reliable learning results as well. This was not only confirmed by comparison to other approaches but also by learning experiments where parts of the mentioned domain knowledge were not used. It seems, that often a good 'naive' intuition about

the physics of a system suffices to provide valuable additional information. Using rudimentary causal domain knowledge in order to speed up the learning process appears widely applicable. Only in rare cases, causal links between a failure in controlling a system and a possibly responsible control action taken may be unavailable. However, if only approximate simulation models are available, it is not clear yet how far the presented approach can be employed.

Further kinds of 'easy to provide' background knowledge are currently explored. E.g. background knowledge may require that the speed of the cart is kept at a relatively low level etc. During the experiments with the pole balancing problem a number of unforeseen intricacies turned out which required special treatment. E.g. the fact that contradictory actions were required for keeping the cart on the track and for keeping the pole in balance - thus priority had to be determined. However, it is believed that a learning system could also be supportive in detecting such problems and that an engineer can easily overcome those problems by specifying stepwise more and more domain knowledge to be used for the learning process.

REFERENCES

[1] C. W. Anderson. Strategy learning with multilayer connectionist representations. In *Proceedings of the 4th International Conference on Machine Learning*, pages 103–114. Morgan Kaufmann, 1987.

[2] I. Bratko. Deriving qualitative control for dynamic systems. In K. Furukawa and S. Muggleton, editors, *Machine Intelligence and Inductive Learning*. Oxford University Press. (new series of Machine Intelligence), to appear.

[3] M. E. Connel and P. E. Utgoff. Learning to control a dynamic physical system. In *Proceedings AAAI 87*, pages 456–459, 1987.

[4] C. C. Lee. A self-learning rule-based controller employing approximate reasoning and neural net concepts. *International Journal of Intelligent Systems*, 6:71–93, 1991.

[5] D. Michie and R. A. Chambers. BOXES: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence*, pages 137–152. Edinburgh: Oliver and Boyd, 1968.

[6] C. Sammut. Recent progress with BOXES. In K. Furukawa and S. Muggleton, editors, *Machine Intelligence and Inductive Learning*. Oxford University Press. (new series of Machine Intelligence), to appear.

[7] C. Sammut. Experimental results from an evaluation of algorithms that learn to control dynamic systems. In *Proceedings of the 5th International Conference on Machine Learning*, pages 437–443. Morgan Kaufmann, 1988.

[8] C. Sammut and D. Michie. Controlling a "black box" simulation of a space craft. *AI Magazine*, (12):56–63, 1991.

[9] O. G. Selfridge, R. S. Sutton, and A. G. Barto. Training and tracking in robotics. In *Proceedings 9th IJCAI'85*. Morgan Kaufmann, 1985.

[10] T. Urbancic and I. Bratko. Learning to control dynamic systems. In D. Michie and D. Spiegelhalter, editors, *Machine Learning, Neural and Statistical Classification*. Ellis Horwood. to appear.

[11]   A. Varsek, T. Urbancic, and B. Filipic. Genetic algorithms in controller design and tuning. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-23(6):1330–1339.