"A Self Organizing Real-time Controller"

David W. Russell, PhD, FIEE
Penn State Great Valley
School of Graduate Professional Studies
Malvern, PA, USA
Brief Bio – David Russell

- Joined Penn State in 1985 and is Professor of EE at UP.
- Fellow of the Institute of Electrical Engineers.
- Americas Editor of the *Int J of Advanced Mfg Technology*.
- Previously VP in a System Integration Company.
- Became Academic Division Head for Engineering and Information Science in 1994.
- Chair-elect of the Mechatronics Forum in the UK.
- About 100 papers, presentations and invited talks in Real-time AI, Manufacturing Systems and Graduate Pedagogy.
- Made presentations in Australia, Austria, Belgium, Bermuda, Brazil, Canada, China, Dubai, England, France, Finland, Germany, Hungary, Ireland, Jamaica, Japan, Malaysia, Mexico, New Zealand, Singapore, Slovenia, Spain, Sweden, Switzerland, Turkey, and in the USA. 😊
Penn State Great Valley

- A Special Mission Campus near Philadelphia
- Three Academic Divisions
  - Education (C&I, SPLED, INSYS)
  - Management (MBA and M.Lead)
  - Engineering and Information Science
- Engineering Division offers:
  - Master of Software Engineering
  - Master of Engineering in Systems Engineering
  - Master of Science in Information Science
  - Graduate Minor in Bioinformatics
- ~1600 Part-time Graduate Students
The School of Graduate Professional Studies at Penn State Great Valley
Outline of Presentation

1. Poorly Defined Dynamic Systems
2. Learning – black box
3. A Learning Controller
4. A Real-World Application
5. Advisors
6. Self Organization
7. Concluding Comments
1. Poorly-defined Dynamic Systems

- Exist in many problem domains from aircraft engines to chemical processes.
- Some systems are even chaotic – swirling around state space endlessly but within defined boundaries (e.g. Lorenz attractors)
- Many mechanically unstable systems – perhaps the best known and most studied is the “trolley and pole” with slipping wheels and wear.
- Computationally difficult to model.
A Conventional Controller

Target Input + error = Plant P(s, v)

Control Signal = V

Read and Sample State Variables

Real-time control

Uses Model and a Control Schema (e.g. PID) to predict control value = v.

Digital (v) to Analog (V) Interface

Sample Rate Clock

System Output
A Learning Controller

Input $X$ + error -> Plant $P(s,v)$ -> Output

Control Signal $= V$

Digital to Analog Interface

Real-time control

Read and Sample State Variables

Clock or Timeout

Refresh

Learning Monitor

Decision Database

ICS, April 24, 2006
2. What is “learning” anyway?

How an infant learns to walk maybe:

- Evolution from Lying to Crawling to Walking
- Curiosity & Discovery of new Domains
- Positive and Negative “rewards”
  - Reinforcement of achievement
  - Warnings of danger
  - Conflict of wills
- Forgetting Failure and Over-success!
Common Learning Processes

- Recalling the Past (memory)
- Learning from Mistakes (mitigate risk)
- Handling Failure and Success
- Having Goals to Succeed
- Making and Implementing a Plan to Succeed
- Exploring Uncharted Areas
- Monitoring and Adjusting Progress
- Repetition
How a Computer can Learn to Play a Game

- Description of Game “board”
- Initial values
- Rules of Play
- Winning & losing daemon.
- Decision making
- Learning by experience

For example:
- let’s look at “noughts-and crosses” (tic-tac-toe 😊)
Example: Tic-tac-toe: Game board & Initial Values

- Board is 3 x 3
- Represented as an array of 9 “boxes” with index 1 thru 9
- Values are blank, “O”, or “X”
- Two players (O and X)
- Initially all boxes are blank
Rules of Winning and Losing

-“O” wins if any row, column or diagonal is all “O” s
-“X” wins if any row, column or diagonal is all “X” s
-Game is a DRAW if no winner and no boxes left vacant
Rules of Play

- New game starts with a clear board
- Random selection of who goes first
- Player can only insert own token ("O" or "X" – unchangeable unlike Othello)
- Must insert into an available space (unlike chess 😞 ... darn!)
- The first one to get three-in-a-row wins (unless you are playing "OXO")
Strategies?

- Try to get the middle space (cell 5)?
- Try to get a corner (cell = 1, 3, 7, 9)?
- Just react to opponent’s move.
- Avoid losing by not ever giving opponent an opportunity two ways to win.
- Realize that there are only 4 or 5 moves per player in each game, so no deep strategy needed ☺
Algorithm: The Payoff Matrix  
– a simple “BOXES” Algorithm

- Each player has counters assigned to each “box”
- Initially each counter is given, say, 50 points
- Play game
- The player reflects on the moves made in the game that has just ended and updates counter.
- Simple Statistics
  - If player WON! Reward each cell used by adding points
  - If Player LOST..Punish each cell used by subtracting points
  - If Game was a DRAW, do nothing
- Forgetfulness
  - Age statistics after every game (e.g. multiply by 0.99)
Playing by Statistics

- Play
- When it is the computer’s turn
  - Scan whole board
  - Note all available “boxes” that are empty
  - Read rank for each of these cells based on data
  - Make move based on the cell with best (e.g. highest) statistical rank
  - If a tie between cells, choose randomly from these cells
- Insert token in selected cell
Gaining Some Preliminary Knowledge

DECISION MATRIX

1. Use equal or random values at start of learning process

2. Have two humans play say 1000 games, updating table values after each game. e.g. add 5 for a win, subtract 3 for a loss and add 1 for a tie

3. Age data

Expert Knowledge Elicitation
## Decision Matrix after 1000 Games

<table>
<thead>
<tr>
<th>Box Rank</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

Box #7 is the best move.
Now Play the Game. Let the Human (O) start

Human chooses cell #1

So Computer deletes cell=1

<table>
<thead>
<tr>
<th>Box</th>
<th>Rank</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>208.19</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>25.97</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>200.01</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>40.28</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>99.50</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>20.43</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>210.32</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>30.53</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>135.29</td>
</tr>
</tbody>
</table>
Computer’s first move

The computer selects its best ranked cell #7 & deletes

<table>
<thead>
<tr>
<th>Box</th>
<th>Rank</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>208.19</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>25.97</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>200.01</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>40.28</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>99.50</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>20.43</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>210.32</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>30.53</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>135.29</td>
</tr>
</tbody>
</table>
### Game progresses

#### Human chooses cell #5

![Game board with a human choosing cell #5](image)

#### Data Table

<table>
<thead>
<tr>
<th>Box</th>
<th>Rank</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>208.19</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>25.97</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>200.01</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>40.28</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>99.50</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>20.43</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>210.32</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>30.53</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>135.29</td>
</tr>
</tbody>
</table>
**Game progresses**

Human chooses cell #5

- Computer selects its next best ranked cell #3 & deletes

<table>
<thead>
<tr>
<th>Box</th>
<th>Rank</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>208.19</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>25.97</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>200.01</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>40.28</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>99.50</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>20.43</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>210.32</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>30.53</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>135.29</td>
</tr>
</tbody>
</table>
Game progresses

Human chooses cell #9 and wins

```
  0  |   X  |
  O  |   0   |
     |   X   |
```

<table>
<thead>
<tr>
<th>Box</th>
<th>Rank</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>208.19</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>25.97</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>199.0</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>40.28</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>99.50</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>20.43</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>209.0</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>30.53</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>135.29</td>
</tr>
</tbody>
</table>
Statistical Weighting

- Age contributing cell data (e.g. multiply by .99)

- Aggressive on Winning
  - If a WIN, reward by 2%
  - If a LOSS, reduce by 5%
  - If a DRAW, reduce by 1%

- Aggressive on not losing
  - If a win, reward by 5%
  - If a LOSS, reduce by 10%
  - If a DRAW do nothing
Decision Matrix after 1001\textsuperscript{st} game

Aggressive on Winning – so punish losers (#3 and #7 by -5%)
# Decision Matrix after 1001st game

Aggressive on Winning (reward winners #1, #5, #9 by +2%)

<table>
<thead>
<tr>
<th>Box</th>
<th>Data</th>
<th>Box Rank</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>208.19</td>
<td>1</td>
<td>210.23</td>
</tr>
<tr>
<td>2</td>
<td>25.97</td>
<td>8</td>
<td>25.97</td>
</tr>
<tr>
<td>3</td>
<td>200.01</td>
<td>3</td>
<td>188.11</td>
</tr>
<tr>
<td>4</td>
<td>40.28</td>
<td>6</td>
<td>40.28</td>
</tr>
<tr>
<td>5</td>
<td>99.50</td>
<td>5</td>
<td>100.48</td>
</tr>
<tr>
<td>6</td>
<td>20.43</td>
<td>9</td>
<td>20.43</td>
</tr>
<tr>
<td>7</td>
<td>210.32</td>
<td>2</td>
<td>205.82</td>
</tr>
<tr>
<td>8</td>
<td>30.53</td>
<td>7</td>
<td>30.53</td>
</tr>
<tr>
<td>9</td>
<td>135.29</td>
<td>4</td>
<td>136.63</td>
</tr>
</tbody>
</table>

Calculations:
- $205.19 \times 0.99 \times 1.02 = 207.20$
- $99.50 \times 0.99 \times 1.02 = 100.48$
- $135.29 \times 0.99 \times 0.95 = 136.63$
### Decision Matrix after 1001 games

<table>
<thead>
<tr>
<th>Box Rank</th>
<th>Data</th>
<th>Box Rank</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>210.23</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>188.11</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>100.48</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>205.82</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>136.63</td>
<td></td>
</tr>
</tbody>
</table>

Box #1 is now the best move
3. A Learning Controller

- Tic-tac-toe algorithm had no real knowledge of the game – just moves based on statistical data
- Can a poorly-identified control system be handled similarly?
- What if the decision matrix did not look for a good cell to move to, but rather gave a control decision based on where in the state space the system was at that moment of time?
Representation of Knowledge in a Control System

- Binary – 0/1, on/off (i.e. a switch)
- Supplemental control e.g. +1% to a control coefficient
- Fuzzy e.g. start braking – (ABS)
- Release when wheels lock (ABS)
- Statistical inference & weights (NN)
Example: The Trolley & Pole

Problem:
To stop pole from falling by moving trolley. The mathematical non trivial solution is not to balance pole vertically, but to swing the cart in a pendular motion using a bang-bang motor to inject energy.

Bang-bang Control:
Full speed LEFT or RIGHT
BOXES Approach to Trolley & Pole

- Bang-bang motor controller
- Instead of game board, use an integer “box” identifier of the system’s location in the state space
- Instead of placing an “O” or “X” on a game board, the enforced action is to re-ad the motor direction (Left or Right)
- Instead of one data count per box, keep two values for each decision. i.e. “L” or “R” decision strength and “L” or “R” cell use data stores
System Identification using State Variables

**State Variables**
- Trolley position (\(x\))
- Trolley speed (velocity = \(dx/dt\))
- Pole Angle (\(\theta\))
- Pole speed (angular velocity = \(d\theta/dt\))

**Measuring the Skill of the Controller**
- \(T_f\) – Time of Failure when the pole falls or the trolley (cart) reaches the end of the track
- Goal is to be able to play the game and make \(T_f\) as long as possible
Mapping the Problem:
a. Divide continuous state space into regions or boxes.

- Normalize \( \{x, dx/dt, \theta, d\theta/dt\} \) e.g. \( \eta = x/X_{\text{max}} \)
- Divide domain space into “n” boundaries
- For example, divide trolley position into 5 regions

<table>
<thead>
<tr>
<th>Fail = 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Fail = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_0 = 0 )</td>
<td>( B_1 )</td>
<td>( B_2 )</td>
<td>( B_3 )</td>
<td>( B_4 )</td>
<td>( B_5 = 1 )</td>
<td></td>
</tr>
</tbody>
</table>

And so on for normalized values of \( dx/dt, \theta, \) and \( d\theta/dt \)
Mapping the Problem:
b. Divide domain into zones

- Normalize \( \{x, dx/dt, \theta, d\theta/dt\} \) e.g. \( \xi = x/X_{\text{max}} \)
- Divide domain space into “n” boundaries
- For example, divide trolley position into 5 regions

<table>
<thead>
<tr>
<th>Fail = 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Fail = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_0=0 )</td>
<td>( B_1 )</td>
<td>( B_2 )</td>
<td>( B_3 )</td>
<td>( B_4 )</td>
<td>( B_5=1 )</td>
<td></td>
</tr>
</tbody>
</table>

A-priori knowledge is only the domain size and boundary positions
Mapping the Problem:
c. Determine index box values for each variable

Sample variables at some rate.
Decide where **each normalized variable** lies in its state space.

For example: Normalized Trolley position \( \theta = 0.26 \) at time = t

And so on assign \( K_2, K_3 \) and \( K_4 \) ..... for \( \frac{dx}{dt}, \theta, \) and \( \frac{d\theta}{dt} \)
Mapping the Problem:
c. Determine index box values for each variable

Sample variables at some rate.
Decide where each normalized variable lies in its state space

For example: Normalized Trolley position \( \dot{\phi} = 0.26 \) at time = \( t \)

\[
\begin{array}{cccccc}
\text{Fail} & 0 & 1 & 2 & 3 & 4 & 5 & \text{Fail} = 0 \\
\hline
\end{array}
\]

\[\dot{\phi} = 0.26\]

\( \text{In region two} \)

e.g. if \( \dot{\phi} = 0.26 \) at some time = \( t \), then \( K_1 = 2 \)

And so on assign, \( K_2, K_3 \) and \( K_4 \) ..... for \( dx/dt, \theta, \) and \( d\theta/dt \)
Mapping the Problem:
d. Identify system with one INTEGER value

- At any (real) time all values of $K_1$, $K_2$, $K_3$ and $K_4$ are non zero then the trolley and pole are within control limits.

- If any $K$ value becomes a “fail” = 0, then the “game” (a.k.a. “run”) is over and system moves to learning mode.

- If no fail – at some sample time – the system can compute one unique integer “$m$” that describes the state space region that the system is in.


- Obviously the coefficients $C_1$, $C_2$, $C_3$ and $C_4$ depend upon the number of zones in each variable.
Mapping the Problem:
e. Simple Integer Arithmetic

If the above configuration is adopted, there are a total of 225 possible state integers that describe the complete state space.

To compute a unique m: C1 = 1, C2 = 5, C3 = 15 and C4 = 75

The algorithm for m is therefore:

\[ m\{K1,K2,K3,K4\} = C_1.K1 + C_2.(K2-1) + C_3.(K3-1) + C_4.(K4-1) \]

so \( m\{1,1,1,1\} = 1 \) and \( m\{5,3,5,3\} = 225 \)
Read the Signature Table

- The Signature Table “Φ” is simply an array initially loaded with random decisions as -1 and +1 values.

- Game rule is that the signature value represents LEFT and RIGHT on motor direction switch in state “box.”

- Using state integer \( m \) as the pointer.

- Select move “u” i.e. switch trolley motor LEFT or RIGHT (“-1” or “+1”) based on value in the knowledge base.

- So the control value, \( u = \Phi (m) = \text{“L” or “R”} \)
Save Data

- Run Time Data.

If the system moves to a new state (m-value), at some real time = t from run start, and the current value of decision in \( \Phi(m) \) is “L” then:
  - Update LEFT cell timer
    \[
    L_{\text{time}}(m) = L_{\text{time}}(m) + t
    \]
  - Increment LEFT counter
    \[
    L_{\text{count}}(m) = L_{\text{count}}(m) + 1,
    \]

- If \( \Phi(m) = “R” \) update values of \( R_{\text{time}}(m) \) and \( r_{\text{count}}(m) \)
Unstable Dynamic System with Learning Controller – in CONTROL mode

REAL SYSTEM UNDER CONTROL

Motor Switched LEFT or RIGHT

Control Decision is “L” or “R”

Read State Variables

Identify “box” = m, Save data

Table V = \( \Phi (m) \)

Read Signature
Unstable Dynamic System with Learning Controller – after FAILURE

1. Move trolley to end stop

REAL SYSTEM – STOPS on FAIL

Motor Switched LEFT or RIGHT

2. Update Signature Table and authorize system restart

3. Auto Restart
Update the Knowledge Base

1. **Update Global System Statistics** after each failure or timeout:

   If the system failed at time $T_f$:
   
   **Global Performance**
   
   $\text{global\_life} = \text{global\_life} \times dk + T_f$
   $\text{global\_use} = \text{global\_use} \times dk + 1$
   $\text{merit} = \frac{\text{global\_life}}{\text{global\_use}} = \text{seconds}$
   
   Now set
   
   $\text{dla} = \text{desired\_level\_of\_achievement} = c_0 + c_1 \times \text{merit}$
   $\text{K} = \text{impatience} = \text{rate of learning}$
   $\Phi(m) = \text{signature table} – \text{contains “L” and “R” values only}$
Update each state ("box")

2. **Individual “Box” Update**

1. “Age” all box history (1 .. m .. MAX), dk = 0.99 etc.

   \[
   \begin{align*}
   L_{\text{life}}(m) &= L_{\text{life}}(m) \times dk \quad / \text{time in box “m” with L decision} \\
   L_{\text{use}}(m) &= L_{\text{use}}(m) \times dk \quad / \text{number of entries while L decision} \\
   R_{\text{life}}(m) &= R_{\text{life}}(m) \times dk \quad / \text{time in box “m” with R decision} \\
   R_{\text{use}}(m) &= R_{\text{use}}(m) \times dk \quad / \text{number of entries while R decision}
   \end{align*}
   \]

2. Update cell with data from last run

   if \( \Phi(m) \) is currently “L” then

   \[
   \begin{align*}
   L_{\text{life}}(m) &= L_{\text{life}}(m) + L_{\text{count}}(m) \times T_f - L_{\text{time}}(m) \\
   L_{\text{use}}(m) &= L_{\text{use}}(m) + L_{\text{count}}(m) \quad \text{etc.}
   \end{align*}
   \]
Evaluate decision & update if necessary—still for each box

3. Compute L and R strength of each cell

   SL(m) = (L_life(m) + K. dla)/(L_use(m)+K)
   SR(m) = (R_life(m) + K. dla)/(R_use(m)+K)

4. Update as necessary – for all “m” values

   if SL(m) > SR(m) then Φ(m)="L"
   if SR(m) > SL(m) then Φ(m)="R"

5. Clear all run time timers and counters, for next run

   L_count (m), L_use (m), R_count(m) etc.
Learning Algorithm Summary

☐ Keep Simple Statistics
   - For each box, keep two data sets for each decision (0/1) - number of entries into that box in any run and a history of the lifetime of system in any box

☐ Forgetfulness
   - Age statistics after every game (e.g. multiply by 0.99)

☐ Update Decision Matrix
   - Signature Table reflects skill, just like in the tic-tac-toe decision matrix
Knowledge

- The primary objective of the automaton is to accrue knowledge in the Matrix to achieve some desired performance goal.

- A second objective is to seek optimality of control – which is difficult in bang-bang systems (for example: requiring a Pontriagin analysis to locate the switching surface).

- The third concern is to examine the learning mechanism to seek computational improvement and the repeated avoidance of learning barriers.
4. A Real-world Application

Problem:
To stop the pole from falling by moving the trolley.

Control:
LEFT/RIGHT
No Neutral

Fixed Length Track

Motor

Stops
System Constraints and Observations

- The system must not “chatter” across a boundary
  
  Fail = 0 | 1 | 2 | 3 | 4 | 5 | Fail =0

- Variables are easy to measure for min/max estimation and normalization

- Inverted pendular motion – does precess off the track

- System is to be controlled without knowledge of the dynamics or model

- Difficult for a human to perform with a joystick!
A real system was built in Liverpool and much data generated, and a Pontriagin Analysis was performed on the un-linearized model.

A comparison of the Pontriagin surface and the decision matrix showed some agreement with a clearly visible complex optimal switching surface.

The agreement varied as the boundary locations in the state space we altered; Some authors use genetic algorithms to search out boundaries before running BOXES.

There is debate as to the need for more or fewer boundaries.
Actual System

Plate 1

Trolley showing stops
Real Data from the System

![Graph showing Trolley & Pole Learning](image)

- **Merit**
  - BOXES
  - Human
  - Random

- **Runs (in 100's)**

---

ICS, April 24, 2006
Run system over again

Merit ~ Avg. run duration in seconds

Graphics Stylized for illustration only – not actual data
5. Advisors

- Use $\Phi(m \pm 1)$ values from neighboring “boxes” in two (weighted) decision making options.

  i. Each cell and its advisors vote by decision value only. Tie breaker is original cell value.
     
     \[
     \text{if } \sum \text{"L"} > \sum \text{"R"} \text{ then } u = \text{"L"} \text{ else } u = \text{"R"}
     \]

  ii. Vote by aggregating cell-strengths – maybe naïve neighbors. Tie breaker is original cell value
     
     \[
     \text{if } \sum_{SL(i)} > \sum_{SR(i)} \text{ then } u = \text{"L"} \text{ else } u = \text{"R"}
     \]

- In either case, the advisors are rewarded or penalized based on their contribution to the decision – this is extremely computationally intensive.
Results with Advisors

TROLLEY & POLE SIMULATION

Runs (in 500's)

Merit

BOXES with Advisors

ICS, April 24, 2006 56
Results with Advisors

Problem is that in the initial stages, the “blind” are leading the “blind” – therefore system delays advising until the signature table is mature
Results with Delayed Advisors

TROLLEY & POLE SIMULATION

Runs (in 500's)

Merit

with Delayed Advisors

BOXES with Advisors
6. Self Organization

- It would seem the boundary values that distinguish the state variable cells is extremely important.

<table>
<thead>
<tr>
<th>Fail = 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Fail = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boundary</td>
<td>0</td>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
<td>B5=1</td>
</tr>
</tbody>
</table>

- At the end of many learning runs, could the data/knowledge suggest a better division of the state space?

- Must not alter number of states, but rather seek to minimize the size of a set of “un-trainable” states – which may then indicate the switching surfaces!
On line Evolution

As the matrix is being updated, look for self similar cells that are advisors (i.e. adjacent) to each cell in turn. Of course, end boundaries \{0, 1\} cannot be moved.

e.g. Two adjacent moveable boundaries \(B_i\) and \(B_{i+1}\) with Left and Right strengths \(SL_i, SL_{i+1}\) and \(SR_i, SR_{i+1}\) and \(\delta B_i\) is a function of \(\{\Phi(i), SL_i, SR_i, \Phi(i+1), SL_{i+1}, SR_{i+1}\}\)

For example:

if \(\Phi(i) = \text{"R"}\) and \(\Phi(i+1) = \text{"L"}\) then \(\delta B_i = 0.05 \cdot (SR_i - SL_{i+1}) / SR_i\)

\[B_i = B_i + \delta B_i,\]

else ....

Example:

If a STRONG “R” is next to a WEAK L, incrementally move the shared cell boundary to increase the size of the strong cell at the expense of the weaker one. Move purely based on and proportional to the cell strengths. Cell strengths measure aggression and resistance.
Self Organization – starting position

e.g. X – cart position - divided into 5 states = k1 \{1,5\}

<table>
<thead>
<tr>
<th>Boundaries</th>
<th>0</th>
<th>0.3</th>
<th>0.5</th>
<th>0.6</th>
<th>0.75</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>Fail</td>
</tr>
<tr>
<td>( \Phi(m) ) value</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
</tbody>
</table>

R or L mean strong cells, l or r mean weak cells
Self Organization

So, Edge boundary 3 from 0.5 to 0.52 and reduce boundary 4 from 0.6 to 0.59

<table>
<thead>
<tr>
<th>M value</th>
<th>0.3</th>
<th>0.5</th>
<th>0.6</th>
<th>0.75</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Fail</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

ICS, April 24, 2006 62
Self Organization - continued

Further edge boundary 3 from 0.52 to 0.55 and further reduce boundary 4 from 0.6 to 0.555.
Evolution

- Eventually, the strong cells will decrease the size of weak ones of a similar or opposing value until they become very small.

- Eventually, STRONG LEFT boxes and the STRONG RIGHT boxes will leave small untrainable cell wedges between them.

- Is this the location of the switching surface?

- Real-time, in-line DATA MINING?
7. Concluding Comments

- A Simple BOXES Method can perform black box control of almost any bang-bang system – some chaotic systems have also been controlled using the algorithm.

- Algorithm can be adjusted to include experience of near neighbors as advisors – but make them accountable.

- Evolutionary proposal will allow control and identification of the switching surface of difficult, maybe non-defined, systems (e.g. biomass etc.).

- Evolutionary approach does not force the system to be restarted because the number of cells remains the same.
Questions?

DATE SAVER
Mechatronics MX 2006, June 19-21, 2006 at Penn State Great Valley
www.gv.psu.edu/mx2006