Action graphs and user performance analysis

Harold Thimbleby
Corresponding author email: harold@thimbleby.net
College of Science
Swansea University, Swansea, Wales, SA2 8PP, UK

ABSTRACT

A user operating an interactive system performs actions such as "pressing a button" and these actions cause state transitions in the system. However to perform an action, a user has to do what amounts to a state transition themselves, from the state of having completed the previous action to the state of starting to perform the next action; this user transition is out of step with the system's transition. This paper introduces action graphs, an elegant way of making user transitions explicit in the arcs of a graph derived from the system specification. Essentially, a conventional transition system has arcs labeled in the form "user performs action A" whereas an action graph has arcs labelled in the form "having performed action P, the user performs Q." Action graphs support many modelling techniques (such as GOMS, KLM or shortest paths) that could have been applied to the user's actions or to the system graph, but because it combines both, the modelling techniques can be used more powerfully.

Action graphs can be used to directly apply user performance metrics and hence perform formal evaluations of interactive systems. The Fitts Law is one of the simplest and most robust of such user modelling techniques, and is used as an illustration of the value of action graphs in this paper. Action graphs can help analyze particular tasks, any sample of tasks, or all possible tasks a device supports — which would be impractical for empirical evaluations. This is an important result for analyzing safety critical interactive systems, where it is important to cover all possible tasks in testing even when doing so is not feasible using human participants because of the complexity of the system.

An algorithm is presented for the construction of action graphs. Action graphs are then used to study devices (a consumer device, a digital multimeter, an infusion pump) and results suggest that: optimal time is correlated with keystroke count, and that keyboard layout has little impact on optimal times. Many other applications of action graphs are suggested.

To appear in *International Journal of Human-Computer Studies*

Keywords

Action Graph; the Fitts Law; Finite state transition system; Lower bounds on task time; User modelling.

1. INTRODUCTION

Predicting how users will perform using an interactive system is a key part of the science of HCI as well as a practical part of usability analysis.

This paper introduces action graphs, which generalise finite state machines to allow analysis of user actions. The dimensions of buttons and their separation along with an action graph can be used to predict time or other costs the user incurs for any sequence of activities. Since times are calculated using programs, *any*

programmable function can be used, such as the Fitts Law, KLM or other model (even financial costs). This paper provides an algorithm (in Java) to convert a standard model into an action graph; our work is reproducible and could be embedded into analysis tools. This makes a significant advance on our previous work (Thimbleby, 2007a; Gimblett & Thimbleby, 2010).

Almost any interactive system can be analyzed with action graphs, though the example case studies in this paper are based on "control panel" devices with a small keypad, rather than typewriter (QWERTY)-based devices; thus this paper is not explicitly concerned with information-based applications (word processing, data entry, diaries, address books, etc) but with the control of systems (such as instrumentation, medical devices, consumer devices) — although an abstract view of a complex application such as a word processor may have interesting control features, say, in its dialog or menu structures, which would be amenable to action graph analysis.

Our action graph case studies suggest that optimal task time and keystroke counts are correlated and, surprisingly, that keyboard layout is not a significant factor in optimal task times. However, such results are but a small contribution of the paper, since action graphs can be used to explore many further issues.

1.1 Conceptual background: tasks, generalisations & abstractions

This paper presents a mathematical framework to address certain HCI questions, and its main benefits are that it permits a complete and automatic analysis of certain issues previously beyond the reach of researchers (except in the very simplest of cases). As a piece of mathematics, it is correct; the key questions, then, are whether it can be applied to HCI in an appropriate and in a useful way?

By way of comparison, "addition" is correct mathematically, but whether and to what extent it can be usefully applied to real-world questions, say, about money and cash depends on various non-mathematical, or at least "non-addition" issues. For example, in "the real world" there are inflation and interest and bank charges and even thieves, so money in a bank account does not quite follow the usual laws of addition without a lot of qualification. Cash, however, is more familiar than HCI theory and certainly far clearer than action graphs, which this paper introduces! We will therefore use the very familiar territory of cash as a conceptual bridge to help make some of the HCI issues of action graphs clearer: very familiar issues with cash and addition have interestingly analogous issues in the less familiar territory of action graphs.

The big picture could be put like this: although one would hardly think of dismissing the abstract idea of addition because of the technicalities of inflation, it might be tempting to dismiss action graphs because of "their" problems when in fact the problems are more to do with the complexity of HCI. In particular, the rigour of action graphs highlight many boundary problems that deserve more research, in a sort of similar way that an apparent failure of addition on your bank account might reveal a thief, or something even more interesting, at work that deserves closer investigation rather than dismissing theory that does not cover everything.

If different sorts of coins are to be added for a cash value, they should be treated with different values. In this paper, we use our approach to add times

due to finger movement, but it could also be used to add times (or even cash values) from other sources. We use the Fitts Law to estimate times for a user to do tasks, but we could have used, for instance, KLM (Card, Moran, & Newell, 1980), which would add further types of time values. Mathematically, this is trivial, but for the first paper introducing the approach it adds a level of complexity — in fact, we side-step this complexity by emphasising lower bounds on times; KLM would *increase* timing estimates, but does not affect hard results from lower bounds. (The second case study, discussed in appendix B.1, introduces "button hold operators," which shows that generalizations like KLM are trivial.)

If cash (e.g., from a loan) is to be added, it may have a time-dependent value. We assume the user interface has a fixed physical layout, as occurs on physical devices such as industrial control panels. The mathematics can handle dynamic, soft key layouts, too, but for the purposes of this paper such dynamic features introduce unnecessary complexity.

If very large amounts of cash are to be added, a computer program may overflow and give incorrect results. We use a computer to perform calculations with action graphs, and as such, we are limited to work within the practical limitations of computers. This means there are some interactive systems that are too complex to be satisfactorily analyzed, but we would argue that such systems raise HCI questions of a different nature than our approach is intended to handle. Moreover, a system that is too complex for a computer to analyze is possibly too complicated for conventional concepts of usability to be applied.

If people do not declare all their capital and cash flow, one will obtain incorrect results. People often ignore illiquid capital — because they are only interested in cash, or perhaps because they are trying to pay smaller insurance premiums. In other words, one has to be clear what the task is, and then analyze it correctly. Our approach uses action graphs. Any task a user performs on a device changes the state of the device's action graph; thus, every task corresponds to a state change. Just as there are some types of monetary value one may not wish to declare, there are some types of state change that one may — or may not — consider to be valid tasks. For example, a type of task one might want to ignore when analysing a ticket machine is "press buttons, insert cash, but don't get a ticket." Undoubtedly the device has a sequence of states corresponding to this failed task! For some analyzes, one might want to know the time cost to the user of failure (presumably it would be very frustrating for it to take a long time before the user discovers they cannot get a ticket), and for other analyses one might wish to ignore it. From a computational perspective, both choices are easy: we can define tasks as any state change, or define tasks as any state change ending with dispensing a ticket — or we can impose any other task criterion that interests us to analyse.

How people wish to use their cash is a question of economics, not just of addition. What tasks a user wishes to perform is beyond the scope of this paper.

Some people may be quite happy not knowing exactly how much cash they have; they don't need to use addition (adding up coins), they just shake and listen to the piggy bank, or use some other heuristic to check they have enough to live by. Although action graphs give precise answers to certain HCI questions, indeed questions that previously were impractical to address in their full generality, they do not address all HCI concerns. They are another tool for the toolbox,

not a replacement toolbox.

Not everybody uses cash; how does addition work with cheques, credit cards, shares, banknotes and other forms of money? A natural question is to ask whether action graphs can handle continuous systems such as virtual reality, speech, action games, and so forth. This question is rather like saying, "I can see how addition works with coins, but how does it work with paper money?" (The answer is, you first need to be able to convert arabic numerals into numbers.) Yes, action graphs can handle continuous systems; you first need to decide an abstraction that ensures the action graphs measure the values of interest, and just like converting the text on a banknote into a value, one will need to convert the duration of (say) a music track into a number. How that is done is an issue beyond action graphs, but once obtained, the numbers then plug into action graphs and analysis can proceed exactly as described in this paper. In fact, it is unlikely that action graphs will help much with sharp usability issues here — does the length of a track affect the usability of a music player? — but the music industry might wish to use action graphs to model costs and profits obtainable over the period while the user is downloading and listening to a track.

Finally, we often want to know the value of a pile of cash, and it is natural to add up its value to find out — addition is obviously useful. The question is, for any mathematical technique, does it tell us things we didn't or couldn't know without it? Finally, are action graphs worth the effort? They are a new, simple technique that answer certain HCI questions; in that sense they are another contribution to the HCI literature. More specifically, we started analysing large interactive systems we thought that cross over, which is defined below, would be a problem — it is a design issue where satisficing users may choose unexpected strategies.[1] (In fact, action graphs were invented to handle cross over.) It turns out that for all devices we have now analysed, cross over is not a significant factor in estimating task times. This is quite a surprise, and ironically suggests that estimates of task times for these types of device can be obtained without action graphs!

The preceding comments have hopefully made the philosophical orientation easier to understand, but the comparison with something so mundane might accidentally make the approach seem equally trivial. In fact, the methodology used in this paper spans disciplines, drawing them closer. We develop some theory and analyse systems, which is superficially like and broadly similar to cognitive modelling — distinctively, cognitive modelling is usually completed with an empirical evaluation, but our approach does not rely on direct human-based evaluation, though some of our analysis relies on published results from empirical experiments. The literature on cognitive modelling cannot be briefly summarised, but see Card, Moran, and Newell (1983); Gray, John, and Atwood (1993); Grossman, Kong, and Balakrishnan (2007); Kieras and Meyer (2000); Kieras, Wood, and Meyer (1997); St. Amant and Horton (2007); Meyer, Abrams, Kornblum, Wright,

[1] *Satisficing* is a word introduced by Simon (1996), combining *satisfy* with *suffice*, and here means users need not adhere to strictly optimal strategies (optimal strategies may be too hard to work out, or take too long) and instead perform heuristically or well-enough in ways we may not be able to anticipate in detail because of the user's arbitrary choices.

and Smith (1988); Matessa, Remington, and Vera (2003).

Research methodology in HCI owes much to the conception of science stemming from Francis Bacon (and his ideas as refined particularly by John Stuart Mill) and is empirical: put briefly, since we do not know adequate theories *a priori*, we should explore the world inductively to determine them. In contrast, Isaac Newton's innovation was to start with simple assumptions, explore the mathematical consequences, then turn to real conditions (Cohen, Nauenberg, & Smith, 2008). If you start from the world, as Bacon recommends, you perhaps never achieve clarity, whereas with Newton's approach, you start with clarity then determine how applicable it is. Following Newton's style, then, the methodology of this paper is to start with mathematics with explicit assumptions, and then to explore the consequences of those assumptions. Real case studies (see section 5.1 for the main case study, and additional case studies provided in appendix B) show the value of the approach, but the approach can be applied far more widely. Obviously, while necessary this alone is not sufficient for a useful contribution; therefore, we also argue that the results we achieve are unexpected and insightful.

Inevitably, Baconian science is driven by what is easy to measure. In an empirical experiment time is easy to measure, but from a system perspective keystroke count is easy. The differences in these practical considerations should encourage research on the tradeoffs between the various approaches. For example, in many contexts time is crucial, but in many others low error rate is crucial. Almost certainly, reducing keystroke counts will have a better impact on overall error rate (e.g., if the probability of error per keystroke is p, then the probability of an error-free sequence of n keystrokes is $(1-p)^n$; this is exponential with n, and therefore reducing n is indicated to reduce error rate); conversely, requiring users to work faster (reducing time) may increase error rates. Now from a system perspective, keystroke count is easy, even trivial, to measure, but this is not sufficient for many purposes — we need new methods to broaden the scope and impact of system-based theories.

1.2 HCI background

In 1985, Newell and Card (1985, p.237) said "striving to develop a theory that does task analysis by calculation is the key to hardening the science [of HCI]," and writing a decade later MacKenzie (1995) anticipated a future scenario:

> "something like this: A user interface is patched together in story-board fashion with a series of screens (with their associated soft buttons, pull-down menus, icons, etc) and interconnecting links. The designer puts the embedded model into "analyse mode" and "works" the interface — positioning, drawing, selecting, and "doing" a series of typical operations. When finished, the embedded model furnishes a predicted minimum performance time for the tasks (coincident with a nominal or programmable error rate). The designer moves, changes, or scales objects and screens and requests a reanalysis of the same task."

Systems like CogTool (John & Salvucci, 2005) are already a great help for designers working from storyboards, but (to date) they only evaluate specific, sequential tasks composed of relatively few steps. This paper will show how to

predict optimal times a skilled user would not be able to do better than for any or all tasks, or from benchmark collections of tasks, composed of any number of choices and steps — all without the designer having to patch a story-board together or "work" the user interface as MacKenzie envisaged. Of course it remains possible to obtain estimated times for particular sequences of user actions (e.g., from story-boarded sequences) if desired. The importance of "automatically" becomes apparent when analyzing devices with thousands or more states: there are then millions of potential tasks.

Card, Moran and Newell's classic *The Psychology of Human-Computer Interaction* (Card et al., 1983) argues that reducing expert time is a key principle of user interface design. Expert users often want "short cuts" such as special keystroke combinations that save work, presumably to save time as much as to reduce the number of actions they have to do.

Projects such as Ernestine were driven by the conviction that "time is money" and that it was worth redesigning user interfaces to make them faster to use (Gray et al., 1993). There is considerable evidence that users optimize timings (e.g., Appert, Beaudouin-Lafon, & Mackay, 2004; Gray & Boehm-Davis, 2000), and eventually will treat optimal or nearly-optimal interaction as routine. Howes, Lewis, and Vera (2009) give evidence that optimal time is a predictor of actual skilled performance time: people are adaptive, and with practice they improve. (Bailey, Wolfson, Nall, and Koyani (2009) provide a review of usability testing and high-impact metrics.)

In safety critical domains, conventional empirical experiments cannot cover all features of devices even of modest complexity; usability inevitably gets relegated to "look and feel" or focuses on a few tasks. Thorough empirical exploration is not possible except for the most trivial of devices. Although action graphs are only a start, more development in analytical approaches is needed to extend the scope of HCI further into systematic analysis, particularly when there is a requirement to do so, as in safety critical domains.

Conventional user evaluation is costly (to pay human participants, buy laboratory time, and to manage the experiments) and must be performed later in the design cycle, after a prototype system has been made available. At this stage, insights are less likely to be fed back into the design: many decisions have already been made, and if the system works well enough to evaluate it, why not ship it? Indeed, production pressures typically mean that companies ignore poor usability provided that systems appear good enough to be shipped. In many environments, then, improving usability has negligible priority after a system "works," for when a system appears functional it is unlikely to be revised even if revision could achieve usability gains.

As soon as a specification of an interactive system is available, or as soon as program code is written, a system model can be obtained (Thimbleby & Oladimeji, 2009; Gimblett & Thimbleby, 2010) that can be used to generate action graphs automatically — this approach is extremely useful in an iterative design process, since the model can be continually regenerated for analysis as the design is modified. Thus, the approach lends itself to predictive analysis, which can have a significant influence on a design because it can be used earlier, cheaper, faster and more often, and at a design phase when improvements are easier to implement.

There is a great need for quantitative predictions about user performance with designs well before actual experiments with users can be contemplated. This is the key point: *predictive estimates of low bounds on time are relevant to design for or to analyze skilled behavior*, for skilled behavior cannot do better than the theoretical low bounds. Other research based on this premise includes Pirolli (2007), and Gray and Boehm-Davis (2000).

Illustrating these issues, in fact, most relevant published work to date — including Kieras et al. (1997); Appert et al. (2004); John and Salvucci (2005) — is based on analyzing manually *predefined* scenarios: that is, given a particular sequence of user actions, estimate the time a user would take to achieve a specific goal. Menu selection (Cockburn, Gutwin, & Greenberg, 2007; St. Amant & Horton, 2007) is a special case where the goal is to make a selection, and where each selection has only one way to make it. Petri nets have been used, but most published papers (e.g., Lacaze, Palanque, Navarre, & Bastide, 2002) only show single-step times, not times for arbitrarily long sequences of actions that this paper handles, though some papers (e.g., St. Amant & Horton, 2007) explore linear sequences of actions. In all cases the system modelling seems to be limited by the difficulty of precise manual analysis; for example, St. Amant and Horton (2007) note that system features, which they ignore, such as short-cuts, would complicate their analysis. We have no such problems in this paper, because our approach is fast, general and completely automatic.

1.3 Action graphs *versus* KLM and CogTool

Researchers using action graphs or methods such as KLM (Card et al., 1980) CogTool (John & Salvucci, 2005) may do the same sorts of things, so it is natural to make a comparison between the approaches.

KLM is usually a manual technique for estimating task times from user behavior, keystrokes, mouse movements and mental operations. It relies on having a task breakdown. Action graphs can provide this task breakdown for any or all tasks a system supports; action graphs allow KLM (or any related analysis approach) to be automated, and allow KLM to be applied without manual effort. In particular, in areas where coverage is required (e.g., for safety critical interactive systems), action graphs allow every task (perhaps millions of tasks) to be analysed automatically for any device. Previously, this has not been possible except, perhaps, in very limited contexts.

CogTool is an interactive tool (with a graphical user interface) with a much more sophisticated underlying model than KLM; it is much easier to use and more accurate. CogTool allows researchers, system designers, usability professionals, to build a story board of a proposed or actual system, and then run a sophisticated psychological model (using ACT/R) on it. A researcher thus obtains realistic estimates of task times (along with breakdowns) from CogTool. ACT/R is a very complex program (because it is a very realistic human performance model), and CogTool uses it as a black box.

Action graphs are a theoretical model, very similar to finite state machines. They allow interactive systems to be implemented and analysed, with the advantage over finite state machines that they directly support analysis of sequences of user actions.

CogTool is open source and runs on commercial Macintosh and PC platforms. Action graphs are theoretical and completely described in the present paper; they are therefore "open source" for all practical purposes.

CogTool is quite a complex system, but the CogTool web site (`cogtool.hcii.cs.cmu.edu`) provides substantial documentation, downloads, and access to the CogTool user community. In a sense action graphs are simple and elegant, but unfortunately they rely on multidisciplinary knowledge, graph theory, algorithms and HCI, so although they are "simple" they have a comparable learning curve to CogTool. In contrast, the present paper is the only documentation on action graphs. An interesting contrast between CogTool and action graphs is that you have to understand CogTool to use it, but action graphs could be used inside an HCI analysis program without the user of that program knowing anything about action graphs: action graphs are a means to an end, not an end in itself.

CogTool could use action graphs as a means of implementing story boards and supporting ACT/R (in the present paper we use action graphs with Fitts Law, but any measure, for instance provided by ACT/R could be used). In fact, CogTool effectively implements a single path through an action graph, as the sequence of ACT/R-annotated actions a story board represents. Thus CogTool analyzes single paths through story boards, whereas action graphs are a natural representation to explore all or any subset of paths, including a single path.

Since action graphs allow automatic analysis of all paths a user might take using a system, they can be used to support analysis of safety critical systems, where coverage (i.e., checking every feature) is essential. KLM and CogTool cannot do this, though if either KLM or CogTool was implemented using action graphs, it would become feasible to explore alternative user strategies, optimal behavior as well as user error.

Since CogTool relies on building a story board by hand, it is impractical to analyze many design alternatives; the story boards tend to be very small in comparison with action graphs, which have no real practical limitations on size. On the other hand, the story board is a natural, visual representation of interaction, and this approach makes CogTool very appealing to its user community. Because of the underlying ACT/R model, the analysis of the single story board is thorough and insightful, though exploring alternatives (and keeping track of them) is tedious. Using CogTool seriously in iterative design would be burdensome: as changes to a design are made, the story boards need to be revised and this will unavoidably create a version control problem with the requirements or specification of the target system.

In contrast, action graphs are used to specify a system, and how that system is originated is outside their scope. A story board would only give one (or possibly a few) alternative paths, and this would not be sufficient. In the present paper, complete system models are automatically derived from running programs using discovery (Thimbleby & Oladimeji, 2009; Gimblett & Thimbleby, 2010), though one could equally obtain system models from specifications (written in any of the many formal specification languages that generate FSMs or BDDs).

1.4 Notations and the role of appendices

The body of this paper assumes a breadth of knowledge covering the Fitts Law, graph theory, lower and upper bounds, order notation, and algorithms. While the ideas may particularly inspire HCI researchers, the paper is also likely to be read in depth by programmers implementing tools based on the ideas.

Unfortunately there is a confusing variety of assumptions and notations used in wider literature, so some short appendices have been provided to supply a coherent summary of and short introduction to the standard notations and concepts used in this paper. These brief appendices also provide references for further reading on the topics.

Appendix A.1	Fitts Law; brief introduction and details of parameters used in the present paper.
Appendix A.2	Upper and lower bounds, including fastman/slowman and bracketing.
Appendix A.3	Order notation, including Ω notation.
Appendix A.4	Graph theory and state machines.
Appendix A.5	Least cost algorithms, and why action graphs are needed for user performance analysis.

The paper develops a theory, then applies it to explore some real case studies. The main case study is presented in the body of the paper, but several other case studies are provided in appendix B, primarily to support the argument that the main case study has the properties ascribed to it because it is typical, rather than arising by chance or (worse!) by contrivance or special selection. (We also vary the case studies to explore some more extreme keyboard layouts, see appendix B.2.) While good conventional HCI experiments take care to control for variability in human users, we are unaware of other HCI experiments that similarly try to manage variability in device design; the space of device design is largely unexplored territory.

Appendix C expands potential critiques of the case study experiments, details that would perhaps have been too technical or too distracting within the body of the paper (which already has a substantial further work section, section 6), as well as exploring some further thought experiments.

Appendix B	Additional case studies.
Appendix C	Additional further work and critiques of basic results.

1.5 Graph theory device models and notations

The theory developed in this paper will generally be embedded in a tool, such as CogTool (John & Salvucci, 2005), so a typical user (e.g., an HCI professional) need not know any technical details. However in this paper, we need to develop and justify the approach. Readers unfamiliar with graph theory notation may wish to refer to appendix A.4.

We represent an interactive device as a graph: a set of vertices $\mathcal{V}$ (states), a set of user actions $\mathcal{A}$, and a transition relation $\mathcal{T} \subseteq \mathcal{V} \times \mathcal{A} \times \mathcal{V}$. It is suggestive to represent elements $\langle u, a, v \rangle$ of the graph by $u \xrightarrow{a} v$. In words, the notation $u \xrightarrow{a} v$ means that if the device is in state u and the user does action a, the device will

transition to state v. In graph theoretic terminology, $u \rightarrow v$ is an arc and a its label.

A sequence of transitions $u \xrightarrow{a} v$, then $v \xrightarrow{b} w \ldots$ is more concisely represented by $u \xrightarrow{a} v \xrightarrow{b} w \ldots$ When we are not concerned with the details of the intermediate steps $a, b, \ldots$ (what actions are, what intermediate states are visited, and how many states are visited), we use the notation $u \rightsquigarrow w$.

Actions $\mathcal{A}$ define names and the geometry of targets (i.e., physical details of the button, its name, shape and location) to perform those actions. For systems with timeouts (like "reset if user does nothing for 10 s" or "hold button for 2 s") actions in $\mathcal{A}$ define the appropriate timings. The model allows for soft keys and touch screens that can display changing, moving, or expanding targets for the user to press or mouse click on; $\mathcal{A}$ is enlarged accordingly to accommodate each variation of input actions, simply by having a distinct action $a \in \mathcal{A}$ for each unique user action. Thus if the "same" button can appear in different places (a common strategy to stop users habituating, and, say, clicking OK without checking a warning), we still need each place to have a separate action for our analysis.

We use the following notation for properties of sequences of actions, σ:

$\lvert\sigma\rvert_{\#}$	the number of user actions; thus if A, B, C are actions then $\lvert ABC\rvert_{\#} = 3$
$\lfloor\sigma\rfloor_T$	the lower bound time to perform the actions
$\sigma_1 \equiv \sigma_2$	if the two sequences of actions have the same effect on the system.

In this paper, the system model $\mathcal{M}$ and the initial state s_i will be readily understood from the context; the standard notation $\mathcal{M}, s_i \models$ *formula* would be used in more formal presentations.

2. CROSS OVERS: TIME NEED NOT CORRELATE WITH ACTION COUNT

One might think that reducing the number of actions a user needs to achieve a goal reduces the time required for the task; this is certainly true in applications like menu selection (Cockburn et al., 2007) where each menu selection has a unique sequence of user actions. However, if there is more than one way to do a task it is possible that a faster way can be found for doing a task that nevertheless requires *more* actions. We are not aware that this issue has been previously explored in the literature, though some papers acknowledge uniqueness as an explicit, but limiting, assumption in their work (St. Amant & Horton, 2007).

Consider the graph shown in figure 1. If the task under consideration is getting from state 1 to state 4, this can be achieved with the least number of actions by doing AB, but under certain conditions it can be achieved in less time by doing AAA, despite taking an extra user action. Although the skilled user is thus faster, they are also getting the system into an extra state (namely 3) and this may have unexpected side-effects; the user is not just being faster, but on occasion may be doing something different.

The graph in figure 1 has the property that AB and AAA are alternative paths connecting state 1 to state 4: put another way, if the goal of a user is to get the device from state 1 to state 4, the user has a choice. (With the notation introduced in section 1.5: $AAA \equiv AB$.) Clearly, AB is a better way to get from 1 to 4 when

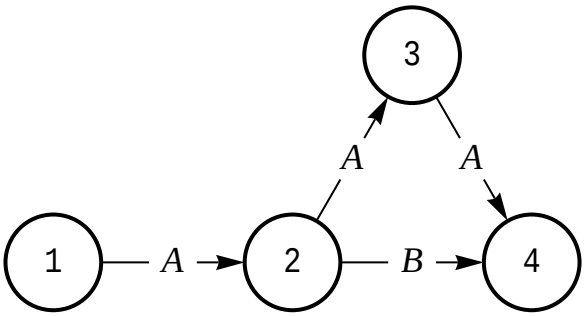

Figure 1. A transition diagram for a simple device or for a small part of a complex device, with four states $(1, 2, 3, 4)$ and two buttons (A and B). Placing the buttons A and B sufficiently far apart ensures the fastest way in terms of time to get from state 1 to state 4 would be by pressing A 3 times, thus taking 3 state transitions. However, the fastest in terms of counting user actions would be to press A then press B, which requires only 2 steps.

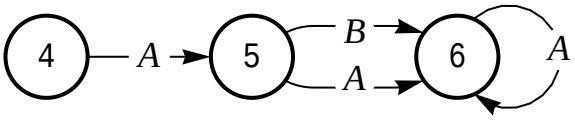

Figure 2. Using the device shown in figure 1 the user may learn that AAA is faster than AB for getting between two states. Another part of the same (or another) device may be as shown in this figure: here, under the same assumptions, to get from state 4 to state 6 (which may appear to the user to be similar to states 1 and 4) it is *still* true that AAA is faster than AB, however this mode of the device also allows AA, which of course is faster even than AAA. The user may never realize that a better strategy exists, especially since they may know that their strategy-in-use is more efficient than the worse alternative.

measured in terms of counting user actions. However, AAA may be faster in terms of time. In particular, if A and B are typical buttons in fixed locations that do not change, then AAA is done by the user as "locate A, press, press, press," and AB is done by "locate A, press, locate B, press." The initial location of A can be assumed to take the same time in each case, as will the pressing of an already-located button, whether A or B. Thus if the time to press button A a second time is faster than the time to locate and press B after locating A, then AAA will be faster than AB.

This example system has the properties: $AAA \equiv AB$, $|AAA|_{\#} > |AB|_{\#}$ yet $\lfloor AAA \rfloor_T < \lfloor AB \rfloor_T$. The paradox is that it seems obvious that button pressing takes time, that moving one's fingers to buttons also takes time, and therefore that more actions should take longer. The example shows that a faster cross over can happen, depending on the interaction of physical layout and task structure.[2]

In this example, there is only one cross over. In general there will be multiple shortest paths between vertices, as figure 4 makes clear, so a cross over is defined over *sets* of optimal paths: if C_{uv} is the set of optimal action count paths and T_{uv} the set of optimal time paths between two given vertices u and v, then there is at least

[2]Recall that times and action counts are expressed in terms of *specific* sequences of user actions: that is, we write $\lfloor AB \rfloor_T$ to mean the lower bound on the time to *do* AB, not to mean the lower bound on the time to do what AB does — otherwise we would have $\lfloor AAA \rfloor_T = \lfloor AB \rfloor_T$ since $AAA \equiv AB$ for this device.

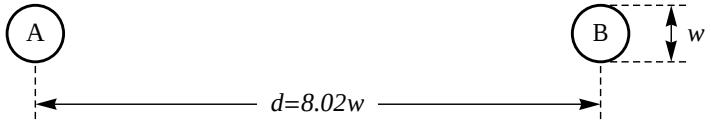

Figure 3. With reference to figure 2, to achieve $\lfloor AAA \rfloor_T < \lfloor AB \rfloor_T$, buttons A and B should be separated so $d > 8.02w$. This figure, drawn to scale, represents a separation of $d = 8.02w$, and thus illustrates how unusual this layout would be as part of a typical multi-button device, given that the separation shown is the minimum. (Most devices have buttons much closer together relative to their diameters.)

one cross over if $C_{uv} \neq T_{uv}$, and there is a definite (i.e., unavoidable) cross over if $C_{uv} \cap T_{uv}$ is empty (as it is for figure 1) and a potential cross over if $C_{uv} \cap T_{uv} \neq \emptyset$ — that is, the user can find optimal paths on either time or action criteria that may or may not be the same. For analyzing a device rather than a particular task, one would generally consider all possible paths, with u and v ranging over all possible states.

With reference to the Fitts Law (appendix A.1), $\lfloor AB \rfloor_T > \lfloor AAA \rfloor_T$ implies $d > w(e^{a/b} - k)$, which occurs when $d > 8.02w$, making the usual assumptions (appendix A.1). Few devices have buttons so far apart, as figure 3 makes evident, but of course this is a consequence of the very simple state space chosen to introduce the problem. In general, it is not difficult to construct state spaces that have cross overs even when buttons are arbitrarily close together. In fact, for a real device, case study 1 (section 5.1), 2.9% of paths involve a cross over. It is worth emphasising that we would expect such a proportion of cross overs to affect user behavior, particularly if cross overs occur within frequent or important tasks.

2.1 Not just time: Generalised complexity

The previous sections show that time to perform a task need not correlate with the number of user actions required to achieve those tasks. In fact, *any* measure that depends on an independent dimension — here button location and the logarithm of distance — can be manipulated to create cross overs. An example would be obtaining a cheaper product from a vending machine by pressing a "discount" button: one extra button press, one cheaper product.

3. EXTENDING TRANSITION MODELS TO ACTION GRAPHS

An interactive state transition system changes state when a user performs an action, such as pressing a button. User performance measures such as the Fitts Law give times, not in terms of actions, but in terms of movements *between* actions. The time to do a movement can only be calculated from *pairs* of arcs, as two are needed to define where the user was for the previous action and where the user moves to for the next action. The conventional state graph (as described above) does not provide this information deterministically, as figure 5 makes clear.

An interesting argument is as follows. Optimal paths satisfy the *principle of optimality*, that parts of optimal paths are themselves optimal, for if they were not, making them optimal would reduce the cost of the overall path. Yet in figure 1 the time optimal path $1 \rightsquigarrow 4$ is $1{\rightarrow}2{\rightarrow}3{\rightarrow}4$, and the time optimal path $2 \rightsquigarrow 4$ is $2{\rightarrow}4$ not $2{\rightarrow}3{\rightarrow}4$ as on the optimal path $1 \rightsquigarrow 4$. Here, substituting a faster

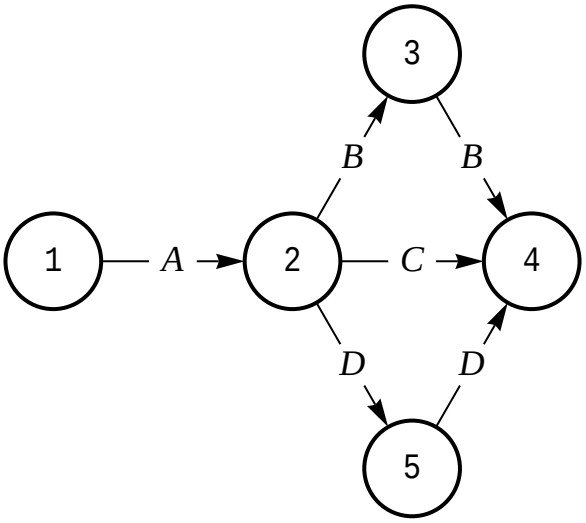

Figure 4. Graphs may have many multiple optimal paths between the same vertices, not just one as in the misleadingly simple figure 1. Illustrated here are three paths $1 \rightsquigarrow 4$; two, possibly three, of them potentially of equal optimal times.

subpath makes the path of which it is a part slower! The explanation is that each device state actually represents several path states, so the state 2 in $2{\rightarrow}3{\rightarrow}4$ need not be, and is this case is not, identical for all purposes as the state 2 in $2{\rightarrow}4$, hence the faster subpath $2{\rightarrow}4$ cannot be substituted in $2{\rightarrow}3{\rightarrow}4$. The problem arises because the user is doing things that matter for determining optimal (or indeed, any other) paths (or path properties), which the standard state machine graph cannot represent properly.

In other words, the conventional device state is not a proper representation of the user model of the state. In the example above, a user model needs multiple interpretations of a single device state. Appendix A.5 provides a purely algorithmic explanation of the same problem.

On any account, then, we need a new approach, with path information properly encoded in individual vertices and arcs; we now provide a solution and call such graphs *action graphs*.

Given a conventional transition system G, construct an action graph G' where vertices $\langle v, m\rangle$ in G' record that vertex v in G is entered by the user performing action m. Each transition in G' therefore corresponds to a specific movement between pairs of actions in G. Hence:

$$\begin{aligned}\mathcal{V}(G') &= \{\langle v, a\rangle \mid u \xrightarrow{a} v \in \mathcal{T}(G)\}\\ \mathcal{T}(G') &= \{\langle v, m\rangle{\rightarrow}\langle w, n\rangle \mid u \xrightarrow{m} v \xrightarrow{n} w \in \mathcal{T}(G)\}\end{aligned}$$

This is a basic action graph with unlabelled (and unweighted) arcs. For this definition we require G to be total; that is, for all vertices $u \in \mathcal{V}(G)$ and all actions $a \in \mathcal{A}(G)$, there are arcs $u \xrightarrow{a} v$ (possibly self-loops $u \xrightarrow{a} u$) in $\mathcal{T}(G)$.

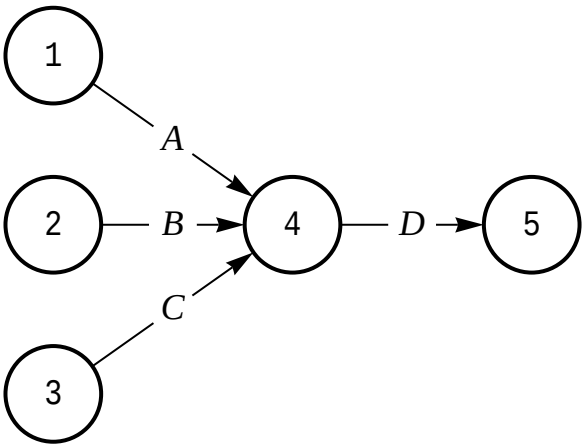

Figure 5. When in state 4, pressing button D changes the state to 5, but the time it takes the user to press D and hence the time to change the state depends on how the user's finger moved prior to pressing D. In this example, the user's finger must have pressed and hence been on A, B or C in order to reach state 4. The user's finger may have moved away from these buttons before pressing D, but the minimum time to press D will be given by the Fitts Law estimate based on one of the motions AD, BD, CD, and the target size of D given the approach angle. In this graph, there are only 3 choices, but the amount of non-determinism grows exponentially on the length of user action sequences, and is unbounded if the graph has cycles. An action graph eliminates this non-determinism from calculations.

It follows directly from the definitions above that *all* properties of G, including all interaction properties, can be derived from G'. However, the vertices and arcs of G' encode more information, specifically user action information, and thus make action graphs considerably more convenient to use to analyse certain properties of G relevant to interaction.

When graphs are derived by discovery (Thimbleby & Oladimeji, 2009; Gimblett & Thimbleby, 2010), total graphs are generated automatically, but it is unusual and visually cluttering to *draw* total graphs. By way of example, figure 6 shows a total graph corresponding to figure 5, where all missing arcs have been corrected by self-loops. In a sense, figure 6 represents the intuitively same behavior as figure 5 — looking at figure 5 we assume that missing arcs "do nothing," which actually means they should be self-loops. On the contrary, for a device like a cash safe with a combination lock, "unspecified" arcs would put the safe in a single reset or alarm state, not leave it in the same state, as we are assuming in this total graph construction here.

If G has $n = |\mathcal{V}(G)|$ vertices and b possible actions (the out-degree of every vertex is b), then G' has nb^2 vertices, b actions, and nb^3 arcs.

The action graph shares many properties with the original graph. If G is deterministic, G' will be. In particular, G' is total. If G is strongly connected, G' will be; in fact, G and G' have the same number of strongly connected components. Although it is a simple exercise to prove such results, they are intuitively valid because the action graph supports the same interaction style or user experience as the original graph, and hence must share the corresponding graph theoretic prop-

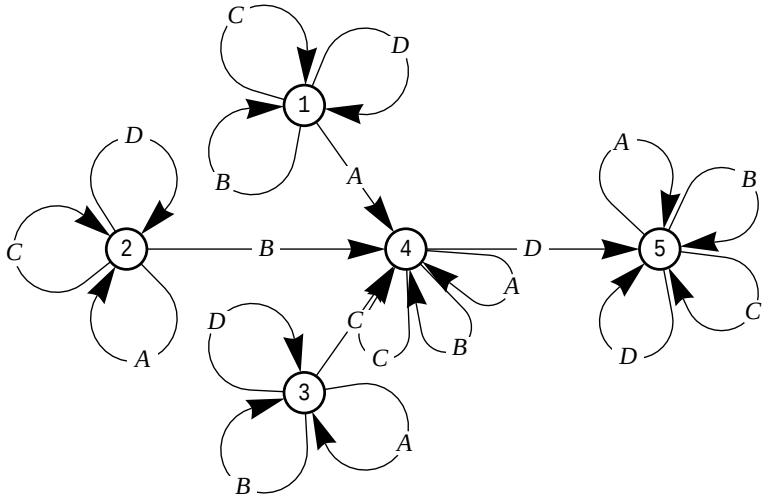

Figure 6. A total closure of the graph of figure 5. This graph is easily constructed automatically, though it is a design implementation error if a user interface is not already closed, as some actions would be undefined in some states. For example, in figure 5 action B in state 1 is not defined (in the closure here it is defined but does not change the state); in fact, in the closure all actions are defined in all states. For an interactive system, the graph potentially has other problems: for example, once the device is in state 5, it does nothing and cannot be reset — and, since there is no start state specified, it isn't clear what reset would do anyway! Such issues, see Thimbleby (2007a), are beyond the scope of this paper, as here we are interested in what tasks the graph specifies, rather than on what tasks the graph *should* specify.

erties. However, a user cannot (by experiments on the interactive system itself) count vertices or arcs, as some may be indistinguishable, so these graph theoretic properties are not related to interaction properties, and indeed the action graph has more vertices and arcs than the underlying graph; in fact, this implies the action graph introduces vertices and arcs that a user cannot distinguish. Put another way, an action graph introduces distinctions that are useful for analysis, but which are irrelevant to a user.

A transition $\langle v, m\rangle \rightarrow \langle w, n\rangle$ in the action graph G' implies if G got to state v by pressing m, then the user moving from the position to perform m to n then pressing n, G will transition to w. Since the arcs in G' thus encode the movements in G as well as G's state transitions, we have enough information to apply the Fitts or similar law as a cost function f on each arc of G':

$$\mathcal{T}(G') \;=\; \{\langle v, m\rangle \xrightarrow{f(m,n)} \langle w, n\rangle \mid u \xrightarrow{m} v \xrightarrow{n} w \in \mathcal{T}(G)\}$$

Figure 7 shows part of the action graph of figure 5 with arcs labeled in this way.

Since by construction f has m, n (and potentially any or all of the sequence of states u, v, w) as a parameters, the function f can encode state or movement-dependent formulæ for each arc in the action graph. If we know some actions in some states are performed by the thumb, where the Fitts Law constants are different (MacKenzie, 2003), this can be taken account of. If we do not know what a user will do, since we are interested in lower bound times, f can evaluate all

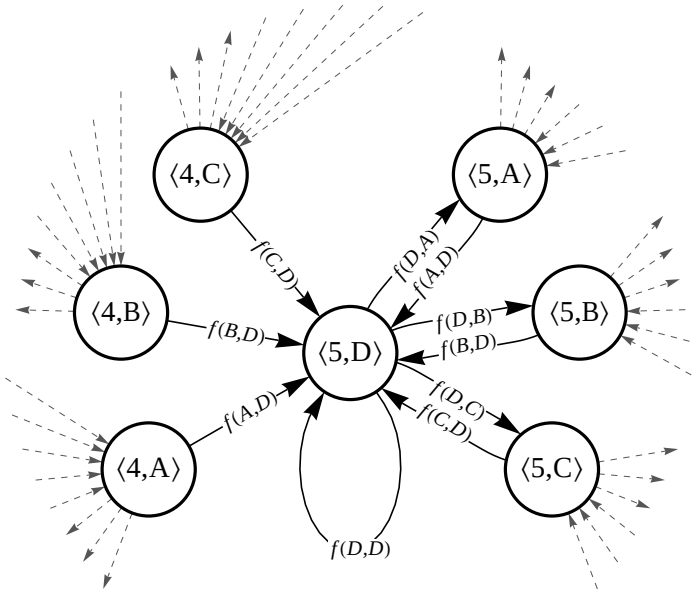

Figure 7. In an action graph, state $\langle u, a\rangle$ records that the original state u is reached by action a, and arcs are labeled $f(m, n)$ meaning that n is pressed (causing the transition) following an immediately previous press of m. The action graph derived from figure 5 (and 6) has 16 vertices and 80 arcs (an increase in size that is typical of moving from a non-deterministic machine to a deterministic machine). Here we show the subgraph of arcs adjacent to $\langle 5, D\rangle$ — the six vertices around the perimeter have additional arcs that are not in this subgraph but are shown (in arbitrary directions) for illustrative purposes.

possibilities and pick the minimum. In particular, for the Fitts Law we can use the effective target size of the target button n given that the finger is moving on a vector in the direction m to n.

In particular, if certain actions $v \xrightarrow{n} w$ have timeouts t_n, then f will not be a conventional Fitts Law for that transition, but t_n plus the Fitts Law timing or t_n itself, etc, depending on the type of timeout encoded in the action. For example, some devices distinguish between just *pressing* a button (i.e., tapping it) and *holding* a button down for a few seconds. To allow for this, the Fitts Law formula is easily extended to $t = t_n + a + b\log(d/w + k)$ where d/w is calculated as normal for a movement towards the relevant button, and $t_n = 2$ or whatever value is appropriate for the state.

What we have achieved is that time (here, as may be estimated by the Fitts Law) labels arcs in a derived graph: action graphs thus allow us to do any calculation with the Fitts Law. The insight means we can use standard algorithms on G', and since every transition in G' corresponds to a transition in G, results can be "back translated" to properties in the actual design, G.

Since vertices in G' distinguish which button the user pressed in G, the shortest times in G' are not simply shortest times in G; instead, the least time for a path $u \rightsquigarrow v$ in G is the least time $\langle u, m\rangle \rightsquigarrow \langle v, n\rangle$ in G' over all $m, n \in \mathcal{A}(G)$. It is on this

basis that the results summarized in section 5.2 (and appendix B) are based.

In general users need not (or may be unable to) distinguish particular individual states, then paths can be defined in terms of sets of states the user treats as equivalent; we would then find the minimum time over all paths between the initial and goal equivalence classes.

4. AN ALGORITHM TO GENERATE ACTION GRAPHS

Action graphs may be large so they are therefore not normally constructed by hand, and since they share the same user interaction graph properties as the simpler underlying graph it is essentially pointless to visualize them (figure 7 has explanatory power *about* action graphs, not about the underlying graph!). Action graphs are, however, trivial to construct by program; for example, basic Java code to create an action graph from a vector of a graph's arcs is as follows:

```
class Arc
{ Object from, action, to; // represent an arc from --action--> to
}

// assumes that the graph is total
// assumes no arcs are repeated (the parameter arcs form a set)
public static void makeActionGraph(Arc[] arcs)
{ for( int u = 0; u < arcs.length; u++ )
    for( int v = 0; v < arcs.length; v++ )
      if( arcs[u].to.equals(arcs[v].from) )
        System.out.println(
          arcs[u].to+"."+arcs[u].action+" -> "+
          arcs[v].to+"."+arcs[v].action+
          " f("+arcs[u].action+","+arcs[v].action+")"
        );
}
```

For readers unfamiliar with Java, note that the operator + used here is string concatenation, not addition.

The program code directly implements the definition of $\mathcal{T}(G')$ given above. A graph is represented as a vector of `Arc`, arcs being a triple of objects: the initial vertex, the action, and the terminal vertex. For brevity in the context of this paper `makeActionGraph()` does not construct a graph, but simply prints a list of its arcs (thus avoiding further Java details irrelevant to this paper).

5. INTERACTIVE SYSTEMS AS FORMAL CASE STUDIES

Although it is routine to take a single systems or procedures (or two very similar designs for A/B comparisons) and evaluate them with many users, to account for individual differences, rarely do HCI studies examine multiple systems or variability between systems. Thus in conventional empirical HCI studies, very little of device design is examined, a point made by Dix (2010) and at greater length by Thimbleby (2007b). It is relatively easy to do a "large" empirical experiment with, say, 20 human participants, to help control for human variability as humans are readily available to participate in most experiments. It takes minutes to re-

cruit another human participant; but as it can take a year to implement another system if too few devices are used. This reduces enthusiasm for pursuing system-based HCI research, which in turn reduces the profile of systems-based research in the HCI culture. In short, full systems are rarely analyzed, and the generalizability of research to other systems or system conditions is largely ignored. Indeed, very few papers in the literature consider more than one system, and most — e.g., St. Amant and Horton (2007) — only consider small components or subsystems of real devices. In this paper, not only do we examine largish systems (e.g., considering average performance over 50,000 unit tasks), but we examine several devices; our automated analysis is easy to do.

Production devices were used as case studies because they were not created specifically to support any claims of this paper, and because they are clearly of realistic complexity that the approach should be able to, and does, handle. In addition, other researchers can readily obtain the devices and replicate our work; indeed the data and methods used here are reproducible, as the methods and algorithms are also disclosed fully in this paper.

Unfortunately we did not have access to the source code of the devices themselves. Instead, reverse-engineering was used to implement faithful models, and we then used discovery to determine the state transitions systems. Reverse engineering is just an artefact of the methodology used in this paper and is unnecessary for real development. The reason we used reverse-engineering was to have realistic models that readers of this paper can check against real devices; it would have been unpersuasive and methodologically weak to show good correlations from device models specially constructed for the purposes of this paper.

Action graphs can be used to evaluate systems formally; we thus need to use no direct user data. For many results in this paper we use the Fitts Law, which is well-established and we have no grounds to doubt (see appendix A.1). We can be reasonably confident that human studies are not going to provide evidence to seriously challenge the Fitts Law, and certainly not human studies with the general physical properties of the real devices that we are interested in; thus we can be confident that human studies are not going to provide evidence to challenge system calculations that are independent of the user. In other words, the case studies apply and evaluate a system-oriented theory for its relevance to design, not a user-oriented or psychomotor theory for its relevance to psychology. In general, empirical data can be obtained for a specific device, and this data can of course be used to initialize an action graph rather than estimates from Fitts or other laws.

Our approach avoids the confounding experimental factors that beset empirical work, including the common "evaluator effect" (Hertzum & Jacobsen, 2001) and low reproducibility, as in our approach there is no human-human participant-researcher based activity (although we do rely on previous work on the Fitts Law). On the other hand, as so far used, action graphs have been used for lower bounds on times — arguably a failing similar to evaluator effects where, generally, an evaluator will want faster times!

In empirical work one can only study specific tasks, not whole systems. While action graphs support that, most results presented here are averages over all possible uses. The method works equally well with specific, predetermined or weighted tasks, as suggested in the approach of Sears (1993), however to do this requires

Figure 8. The right hand side of the PVR front panel, with button locations and outline shapes shown. (To save space, the "insert tape" location is not shown; in our model the insert tape action requires the user to hit the tape-sized target and press it.)

further data or assumptions about use beyond the scope of the present paper.

5.1 Case study 1 (personal video recorder)

Figure 8 shows the front panel layout of a JVC model HRD-580EK personal video recorder (PVR). The PVR has a complexity typical of such consumer devices.

The device is considered to have 8 rectangular buttons. The graph (as relevant to these buttons) has 28 states, although the remote control accesses more functionality than the front panel and, for example, allows program time settings which is not possible from the front panel. A full discussion of the device and its structure can be found in Thimbleby (2007a), from where the device definition was taken.

The model was used to calculate lower bounds on button press counts and lower bounds on cumulative Fitts Law times for all pairs of different states, using the techniques described earlier. The shortest paths, whether for button press counts (from the device graph) or Fitts Law times (from the action graph), effectively models the user tasks as unit tasks (Card et al., 1983). Figure 9 shows lower bounds on times plotted against lower bounds on counts for each pair of states. The linear coefficient of determination (square of the correlation coefficient) is $R^2 = 0.95$ ($N = 28^2 = 784$). This result is comparable to what Fitts Law experiments achieve in user studies under laboratory conditions.

Comparable results are also obtained from our additional case studies covering a variety of devices, details of which are provided in Appendix B.

5.2 Summary of results

Table 1 summarises all case study results (from section 5.1 above and from appendix B), using the Fitts Law constants from MacKenzie (2003, p45), corresponding to index-finger movement as appropriate for these devices, $t = 0.165 + 0.075 \log(d/w + 1)$ using natural logs and time in seconds, where d is the distance to the center of the target and w its width in the direction of movement. MacKenzie (2003) suggests these constants for mobile phones, which are on the same scale as the devices considered here.

Allowing for the hold button (which adds 2 or 4 seconds to some paths on the Fluke$_{2-5}$ devices where its use is required) the results are very similar, yet as figure 11 indicates, the case study devices are structurally very different. Comparison of figures 8, and the appendix figures B.1, B.3 and B.4 will make clear that the device button layouts are also very different as well.

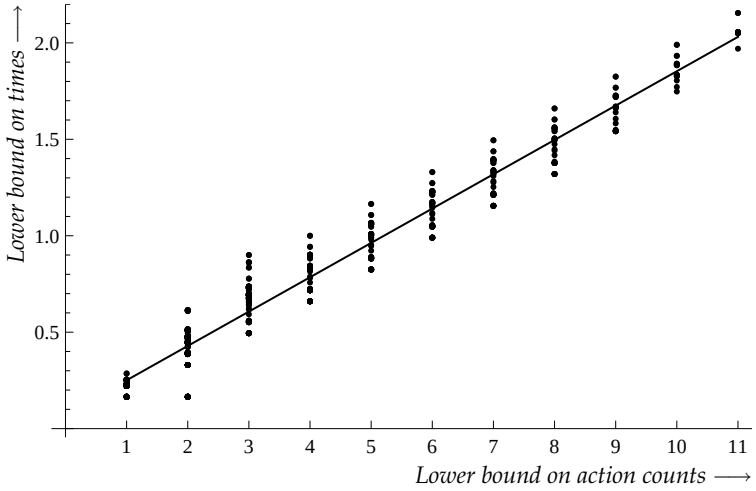

Figure 9. A plot of the Fitts Law timings (vertically) against button press or action counts (horizontally) for all possible tasks on a PVR; that is, the model timings including Fitts law timings (vertically) are plotted against the model with fixed inter-button times, and each point is one task. The graph shows the best-fit line $y = 0.073 + 0.18c$ ($R^2 = 0.95, N = 784$). Note that while each point in the graph represents a particular task of getting from one state to another, the path taken by the lower bound on time and the lower bound on action count will generally be different. Many data points are indistinguishable in this plot, and hence the regression line visually appears to have poor fit; see figure 10 for a plot visualization that makes the good regression clearer.

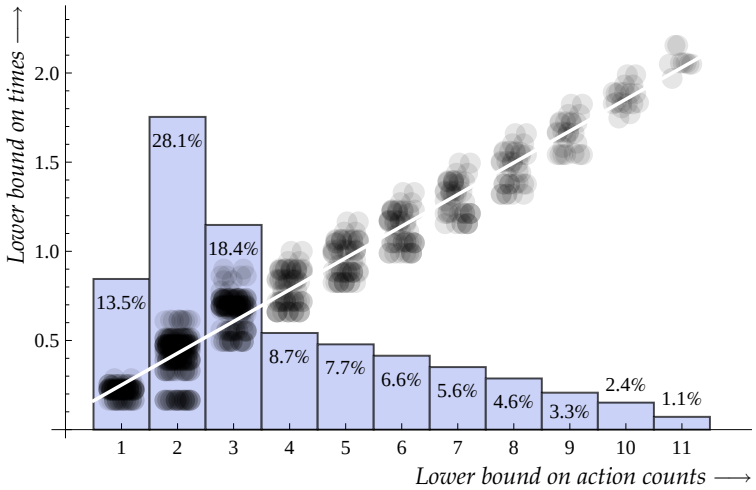

Figure 10. To help visualize the density of points in the data plotted more simply in figure 9, data points are shown as enlarged circles (with centers given a small random horizontal spread in the x axis to the correct coordinates) and plotted with an α-transparency of 4%. Thus areas that represent more points are darker, and because of the random horizontal spread, the visual distribution better discriminates the point density. A bar chart, behind the scatter plot, shows the proportions of the total data represented by each action count value. The α technique is discussed in Theus and Urbanek (2009).

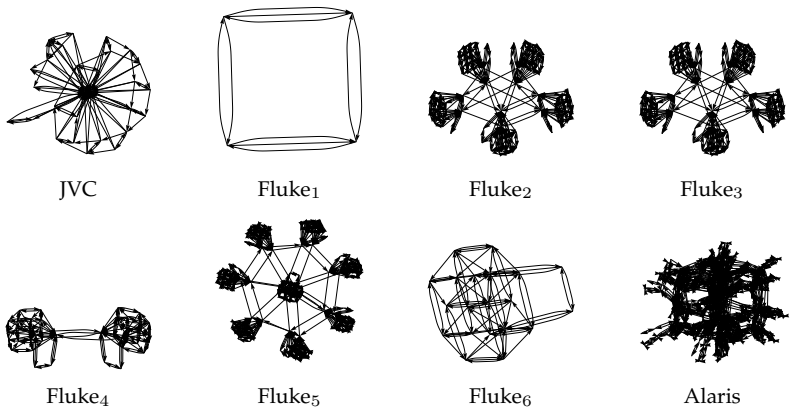

Figure 11. Transition diagrams for the case studies; top left is the PVR, bottom right is the infusion pump, and the other diagrams are for the meter. Although the diagrams are too small to see details, they clearly show the devices are structurally diverse, with the exception of the meter's ACV and DCV modes (Fluke$_{2,3}$ models, top right), which are isomorphic — as borne out by the identical results summarized in the table 1. Note that the diagrams are scaled to the same geometric size even though they have varying numbers of states.

The case study analyzes collectively suggest that if you want predictions of optimal task times to help improve a system design (i.e., to evaluate the underlying task structure and optimal timings imposed by the device), then it is sufficient to use $\Omega(n)$ where n is the lower bound on the number of button presses to achieve the relevant goal or goals. Ironically, the correlation is so good that going to the effort of analyzing Fitts Law times (which action graphs enable) does not seem worthwhile; the new approach is effectively zero-cost yet obtains results as good (in terms of correlations) as costly empirical experiments. Furthermore, the wide range of button layouts explored in case study 4 (described in appendix B.3) make negligible difference to the results, though rearranging physical button layout can be optimized to improve speeds by greater factors for *specific* faster tasks (Sears, 1993).

Low bound estimate of task times is approximately 0.2 s times the low bound on the number of button presses (0.2 being an unweighted average over the device models). Since the constant term varies from -0.14 to $+0.09$ (or, interestingly, to $+0.3$ for devices needing button holds) across the devices considered, the error in absolute time predictions will be high for low numbers of button presses. However, the error in predicting possible timing improvements during a device redesign will be low as the constant term is irrelevant.

6. POSSIBLE FURTHER WORK

Action graphs raise rich research issues that go beyond the scope of a single paper. We raise a small selection of future possibilities in the body of the paper, in this section, continued in appendix C for a variety of deeper issues (for an overview of appendix C, see section 6.6 below).

Case study	Section	Number of tasks	R^2	Lower bound time in terms of lower bound c of button presses
JVC	5.1	756	0.95	$0.073 + 0.18c$
Fluke_1	B.1	12	1.00	$-0.14 + 0.30c$
Fluke_2	B.1	$9,900$ $\begin{cases} 3,052 \\ 5,696 \\ 1,152 \end{cases}$	$\begin{cases} 0.96 \\ 0.97 \\ 0.97 \end{cases}$	$\begin{cases} 0.026 + 0.20c \\ 2.2 + 0.19c \\ 4.2 + 0.19c \end{cases}$
Fluke_3	B.1	$9,900$ $\begin{cases} 3,052 \\ 5,696 \\ 1,152 \end{cases}$	$\begin{cases} 0.96 \\ 0.97 \\ 0.97 \end{cases}$	$\begin{cases} 0.026 + 0.20c \\ 2.2 + 0.19c \\ 4.2 + 0.19c \end{cases}$
Fluke_4	B.1	$1,560$ $\begin{cases} 856 \\ 704 \end{cases}$	$\begin{cases} 0.95 \\ 0.97 \end{cases}$	$\begin{cases} 0.0064 + 0.20c \\ 2.2 + 0.20c \end{cases}$
Fluke_5	B.1	$25,440$ $\begin{cases} 6,760 \\ 16,664 \\ 2,016 \end{cases}$	$\begin{cases} 0.93 \\ 0.87 \\ 0.97 \end{cases}$	$\begin{cases} 0.059 + 0.19c \\ 2.3 + 0.18c \\ 4.2 + 0.19c \end{cases}$
Fluke_6	B.1	380 $\begin{cases} 348 \\ 32 \end{cases}$	$\begin{cases} 0.92 \\ 1.00 \end{cases}$	$\begin{cases} 0.013 + 0.20c \\ 2.0 + 0.24c \end{cases}$
$\text{Alaris}_{\textit{actual}}$	B.2	$51,104$	0.99	$-0.031 + 0.25c$
$\text{Alaris}_{\textit{packed}}$	B.3	$51,104$	0.99	$-0.025 + 0.25c$
$\text{Alaris}_{\textit{circular}}$	B.3	$51,104$	0.99	$-0.048 + 0.25c$

Table 1. Summary of all case studies and linear regression models. Regressions have been split for models $\text{Fluke}_{2\ldots5}$ to account for one or two 2 s button holds; the constant terms in the linear models are approximately $2n$ as expected, for $n = 0, 1, 2$ depending on how many times a button has to be held down for 2 s on the device. The three Alaris models are the actual manufacturer's and two hypothetical variants with contrasting keyboard layouts; see appendix B.3 and figure B.4 for details.

6.1 Basic generalizations of action graphs

Action graphs support many straight-forward generalizations, e.g.,

—To find the path (not just the time) the user takes corresponding to a lower bound time, arcs are labelled with the action n as well:

$$\mathcal{T}(G') \;=\; \{\langle v, m\rangle \xrightarrow{\langle f(m,n),n\rangle} \langle w, n\rangle \mid u \xrightarrow{m} v \xrightarrow{n} w \in \mathcal{T}(G)\}$$

and then use standard shortest path algorithms (Cormen, Leiserson, Rivest, & Stein, 1986). Note that there may be several different paths all with the same lower bound on time.

—More sophisticated models than the Fitts Law would allow for the user's perceptual and cognitive processing. For example, to use the Hick-Hyman Law (Seow, 2005) to add delays due to the varying complexity of choice users face, arcs may be labelled $f(m, n) + h(v)$ where $h(v)$ adds a time $c \log(\delta^+(v) + 1)$, where δ^+ is the non-trivial out-degree of v in G (the standard graph theoretic out-degree for the total closure of G is a constant, the number of actions). An example of this approach is given in Soukoreff and MacKenzie (1995) for estimating timing bounds on soft keyboard use. The non-trivial out-degree

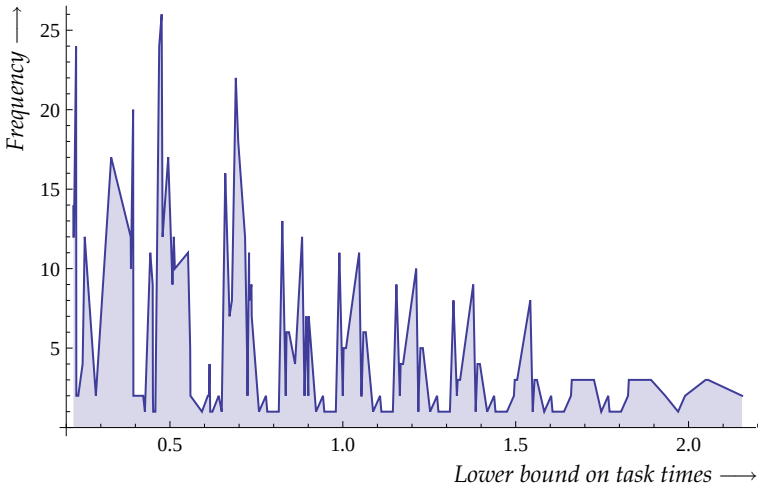

Figure 12. Graph of all lower bounds on task times from case study 1 (section 5.1). The designer should provide a justification for this design outcome, as at face value, the design appears to be suboptimal — since the integral of the graph is a proportional to the total number of possible tasks, increasing the frequency of some low-cost tasks would reduce the number of high cost tasks, and hence make the device more efficient to use on average. Once the designer has provided a cover story, which might simply be based on empirical weights of how often tasks are performed (and perhaps an auditor concurs), the design tool could hide these tasks from the visualization, to allow the designers to better focus their attention on other issues, for example on the relatively few, very slow tasks.

varies in soft-key applications; it can be reduced by using techniques from Thimbleby (2007a).

—We could analyze costs other than time: utility, saving money, energy, etc. A very simple example is that all arcs $\langle v, m\rangle \rightarrow \langle w, m\rangle$ represent the effects of user stutters (repetitions) in G.

—The approach obviously can be used to find properties other than lower bounds.

6.2 Handle weights in a design tool

With an appropriate design tool, one might handle weights and benchmark tasks sets interactively. Figure 12 shows a graph of *all* predicted task times from case study 1 (section 5.1). One imagines using a tool with the analyst moving a cursor over the graph and being told what the tasks are — perhaps the designer would be interested in very slow tasks? The designer would mark these tasks for improvement; or perhaps they would mark tasks with anomalous times as "OK" so they are deleted from the visualization to allow the designer to focus on more worrisome tasks. In this envisioning of a design tool, in the first place, *all* tasks would need to be considered, which is what the previous case studies did.

6.3 The data comes out at the end

An obvious use of the theory presented in this paper is to help improve system designs in an iterative design process aiming to improve user performance, yet

clearly the case studies were performed on commercial, finished devices, not on designs in progress.

Only now that it is established that there *is* a correlation, this opens the possibility of using lower bounds on action counts as a convenient design analysis metric in order to predict user timings. Whether and to what extent designers will find this use of the theory effective in improving real systems is an opportunity for further work.

6.4 A library of models

The biggest problem in undertaking analytical work of the sort pursued in this paper is knowing *exactly* what the device being analyzed is. The case studies in this paper were reverse-engineered, with the models being checked carefully against the physical devices and their user manuals. The unfortunately process takes a few months per device. Manufacturers and device designers, however, can instead use processes that ensure implementations are refined from known specifications — they have to do this anyway to build real devices.

6.5 Replication

It is a methodological weakness of the present paper that the theory, all the models, all the programs and all the experiments were implemented by a single author; replication and extension of this work would be welcomed.

6.6 Additional ideas

Action graphs open up a rich range of issues for further research, as well as stimulating further critical thought about both existing and new issues, collectively going way beyond the scope of a single paper. In addition to the ideas above, covered in sections 6.1–6.5, Appendix C discusses many additional ideas:

Appendix C.1	Cyclic movement.
Appendix C.2	Explore laws other than the Fitts Law.
Appendix C.3	Correlations of predictions aren't surprising.
Appendix C.4	Obtaining empirical timings.
Appendix C.5	Further studies of layout.
Appendix C.6	Soft buttons.
Appendix C.7	Using timing information for optimizing layout.
Appendix C.8	More than one finger.
Appendix C.9	Correlations of 1.0 in the case studies.
Appendix C.10	Abstracting data entry decreases correlation.
Appendix C.11	Geometric scale.
Appendix C.12	Transferring learned sequences.
Appendix C.13	Data uniformly weighted.
Appendix C.14	Automatically weighting "interesting" tasks.
Appendix C.15	The Fitts Law ignores other important timings.
Appendix C.16	The two-dimensional Fitts Law.
Appendix C.17	Exploring the cross over effect further.
Appendix C.18	The linear target task.
Appendix C.19	Confidence intervals.
Appendix C.20	Origins of variance.
Appendix C.21	Web use.
Appendix C.22	The paradox of the active user.

7. CONCLUSIONS

A skilled user's performance is limited by the optimal bounds on user performance, as determined by the device design. Usability depends on efficient use of interactive systems, and to design efficient systems requires analysis or evaluation of the time complexity of the designs with due considerations of relevant trade-offs, such as error rate.

This paper introduced action graphs and gave a theory and algorithm for obtaining lower bounds on task times. The work is placed within a standard mathematical framework, using graph theory. Once the framework is implemented (by combining this paper with standard algorithms), it has essentially no overhead in use, whereas empirical approaches always take organisation and participant time, so the marginal cost of analysis is negligible — in particular, during iterative design analyzes can easily be performed repeatedly to explore variations.

Models for lower bounds on time and count give *different* sequences of user actions to achieve the same tasks and thus need not necessarily be correlated, for the reasons discussed in section 2. We showed that when operating a pushbutton device, the best possible user time may not be the obvious

$$t = \Omega(\text{lower bound on count}).$$

We first illustrated the idea with a simple, illustrative, device (section 2), but for the various real devices studied, even though they have cross overs, the following linear relation applies

$$\text{lower bound user time} \approx 0.2 \times (\text{lower bound button count}) \text{ s}$$

We conclude it may not be worth estimating times (unless the application requires exact times) when lower bounds on button counts are easier to measure. Indeed, lower bounds on button counts for any task can be calculated with no uncertainty: the measures are objective and do not depend on particular users, training, or other experimental variables. *Lower bounds on button counts can be used to optimize user interface design for skilled use.* The caveat of course is that, for other domains than panel user interfaces, this result should first be checked — and this can be done using action graphs.

Of course, empirical work can still calibrate an analysis, and could do so incrementally over a period of time, replacing calculated values of f with actual user times for those transitions (which would of course then include thinking time, and other delays that KLM, GOMS, CogTool, etc, estimate), but, typically, absolute time differences are of no more value in design trade-off considerations than relative improvements, for which no calibration is necessary.

Further, we explored alternative button layouts. The erroneous intuition that button layout must have a large effect on task performance is possibly because we over-rate button location or we over-rate a few salient tasks, ignoring the larger number of alternative tasks that are affected in compensating ways by changing layout. Overall, the layout does not seem to matter much for a balanced portfolio of tasks.

An advantage of a rigorous approach such as this paper followed is that many assumptions are made explicit, and therefore now beg to be explored in further research. It is unsurprising, then, that this paper raises numerous points that deserve further examination; section 6 (which continues in appendix C) presents selected ideas for future work that build on the ideas and results presented here in the main paper.

8. NOTES

Author biography

Harold Thimbleby is a computer scientist researching interaction programming, how to design and program effective and dependable user interfaces, with a particular interest in interactive medical devices. He is Professor of Computer Science in the College of Science, Swansea University, Wales.

Acknowledgments

Ann Blandford provided the initial impetus for this work by claiming that action counts had little psychological relevance to behaviour. Duncan Brumby, Paul Cairns, Abi Cauchi, Stu Card, Andy Cockburn, Alan Dix, Parisa Eslambolchilar, Jeremy Gow, Wayne Gray, Michael Harrison, Tony Hoare, Andy Howes, Dick Pew, and numerous referees and editors made very useful comments.

Support

Work supported by the UK Engineering and Physical Sciences Research Council grants [EP/F020031, EP/F059116]. The author was a Royal Society-Leverhulme Trust Senior Research Fellow during this research, and gratefully acknowledges this support. All graphs were plotted by the author in *Mathematica*.

Author's present address

Post: FIT Lab — Future Interaction Technology Laboratory, Swansea University, Swansea, Wales, SA2 8PP; Email: harold@thimbleby.net; URL: harold.thimbleby.net

References

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Lawrence Erlbaum Associates.

Appert, C., Beaudouin-Lafon, M., & Mackay, W. E. (2004). Context matters: Evaluating interaction techniques with the CIS Model. In *People and computers XVIII* (pp. 279–296).

Bailey, R. W., Wolfson, C. A., Nall, J., & Koyani, S. (2009). Performance-based usability testing: Metrics that have the greatest impact for improving a system's usability. In M. Kurosu (Ed.), *Human centered design* (Vol. 5619, pp. 3–12). Springer.

Balakrishnan, R. (2004). "Beating" Fitts' law: Virtual enhancements for pointing facilition. *International Journal of Human-Computer Studies*, *61*(1), 857–874.

Beamish, D., Bhatti, S. A., MacKenzie, I. S., & Wu, J. (2006). Fifty years later: A neurodynamic explanation of Fitts' law. *Journal Royal Society Interface*, *3*(10), 649–654.

Bi, X., Smith, B. A., & Zhai, S. (2010). Quasi-Qwerty soft keyboard optimization. In *ACM CHI'10: Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 283–286). New York, NY, USA: ACM Press.

Bootsma, R. J., Fernandex, L., & Mottet, D. (2004). Behind Fitts' Law: Kinematic patterns in goal-directed movement. *International Journal of Human-Computer Studies*, *61*, 811–821.

Card, S. K., English, W. K., & Burr, B. J. (1978). Evaluation of mouse, rate controlled isometric joystick, step keys and text keys for text selection on a CRT. *Ergonomics*, *21*, 601–613.

Card, S. K., Moran, T. P., & Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, *23*, 396–410.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Lawrence Erlbaum Associates.

Cardinal Health. (2006). Alaris GP Volumetric Pump. (1000DF0009 Issue 3).

Cockburn, A., Gutwin, C., & Greenberg, S. (2007). A predictive model of menu performance. In *ACM CHI'07: Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 627–636). New York, NY, USA: ACM Press.

Cohen, I. B., Nauenberg, M., & Smith, G. E. (2008). *A guide to the Principia Mathematica*. Folio Society.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (1986). *Introduction to algorithms* (2nd ed.). Boston, Massachusetts: MIT Press.

Dix, A. J. (2010). Human-computer interaction: A stable discipline, a nascent science, and the growth of the long tail. *Interacting with Computers*, *22*, 13–27.

Drewes, H. (2010). Only one Fitts' Law formula — please! In G. Fitzpatrick & S. Hudson (Eds.), *ACM CHI'10: Proceedings of the SIGCHI conference on*

Human Factors in computing systems (pp. 2813–2822). New York, NY, USA: ACM Press.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, *47*, 381–391.

Fu, W.-T., & Gray, W. D. (2004). Resolving the paradox of the active user: stable suboptimal performance in interactive tasks. *Cognitive Science*, *28*(6), 901–935.

Fu, W.-T., & Pirolli, P. (2007). SNIF-ACT: A model of user navigation on the World Wide Web. *HCI Journal*, *22*(4), 355–412.

Gajos, K. Z., & Weld, D. S. (2004). SUPPLE: Automatically generating user interfaces. In *IUI'04* (pp. 93–100). New York, NY, USA: ACM Press.

Gimblett, A., & Thimbleby, H. (2010). User interface model discovery: Towards a generic approach. In G. Doherty, J. Nichols, & M. D. Harrison (Eds.), *Acm sigchi symposium on engineering interactive computing systems, eics 2010* (pp. 145–154). ACM.

Gray, W. D., & Boehm-Davis, D. A. (2000). Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of Experimental Psychology: Applied*, *6*(4), 322–335.

Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world performance. *Human-Computer Interaction*, *8*(3), 237–309.

Grossman, T., Kong, N., & Balakrishnan, R. (2007). Modeling pointing at targets of arbitrary shapes. In *ACM CHI'07: Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 463–472). New York, NY, USA: ACM Press.

Guiard, Y., & Beaudouin-Lafon, M. (2004). Fitts' law 50 years later: Applications and contributions from human-computer interaction. *International Journal of Human-Computer Studies*, *61*, 747–750.

Halverson, T., & Hornof, A. J. (2007). A minimal model for predicting visual search in human-computer interaction. In *ACM CHI'07: Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 431–434). New York, NY, USA: ACM Press.

Hertzum, M., & Jacobsen, N. E. (2001). The evaluator effect: A chilling fact about usability evaluation methods. *International Journal of Human-Computer Interaction*, *13*(4), 421–443.

Howes, A., Lewis, R. L., & Vera, A. (2009). Rational behaviour under task and processing constraints: Implications for testing theories of cognition and action. *Psychological Review*, *116*(4), 717–751.

John, B. E., & Salvucci, D. D. (2005). Multi-purpose prototypes for assessing user interfaces in pervasive computing systems. *IEEE Pervasive Computing*, *4*(4), 27–34.

Kieras, D. E., & Meyer, D. E. (2000). The role of cognitive task analysis in the application of predictive models of human performance. In J. M. C. Schraagen, S. F. Chipman, & V. F. Shalin (Eds.), *Cognitive task analysis* (pp. 237–260). Lawrence Erlbaum Associates.

Kieras, D. E., Wood, S., & Meyer, D. (1997). Predictive engineering models

based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *ACM Transactions on Computer-Human Interaction*, *4*(3), 230–275.

Knuth, D. E. (1997). *The art of computer programming* (3rd ed., Vol. 1). Addison Wesley.

Kvålseth, T. O. (1980). An alternative to Fitts' law. *Bulletin of the Psychonomic Society*, *16*(5), 371–373.

Lacaze, X., Palanque, P., Navarre, D., & Bastide, R. (2002). Performance evaluation as a tool for quantitative assessment of complexity of interactive systems. In *Proceedings of the 9th international workshop on interactive systems. design, specification, and verification* (Vol. 2545, pp. 208–222). Springer.

MacKenzie, I. S. (1991). *Fitts' Law as a performance model in human-computer interaction*. University of Toronto: Toronto, Ontario, Canada.: Doctoral dissertation.

MacKenzie, I. S. (1995). Movement time prediction in human-computer interfaces. In R. M. Baecker, W. A. S. Buxton, J. Grudin, & S. Greenberg (Eds.), *Readings in human-computer interaction* (2nd ed., pp. 483–493). Kaufmann.

MacKenzie, I. S. (2003). Motor behavior models for human-computer interaction. In J. M. Carroll (Ed.), *HCI models, theories, models and frameworks* (pp. 27–54). San Francisco, CA: Morgan Kaufmann.

MacKenzie, I. S., & Buxton, W. (1992). Extending Fitts' Law to two-dimensional tasks. In *ACM CHI'92: Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 219–226). New York, NY, USA: ACM Press.

Matessa, M., Remington, R., & Vera, A. (2003). How Apex automates CPM-GOMS. In *Proceedings of the fifth international conference on cognitive modeling* (pp. 93–98). Bamberg, Germany: Universitas-Verlag Bamberg.

McGuffin, M. J., & Balakrishnan, R. (2005). Fitts' Law and expanding targets: Experimental studies and designs for user interfaces. *ACM Transactions on Computer-Human Interaction*, *12*(4), 388–422.

Meyer, D. E., Abrams, R. A., Kornblum, S., Wright, C. E., & Smith, J. E. K. (1988). Optimality in human motor performance: Ideal control of rapid aimed movements. *Psychological Review*, *95*(3), 340–370.

Newell, A., & Card, S. K. (1985). The prospects for psychological science in human-computer interaction. *Human-Computer Interaction*, *1*, 209–242.

Olafsdottir, H. B., Guiard, Y., Rioul, O., & Perrault, S. T. (2012). A new test of throughput invariance in Fitts' Law: Role of the intercept and of Jensen's Inequality. In *Proceedings of the bcs hci 2012 people & computers* (Vol. XXVI, pp. 119–126).

Pirolli, P. (2007). *Information foraging theory*. Oxford University Press.

Schedlbauer, M. J. (2007). An extensible platform for the interactive exploration of Fitts' Law and related movement time models. In *CHI'07 extended abstracts: Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 2633–2638). New York, NY, USA: ACM Press.

Sears, A. (1993). Layout appropriateness: A metric for evaluating user interface widget layout. *IEEE Transactions on Software Engineering*, *19*(7), 707–719.

Seow, S. C. (2005). Information theoretic models of HCI: A comparison of the Hick-Hyman Law and Fitts' Law. *Human-Computer Interaction*, *20*, 315–353.

Simon, H. A. (1996). *The sciences of the artificial* (third ed.). MIT Press.
Smits-Engelsman, B. C. M., Galen, G. P. V., & Duysens, J. (2002). The breakdown of Fitts' law in rapid, reciprocal aiming movements. *Experimental Brain Research*, *145*, 222–230.
Soukoreff, R. W., & MacKenzie, I. S. (1995). Theoretical upper and lower bounds on typing speed using a stylus and soft keyboard. *Behaviour & Information Technology*, *14*(6), 370–379.
Soukoreff, R. W., & MacKenzie, I. S. (2002). Using Fitts' law to model key repeat time in text entry models. In *Poster presented at Graphics Interface 2002.*
Soukoreff, R. W., & MacKenzie, I. S. (2004). Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI. *International Journal of Human-Computer Studies*, *61*(6), 751–789.
St. Amant, R., & Horton, T. E. (2007). Model-based evaluation of expert cell phone menu interaction. *ACM Transactions on Computer-Human Interaction*, *14*(1), 1–24.
St. Amant, R., Horton, T. E., & Ritter, F. E. (2007). Model-based evaluation of expert cell phone menu interaction. *ACM Transactions on Computer-Human Interaction*, *14*(1), 1.
Theus, M., & Urbanek, S. (2009). *Interactive graphics for data analysis*. CRC Press.
Thimbleby, H. (2001). Permissive user interfaces. *International Journal of Human-Computer Studies*, *54*(3), 333–350.
Thimbleby, H. (2007a). *Press on*. Boston, Massachusetts: MIT Press.
Thimbleby, H. (2007b). User-centered methods are insufficient for safety critical systems. In A. Holzinger (Ed.), *Lecture Notes in Computer Science* (Vol. 4799, pp. 1–20). Springer.
Thimbleby, H. (in press). Clarifying the Fitts Law. *ACM Interactions*.
Thimbleby, H., & Addison, M. A. (1996). Intelligent adaptive assistance and its automatic generation. *Interacting with Computers*, *8*(1), 51–68.
Thimbleby, H., & Gow, J. (2008). Applying graph theory to interaction design. In J. Gulliksen (Ed.), *DSVIS 2007: Engineering Interactive Systems* (Vol. 4940, pp. 501–518). Springer Verlag.
Thimbleby, H., & Oladimeji, P. (2009). Contributing to safety and due diligence in safety-critical interactive systems development. In G. Calvary, T. C. N. Graham, & P. Gray (Eds.), *ACM SIGCHI symposium on engineering interactive computing systems, EICS'09* (pp. 221–230). New York, NY, USA: ACM Press.
Zhai, S., Hunder, M., & Smith, B. A. (2002). Performance optimization of virtual keyboards. *Human-Computer Interaction*, *17*, 89–129.

APPENDICES

A. NOTATION AND BASIC THEORIES

A.1 The Fitts Law

Originally introduced by P. M. Fitts (1954), the Fitts Law has had considerable attention, such as Bootsma, Fernandex, and Mottet (2004); Cockburn et al. (2007); Schedlbauer (2007), and see MacKenzie (1991, 1995); Meyer et al. (1988) for literature reviews and experiments, and Guiard and Beaudouin-Lafon (2004) for a journal special issue. McGuffin and Balakrishnan (2005) discuss variable-size targets.

The Fitts Law is a psychomotor law that has been well-studied and found to be remarkably robust. The Fitts Law estimates aimed movement times, based on distance and target size. Since the seminal work of Card, English, and Burr (1978), the Fitts Law has been widely used in HCI: for example, given the position and size of buttons it provides an estimate of the lower bound on the time it would take a user to move their finger from one to the other, thus leading to design insights.

The Fitts Law is usually expressed in some form equivalent to

$$t = a + b \log(d/w + k),$$

where t is the time, d is the distance to move to the center of the target, w is the width of the target in the direction of movement, and a, b, k are constants, typically determined by a mixture of theory and best fit to experimental data. Many researchers, and certainly usability professionals, would generally take a, b, k direct from the literature rather than performing their own analyzes or experiments.

Some authors use superficially different formulations of the Fitts Law, such as $a + b\sqrt{d/w}$ (Meyer et al., 1988) or, more generally, $a + b(d/w)^n$ (Seow, 2005) and $ad^b w^c$ (Kvålseth, 1980). All that matters for the present paper is that the numerical values are sufficiently close to user performance — *how* the timings are computed is not our concern.

Different authors use different units and different logarithm bases, and some prefer to set $k = 1$ for theoretical reasons rather than treat it as a parameter in a best fit to empirical data. Soukoreff and MacKenzie (2004) suggest a normative approach and following ISO standard 9241-9; whereas Drewes (2010) argues, somewhat cynically, that experimenters choose a formulation of the Fitts Law to best suit their data; certainly, fixing k ensures that determining a and b from experimental timings simplifies to linear regression.

The numerical values of the constants depend on the scale, the units chosen and on the logarithm base. Many authors do not explicitly specify how the distance d is measured. In the present paper, d is always measured between target centers. While some authors use milliseconds for time, this paper uses SI units (seconds, metres) and natural logarithms (it doesn't affect the results).

There are various cases and issues to note:

—Setting $a = 0, b = k = 1$ and using $\log_2$, then $t = \log_2(d/w + 1)$. This is called the Index of Difficulty (ID or IOD) and may be thought to be measured in bits (because of the logarithm base); the general Fitts Law is then expressed as $t = a + b \times \mathrm{ID}(d, w)$. The ID is a uniform comparison measure as it involves no participant-dependent or experimental constants; in fact, Fitts Law $= \Omega(\mathrm{ID}(d, w))$. However, the notion of ID as a simple information theoretic measure is problematic (Beamish, Bhatti, MacKenzie, & Wu, 2006), though this is a topic beyond the scope of this paper.

—The base of logarithm varies from author to author, but this is accommodated by a linear change in b, since for any two logarithm bases m, n we have $\log_m x = \log_n x \times \log_m n$.

—Setting $a = 1, b = 0$ gives a constant $t = 1$ to press a button. This special case effectively counts button presses, since it assigns 1 s per user action.

—Setting $a = 1.1, b = 0$ gives an empirically-based simplification of the Fitts Law, as used in KLM (the keystroke-level model) of Card et al. (1980).

—If $d = 0$ the user must be repeating the same action without any intervening movement (other than raising and lowering the finger), and generally a special-case value for t is chosen. Otherwise, a and b are chosen for a best fit to empirical data over all tested values of d. The algorithm presented in this paper has no problem with "special case" values — the algorithm merely needs a programmed function that calculates a time from user actions, such as

$$t = \begin{cases} a + b\log(d/w + k), & d > 0 \\ t_0, & d = 0 \end{cases}$$

For simplicity we will assume the function is a uniform Fitts Law regardless of d, and hence the time predicted for $d = 0$ is $t_0 = a + b\log k$, which for $k = 1$ is of course $t_0 = a$. This assumption is not uncontroversial (Soukoreff & MacKenzie, 1995; Soukoreff & MacKenzie, 2004, etc), primarily because experimental determinations of the parameters usually do not include an experimental $d = 0$ condition, and therefore using the law for $d = 0$ in such cases may seem to be an unwarranted extrapolation.

Thus with the constants used in this paper (see section 5.2) we take $t_0 = a = 0.165$ s. In fact, the literature suggests higher values for t_0, such as 0.195 s in Soukoreff and MacKenzie (2002), but it is important to note that in our case studies increasing $t_0 > a$ would increase the correlations we report (for example, for case study 3 in appendix B.2, R^2, the square of the linear regression correlation coefficient, would increase from 0.979 to 0.990); thus, for our research, the smaller value $t_0 = a$ used makes our results more conservative.

—Although the precision d/w is scale-free (e.g., the Fitts Law *formula* makes the time to acquire a target 1 cm wide after a 10 cm movement equal to the time to acquire a target 1 km wide after a 10 km movement) for large d and w, terms $\Omega(d - w)$ are required to better estimate timings. Our approach can handle this, for instance it can be applied without loss of generality to mobile or other user interfaces on geographical scales.

Finally, the Fitts Law is not without controversy beyond the scope of the present paper:

—Here, like other users, such as CogTool (John & Salvucci, 2005), we are only interested in the Fitts Law as an estimator of times, and all that matters is that a computer program can estimate times. Other authors are interested in the structure of the mathematical form of the law: for instance that if the logarithms are base 2, then the logarithm term measures bits; or they may worry that logarithms of fractions are negative, so steps are taken to compensate; or that if the term a is not zero, then the best fit seems to predict non-zero times for zero movements. These differences are a confusing handicap to the research literature (Olafsdottir, Guiard, Rioul, & Perrault, 2012); as hinted above, authors should make clear when they differ from ISO standard 9241-9.

—The name of the Fitts Law, too, is controversial. Some authors use *Fitts's Law*, others *Fitts' Law*. Perhaps because *Fitts' Law* sounds the same as the incorrect *Fitt's Law*, there are numerous errors in the literature. The present paper has used the compound noun form throughout, following the recommendations of Thimbleby (in press).

A.2 Lower and upper bounds

Of all possible ways of performing a task, if we have a system model we can find the optimal sequences of actions to perform the task. If a device is found to require at least n keystrokes to perform some task, then a user cannot perform it in fewer keystrokes. We are justified in saying n is a strict lower bound on the number of keystrokes for that task. However, for further insights we may also want to find bounds on timings — this is central concern of this paper.

Lower bounds on times means times a user *cannot* do a task faster; in reality a user will probably make errors (e.g., slips) or not know optimal strategies (e.g., they will make mistakes) so their actual times will be greater than the lower bound. Thus neither accounting for user errors nor ecological validity can reduce optimal times. The optimal times give a strict and useful low bound on the best times we can expect in real use. This is why it is useful for a designer to predict lower bounds on times, for they provide a limit on the best possible performance of any user under any circumstances.

Of course, a trivial low bound on the time for a user to do any task is 0 s; this, of course, is a *loose* low bound and not particularly insightful! In contrast, a *tight* low bound is more informative, and ideally highly indicative of the best times a user can achieve under conducive circumstances.

Upper bounds on times are generally less relevant to designers, because however well we might design an interface, a user could respond slowly and give arbitrarily poor times. In the case that the user has a real-time task (say, deploying landing gear on an aircraft) one would want assurance that the design upper bound on the user time was less than the lower bound on time for the aircraft to land. Determining upper bounds (not necessarily on time) may also be of use in financial and gambling applications, where a user should not normally be permitted to lose more than some predetermined limit, and the system they are using may be designed to enforce this.

Card et al. (1983) use the terms *fastman* and *slowman* in a superficially similar way. In their approach, empirically-based parameters are chosen to create families of predictive models of human performance; the fastman parameters are chosen to give best performance, and slowman worst performance. User performance should lie between slowman and fastman predictions. (So-called *middleman* predictions provide typical performance.) Although Card, Moran and Newell refer to these predictions as lower and upper bounds, they are merely convenient points on a distribution of possible predictions; perhaps 5% of users excel the predictions of the fastman models but we might not consider this significant.

Kieras and Meyer (2000) introduce *bracketing* to pair the fastest possible speed and the slowest reasonable speed. Thus reasonable user behavior should fall within the brackets. This approach contrasts with lower and upper bounds, where *any* user behavior, whether reasonable or not, *must* lie between the given bounds.

A.3 Mathematical notation for bounds

This paper uses the standard mathematical notation "big Omega" (Ω) for expressing lower bounds on times. The Ω notation is related to "big Oh" (O) notation, which is very widely used in computer science (Cormen et al., 1986; Knuth, 1997) because we often want to design fast algorithms whose *worst* possible behavior is known; O is the notation to use here. In this paper, we are interested in the user interface's *best* case behavior, and for this Ω is an appropriate notation.

The Ω notation expresses low bounds on arbitrary functions. We say $f(n) = \Omega(g(n))$ when for some fixed n_0 and $k > 0$, $f(n) \geq kg(n)$ for all $n \geq n_0$. Although Ω notation is well-defined, it is idiosyncratic (e.g., it cannot appear on the left hand side of an equation) and requires careful use; further discussion is beyond the scope of this paper. Knuth (1997) is a very good reference.

We can say useful, precise things with Ω. For example, the time $t(n)$ a user actually takes based on the least number of actions n found from analysis of a device may be $t(n) = \Omega(n)$ because $\forall n \geq 1 \colon t(n) \geq 0.05n$ if users operating the device cannot possibly do actions faster than, say, once per 50 ms.

An advantage of Ω notation is that it does not make the value of the constant, k, explicit; it may even be unknown.

A.4 Basic graph and state machine terminology

A full discussion of graphs and state machines can be found in Thimbleby (2007a). This paper uses graph theoretic and state machine terminology almost interchangeably, as follows:

State machine	Graph theory	Notation	Set
State machine	Directed multi-graph	G, G'	
State, mode	Vertex	$u, v \ldots$ and $1, 2 \ldots$	$\mathcal{V}$
Transition, action	Arc	$u \xrightarrow{a} v$	$\mathcal{T}$
Button	Label	$a, b \ldots$ and $A, B \ldots$	$\mathcal{A}$
Action sequence, task	Path	$u \rightsquigarrow v$	

A state machine is a fixed graph with a variable current state, s. If there is an arc from the current state s corresponding to an action a, then that action changes the current state to the adjacent state: the state machine performs the transition $s \xrightarrow{a} s'$ and transitions to state s', the new current state. In this paper, we assume every action is defined in every state, so an action b that "does nothing" must be represented as $u \xrightarrow{b} u$ for all states u. Of course, a program implementing a state machine may chose, as an optimization, not to represent arcs of the form $u \xrightarrow{b} u$, and test whether there is an arc before performing a transition.

An edge is a pair of vertices, and an arc is an ordered pair of vertices, but some authors prefer the term "directed edge" to arc, and in the context of a directed graph, the unqualified term "edge" obviously means arc.

Some authors use mode to mean a set of states sharing a common property (for example, that all vertices in the mode share a common transition). Thus, for these authors, "on" may be a mode, since for all states in a device's on mode, the action "off" may put the device into the state (or mode) off. The present paper does not need to make a state/mode distinction, though it is clearly important in "mode

confusion" discussions (since "state confusion" *within* a mode is unproblematic for the task at hand, as the confused states share the same properties and hence are equivalent for the purposes of the user).

Vertices and arcs are named with a bijection; in the paper, we tend to number particular vertices $1, 2, 3 \ldots$ and label arcs with button names from the letters of the alphabet, $A, B, C \ldots$; we also allow generalized arc labels, including weights — $\langle f(m,n), n \rangle$ being a case in point (as used in section 6.1).

Directed graphs are very easy to represent visually, drawing vertices as dots or circles, and arcs as arrows between them. Although visualizatons of graphs may easily become too complex to have any useful interpretive value, graphs can be processed and analyzed automatically and there is practically no limit to the complexity they can handle with very simple concepts. This is an important point since the only graphs we see and understand explicitly are very simple, and one might mistakenly conclude that graphs have to be simple!

Similar approaches are covered in Thimbleby (2007a); Thimbleby and Addison (1996); Thimbleby and Gow (2008); Thimbleby and Oladimeji (2009).

A.5 Least cost algorithms

We explain how least cost paths algorithms work in order to show that they are inadequate for working with the Fitts Law or otherwise taking account of geometry (or indeed physical interlocks, etc).

Suppose the cost of A is 1 and the cost of B is 2, then how do we find the least cost path from state 1 to state 4 in the device illustrated in figure 1 (see main paper)? Many algorithms can find least cost paths in a graph; the Floyd-Warshall algorithm (Cormen et al., 1986) starts with a representation of the graph as a matrix. Matrices are values organized into rows and columns; so we put the cost of the arc $u \rightarrow v$ into position at row u and column v:

$$\begin{array}{c} \textit{Costs} \\ \begin{pmatrix} \infty & 1 & \infty & \infty \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & \infty \end{pmatrix} \end{array} \qquad \begin{array}{c} \textit{User actions} \\ \begin{pmatrix} - & A & - & - \\ - & - & A & B \\ - & - & - & A \\ - & - & - & - \end{pmatrix} \end{array}$$

The cost matrix shows costs for getting between *all* pairs of vertices, and where there isn't an arc between vertices the cost is ∞ (because there is no single step to do it). In this example the costs are arbitrary numbers for illustrating times for taking single steps, but by comparing costs with the adjacent user action matrix we can see that we have chosen to give the same cost to the same action regardless of state.

Typically we zero the diagonal, as there is no cost associated with staying in the same state. Given a cost matrix with a zero diagonal, then, and using the Floyd-Warshall algorithm (or equivalent) we obtain the least costs for the shortest paths between all pairs of states:

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

Thus the least cost path $1 \rightsquigarrow 4$ has cost 3; the user could do this with the actions AAA or AB, the overall cost being the same whichever way they go.

Some entries are *still* ∞, meaning that it is impossible for a user to get between these states: for example, there is no path from state 4 to any other state. Thus the example is not strongly connected, a situation that may be considered an error for many interactive devices, but it may also be inevitable — for instance, with a fire extinguisher, once it has been used (reaching state 4) there can be no going back to "unuse" it; see Thimbleby (2007a).

The problem with trying to use this technique with timing costs from the Fitts Law is that there is no single step cost matrix. Times depend on where the user's fingers were *before* they do an action; hence costs cannot be functionally associated with arcs.

Consider, if A is the action "pressing button A," when the user presses A then their finger is at the location of A, but the current state only defines what they can now do having pressed A. The current state defines what they can do, but the time it takes them to do this depends on how they got to the current state, a step before the current action. The cost matrix above cannot represent this information.

B. ADDITIONAL CASE STUDIESS

The first case study was a personal video recorder, and was described in section 5.1.

B.1 Case study 2 (digital multimeter)

The second case study is quite different to the consumer PVR of case study 1. The Fluke 114 is a professional measuring instrument, a digital multimeter with five buttons and a 7-position knob. Our graph model of it, taken from Thimbleby (2007b), has 425 states and 4,250 arcs, so it is much more complex than the PVR. Our model is accurate but ignores certain "start up" modes that the Fluke 114 allows, such as disabling beeping (which is achieved by holding a button down while switching on); such features do not affect the behavior relevant to this paper, as they are not relevant to hand or finger movement during use. The buttons are both circular and rectangular, another difference to the PVR. Their shape and position are modeled to within 0.1 mm (see figure B.1).

Each of the knob positions effectively creates a distinct device with its own behavior, one for each of the different sorts of measurement the meter supports, so we can treat it as six separate devices. We ignore the knob position "Off," as this is a trivial "device" with only one state the user cannot do anything with! Treating the meter as a collection of devices also allows us to ignore the confounding timing issues of turning knobs, and it provides six correlations rather than a single aggregate value. We thus have case studies Fluke_1 to Fluke_6, and as figure **??** makes clear, these are interestingly different: fluke_1 is an automatic voltage low-

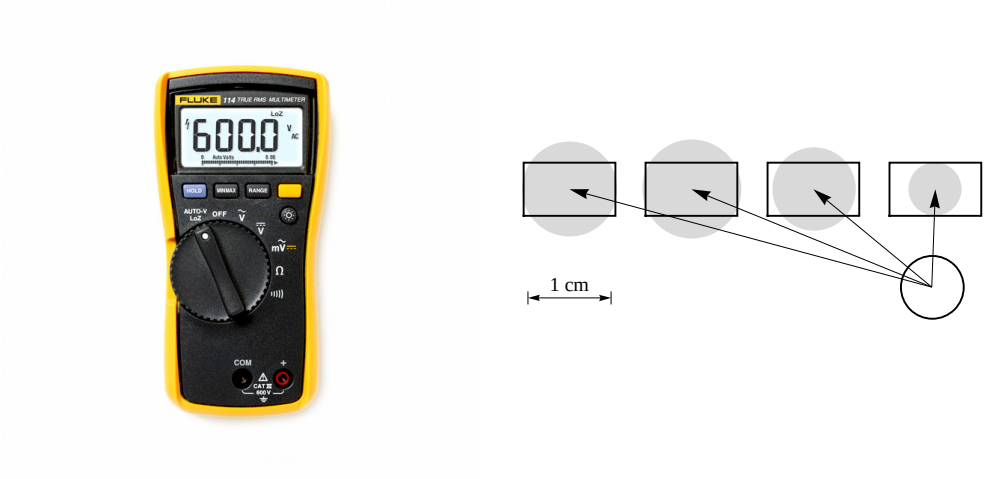

Figure B.1. The Fluke multimeter and a schematic of its button layout (a row of four rectangular buttons and a single circular button beneath them), also illustrating how effective target size (gray circles) depends on the direction of travel when moving from the lower-right button.

impedance mode; fluke_2 and fluke_3 measure volts AC and DC (and are otherwise identical); fluke_4 measures mV, so its ranges are constrained — the symmetry in its state diagram arises because of the similarity between measuring mV AC and mV DC; fluke_5 measures resistance, so its ranges are just Ω, kΩ and MΩ; and fluke_6 measures continuity. These "devices" are operated in different ways, but share the same button layout.

Repeating the same methodology as for case study 1, we obtain correlations of $R^2 = 1.0, (0.96, 0.97, 0.97), (0.96, 0.97, 0.97), (0.95, 0.97), (0.93, 0.87, 0.97), (0.92, 1.0)$, where the bracketed correlations arise because some tasks require pressing and holding some buttons for 2 s, and the correlations have been split into groups depending on how many 2 s button holds were required. Figure B.2 illustrates how three separate linear correlations arise for one model. The two cases of $R^2 = 1.0$ arise because the device has tasks requiring for their optimal solutions pressing only either one or two button presses; it is therefore possible to get an exact linear fit regardless of the Fitts Law timings.

B.2 Case study 3 (infusion pump)

The Cardinal Health Alaris GP volumetric infusion pump (Cardinal Health, 2006) is an interactive medical device with 14 buttons, designed to provide patients with controlled delivery of drugs. The Alaris GP has round buttons, though curiously they look like ellipses — the graphical design of the touch sensitive area is deceptive. We model the device thoroughly using an interactive *Mathematica* program with a realistic graphical animation (figure B.3) that allows user testing to confirm the program is an accurate interaction simulation; we then use discovery (Thimbleby & Oladimeji, 2009; Gimblett & Thimbleby, 2010) to derive a graph

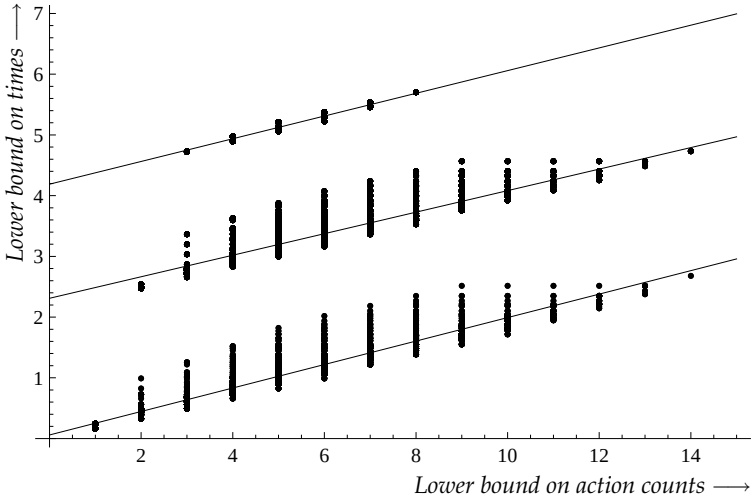

Figure B.2. Data obtained using the same methodology as in figure 9 (Fitts Law timings, vertically, against action counts, horizontally), except that the Fluke multimeter has 2 second holds for some actions. It is clear from the plot (which shows all data from the Fluke$_5$ model; see table 1) that all possible tasks can be achieved with $0, 1$ or 2 hold actions. Best-fit lines have been drawn considering each case separately, which involve $N = 6,760$ (no hold), $16,664$ (one hold), and $2,016$ (two hold) possible tasks. There are fewest sequences with 2 holds as relatively few tasks require such complex and slow operations — the device appears to be well-designed in this respect. Note that the regression lines look to have poor fit: this is because many data points are nearly coincident and hence indistinguishable — see figure 10 for further discussion.

model.[3]

The Alaris allows the user to enter fractional numbers, such as the infusion rate; thus the state space, treating each number as distinct, is enormous. To keep the model to a manageable size, we treat each of the numbers a user can enter (e.g., drug volume to be infused) as either 0 or 1, representing zero or any value not zero, respectively. Even so, the Alaris GP is a non-trivial device: it is modeled with a graph G of 352 states and $4,928$ (i.e., 352×14) arcs (ignoring internal transitions in the model that the user cannot activate), producing an action graph G' of $4,928$ vertices and $68,992$ arcs.

This abstract model of the Alaris GP allows us to study tasks such as "enter any setting" rather than "enter 23.4 mL per hour," for example, requiring specific numbers; the Fitts Law analysis as done for this paper is therefore abstracting timings for entering different numbers, but not timings for going through the various modes to enter all required numbers, or to stop, start or hold infusions, the primary tasks for the device.

We reprogrammed the analysis directly from the descriptions provided in this paper as an efficient C program running on a 2.7GHz processor, and it took 12

[3]The Alaris pump is serial number 800606589. Unusually for a safety-critical device it uses volatile memory, and after a depleted battery incident it became non-functional. Subsequently, Cardinal Health upgraded the Alaris GP user interface, so our simulation is now obsolete relative to available products and can no longer be checked against a physical device.

Figure B.3. A screenshot of the Cardinal Health Alaris GP volumetric infusion pump simulation. Note the 14 buttons, including 3 soft buttons just under the LCD screen. As shown, a yellow alarm light has illuminated (top left on the device) indicating an error condition — for instance that the pump has run out of drug or detected air in the drug line.

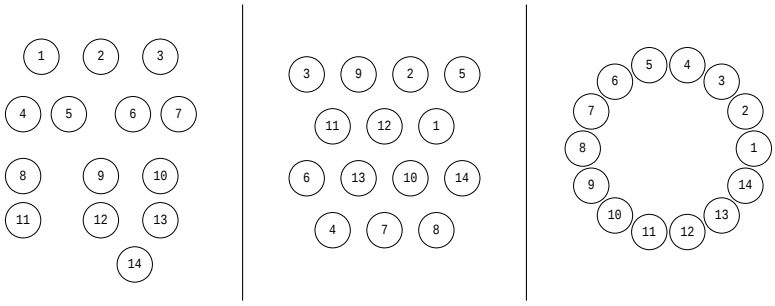

Figure B.4. Three alternative button layouts, drawn to scale, for the Alaris GP infusion pump: Alaris$_{actual}$ (compare with figure B.3), Alaris$_{packed}$ with button positions randomized and closely packed, and Alaris$_{circular}$. The same-numbered buttons in each layout have the same effects.

minutes to compute results. Repeating the same methodology as for the previous case studies, we obtained another high correlation $R^2 = 0.98$ ($N = 51,104$).

As with the other case studies, t_0 (the Fitts Law prediction for times with zero movement) was taken as $a = 0.165$, since $b \log(d/w + 1)$ is zero when d is zero. As expected, then, increasing t_0 to 0.195 s, from Soukoreff and MacKenzie (2002), *increases* R^2 from 0.98 to 0.99.

B.3 Case study 4 (infusion pump with modified key layouts)

The most complex case study is the Alaris GP infusion pump, case study 3. It is interesting to make radical changes to the button layout, and repeat the analysis for two additional button layouts. Figure B.4 shows the original button layout and the two alternatives, Alaris$_{packed}$ and the Alaris$_{circular}$ — in contrast to the Alaris$_{actual}$ of case study 3. Results are very similar to case study 3, and are summarised in table 1 in the main paper.

C. ADDITIONAL FURTHER WORK AND CRITIQUE OF RESULTS

C.1 Cyclic movement

The Fitts Law breaks down for cyclic movement (Smits-Engelsman, Galen, & Duysens, 2002). This is a factor that was ignored in our analysis, and since cyclic times are faster, this *reduces* the lower bound estimates of timings and hence may be an important consideration.

C.2 Explore laws other than the Fitts Law

The Fitts Law is only one of many functions to estimate timings. We showed, in section 6.1, that Hick-Hyman can be used within the present approach for user interfaces with soft buttons (and hence a varying number of user choices). KLM (Card et al., 1980) would be straight forward to use, but more sophisticated approaches, such as ACT/R, assume a cognitive model (St. Amant, Horton, & Ritter, 2007), and as CogTool shows so well (albeit with linear paths), it can clearly be integrated into a action graph model.

The Fitts Law only models movement; it does not model perception or cognition. St. Amant et al. (2007) suggested that the Fitts Law consistently underestimates times against more sophisticated psychological user models, such as GOMS and ACT/R. GOMS could be introduced into the framework discussed here, but ACT/R is a different matter. The right trade-off is clearly a matter of detailed research.

C.3 Correlations of predictions aren't surprising

One might object that the results are correlations between action counts and *predicted* times, not *measured* times. There are several responses to this critique:

—The predicted times are computed lower bounds, and no empirically measured times can obtain lower results (except by disobeying the Fitts Law, which is unlikely). If we are interested in low bounds, for instance for studying skilled performance, this is exactly what is needed.

—The correlations are not correlations of anything with empirical user performance as such but between two properties of a device, action counts and timings. The correlations are higher than we expected because real devices do not support many tasks with cross over effects.

—Alternatively, one could say the correlations are between an abstract measure of a device and an empirically-based law, namely the Fitts Law. To the extent that the Fitts Law embodies real data (and of course it does), the correlations are significant correlations between action counts and measured times, albeit timings measured in an experimental context (in fact, rigorously generalisable) rather than on the particular devices used in the case studies.

C.4 Obtaining empirical timings

Rather than use the Fitts Law, one could obtain actual timings. A graph is readily simulated, allowing user experiments and recording of empirical times, rather than using calculated times for all transitions. This would be easy to integrate into the present graph model approach. A Java framework has been suggested

(Schedlbauer, 2007) but this does not provide a graph model; while Thimbleby (2007a) provides a general graph-based simulation framework, which could be used for further experiments.

C.5 Further studies of layout

Case study 4 (infusion pump) compared three very different button layouts, but even if we keep the geometry fixed (permuting the buttons, but retaining the original shape) there are $14! = 87,178,291,200$ different permutations of buttons. Exploring three layouts, even such very different layouts, might be criticised as hardly a representative sample!

In general, with n buttons there are $n!$ layout permutations, but in special cases there may be fewer than this:

—For a circle (more precisely, the vertices of an n-gon), the time taken does not depend on the orientation of the circle, nor which direction (clockwise or anticlockwise) the buttons are positioned. There would be $(n-1)!/2$ permutations to consider.

—Symmetries in the transition system (the simplest case being two buttons that do exactly the same thing) also reduce the number of permutations to consider.

From figure 9, the Fitts Law times can be seen to vary by about a factor of 2. This suggests that changing the physical layout of buttons could have a significant impact on timings. It remains to be studied whether there are significantly better layouts (even ignoring device semantics, such as it may be unwise to place the Off button centrally). It is possible, for instance, that any layout would have such a range of timings, and that the actual gains possible are less than expected.

Note that the work of Sears (1993) considered optimizing button layout, but he worked with considerably simpler systems.

C.6 Soft buttons

Our algorithm (section 4) trivially accommodates different values of parameters (e.g., the Fitts Law parameters a, b, k) depending on the design of the user interface: the same values need not be used across the design — button shapes, distances and hold options (i.e., $t_n \neq 0$) on some buttons may depend on the current state, and different actions may be realised by different types of controls (buttons, levers, sliders, etc). Devices with soft buttons (displayed graphically on screens) allow changing button size and location depending on the state of the device, perhaps with the intention of making pressing certain buttons in certain states harder or easier, as the case may be (Balakrishnan, 2004).

C.7 Using timing information for optimizing layout

Timing information can be used to search for efficient physical button layout (Sears, 1993; Zhai, Hunder, & Smith, 2002). We note, however, that layout optimization to date has used single-step sequential tasks or very simple systems, restrictions this paper goes significantly beyond.

C.8 More than one finger

A user can use more than one thumb or finger and, for example, might chose to press the digit closest to the target button; if so times would be reduced because distances are reduced. Text entry tasks and keyboard design (e.g., QWERTY and variations) make a case in point: users have quite linearised tasks (typing text) and all digits are normally in use (Bi, Smith, & Zhai, 2010). Collecting accurate data on how people really use devices would be insightful.

C.9 Correlations of 1.0 in the case studies

If all tasks on a device can be performed in one or two user actions with no choices (or all in only two or three and none in one, etc), then the data may effectively have only two points, and can therefore be perfectly fitted by a linear model, even though the underlying Fitts Law model is non-linear. Table 1 (in the main paper) provides two examples of this phenomenon arising in small models. However, this ceases to be a good explanation of correlations when there are thousands of data points, as in other models.

C.10 Abstracting data entry decreases correlation

The Alaris model (case study 3; appendix B.2) obtained correlation $R^2 = 0.98$, but the model abstracted data entry. In fact, this correlation would be worse had we used a full model of the Alaris GP, because in a complete model it would be efficient for the user to keep their fingers over the increment/decrement buttons as they increased/decreased a drug dose: they would be making fewer movements between different buttons, so the variance in their timings would be lower. By abstracting out different numbers (mapping them to $\{0, 1\}$ as described in appendix B.2), long sequences of the user adjusting numbers are not considered, and these sequences would have very high correlations because finger movement would be reduced making the Fitts Law times for many transitions constant.

Constructing a model that is expected to *decrease* the correlation but still achieving a high correlation tends to strongly confirm our claim that lower bounds on times are correlated with lower bounds on action costs.

C.11 Geometric scale

Perhaps our models have too little variability in $\log d/w$ for the Fitts Law to be really relevant? This is unlikely because the case studies are exactly the sort of pushbutton interfaces that the Fitts Law has historically been widely applied to throughout the HCI literature.

As a check on this, increasing the distances but not target sizes by a factor of 1,000 (way beyond the empirical calibration range of the coefficients) gives a lower correlation ($R^2 = 0.52$ instead of $R^2 = 0.96$ for case study 1), and one would indeed expect a lower correlation as the $\log$ term becomes more dominant. In other words the analysis works as shown at the right scale, and increasing scale decreases correlation (because the standard deviation of time over different physical paths increases logarithmically). However, even at this scale, the correlations of the minimum and maximum times are 0.88 and 0.85 respectively; essentially, the *bounds* on timings remain closely correlated even when the geometry is severely distorted.

C.12 Transferring learned sequences

Action graphs allow us to analyze interesting cognitive and interaction issues beyond the scope of this paper. Here are some examples.

So far as a user is concerned, figures 1 and 2 both accept the same sequences of actions making corresponding state transitions. In figure 1 we have $AAA \equiv AB$ and $\lfloor AAA \rfloor_T < \lfloor AB \rfloor_T$ and the user may learn and perhaps prefer the faster way; but in figure 2 we have $AA \equiv AB$ as well, and $\lfloor AA \rfloor_T < \lfloor AAA \rfloor_T$, so there is an even faster sequence the user may overlook if they transfer their learning from their experience of the system represented by figure 1.

In general, then, it is possible that a user will learn faster sequences of actions (e.g., from the system of figure 1) that do not transfer to other tasks or devices, with the consequence that the user will unwittingly use devices (e.g., that of figure 2) inefficiently.

Many cognitive effects may slow users down (Gray et al., 1993; Fu & Gray, 2004), but again it must be emphasized that slowing users down does not affect optimal behavior — optimal behavior is a property of the system, irrespective of a particular user's behavior. (See appendix C.15 for further discussion.) What would be interesting if any effects could speed up users, as this would challenge lower bound estimates, but to do so would require finding a way around the ballistic mechanisms underlying the Fitts Law, and this is surely not a cognitive issue. Interestingly, we will see later, at least for the variety of devices studied in this paper, that rearranging the keyboard (which does affect individual movement times) has little overall effect on optimal performance.

C.13 Data uniformly weighted

Since the correlations obtained are based on *all* possible tasks uniformly weighted, we cannot be accused of "cherry picking" samples that generate high correlations.

The user will rarely do all tasks equally frequently, and almost certainly not all state transitions. A user may be more or less likely to undertake some tasks than others, and a designer may be more concerned with a representative set of benchmark tasks rather than all possible tasks; under these circumstances the correlation may not hold up, as fortuitously the tasks the designer considers may be ones with low correlation, as discussed in section 2 of the main paper. However, it should be noted that the algorithm has no problem handling benchmark tasks or assigning non-uniform distributions of weights; indeed, one would imagine that a design analysis tool would do exactly this. Sears (1993) makes a similar point.

If we assume that devices are designed to be easier to use for frequent state transition paths, Zipf probabilities are a plausible distribution to apply (Thimbleby, 2007a): this very different weighting makes less than $\pm 1\%$ difference in R^2 for all the case studies.

C.14 Automatically weighting "interesting" tasks

In this paper's case studies, all tasks have equal weight, but clearly with such complex devices, some tasks are certainly going to be of less interest to a user than others. For example, with the PVR (case study 1, discussed in section 5.1), the only interesting tasks might be those that end in one of the following classes of state: off, playing, recording, timer-delayed recording. Or with the infusion pump

(appendix B.2), interesting tasks would end in one of the following classes of state: off, infusing, paused. It is thus easy to identify subtasks, such as entering times to record, since these do not terminate in any of the interesting states.

Even though one may disagree with the specific details suggested above, the crucial point is that there are very few interesting termination states, and therefore they can be enumerated explicitly before performing an automatic analysis. Hence, large classes of state transitions can be weighted so that results such as the paper presented give accordingly greater weight to interesting tasks.

If we give "uninteresting" tasks zero weight, the time to analyze a system is significantly reduced, since there is no need to examine any task with zero weight at all, and the analysis can skip it. The time to analyze a system would then drop from $O(n^2)$ to $O(n)$ — for large n this could be a useful gain.

C.15 The Fitts Law ignores other important timings

The case studies reported in this paper are based on a "pure" Fitts Law model, and they therefore show the correlation between user actions and a lower bound on task time. The simplest generalization would be to introduce other operators from KLM (Card et al., 1980) or from the GOMS models (Card et al., 1983), such as mental times; this would be trivial, and merely requires a per-transition modification of the constant a in the Fitts Law formula. As in CogTool (John & Salvucci, 2005), one could annotate the states with the types of operators required, or derive the operators from a generic description of states — though for the scale of devices considered here, this approach is impractical. More generally, an ACT/R model (Anderson & Lebiere, 1998) or EPIC model (Kieras et al., 1997) could be used, rather than using a simple formula — in fact, both *are* equivalent to formulæ, but they happen to be computed by running program code.

C.16 The two-dimensional Fitts Law

The Fitts Law considers the target width w to be measured in the direction of movement. This source of variability is often ignored; MacKenzie and Buxton (1992) demonstrate from their experimental data that accounting for shape is important in using the Fitts Law, as we did here.

Interestingly, MacKenzie and Buxton (1992) argue that the accurate 2D model is hard to use because it requires knowledge of the approach angle; they suggest doing manual calculations (e.g., for a specific scenario). The analysis done in this paper used a full directional 2D model, and of course our analysis aggregates every possible task as well, which would be impractical for a manual calculation.

Figures C.2 and B.1 exhibit variation in target sizes, as measured in the direction of finger travel. Figure C.1 shows a plot of Fitts Law times taking the geometry of buttons for case study 1 into account plotted against the Fitts Law times *ignoring* button shape, treating all buttons as fixed diameter circles. The correlation is still good ($R^2 = 0.87$), suggesting that ignoring button shape may be safe. The apparent conflict with MacKenzie and Buxton (1992) is to be due to their concern with isolated *actions*, as opposed to our primary concern with *unit tasks*, which are sequences of actions.

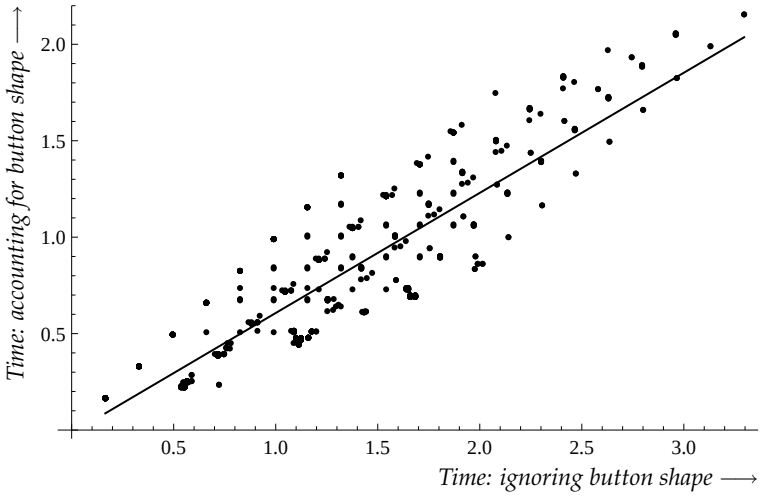

Figure C.1. A plot of the exact Fitts Law timings (y axis) against approximate Fitts Law timings, which ignore button shape (x axis) for all possible tasks on a PVR. The graph shows a best-fit line $y = -0.016 + 0.62x$ ($R^2 = 0.79$). The calculated times accounting for shape are generally faster than the times ignoring button shape (i.e., the slope of the best fit is 0.62, less than 1) as the direction-independent targets are taken to be circles of diameter the smaller of the (width,height) of the 2D rectangular buttons, and therefore the Fitts Law predicts longer times. Effectively, the horizontal axis corresponds to timings for circular targets that fit inside the rectangular buttons — see figure C.2. The high correlation shows that little is gained by correctly accounting for 2D shape and approach angle. Close analysis of the plot shows several independent but better correlated linear regressions (most with gradient 1): these correspond to tasks that do not use certain outlying buttons, and therefore the variance due to changes in button approach angle are reduced.

C.17 Exploring the cross over effect further

The cross over described earlier (in section 2) is evidently not a significant issue in determining user times for lower bounds on the timings of use of the real devices considered. The insensitivity to key layout is also a surprise. We now explore some reasons why these results were obtained.

The physical space that buttons occupy is limited, and to get a cross over requires buttons to be placed appropriately. A simple argument suggests that a device is unlikely to have many cross overs, meaning that cross overs are not a general problem in user interface design, though of course they may be very important for particular tasks, devices or actions.

Consider figure C.3. The figure shows a hypothetical arrangement of buttons in a 6×6 grid. Buttons A and B have been placed as far apart as possible in this grid, and for the sake of argument the scale of the button placement is such that two buttons are spaced just over 8 times their diameter ($6\sqrt{2} > 8.02$, as discussed above, is a requirement for the effect) — obviously a big enough grid has distances arbitrarily greater than 8 diameters, but 6×6 is conveniently easy to visualize. It is clear that if another button is placed anywhere on the grid that the distance from A to it cannot be greater than 8 diameters, so it will not exhibit the cross over.

The limitation of this argument is that it supposes the cross over is apparent

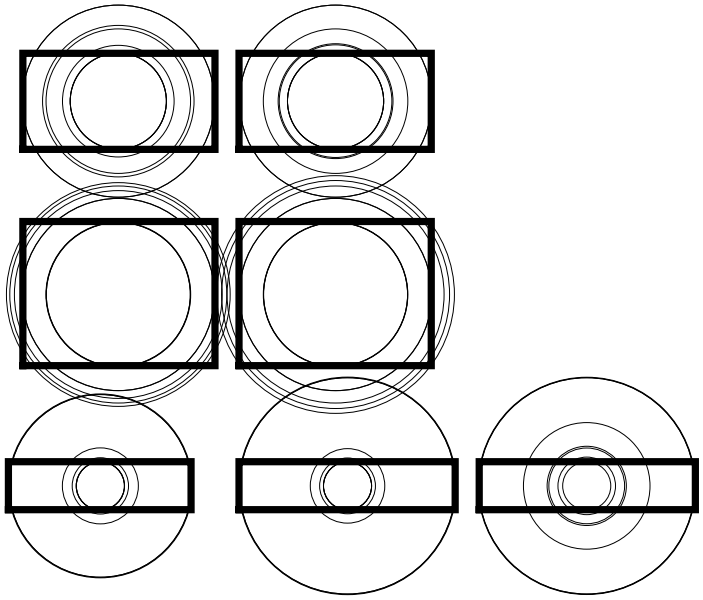

Figure C.2. Visualizing target size variability for case study 1. The rectangles are the buttons themselves, and the circles are not the targets but visualize the effective size of the target for each possible direction of travel from the center of every other button.

comparing one or two steps, as illustrated in figure 1. Nevertheless the argument generalizes to sequences involving arbitrarily many steps,

$$\begin{array}{l} a = \text{a sequence of } m \text{ buttons} \\ b = \text{a sequence of } n < m \text{ buttons} \end{array} \qquad a \equiv b, |a|_{\#} > |b|_{\#} \text{ but } \lfloor a \rfloor_T < \lfloor b \rfloor_T$$

The more steps that are needed, the faster the grid will run out of blank positions for introducing more buttons maintaining the desired effect! (Buttons could be made smaller to increase times, but the argument does not depend on the scale of the grid — and even with small buttons, one eventually runs out of physical space.) On a graphical user interface, and some devices with dynamic (e.g., illuminated) button labels, it is possible to have soft buttons that "recycle" space, as not all buttons need to be visible at the same time (Gajos & Weld, 2004), however this does not change the geometric trade-off — it just increases the number of buttons.

It is worth pointing out that any non-trivial device with 6×6 buttons is likely to have a very large state space: if all tasks take just 2 button presses, and the device is not significantly permissive (Thimbleby, 2001), then the state space would have size up to $2^{36} \approx 10^{11}$; for an $n \times n$ grid and b button presses, the state space is $O(b^{n^2})$.

Some user interfaces are designed to have large numbers of states that have no use: security systems have astronomical numbers of tasks, but all but one of the tasks amount to "block user without correct key." However, in this case the

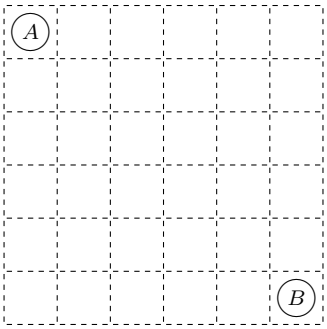

Figure C.3. A possible button layout placing the buttons A and B for the device represented in figure 1 as far apart as possible in a (mostly blank) 6×6 grid.

systems *are* strictly permissive because they allow a user without security clearance to be blocked in many ways — the purpose of permissiveness is to make user interfaces easy, and a security interface makes it easy to block unauthorized users. (If the security system required two correct button presses from the choice of 36, as above, it would only need 4 states.)

It follows that there will be *many* more tasks than the geometrically-limited opportunities to have cross over timings — provided that the buttons get a "fair" use in the state space (if some buttons are unused, the state space will be smaller and the proportion of cross overs could therefore increase). On real devices, since buttons cost money and cause visual complexity that might put off customers, unused buttons are generally avoided if possible; thus it seems most buttons will do things in most states, and therefore the general argument is likely to apply to real devices.

In short, to get a cross over there is a trade-off: the button geometry is extreme (e.g., figure 3) or the transition system is contrived (e.g., figure C.3). In the middle ground of this trade-off, the correlation between lower bounds on time and button counts are likely to be high: and this is indeed what we found with the real case studies.

For a special, linear, case similar to this analysis, see appendix C.18.

C.18 The linear target task

Imagine a thought experiment, the "linear target task" where the task is to press n target buttons B_1 to B_n in order (for example, left to right) with centers uniformly separated by distance d; the button layout is shown in figure C.4. The Fitts Law predicts the low bound on the mean time to complete the task is $n[0.165 + 0.075 \log(100d + 1)]$ using the parameters assumed earlier.

Figure C.5 shows a contour plot of the variation in time as a function of n and d. The contours show, as expected, that as distance increases, the variation due to changes in button distance becomes less significant than the variation due to changes in action counts. Similar plots can be obtained for the case studies.

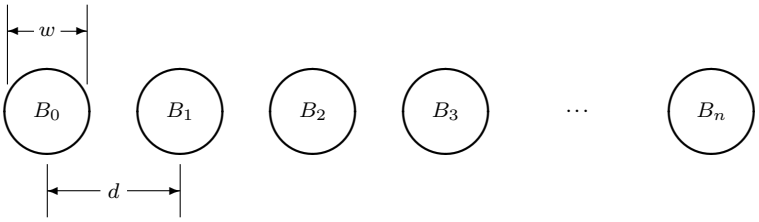

Figure C.4. The linear target task. Buttons diameter w are uniformly spaced along a straight line with separation d. Starting from the initial position B_0, the task is to press the n buttons B_1 to B_n in order.

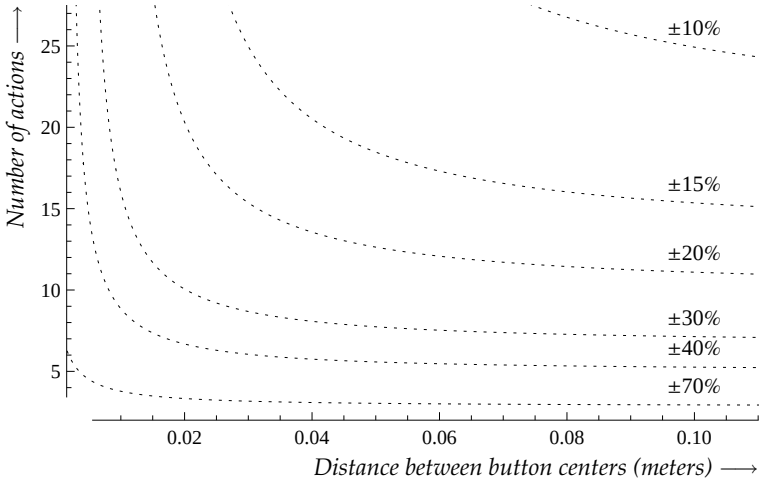

Figure C.5. Linear press task contour plot of relative variation in Fitts Law timings against distance $d \pm 0.005$ (the radius of the buttons is taken as 0.005 m) and variation $n \pm 1$ in task button counts. For example: over distance 0.035–0.045 m and a button count of 13–15 the relative change of the lower bound on the Fitts Law time will be about 20%; increasing the count n will have a larger impact than increasing the distance d, and increasing either will decrease the relative change. Overall, the graph shows that variation in button counts, that is, varying lower bounds on steps to perform a task, has a larger effect on variation on the Fitts Law timings than does variation in distance, but the effect decreases as distance increases. (The contours have been represented as dotted lines, as strictly they are only defined for integral n.)

C.19 Confidence intervals

This paper used the Fitts Law to help analyze and evaluate user interfaces, for instance by helping provide lower bounds on possible user timings for unit tasks, but experiments (on which the Fitts Law parameters are based) generally seek a best-fit result, and in some cases are more interested in reliable and general psychological insight than with specific design insight that can be used by designers and evaluators of *other* user interfaces under development.

Typically, experiments eliminate outlier participants, since their exceptional

behavior may be better explained other than by adherence to the law under study. If a participant was found to be unusually and highly skilled, say, this would be a good explanation of their outlying performance, and could be a reason to exclude their data; sometimes participants are dropped because their performance is several standard deviations outlying and therefore obviously not representative. On the other hand, to eliminate outliers reduces the value of the results for lower or upper bound predictive analysis, because some outliers may then outperform low bounds based on cleaned data.

There are always going to be some people who on occasion beat calculated tight lower bounds based on best-fits to experimental behavior (one might expect, say, 1% of people to outperform a calculated lower bound). An adequate statistical treatment of this confidence issue is beyond the scope of the present paper, but would be useful further work, particularly in guiding experimental design.

It would be unfortunate if a device was designed with required bounds on performance, which are then calculated with insufficient clarity on confidence levels, and then used in a context that somehow exacerbated outlier performance. An example would be predictions for mobile phone design, but which is then used as a secondary task for a car driver who does not have the visual attention resources available that experimental participants had in the original experimental conditions that calibrated the Fitts Law parameters used.

C.20 Origins of variance

As figures 9 and 10 make clear, the variance of the data varies with the number of button counts, seemingly, then, a heteroskedastic distribution. (Linear regression assumes the variance in the data *error* is a constant, and therefore a heteroskedastic distribution caused by varying error variance might invalidate a simple linear regression.) However, there is no error as both the x and y values are obtained from deterministic formulæ: the data represent exact values. The apparent variation occurs because a typical graph supports few very short paths and few very long paths, and for intermediate paths, there are a relatively larger number of differently-located buttons that can be pressed.

Because of the logarithmic dependence of the Fitts Law on normalized distance, the further apart buttons, the less variation in time will be due to any design variation in the distance between buttons. In contrast, the variation due to change in the number of button presses to complete tasks remains constant irrespective of the number of button presses required. It follows that if tasks take enough button presses, then more of the variation will be due to the variation in the number of button presses required rather than their location.

Consider a 3-button linear target task, as in figure C.6, and the variation as the middle button is moved between the two fixed buttons. A lower bound on the total skilled task time is the sum of the two Fitts Law times for distances d and $e - d$; if we choose $e = 0.1$ m, $w = 0.01$ m, the total time is $0.33 + 20.1 \log(11 - 100d) + 0.075 \log(100d + 1)$ s. As d moves from 0.01 through 0.09, the maximum range without buttons actually overlapping, the total time varies less than 7.4% even for such a large variation in distance. In all devices considered in this paper, e/w (the relative button spacing) is smaller, and the lower bound time variation possible with different button layouts would therefore be smaller still.

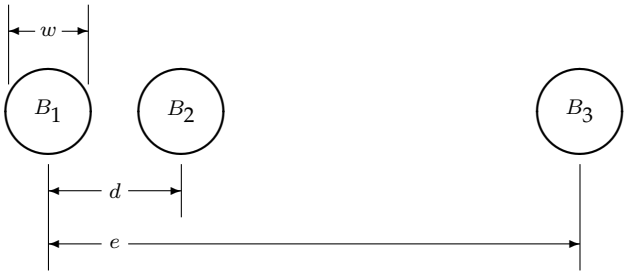

Figure C.6. A 3-button linear target task. Buttons diameter w are spaced along a straight line with separations d and $e - d$ between centers. The task is to press the 3 buttons B_1, B_2, B_3 in order. We consider the total task timings as the location d of button B_2 is varied from touching B_1 to touching B_3, that is over $w \leq d \leq e - w$.

C.21 Web use

The primary concern of this paper is physical devices, with physical buttons, covering consumer devices and industrial control panels. The web generally provides a very different user experience, often with buttons embedded in graphics or text. For some tasks and for some styles of interaction, then, it is nor surprising that layout *is* significant (Fu & Pirolli, 2007; Halverson & Hornof, 2007).

Web use is very different from using interactive devices with physical control panels, but the theory in this paper makes no distinction: users cannot do better than the theory this paper predicts, whether or not for web use, therefore their performance variation must be explained elsewhere; indeed, these papers include empirical data and are not concerned with just pure movement times. We would infer that typical web browsing is less sophisticated than interactive device use in terms of interaction paths through the graph, and that cognition, eye movement and other human factors not addressed by the Fitts Law must be a major factor on human performance. System response time, screen updates and other device factors may also be relevant. Clearly there is much further work to be done, particularly if the differences are to be explained quantitatively.

C.22 The paradox of the active user

One interest of the paper was to explore the contrast between optimal numbers of actions and optimal times, as these can result in the user taking different sequences of actions, which is an issue designers may wish to identify and then address during iterative design. However, it should be noted that an optimal user in this sense is different from an expert user, who may paradoxically use a system more slowly: this is the "paradox of the active user" (Fu & Gray, 2004).

The paradox of the active user should not be confused with the effects discussed in this paper (section 2), as its possible mechanisms are completely different. In this paper, by definition, an optimal user takes less (in fact, least) time or least number of actions. In the paradox of the active user, a user takes longer because they have learned a locally efficient procedure that does not generalise to some more complex task. The so-called paradox of the active user is that an ex-

perienced user may *not* be optimal. The explanation of the paradox of the active user in Fu and Gray (2004) involves the user seeing and interpreting the system display. The cross-over effect discussed this paper occurs regardless of the system display, and the display state has no role in the effects here.