Usability Analysis with Markov Models

Specifications and *Mathematica* details

This *Mathematica* notebook should be read in conjunction with the full paper. As well as a few calculations used in the paper, this notebook contains the *Mathematica* appendix from the paper and code to draw the graphs used in the paper. If you have *Mathematica* all of this notebook can be run and tried out.

Example Mathematica code

This Appendix gives example *Mathematica* code to show how easily Markov analysis is to do. In particular, by making minor changes to the definitions here, other devices can easily be analysed in the same way: supporting our claim for the approach being reusable.

An example device definition

The *Mathematica* code shown in this Appendix is the complete code to calculate the numbers quoted in the body of the paper for the microwave cooker, including the conversion of Sharp's definition of the cooker to a probability transition matrix.We will also show how easily its user interface can be simulated.

The code starts by loading the standard *Mathematica* package for combinatorics (to load a shortest path function, which we will need for calculating the designer's optimal transition matrix), and a utility routine.

```
<< DiscreteMath`Combinatorica`
IndexOf[vector_, e_] := Position[vector, e][[1, 1]];
```

Here is Jonathan Sharp's definition of the device.

```
device =
 ⎛ "clock"   "clock"   "clock"   "clock"   "clock"   "clock" ⎞
 ⎜   qd        qd        qd        qd        qd        qd    ⎟
 ⎜"timer1"  "timer1"  "timer2"  "timer1"  "timer2"  "timer1"⎟ /.
 ⎜ "clock"   "clock"   "clock"   "clock"   "clock"   "clock" ⎟
 ⎝ "clock"     qd     "power1"  "power2"  "power1"  "power1"⎠
  qd Æ "quickDefrost";

stateNames = {"clock", "quickDefrost",
   "timer1", "timer2", "power1", "power2"};
buttonNames = {"clock", "quickDefrost", "time", "clear", "power"};
```

Display the table as in the paper:

```
neatTable["Jonathan Sharp's Microwave cooker",
 device, stateNames, buttonNames]
```

Buttons	—States—					
	clock	**quickDefrost**	**timer1**	**timer2**	**power1**	**power2**
clock	clock	clock	clock	clock	clock	clock
quickDefrost	quickDefrost	quickDefrost	quickDefrost	quickDefrost	quickDefrost	quickDefrost
time	timer1	timer1	timer2	timer1	timer2	timer1
clear	clock	clock	clock	clock	clock	clock
power	clock	quickDefrost	power1	power2	power1	power1

JonathanSharp'sMicrowavecooker

We now work out from the device specification how many states and buttons there are:

```
numberOfStates = Length û stateNames;
numberOfButtons = Length û buttonNames;
```

Example analysis and graph drawing

The first analysis discussed in the paper was for tasks getting from state **power1** to **power2.**

```
start = IndexOf[stateNames, "power1"];
goal = IndexOf[stateNames, "power2"];
```

The random user matrix (called *P* in the paper) is directly calculated from **device**; all button presses are treated as equiprobable — by adding 1/numberOfButtons to elements of the matrix.

```
randomUser = Table[0, {numberOfStates}, {numberOfStates}];
Do[randomUser⟦i, IndexOf[stateNames, device⟦b, i⟧]⟧ +=
  1 / numberOfButtons, {b, numberOfButtons}, {i, numberOfStates}];
```

The designer's matrix is based on the optimal route from the start to the goal states. Notice how the random user matrix (which, conveniently, has non-zero elements precisely where there are transitions) is converted to a **Graph** type to find shortest paths. In the paper, this matrix was called *D*.

(The definition depends on the choice of start and goal states.)

```
designer = Table[0, {numberOfStates}, {numberOfStates}];
Do[Module[{p = ShortestPath[ Graph[randomUser, {}], i, goal]},
   designer[[i, If[ Length[p] > 1, p[[2]], i]]] = 1],
  {i, numberOfStates}];
```

Define the vector of ones of appropriate size, the identity matrix of suitable dimensions, and a utility function for submatrices:

```
ZeroRowCol[matrix_, rc_] :=
  Table[If[ i ä rc || j ä rc, 0, matrix[[i, j]]],
   {i, Length û matrix}, {j, Length û matrix}];
```

We give the definition of the mean first passage time in its most direct form.

```
meanFirstPassage[_, start_, start_] := 0;

meanFirstPassage[matrix_, start_, goal_] :=
  Module[{One = Table[1, {Length û matrix}],
    Id = IdentityMatrix[Length û matrix]},
   (Inverse[Id - ZeroRowCol[matrix, goal]] . One)[[start]]
  ];
```

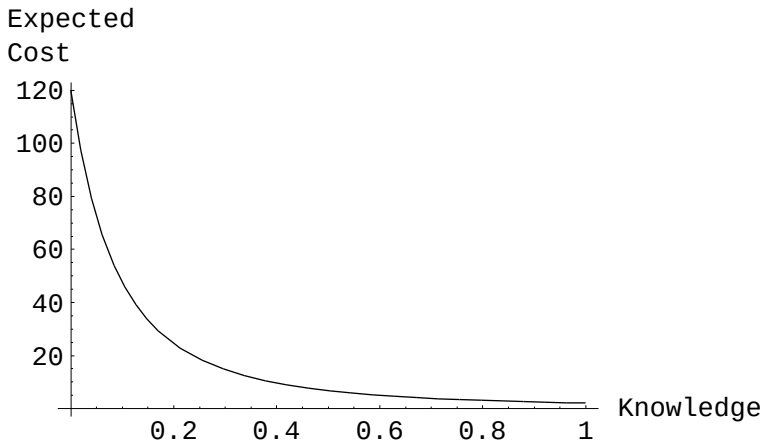
The function may be used thus:

```
meanFirstPassage[randomUser, start, goal]
```

```
120
```

The expected time to get from the start state (**power1**) to the goal state (**power2**) is 120 button presses. The knowledge/usability graph can easily be plotted:

```
Plot[meanFirstPassage[k designer + (1 - k) randomUser, start,
   goal], {k, 0, 1}, AxesLabel Æ {"Knowledge", "Expected\nCost"}];
```



Simulating the user interface

To simulate the device, we use a global variable to keep track of the state of the device as buttons are pressed. For the sake of argument, we start the device in state clock.

```
state = "clock";
```

When a button on the simulation is pressed, *Mathematica* will arrange for the function **press** to be called, with the button as a parameter. This function uses the device specification to determine the next state. The next few lines of the function locate the device's display cell in the current *Mathematica* notebook (*Mathematica* can have several notebooks — that is, windows — running together, which is why the variable *nb* is required); the text displayed in that cell is selected and replaced with the name of the new state.

The simple definition of press given below shows the name of the current state in the display. It is possible to display any image in the display, not just plain text, but to do so would take us beyond the scope of this example.

```
press[theButton_] :=
 Module[{nb = ButtonNotebook[]}, state = device[[
    IndexOf[buttonNames, theButton], IndexOf[stateNames, state]]];
  NotebookFind[ nb, "display", All, CellTags];
  SelectionMove[nb, All, CellContents];
  NotebookWrite[nb, Cell[state]];
 ]
```

Mathematica cells can be displayed in several ways. The following code is the definition of a row of buttons to control the device. To use the buttons, *Mathematica* would change the display mode of the cell, and show a row of actual buttons as shown in a Figure in the paper.

clock | quickDefrost | time | clear | power

clock

In the code given above, the button names and their actions were `hard coded.' However, *Mathematica* can generate button definitions automatically from a device specification, such as the one given earlier in this Appendix. Thus, the user interface itself can be implicitly defined by the *same* device specification. This, of course, is very important to make the analysis — both mathematical and empirical — use consistent specifications; they can be changed easily and only in one place.

```
CellPrint[
  Cell[BoxData[RowBox[Map[ButtonBox[#, ButtonFunction ¶ press[#],
        ButtonEvaluator Æ Automatic] &,
      buttonNames]]], Active Æ True]];
```

clock | quickDefrost | time | clear | power

The device's simulated display is a simple cell, with an appropriate name so that the **press** function can locate it. In its 'raw' form it is just `Cell["", CellTags->"display"]`: normally it would be displayed as shown in a Figure in the paper.

The definitions of the buttons and device display given in this Appendix have been simplified. If desired, *Mathematica* allows them to contain further 'typographical' details, such as their font, size and colour. For example, the device's display could easily be made to look more like a typical LED display of green text on a black background, by writing `FontFamily->"Courier", FontColor->RGBColor[0,1,0], Background->GrayLevel[0]` as options in the definition of the cell (see below). For simplicity, we omitted such details from the definitions given here.

clock

Checking the paper's data

Some checks, so we can check against the body of the paper:

```
randomUser // TraditionalForm
```

$$\begin{pmatrix} \frac{3}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 \\ \frac{2}{5} & \frac{2}{5} & \frac{1}{5} & 0 & 0 & 0 \\ \frac{2}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} \\ \frac{2}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} \end{pmatrix}$$

```
designer // TraditionalForm
```

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The combination lock

```
lock = ( 1 - p  p
         0      1 );
```

```
meanFirstPassage[lock, 1, 2]
```

$$\frac{1}{p}$$

```
meanFirstPassage[(1 - k) lock + k ( 0  1
                                    0  1 ), 1, 2]
```

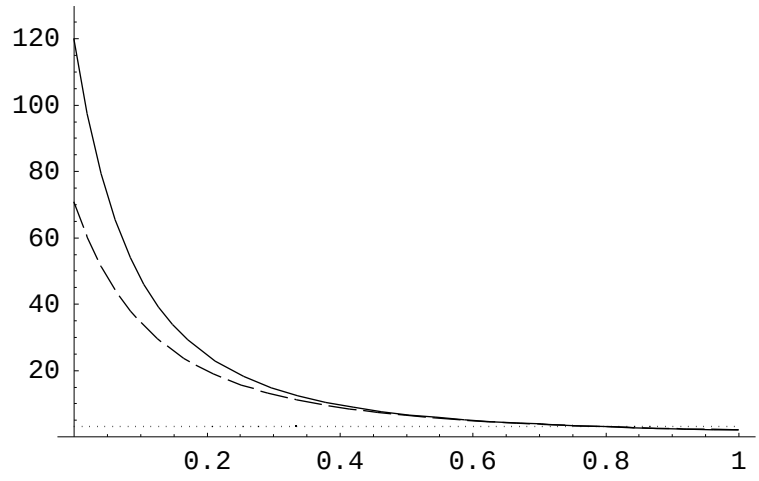
$$\frac{1}{k + p - k\,p}$$

Generating Figures

```
normalise[m_] :=
  Table[m[[i, j]] / Plus @@ m[[i]], {i, Length[m]}, {j, Length[m]}];
zeroDiagonal[m_] := Table[If[i ≠ j, 0, m[[i, j]]],
   {i, Length[m]}, {j, Length[m]}];
LEDUser = normalise @ zeroDiagonal @ randomUser;
LEDUser // TraditionalForm
```

$$\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ \frac{2}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \end{pmatrix}$$

```
sharpGraph = Plot[
   {3, meanFirstPassage[k designer + (1 - k) LEDUser, start, goal],
    meanFirstPassage[k designer + (1 - k) randomUser,
     start, goal]}, {k, 0, 1}, PlotRange → {0, 130},
   AxesOrigin → {0, 0}, TextStyle → {FontSize → 10},
   PlotStyle → {Dashing[{.001, .01}],
     Dashing[{.04, .01}], Dashing[{1, 0}]}];
```



We save the graph as an epsf file so that we can put it in the paper.

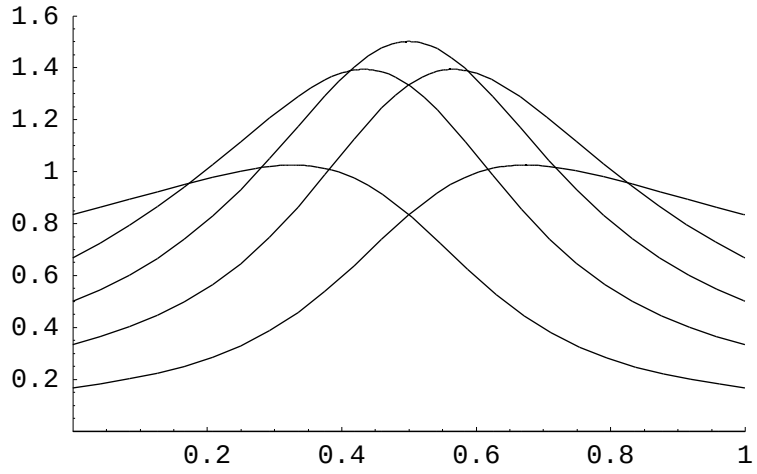
```
SetDirectory["Macintosh HD:Markovfiles"]
```

```
Macintosh HD:Markovfiles
```

```
Export["graph.epsf", sharpGraph, ImageSize → 72 * 4.5];
```

$$\text{ring} = \begin{pmatrix} 0 & p & 0 & 0 & 0 & 1-p \\ 1-p & 0 & p & 0 & 0 & 0 \\ 0 & 1-p & 0 & p & 0 & 0 \\ 0 & 0 & 1-p & 0 & p & 0 \\ 0 & 0 & 0 & 1-p & 0 & p \\ p & 0 & 0 & 0 & 1-p & 0 \end{pmatrix};$$

```
ringGraph =
  Plot[Evaluate@Table[meanFirstPassage[ring, 1, r] / 6, {r, 1, 6}],
   {p, 0, 1}, PlotRange -> {{0, 1}, {0, 1.6}},
   TextStyle → {FontSize → 10}];
```




```
Export["ring.epsf", ringGraph, ImageSize → 72 * 4.5];
```

The Nokia 2110 is a bit harder!

```
nokiaMenu["Standby"] =
  {"Recent calls", "Messages", "Call divert", "Phone settings",
   "Security options", "Duration and cost", "Network selection",
   "Memory functions", "Personal reminders", "In-call options",
   "FAX or data call", "Ringing options", "Number editor"};


nokiaMenu["Recent calls"] = {"Dialled calls",
   "Received calls", "Missed calls", "Erase all recent calls"};


nokiaMenu["Messages"] =
  {"Listen to voice messages", "Read messages", "Write messages",
   "Show deliver reports", "Message settings"};


nokiaMenu["Message settings"] = {"Message centre number",
   "Message sent as", "Accept reply costs", "Delivery reports",
   "Message validity", "Set mailbox number"};
```

```
nokiaMenu["Call divert"] =
  {"Divert all calls", "Divert when busy",
   "Divert when not answered", "Divert if not reachable",
   "Divert all data calls", "Cancel all diverts"};


nokiaMenu["Phone settings"] = {"Lights", "Ringing volume",
   "Ringing tone", "Keypad tones", "Warning tones",
   "Automatic redial", "One touch dialling", "Automatic answer",
   "Cell info display", "Own number sending", "Call waiting",
   "Restore factory settings", "Menu list", "Language"};


nokiaMenu["Security options"] = {"PIN code request",
   "Security level", "Call barring", "View fixed dialling",
   "Change access codes", "Closed user group"};
    nokiaMenu["Call barring"] =
  {"Outgoing calls", "International calls",
   "Int except to home country", "Incoming calls",
   "Incoming calls if abroad", "Cancel all barrings"};


nokiaMenu["Change access codes"] =
  {"Change security code", "Change PIN code",
   "Change PIN2 code", "Change barring password"};


nokiaMenu["Duration and cost"] = {"Call duration",
   "Call costs", "Call costs limit", "Show costs in"};


nokiaMenu["Memory functions"] =
  {"Memory selection", "Memory status", "Copy between memories",
   "Memory erasing options", "Show own number"};


nokiaMenu["Personal reminders"] =
  {"Welcome note", "Countdown timer"};


nokiaMenu[_] = {};
```

```
makeGadget[fcn_, root_, Quitprob_] :=
  Module[{defList = {}, vcounter = 0, gcounter = 0,
    transition, domenu}, transition[from_, button_, to_] :=
    AppendTo[defList, {from, to,
      If[button ä "Quit", Quitprob, (1 - Quitprob) / 3]
     }];
   domenu[s_] :=
    Module[{i, m = fcn[s], view, goal},
     view = "View options " <> ToString[++vcounter];
     transition[s, "Select", view]; transition[view, "Up",
      m[[Length[m]]]]; transition[view, "Down", m[[1]]];
     transition[view, "Quit", s ];
     For[i = 1, i £ Length û m, i++,
      transition[m[[i]], "Down", m[[If[i ä Length[m], 1, i + 1]]]];
      transition[m[[i]], "Up", m[[If[i ä 1, Length[m], i - 1]]]];
      transition[m[[i]], "Quit", s];
      If[ fcn[m[[i]] ] π {},
       domenu[m[[i]]],
       (* has something that can be selected *)
       goal = "Function " <> ToString[++gcounter];
       transition[m[[i]], "Select", goal];
       (* we assume any operation gets out of a function *)
       Map[transition[goal, #, m[[i]] ] &,
        {"Up", "Down", "Quit", "Select"}];
      ]
     ]
    ];
   domenu[root];
   Print[gcounter, " functions"];
   Print[vcounter, " View Options items"];
   Return[defList]
  ];

symbolicTransitions = makeGadget[nokiaMenu, "Standby", p];
```

```
64 functions

12 View Options items
```

```
vocabulary = Union û Flatten[
    symbolicTransitions /. {from_, to_, prob_} Æ {from, to}];

WhichNat[s_] := Position[vocabulary, s][[1, 1]];

ToNats[s_] := s /. {a_, b_, c_} ¶ {WhichNat[a], WhichNat[b], c};

FromNat[n_] := Part[vocabulary, n];

standbyN = WhichNat["Standby"];
goal = WhichNat["Incoming calls"];
numericTransitions = ToNats[symbolicTransitions];
```

```
numberOfStates =
 Max @@ Flatten[numericTransitions /. {from_, to_, _} Æ {from, to}]
```

```
152
```

```
g = Table[0, {numberOfStates}, {numberOfStates}];
Scan[(g[[#[[1]], #[[2]]]] += #[[3]]) &, numericTransitions]
(* some buttons do nothing in some states,
 so set diagonal of matrix, so each row adds to 1 *)
Do[g[[i, i]] = 1 - Plus @@ g[[i]], {i, numberOfStates}];
```

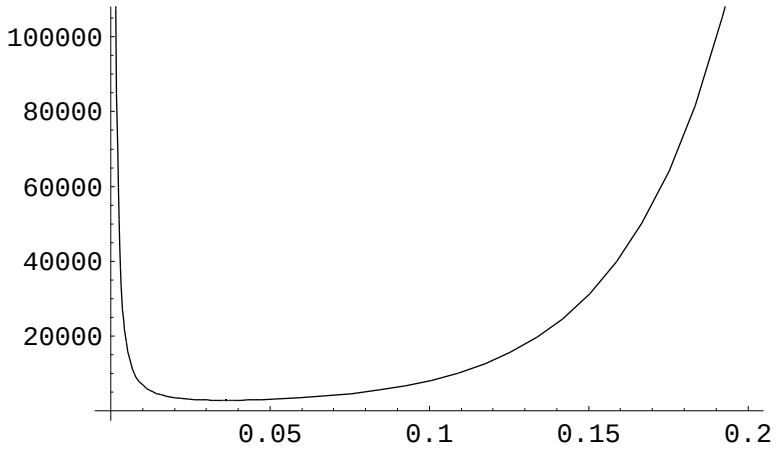
```
quitNokia[quitProb_] :=
  meanFirstPassage[g /. p Æ quitProb, standbyN, goal];
```

What is the expected time when the quit button is pressed with probability 0.25?

```
quitNokia[0.25]
```

```
602235.
```

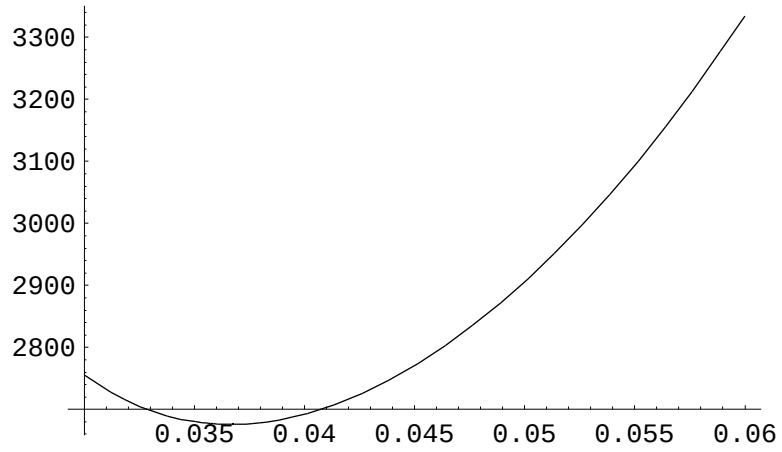
```
nokiaGraph = Plot[quitNokia[p], {p, 0.001, 0.2}];
```




```
Export["nokia.epsf", nokiaGraph, ImageSize Æ 72 * 4.5];
```

Find the minimum value:

```
Plot[quitNokia[p], {p, 0.03, 0.06}];
```




```
FindMinimum[quitNokia[p], {p, {0.03, 0.06}}]
```

```
{2675.34, {p → 0.0366612}}
```