Predictions of low bounds on skilled performance from state transition descriptions

Harold Thimbleby
FIT Lab—Future Interaction Laboratory
Swansea University, Swansea, Wales

Lower bound predictions of skilled human performance times on a large set of tasks can be obtained given only a state transition description of the device as input. The approach shows that, in the studied devices, and perhaps more generally, there is a remarkable correlation between the optimal number of actions required to perform a task and the Fitts Law predicted time: optimal task times and optimal action counts are correlated comparably to conventional experimental correlation of Fitts Law estimates and empirical user times. It follows that a description of neither the interface layout, nor the distance between buttons, is required for estimating low bounds on user timings. In other words, designers can now with some confidence (with hedges as discussed) modify and re-analyze designs to help improve task times prior to obtaining empirical data of actual timings.

We justify and provide in full an algorithm that gives optimal task times for any tasks, and which allows for user choice. The algorithm, based on the Fitts Law or other rules (e.g., information foraging laws), can be used early in the design process before empirical data is available or when only estimates are available.

A discussion of scope and validity of the contribution is also included.

Categories and Subject Descriptors: H.5.2 [User interfaces]: Theory and methods; D.2.8 [Metrics]: Performance measures; K.7.m [Miscellaneous]: Codes of good practice

Additional Key Words and Phrases: The Fitts Law, Finite state transition system, Optimal task time, User modeling, Validity.

1. INTRODUCTION

This paper defines a rigorous framework and shows how to predict optimal times for discrete interactive system use, and it shows that the characteristic path length of state transitions systems is a very good predictor of optimal times.

User evaluation is costly and often must be performed late in the design cycle once a prototype system is available to evaluate, and at this stage insights from evaluation 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? Insights into designs, to have any effect, are needed before resources become overwhelmed with the onslaught of requests from marketing and users—getting a good foundational structure for a design has to be done early, for it is costly to refactor later, and production pressures typically mean that companies ignore poor usability provided systems are good enough to be shipped. In many environments, improving usability has negligible priority once a system "works," and when a system appears functional it is unlikely to be revised even if revision could achieve usability gains.

Another problem is that complex designs may take inordinate amounts of time to evaluate thoroughly with users, which is a major problem for evaluating safety-critical systems. In some areas, such as aviation, laboratory evaluation of user interfaces is very costly, and repeated evaluation through a period of iterative design may be prohibitively costly. Once a system has approval (say for use as a medical device), revising it to improve its perfor-
mance may entail another costly approval process. A more subtle problem is that revising a design after it has been released may expose a company to lawsuits, since the company is implicitly admitting that the earlier versions were substandard in some sense that lawyers may exploit.

A essential complement to user evaluation, then, is to use predictive methods based on theory (generally informed with empirical data) to predict how a system is likely to behave under use but which work before it is ready for human use. Predictive methods can have significant influence on a design because they can be used earlier, more cheaply, faster and more often, when potential improvements to a design are cheaper to consider. Predictive methods are best applied to abstract, early, design models. Predictive methods are not only useful to help refine designs, but they may also be used to stop some poor designs even turning into reality.

As soon as a specification of an interactive system is available, in principle a state model of interaction can be obtained; certainly, as soon as program code is written, a state model can be obtained from the program automatically [37]. (We describe such state models below.) Web sites are interactive systems, and they provide state models directly from the links between pages; some interactive systems are simulated as web sites during prototyping, and this approach combines the two methods. From a state model, a designer can work out whether a user can perform certain tasks, and if so, how they are able to do the tasks. For example, the designer may wish to predict how a skilled user would use a proposed design, and then explore variations on the design to see how to improve it.

1 Systems like CogTool [19] are a great help for designers working from storyboards, but (to date) they only evaluate specific, sequential tasks composed of a few steps. This paper shows how to predict from any or all tasks, or from benchmark collections of tasks, and composed of any number of choices and steps. For example, from all possible ways of performing a task, we can find fastest times that a skilled user would not be able to surpass.

2 State machine models, which are available early in the design process, have hitherto been unsuitable for working with human performance models such as the Fitts Law. This paper shows how state models can be generalized to provide the required information. (Although we use the Fitts Law for concreteness, the technique can be used with other laws, or with any combination of laws.)

3 This paper shows that no correlation should be expected between optimal times and least numbers of actions, but using the techniques developed here, we show that for a variety of interactive systems that there is a good correlation between optimal times and least number of actions. This opens up the useful, and far easier, possibility of using least numbers of actions as a predictive tool in design, at least for discrete systems.

---

1 The law may be written “Fitts’s law” or “Fitts’ law” but these forms create ambiguities, especially when spoken, as to the mix of possessives and plurals: the usual forms also create division amongst pedants whether one writes Fitts’ or Fitts’s. Indeed, there are numerous references in the literature to a Fitt’s Law, which is incorrect, but which probably arose because of the ambiguity. Similar considerations apply to laws and principles due to Charles, Descartes, Stokes, van de Waals, and others. Writing in the unambiguous compound noun form such as “the Curie Principle” avoids the possibility that the phrase is misinterpreted as a principle due, in this case, to somebody called Curies (a possible name ending in s, but an incorrect back-formation) or due to the two Curies, Marie and Pierre. (Although Marie and Pierre famously worked together, in fact the Curie Principle is named after Pierre Curie.)
A further contribution is that we show how standard mathematical notation allows us to write estimates of times with precision at the same time making clear that predictions give a low-bound on times. The work in this paper justifies the use of this notation.

This paper thus builds a bridge between computation (here, state machines) and the usability of interactive systems, and provides a theoretical framework and an algorithm for supporting further research, as well as for integration into design tools. The core of this paper applies some standard mathematical notation and rigor to concerns of HCI but, for convenience, also provides Java code that can be run to obtain results without working through the details. By taking a rigorous approach this paper raises many further ways to consider analyzing task times; the paper explicitly raises a variety of research questions (particularly in section §7), some of which may be addressed starting from the simple Java code supplied.

1.1 Background

In 1985, Newell and Card [28, 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 I. Scott MacKenzie anticipated a future scenario [23]:

“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.”

The present paper justifies and develops an algorithm that automatically finds optimal times so the designer does not have to patch a story-board together or “work” the user interface as MacKenzie envisaged, though the designer can still obtain estimated times for particular sequences of user actions (e.g., from story-boarded sequences) if they wish. The importance of “automatically” becomes apparent when devices with thousands or more states, involving choices, cycles and so on—as when there are thousands of states, there are potentially millions of tasks (the square of the number of states); in the case studies, below, we will exhibit our algorithm on systems of thousands of states. The algorithm does not find minimum times based on error rates, since errors increase times, instead it finds a minimum or optimum time a user cannot better (assuming the validity of the Fitts Law). We will show that optimal times are correlated to optimal action counts. This result is surprising. Optimal action counts are very much easier to estimate, so this is a useful result.

Some related previous work, such as [3; 19], 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 [9] 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 published papers [21] only show single-step times, not times for arbitrarily long sequences of actions that this paper handles. Sometimes very detailed psychological models are used, which make good time estimates if the user behaves as predicted.
Psychological interest concerns experiments to estimate user times accurately, to explore psychomotor performance and behavior—and user errors, ecological validity, etc. Addressed as a design question, however, user performance is a given from which time estimates should be obtained from design specifications as soon as possible, preferably before anything is built, and without fixing the choices users make. To drive early and fast iterative design, there is a great need for quantitative predictions about user performance with designs well before actual experiments with users can be contemplated—the ideas introduced in this paper could, for example, be embedded in conventional program development tools to be used directly by developers.

1.2 Scope, methodology and validity

The mathematical analysis developed in this paper applies to discrete action user interfaces, but the ideas are actually more general than the particular psychomotor timing law that is used to illustrate its use (the Fitts Law, discussed below in section §1.4). For brevity, the scope of this paper is restricted to pushbutton or control-panel type user interfaces, whether physical or presented in a GUI; the standard territory of the Fitts Law. Discrete interactive devices may have widgets other than pushbuttons (e.g., knobs, sliders, changing-scale targets, dials, rotating combination locks, and in safety critical devices there may be features like flaps that must be lifted and other interlocks)—these are amenable to the approach but are not discussed. This paper thus uses the Fitts Law formula for concreteness and clarity and because of its robust experimental heritage, but the approach applies to any law or formula that depends on the user’s past and present actions, including generalizations of the Fitts Law [15].

The approach developed in this paper could use a more general model (such as GOMS [6] or ACT/R [2]); or we could use an arbitrary law more appropriate to some specific problem in hand—or analyze complexities other than time, such as cost, utility, saving money or energy. Information foraging theory has been developed on the premiss that rational users optimize outcomes relative to task times [29], and although this is not equivalent to purely optimizing task times (except for trivial systems), the algorithm developed in this paper can easily accommodate such a generalized cost, for instance taking the value to the user of a goal state divided by a Fitts or other time estimate.

Timing information can be used to search for efficient physical button layout [38]; this is an issue beyond the scope of this paper. We note, however, that layout optimization to date uses single-step sequential tasks, a restriction this paper goes beyond.

This paper could be criticized because it makes simplifying assumptions; but this is a criticism of any theoretical contribution. The point is that by abstracting from certain details (for example, the user’s cognitive abilities) allows us to reason rigorously about the details that we are concerned with. As it happens, this paper uses the Fitts Law; it is interesting to note that the Fitts Law works well even with users with cognitive and physical limitations [27], and therefore abstracting from such details is legitimate.

It is interesting to note that research methodology in HCI owes much to popular views of science stemming from Francis Bacon (and his ideas as later refined by John Stuart Mill) and is empirical: briefly, that since we do not know theories a priori, we should explore the world inductively to determine them. In contrast, Isaac Newton’s innovation was to start with the simplest of assumptions, rigorously explore the mathematical consequences, then turn to more complex real conditions [10]; the point being that if you start from the world, as Bacon recommends, you perhaps never achieve any clear reasoning or ability to predict,
Calculating optimal times

let alone know the limits of assumptions of predictions. Following Newton’s style, then, the methodology of this paper is to start with simple, clear mathematics with clear assumptions, and then to explore the consequences of those assumptions in real applications.

Validation and significance are important issues for any scientific contribution, though it is rarely discussed explicitly [34]. This paper makes several sorts of novel contribution, which may be assessed in different ways:

(1) The paper makes a purely theoretical contribution, significant for HCI. This can be validated by inspecting the mathematics, and reviewing the cited literature [39]; for example, we use standard mathematical notation (e.g., \( \Omega \) notation) and concepts (e.g., graphs). The significance is established by the literature (e.g., [23; 28]) envisioning this sort of contribution, as mentioned above.

(2) An important part of establishing the validity of conventional papers is the review of the literature and its clear presentation for the readers of the paper; the work should build on and extend existing work in a way that is appropriate to the field(s) reviewed. In the present paper, the literature review occurs particularly in sections §1.1 (Background), §1.3 (Time complexity), §1.4 (The Fitts Law), §1.7 (Algorithms for finding optimal times), and §2.1 (Conventional interactive device models). The unusual split of the literature review and its unusually detailed explanatory nature is because this paper combines and relies on background from several different contributing disciplines.

(3) We show that standard algorithms are inadequate for the optimal task time problem, and we show how to extend them (more precisely, we show how to reduce a suitably annotated finite state machine to a new data structure that standard algorithms can obtain optimal task times from). Program source code is provided that implements the reduction, and it can be used as an aid to further validation of the claimed results, that is, readers of this paper can run the code to test it, or to test its use on various designs of their own devising. However no validation of the code is offered as this paper is not a software engineering paper; the code is, in fact, quite straight forward. The significance of this algorithm is that it can be embedded in new or existing predictive modeling and analysis tools, which then makes the insights of the theory available to practitioners (figure 12 illustrates an example use of the theory for a design tool). Once embedded in product design tools, there will then be a new question of whether the tools using the theory rather than theory itself are valid for use in the field.

(4) The paper further claims that optimal times are correlated with optimal action counts, averaged over all tasks; that is, the majority of performance time variation within a set of tasks belonging to a device is caused by number of actions and not by the time required to move between buttons. Validating evidence is provided within the paper based on a number of published case studies (which were obtained from the web site of [31]). The correlations are robust and in the absence of any hypothesis as to why further case studies might be different, the validation is sufficient for this paper. Sufficient detail is provided in the paper for the approach to be replicated and hence validated with further case studies, hence supporting possible future work that might aim to circumscribe the present results.

(5) That the correlations occur with data obtained from an existing web site (albeit by the same author) lends more credibility to the claims; that is, the data was not concocted after the fact (or at the same time as the work reported in the paper) to establish the
correlations. Unfortunately we are unaware of other published data that can easily be used to extend the range of case studies.

(6) This paper uses the Fitts Law as an example psycho-motor theory, and the validity of some of the claims depend in turn on the validity of this law (for example, the correlation mentioned above is of no interest if the law itself is invalid). However, the Fitts Law is a robust result that has been widely validated elsewhere [27].

(7) A crucial aspect of validity is circumscription, that is, being clear about the domain of validity so that rigorous and specific predictions can be made only when it is appropriate to do so. It goes without saying that this paper is not valid as a piece of French literature, say, but it is a lot less obvious what sort of validity to which it may aspire within its own field, here, human computer interaction. Typically, circumscription is one purpose of the literature review and related discussion in a paper; but this paper also adds an explicit section (§1.2) on its scope, since much other work in the field addresses problems in a more discursive way and, hence, a focused scope such as this paper has is unusual. The work reported in this paper applies to finite discrete systems, and therefore, for example, it makes no claims about continuous tasks, like many gesture-based games that have no definite end point.

(8) No paper is perfect, and an actual or implied claim to perfection would surely undermine belief in validity, as would absence of any self-criticism! As is conventional, therefore, this paper critically reviews its claims and suggests some ways forward; see in particular section §6 (Further work).

(9) Bearing in mind the possibility that the paper reports experimental data derived from faulty programs, the entire process was round-tripped in an independent case study. The initial case studies (1–7) motivate the paper, but it is possible that their common programming hides some common-mode error, which, if present, could lead to consistent but invalid results. For the final case study (8), therefore, another interactive device was chosen as a model, and all programming was repeated in a new programming language, working directly from the descriptions given in the present paper—thus sharing no code with the previous case studies, from which the bulk of the paper has been written. With this final case study, we achieve very a similar result. This result lends additional confidence to the quality of the programming underlying all of the claims this paper makes.

(10) Finally, ethics is an important component of claims to validity. The present paper raises no external ethical issues, which is unusual in this field: work with human participants, the environment, etc, would normally raise ethical issues that reflect on validity (for example, were participants vulnerable or coerced to obtain better results?). Internal ethics covers issues such as fraud and professional ethics, such as prior publication and self-plagiarism. Here are some example ethical issues impacting validity for this paper:

—A preliminary paper [33] was presented at a specialist workshop, but was not disseminated beyond the participants. The present paper is a major development from that, in terms of explanation, clarity, and, importantly, generality.

—Figure 5 in this paper is superficially similar to figure 9.5 in [31], but the figure and results provided here use more accurate data and use a 2D version of the Fitts Law. The general case is not discussed in [31]; in fact, this short section of [31] was written as a brief summary of the original ideas behind the present paper, but...
by chance it was published earlier.

These different ways triangulate [34], collectively providing greater conviction of this paper’s validity. There are complimentary ways of achieving the same results: you can reason about the mathematics, you can run the program, you can obtain your own correlations from your own data. That these approaches are very different increases one’s confidence in the claims.

Although there is a strong tradition in the philosophy of science literature [8] and very occasionally from reflective papers that happen to be published in the topical literature [16], the points above are rarely discussed within actual research contributions reflectively; sadly, therefore, many papers only lay implied claims to their own validity—the danger being that their unexamined validity may be weaker than their readers (and authors) would like to believe. It is hoped that the nature of the explicit discussion above is an additional contribution of this paper to the field, and like all contributions, may itself be criticized, refined and developed in further work.

1.3 Time complexity

In user interface design we are often interested in designs that lead to or otherwise support and encourage the best task times for the user; indeed Card, Moran and Newell’s classic *The Psychology of Human-Computer Interaction* [6] argues that reducing expert time is a key principle of user interface design. Classic projects such as Ernestine were driven by the conviction that “time is money,” and therefore that it was worth redesigning user interfaces to make them faster to use [14]. Worst case times are less relevant to designers, because however well we might design an interface, a user could respond slowly and give arbitrarily poor times. Bailey et al. provides a recent review of the state of the art of usability testing and high-impact metrics [4].

The best case behavior of an interactive program is dependent on the least time a user could take using the interface optimally (for a sufficiently fast program this is effectively equal to the least time a user takes). Optimal times are error-free times; if a user makes any errors they will take at least the optimal time. Note that optimal times means times that a user cannot better; in reality a user will probably make errors (e.g., slips) or not know optimal strategies (e.g., mistakes)—thus their times will be greater and certainly never less than the optimal. Thus neither accounting for user errors nor ecological validity can reduce optimal times. In short, a user cannot respond better than the optimal times, so the optimal times give a strict and useful bound on the best interactive times we can expect in real use. This is why it is useful for a designer to predict optimal times, for they provide a limit on the best possible performance of any user whatsoever. There is considerable evidence that users optimize timings (e.g., [3; 13]), and eventually will treat optimal or nearly-optimal interaction as routine. Howes et al. [17] is evidence that optimal time is a reasonable prediction of actual skilled performance time: people are adaptive, and with practice they improve. Optimal performance, given the perceptual-motor system constraints that underpin the Fitts Law, defines the asymptotic bound on the improvement of skill.

Theoretical optimal times are important. A designer could compare real user behavior with theoretical times, and—if they are very different—seek ways to revise the design or to better train the user to exploit the existing design more effectively. If there are estimates of theoretical times, some laboratory work can be simplified, particularly for complex systems
that are hard to evaluate thoroughly. If theoretical times are reliable, some design iteration
can be done before any users are involved, very early in the design cycle. As we shall show
there is an algorithm for determining theoretical times, so finding optimal times could be
programmed into a design tool to enable any designer to use the insights of this paper
without knowing the detailed theory.

This paper uses the standard notation “big Omega” ($\Omega$) for expressing optimal times.
$\Omega$ notation expresses low bounds on functions: $f(n) = \Omega(g(n))$ when for some fixed $n_0$
and $k > 0$, $f(n) \geq kg(n)$ for all $n \geq n_0$. The $\Omega$ notation is related to “Big Oh” ($O$)
notation, which is widely used in computer science [11] because we often want to design
fast algorithms whose worst possible behavior is known; $O$ is the notation to use to reason
about worst case behavior. In this paper, we are interested in the user interface’s best case
behavior, and for this $\Omega$ is the appropriate notation.

The form $f(n) = \Omega(g(n))$, spoken as “$f(n)$ is big Omega of $g(n)$,” says that $f(n)$ is a
function that is equal to or at least a constant (positive) factor larger than $g(n)$ for all suffi-
ciently large values of $n$. Note that although $\Omega$ notation is well-defined, it is idiosyncratic
and requires careful use; for example, writing $\Omega(g(n)) = f(n)$ would be meaningless.

We are interested in the time $t(n)$ a user takes based on our knowledge of the least
number of actions $n$ found from appropriate analysis of the device. Let us suppose, by
way of example, that users cannot possibly do actions faster than every 50ms, then we
would have $t(n) = \Omega(n)$ because for all $n \geq 1$, $t(n) \geq 0.05n$. The advantage of the $\Omega$
notation is that it does not make the value of the constant (here, for instance, $k = 0.05$)
explicit; in general we don’t know exactly what $k$ might be as it can depend on the user,
their training and skill, and so forth. In short, $\Omega$ is helpful because it allows us to make a
precise statement without specifying some details. Moreover, if a device design has some
time complexity $\Omega(f)$ and if we can change it to $\Omega(f')$ where $f' < f$ then we have an
improved design with respect to timings. Our aim, then, is to provide an algorithm to
find $f$.

When analysing the time a non-interactive program running on a CPU takes, we can look
up in the manufacturer’s specification of the computer how long multiplication, addition
and other operations take, then calculate how long the program will take. Many processors
are pipelined and sometimes the calculation is quite complex. For interactive programs,
we are interested in the times individual user actions take, such as pressing buttons on a
keyboard. Just as with timing computer processors, there are many complexities: does
the user use one or two hands, do they spend time thinking before they know what they
should do, and should we also include the time it takes for them to see and interpret the
display . . . to mention just a few issues? There are very powerful and general techniques
for estimating user time using realistic models of user behavior, such as ACT-R/PM [2],
but these approaches, being based on simulations of the user, are unable to determine what
optimal use of a typical program is. That is, the behavior of the user and the behavior of
the program are intertwined.

1.4 The Fitts Law

Originally introduced by P. M. Fitts in 1954 [12], the Fitts Law has had considerable atten-
tion, such as [5; 9; 30], and see [22; 27] for substantial literature reviews and experiments.
Reference [26] provides a recent discussion and application for changing-size targets; and
[23] is a standard review—which calls the problem the present paper addresses an “em-
bedded model” but leaves it a “future possibility.”
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 a logarithmic trade-off between distance and accuracy in the user’s movements. Given the position and size of buttons, the Fitts Law makes an estimate of the time it would take a user to move their finger from one to the other. Since a user’s times will depend on their training, skills and physical abilities, the Fitts Law is often used to obtain an “index of difficulty” which is not subject-specific, and therefore is a more objective measure of the difficulty of doing operations.

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 target, \( w \) is the width of the target in the direction of movement, and \( a, b, k \) are constants, typically determined by a best fit to experimental data. Note that the constants can combine both user response and system response times, and they can vary with the device as well as with the user (or experiments). Different authors use different units and logarithm base, and some prefer to set \( k = 1 \) for theoretical reasons.

The numerical values of the constants depend on the units chosen and on the logarithm base, and although \( d/w \), the measure of the relative precision of the user hitting the target, is dimensionless, the constants \( a, b, k \) depend on the geometric scale. For example, although the relative precision \( d/w = 0.1 \) would be the same in both cases, the time to acquire a target 1cm wide after a 10cm movement is not going to be equal to the time to acquire a target 1km wide after a 10km movement! In this paper we use metric units and natural logarithms, and the scale will be limited to typical handheld device or control panel sizes: buttons of size around 1cm and movements typically less than 10cm.

There are various special cases:

— 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) and is measured in bits. The ID is a uniform comparison measure as it involves no subject-dependent constants; in fact, Fitts Law = \( \Omega(\text{ID}) \).

— Setting \( a = 1, b = 0 \) gives a constant \( t = 1 \) to press a button. This special case counts button presses rather than times user actions.

— If \( d = 0 \) the user must be repeating the same action without any intervening movement, and generally a special-case value for \( a \) is chosen. Otherwise, \( a \) and \( b \) are chosen for a best fit to empirical data over all relevant values of \( d \). The algorithm presented in this paper has no problem with “special case” values—the algorithm merely needs a function that gives a time from user actions, and for concreteness we assume the function is the Fitts Law.

— Some devices distinguish between just pressing a button (i.e., tapping it) and holding a button down for a few seconds. The Fluke multimeter considered later is an example of this: buttons can either be pressed briefly or held for at least 2 seconds, when they have a different effect. To allow for this, the Fitts Law formula is easily extended to \( t = h + a + b \log(d/w + k) \) where \( d/w \) is calculated as normal for a movement towards the relevant button, and \( h \) is 2 (i.e., for a button hold of 2s) or zero, depending on whether the button is held or not. The time to press and release a button is wrapped up in the experimental value for \( a \), so this is essentially unchanged. Figure 10 shows a plot of Fitts Law timings against actions for the Fluke, and makes the delays involved in holding buttons clear.

Note that the algorithm we develop can accommodate different values of \( a, b, c, d, h, k \).
Fig. 1. A transition diagram for part of a 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.

Fig. 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 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.

and w depending on the design of the user interface, and the same values need not be used uniformly across the design—obviously button shapes and distances will in general vary, but also the possibility of hold options (i.e., $h \neq 0$) on some buttons may depend on the current state. In particular, if a button has several hold variations (so it does different things if the user just taps it, holds it for 2 seconds, or holds it down for 10 seconds, etc) this can be handled simply by introducing a variety of “virtual” buttons that are only used in the relevant states, each with an appropriate $h$.

1.5 Time need not correlate with actions

One might think that simply 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 [9] 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.

How this behavior can arise can be understood by considering the finite state machine 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 action.

The state machine 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. Clearly, AB is a better way to get from 1 to 4 when measured in terms of counting user actions. However, AAA may be faster in terms
Calculating optimal times

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 $AB$.

The Fitts Law estimates the time taken to move between two positions, in this case the time taken to move a finger between button locations; time is estimated as a function of normalized distance, the distance moved $d$ divided by the size of the target $w$. Expressing the Fitts Law in the general form $t = a + b \log(d/w + k)$, it is faster to press $AAA$ than $AB$ if $d > w(e^{a/b} - k)$. For the parameters used in this paper ($a = 0.165, b = 0.075, k = 1$; see below) this occurs for this device when $d > 8.02w$, that is when the buttons are further apart than approximately 8 times their diameter (if assumed circular and of equal diameter). These “cross over” criteria will be different for different tasks, for different devices (different devices have different formulae) and, of course, the values of $a, b, d, k, w$ will depend on the user, the task and the physical layout and geometry of the device.

In a complex device, with many more path choices between states, we would be justified in expecting a poor correlation between least cost actions and optimal timings. The complication that a user doing more actions can be faster depending on their previous history of actions does not arise in the analysis of sequential tasks such as menu selection, since the number of steps to complete a task cannot be varied. This paper provides a general technique for handling this problem, and using it shows that there is (at least for the case studies) a very good correlation—this is a surprising result.

As figure 2 shows, the correlation or low correlation issue is not merely of theoretical interest: it is possible that a user will learn optimal sequences of actions that do not transfer to other tasks, modes or other devices, with the consequence that the user will unwittingly use devices inefficiently. Although the example in figure 2 is contrived to achieve $|AA| < |AAA| < |AB|$ all with equal effects, in general if a user finds an optimal path between two states, the same path will also take any third state to some other, but usually suboptimally. This will be a problem if the initial and final states in the two cases are confusable.

### 1.6 Task/action mappings

Users engage in sequences of actions to perform tasks. The inverse problem, what a user should do to achieve a task, is called the task/action mapping problem (see, e.g., [18]). Since every action a user performs has a cost, and every task a user completes is composed of a sequence of user actions, it follows that it is in principle an algorithmic problem to find an optimal solution to any task/action mapping. To express this problem algorithmically presupposes adequate definitions of task, action and cost, as well as the structure of valid tasks as sequences of actions.\(^2\)

It should be emphasized that the optimal task/action mapping being algorithmic does not imply that a user follows any particular algorithm; users, for example, need not perform tasks optimally. However if a user operates a device optimally, then their actions are

\(^2\)Strictly speaking there may be no algorithm (e.g., when the user interface is a universal Turing Machine), but these cases are obviously not ones where a task/action mapping makes any sense (e.g., universal machines don’t support specific tasks); furthermore if the task/action mapping is in principle infeasible the user interface cannot be easy to use—and we don’t need an algorithm to tell us.
equivalent to an optimal algorithm. Furthermore, an optimal algorithm could still allow the user choices: for example, doing the actions $AB$ or $BA$ in different orders might achieve the same outcome in equal, optimal, times.

A user’s task may be something like “listen to a track.” If the user is interacting with a state transition system, a definition of task is to take the machine from any state in one set of states (including the state the device is in before the user starts their sequence of action) to any state in another set of goal states (which complete the task). Every task a device supports will have at least one such sequence, though the same task may have many sequences of actions that achieve it. In contrast, previous work using the Fitts Law defined tasks as mapping to unique sequences of actions: e.g., menus [9] and typing [38] (where the task of typing is conceptually simplified to typing pairs of letters, digraphs, for analysis).

A convenient definition of a task cost is the total or average number of user actions it takes the user to perform a task to completion. This cost is identical to the total number of state transitions (assuming there are no transitions in the system hidden from the user). In this case, finding a least cost solution to a task/action mapping is a routine matter of finding shortest paths in the graph of the system. This is easy to do using any standard algorithm [11].

Unfortunately, there is no strong reason to suppose that minimizing user action counts is a reliable way of determining user behavior. For example, actions need not have equal costs to users. Time would be a better measure, but it is harder to estimate from a system model, since there is no time data (and certainly no user time data) in such a system-based model.

We could make the weak assumption that users must take at least some constant time $c$ to perform any action; then if the user takes at least $n$ steps to complete a task, the time $t$ the user takes must be at least $cn$, that is $t = \Omega(n)$. In general we do not know the value of the coefficient $c$, and indeed it will be different for different sorts of user action, but $\Omega$ notation conceals this ignorance. We may easily calculate a least $n$ for any specified task using a shortest path algorithm, but $t = \Omega(n)$ only tells us that if $n$ gets bigger then the minimum time for a user to do that task will increase at least linearly with the value we find for $n$. If we can redesign the device to make a minimum $n$ smaller, a user could be faster, but we won’t know exactly by how much. In fact, $\Omega$ does not permit us to say that if we reduce $n$ the user would be necessarily faster—because we don’t know (and $\Omega$ does not specify) that the user is taking the minimum possible time to start with. What $\Omega$ does is make explicit our ignorance—and the desirability of a better approach.

A correct summary is

$$\text{optimal user time} = \Omega(\text{optimal button count})$$

but the hidden constant in this $\Omega$ is not what you think it is: the device may have behavior as discussed earlier, with faster ways of achieving tasks than taking the total time to do each of the least number of actions in sequence.

This paper will show from case studies that using a Fitts Law model to estimate each specific time in a sequence of actions that there is a close linear relation where the coefficient is known and fixed:

$$\text{optimal user time} \approx k \times (\text{optimal button count})$$

This result is not only tighter than the best that can be expressed with $\Omega$. It must be emphasized that models for optimal time and models for optimal count give different se-
quences of user actions to achieve the same tasks and are not necessarily correlated. Thus the good linear correlation we obtain is surprising. From our results we will conclude it is not worth estimating times when optimal button counts are considerably easier to measure. Indeed, optimal 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. Optimal button counts can be used to optimize user interface design.

1.7 Algorithms for finding optimal times

An interactive device can be represented as a graph [31], such as the graph in figure 1: in the established terminology of graphs an interactive device is a set of vertices and a set of arcs, usually represented as labeled arrows connecting pairs of vertices. In the graph shown in figure 1, the vertices are \{1, 2, 3, 4\} and the arcs are the ordered pairs of vertices \{(1, 2), (2, 4), (2, 3), (3, 4)\}, which have labels from the set \{A, B\}. In the case of interactive devices, the graph labels represent user actions.

Each action is often associated with a cost—the user doing different things A, B . . . costs different amounts, or takes different amounts of time, etc. A standard problem, then, is to find least cost paths in the graph. Note that if we were simply counting user actions (and looking for least-cost ways to use a device in terms of minimizing action counts) then all actions would be assigned equal cost (for instance, 1).

By way of example, 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?

Many algorithms can find least cost paths in a graph. The Floyd-Warshall algorithm [11] starts with a representation of the graph in question 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 \(v\):

\[
\begin{pmatrix}
\infty & 1 & \infty & \infty \\
\infty & \infty & 1 & 2 \\
\infty & \infty & \infty & 1 \\
\infty & \infty & \infty & \infty \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
A & -- & -- \\
-- & A & B \\
-- & -- & A \\
-- & -- & -- \\
\end{pmatrix}
\]

In this example the costs are arbitrary numbers for illustrating times for taking single steps, and by comparing with the user actions matrix we can see that we have given the same cost to the same action, though there is actually no need to do so.

The cost matrix does not show the costs for getting between any pair of vertices: where there isn’t a direct arc between vertices the cost is infinite, \(\infty\), because there is no single step to do it. In particular, the infinite diagonal of the cost matrix above implies it is not possible to stay in the same state for this graph for any user action; this highlights a technical issue—since a user can generally do any action in any state, every action must be defined in every state. If we had done this for the example graph then the diagonal of the matrix would be finite. Typically we choose to zero the diagonal, as for most purposes there should be no time or other cost associated with staying in the same state—it costs the user nothing to do nothing, and doing nothing will always stay in the same state.

Given a cost matrix with a zero diagonal and using the Floyd-Warshall algorithm (or equivalent algorithm) on the figures above obtains the least costs for the shortest paths between all pairs of states:
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. The least cost path 1→4 has cost 3, and the user could do this with the actions AAA or AB, the overall cost being the same whichever way they go.

2. OPTIMAL TIMINGS FROM DEVICE MODELS

The problem with trying to use these techniques with any costs from the Fitts Law is that we do not have the single step cost matrix. Times (i.e., the costs) depend on where the user’s fingers were before they do an action; in this case, costs cannot be associated with single step arcs, which the matrix representation requires, but are a function of the previous action as well.

Consider, if A is the action “pressing button A,” when the user pressed A then their finger is now 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; that’s a step before the current action, and the cost matrix does not represent this information at all.

2.1 Conventional interactive device models

We now define a graph model to cover a very wide range of interactive systems; the literature supports this approach, and there are various ways an interactive device can be represented as an augmented directed graph [31; 35; 36; 37]. Here, we represent an interactive device as a set of vertices \( V \) (or states), a set of user actions \( A \), a label relation \( L \subseteq V \times A \times V \), and, for feedback to the user, a set of indicators \( I \) and an output function \( O:V \rightarrow \mathcal{P}I \). Typically we assume the device has a single user and has no other input (such as location sensors) not under control of the user; hence transitions in the graph are one-to-one with user actions (in fact, our use of the Fitts Law in this paper happens to put the graph one-to-one with the actions of a single finger).

The choices encapsulated in \( A \) and \( I \) determine the physical realization of the device. We shall assume \( A \) defines positions of actions and sizes of targets as this data is required to use the Fitts Law. The indicators \( I \) will be LED lights, text messages and so forth. The label relation \( L \) defines labeled arcs of the graph in a natural way. For a well-defined device, the label relation will be a total function \( L:V \times A \rightarrow V \), since every action must be defined and with a deterministic outcome for every state. (This definition differs in details from [36].)

It is suggestive to represent elements \( (u, a, v) \) of \( L \) by \( u \overset{a}{\rightarrow} v \). In words, the notation \( u \overset{a}{\rightarrow} v \) means that if the device is in state \( u \) and the user does action \( a \), the device will transition to state \( v \). A sequence of such actions and transitions can be represented by \( u \overset{a}{\rightarrow} b \overset{c}{\rightarrow} w \), etc, and, when we are not concerned with the details of intermediate steps, by \( u \leadsto w \).

\(^3\)The example is not strongly connected, which 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 would be no going back to “unuse” it; see [31].
Given such a graph, an interactive device can be run by (i) having a defined initial state, \( s = s_0 \); (ii) tracking the current state, \( s \), and updating it as the user performs actions \( a \) by \( s' = L(s, a) \). As the simulation proceeds, the user sees the system change from \( O(s) \) to \( O(s') \).

This model covers a very wide range of device types accurately, and is therefore a suitable as a basis for the remainder of this paper. The following points briefly address standard objections to using such models:

— A graph 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: \( O \) defines the screen contents, and \( A \) is enlarged accordingly to accommodate each variation of input actions the touch screen displays.

— When an interactive device allows simultaneous user actions \( a, b \) (for example, the device may have a modifier or shift key that is intended to be pressed at the same time as other keys), we require \( u_2 = v_2 \) whenever \( u_0 \xrightarrow{a} u_1 \xrightarrow{b} u_2 \) and \( u_0 \xrightarrow{b} v_1 \xrightarrow{a} v_2 \); this is a trivial property to check. An alternative approach is to introduce new actions, e.g., “\( a \parallel b \)”, for each possible simultaneous action, and compile these to graphs that have the appropriate properties.

— When a device has time outs (i.e., internal actions that occur after a delay when the user performs no other action), then we introduce new “actions” \( \tau_t \) that occur as necessary, automatically after the given delay \( t \).

2.2 Extending transition models to accommodate the Fitts Law etc

If we wish to estimate the time to achieve any goal from any state, then psychomotor models such as the Fitts Law can be used. The Fitts Law gives 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 the first is needed to define where the user was and the second to define where the user moves to. We need to build a graph with this information in each arc separately, not spread between adjacent arcs.

Construct a graph \( G' \) where vertices \( \langle v, m \rangle \) in \( G' \) record that vertex \( v \) in \( G \) can be entered by the user performing action \( m \). This ensures that each arc in \( G' \) corresponds to a specific movement and action, and therefore one with a specific time \( f \) that can be worked out from that movement and action.

\[
V(G') = \{ \langle v, a \rangle \mid u \xrightarrow{a} v \in L(G) \}
\]

\[
L(G') = \{ \langle \langle v, m \rangle \xrightarrow{f} (w, n) \mid u \xrightarrow{m} v \xrightarrow{n} w \in L(G) \}
\]

For this definition we require \( G \) to be total; that is for all vertices \( u \) in \( G \) and all actions \( a \) in \( G \), there are arcs \( u \xrightarrow{a} v \) (possibly self-loops \( u \xrightarrow{a} u \)). Figure 3 shows the total graph corresponding to figure 1.

As each vertex in \( G' \) has all incident arcs with the same user action, every arc in \( G' \) has a fixed prior user action: hence the Fitts Law can be applied functionally to arcs in \( G' \) directly:

\[
L(G') = \{ \langle \langle v, m \rangle \xrightarrow{f(m,n)} \langle w, n \rangle \mid u \xrightarrow{m} v \xrightarrow{n} w \in L(G) \}
\]
Fig. 3. The total closure of the graph of figure 1. This graph can be constructed automatically, though if a user interface is not closed this would generally be an implementation error indicating that some actions are not defined in some states: for example, in the closure, but not in figure 1, action B in state 1 is defined but as it happens it does not change the state. In contrast, in figure 1 and the closure, both A and B are defined in state 2.

where the function \( f \) uses the Fitts Law formula applied to the distance between the buttons \( m \) and \( n \) and the effective target size of the button \( n \) given that the finger is moving on a vector in the direction \( m \) to \( n \). The values \( f(m, n) \) only depend on the two vertices on the arc and the choice of \( f \)—we can use different formulae for different arcs, for instance if we know one button is to be used by the thumb, then the Fitts Law constants are different than index the finger constants used in this paper [24].

Crucially what we have achieved is that timing is only a function of individual arcs in a derived graph: all the relevant information is embedded in each pair of adjacent states in \( G' \). We can now use standard shortest path algorithms. The mathematics is not as complicated as it may look; for example, code to create the graph in Java is as follows:

```java
class Arc
{
    Object a, u, v; // action a takes state u to state v
}

class Fitts
{
    double m, a, b;
    double cost(double m, double a, double b)
    {
        double cost = Math.abs(a - b) + 2 * b / (m + b);
        return cost;
    }
}

class Actor
{
    String name;
    Actor(String name)
    {
        this.name = name;
    }
}

class State
{
    String label;
    State(String label)
    {
        this.label = label;
    }
}

class Button
{
    String button;
    Button(String button)
    {
        this.button = button;
    }
}

class Graph
{
    ArrayList<Arc> arcs;
    Graph()
    {
        arcs = new ArrayList<Arc>();
    }
    void addArc(Arc arc)
    {
        arcs.add(arc);
    }
    void printArcs()
    {
        for(Arc arc : arcs)
        {
            System.out.print(arc.a + " from " + arc.u + " to " + arc.v + " cost = " + arc.cost);
        }
    }
}

class Interface
{
    ArrayList<Button> buttons;
    Interface()
    {
        buttons = new ArrayList<Button>();
    }
    void addButton(Button button)
    {
        buttons.add(button);
    }
    void printButtons()
    {
        for(Button button : buttons)
        {
            System.out.print(button.button + " ");
        }
    }
}

class Main
{
    public static void main(String[] args)
    {
        Graph g = new Graph();
        g.addArc(new Arc(Actor.valueOf("A"), State.valueOf("1"), State.valueOf("2")));
        g.addArc(new Arc(Actor.valueOf("B"), State.valueOf("2"), State.valueOf("3")));
        g.addArc(new Arc(Actor.valueOf("A"), State.valueOf("3"), State.valueOf("4")));
        g.addArc(new Arc(Actor.valueOf("B"), State.valueOf("4"), State.valueOf("1")));
        g.printArcs();
    }
}
```

where we assume a graph is represented as an array of \( Arc \), arcs being represented as a triple of objects: the initial vertex, the terminal vertex, and the action. For simplicity in the context of this paper the function \( makeFittsGraph() \) does not actually construct
Calculating optimal times

Having obtained $G'$, its shortest paths can be found using the Floyd-Warshall algorithm or equivalent. Now, since shortest paths in $G'$ distinguish what button the user pressed at each step, the shortest paths in $G'$ don’t immediately give shortest time paths in $G$. However the relation is straightforward: the least time over all $\langle u, m \rangle \sim \langle v, n \rangle$ in $G'$ for all $m, n \in A(G)$ gives the least time for a path $u \sim v$ in $G$. It is on this basis that the correlations discussed in the next sections (§3 and §4) are based; however, for more specific studies it may be reasonable to assume that users are unlikely to know or be able to identify specific device states. In this case, the user’s task would be specified more generally as getting the device from any of a set of states the user sees as equivalent to any target state that satisfies a goal; we would thus find the minimum time over all pairs of paths between the two sets of states.

2.3 Extensions

The model supports many straightforward extensions, e.g.,

—To find the optimal path, arcs are labelled with the action $n$ as well: $\langle v, m \rangle \xrightarrow{f,n} \langle w, n \rangle$ and then use standard algorithms [11].
—To use the Hick-Hyman Law to add delays due to the choice users face, arcs are labelled $\langle v, m \rangle \xrightarrow{f+h} \langle w, n \rangle$ where $h$ adds a time $c \log(d(v) + 1)$, where $d$ is the effective out-degree of a state (the graph theoretic out-degree is a constant). The effective out-degree varies in soft-key applications (or GUI interaction where displayed user choices change with the state), it can be reduced by user training, or by using techniques from [31].
—To handle timeouts like “reset if user does nothing for 10s,” introduce arcs with weight equal to the duration of the timeout. There are other sorts of timeout (e.g., “alarm if user does not enter correct code within 30s”) and some may need careful treatment in the model.
—More sophisticated models than the Fitts Law allow for the user’s perceptual and cognitive processing. The current approach allows for this; for example if each state in the graph is labelled with the displays, then when they are changed, time can be added in exactly the same was as for the Hick-Hyman Law. Mental Operators from models such as GOMS [6] are also easily added.

4The simple println assumes that the form (string,string) represents vertices in $G'$. In general something more sophisticated (such as a constructor) must be used: if $G$ contains strings that are empty or contain dots using println is ambiguous.
3. CASE STUDY 1

Figure 4 shows the front panel layout of a JVC model HRD-580EK personal video recorder (PVR). The PVR is a non-trivial device and has a complexity typical of a mid-range consumer device.

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 PVR functionality than the front panel. This extra functionality is not considered here. A full discussion of the device and its structure can be found in [31], from where the device definition was taken.

The model was used to calculate optimal button press counts and optimal Fitts Law times for all pairs of states, using the techniques described earlier. (The average optimal button count is in fact the characteristic path length [31] of the state machine.) The Fitts Law constants were taken from [24, p45] and correspond to index-finger movement as appropriate for this device: $a = 0.165, b = 0.075, k = 1$ using natural logs and time in seconds. Optimal times were then plotted against optimal counts for each pair of states, as shown in figure 5. The following observations are in order:

— the linear coefficient of determination (square of the correlation coefficient) is $R^2 = 0.96$. This result is comparable to what Fitts Law experiments achieve in user studies under laboratory conditions;
— rearranging physical button layout will improve speeds by greater factors for faster tasks;
— but if you only want to improve the interaction programming the correlation is so good

---

Reference [24] suggests these constants for mobile phones, which are roughly the same scale as the devices considered here, but unfortunately the paper does not give a precise scale. Mathematically the units for distance do not matter in the Fitts Law, and few authors specify a scale for which their empirically-derived constants are valid: however, scale does matter, as for instance there must be time discontinuities between scales less than 1m and greater than 10m.
Calculating optimal times

Fig. 6. Transition diagrams for the eight case studies; top left is the PVR, bottom left 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 (top right), which are isomorphic. (Note that the diagrams are scaled the same size even though they have varying numbers of states.)

Fig. 7. Exact button layout for the Fluke 114, also illustrating how effective target size (gray circles) depends on the direction of travel.

that going to the effort of calculating Fitts Law times does not seem worthwhile.

4. CASE STUDIES 2–7

The next case studies have been chosen to be 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 [32], has 425 states and 4,250 arcs, so it is much more complex than the PVR. The 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. The buttons are both circular and rectangular, another difference to the PVR. Their shape and position have been modeled to within 0.1mm (figure 7).

Each of the knob positions effectively creates a distinct device with its own behavior, one for each the different sorts of measurement the meter supports, so we can treat it as six separate devices (see figure 6) by ignoring the knob position “Off,” which with only one state is trivial. Treating it as a collection of separate devices also allows us to ignore
Fig. 8. Visualizing target size variability for case study 1. The circles are not the targets, but visualize the size of the target for each possible direction of travel from the center of every other button.

Fig. 9. A plot of the exact Fitts Law timings (vertically) against approximate Fitts Law timings, which ignore button shape and size (horizontally) for all possible tasks on a PVR. The graph shows a best-fit line $y = -0.04 + 0.72x$ ($R^2 = 0.87$).

the different timing issues of turning knobs, and it provides six correlations rather than a single aggregate value.

Repeating exactly the same methodology as for case study 1, we obtain excellent correlations ($R^2$) of 0.96, 0.97, 0.97, 0.96, 0.97, 0.95. The Fluke allows some buttons to be held down for 2 seconds to obtain additional effects. Figure 10 illustrates the three separate linear correlations when we also consider these actions.
Calculating optimal times

Optimal times

Fig. 10. Data obtained using the same methodology as in figure 5 (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) that all possible tasks can be achieved with either zero, one or two hold actions. Best-fit lines have been drawn considering each case separately, which involve 2,958 (no hold), 5,746 (one hold), and 1,296 (two hold) possible tasks. Note that the regression lines look to have poor fit; this is because the thousands of data points are not uniformly distributed, although many are nearly coincident and hence indistinguishable.

5. CASE STUDY 8

The final case study is the Cardinal Health Alaris GP volumetric infusion pump [7], 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 11) that allows user testing to confirm the program is an accurate interaction simulation; we then use discovery [37] to derive a graph model.6

Uniquely in the case studies in this paper, the Alaris allows the user to enter fractional numbers, such as the infusion rate; thus the state space, treating each number as distinct, is consequently enormous. To keep the model of 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 here with a graph $G$ of 352 states and 3,872 arcs, producing a Fitts 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 ignoring 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 from the descriptions provided in this paper (cf section

---

6The 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 have upgraded the Alaris GP user interface, so our simulation is now obsolete and can no longer be checked against a physical device.
Fig. 11. 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 indicating an error condition.

1.2, point 9), and repeating the same methodology as for the previous case studies, we obtained another high correlation $R^2 = 0.985$ ($N = 51, 104$).

Although good, this correlation would be worse had we used a complete 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 above), 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 essentially constants.

In this paper, constructing a model that is expected to decrease the correlation but still achieving a high correlation tends to strongly confirm our claim that optimal times are correlated with optimal action costs.¹

6. SOME POTENTIAL CONFOUNDING FACTORS

The high correlations imply the “cross over” described earlier is not a significant issue in determining user times for optimal use of real devices. The correlations are high, but do they behave as we expect? Is the model realistic?

6.1 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 they are exactly the sort of pushbutton interfaces that the Fitts Law has historically been widely applied to. As a check on this, increasing the

¹In a typical usability study rather than a paper, however, it would be unusual to study all features of a device like the Alaris GP at once, since the mode interface and the number interfaces are very different, but the various number entry interfaces (volume infused, rate, volume to be infused, etc) are essentially all the same. Since each number entry involves about 1,000 states just to represent a number, it would be in practice simpler and clearer to consider them separately—e.g., analyzing two number interfaces would create $10^6$ states that are no more informative than $10^3$ (unless number input is inconsistent across the device), but would take perhaps a thousand times longer to analyze.
Calculating optimal times

distances but not target sizes by a factor of 1,000 (i.e., to tens of meters, beyond the empirical calibration range of Fitts coefficients) indeed gives a low correlation ($R^2 = 0.52$). 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.

6.2 The two-dimensional Fitts Law

The Fitts Law considers the target width $w$ to be measured in the direction of movement. Detailed discussion of the Fitts Law taking account of the 2D shapes of buttons can be found in the paper [25] by MacKenzie and Buxton: they demonstrate from experimental data that accounting for shape is important in using the Fitts Law. Although it is easy to model this, as we did above, this source of variability is often ignored [25]. Interestingly, [25] argues 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 2D model, and of course our analysis aggregates every possible scenario as well.

Figures 7 and 8 exhibit variation in target sizes, as measured in the direction of finger travel. Figure 9 shows a plot of Fitts Law times taking the geometry of buttons into account plotted against 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 is due to their concern with isolated actions, as opposed to our concern with multiple actions with optimal times.

6.3 Data uniformly weighted

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, an 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 considered may be ones with low correlation, as discussed in section §1.5. However, it should be noted that the algorithm has no problem handling benchmark tasks or assigning non-uniform distributions of weights to times; indeed, one would imagine that a design analysis tool would do exactly this.

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 [31]: this very different weighting makes less than ±1% difference in $R^2$ for all the case studies. Another approach would be to weight target states, adapting the information foraging approach [29].

Another approach would be to handle weights and benchmark tasks sets interactively. Figure 12 shows a graph of all predicted task times from the first case study. One imagines 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 current paper does.
Fig. 12. Graph of all optimal task times from case study 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.

6.4 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 an optimal task time (averaged over all tasks). The simplest generalization would be to introduce other operators from the GOMS models [6], such as mental times; this would be trivial, and merely requires a per-state modification\(^8\) of the constant \(a\) in the Fitts Law formula. (Note that if almost all transitions have a new times added, this would hardly affect the correlations reported in the case studies.) More generally, an ACT/R model [2] or EPIC model [20] could be run, rather than using a simple formula, as assumed in this paper—in fact, both are equivalent to a (complex) formulas, but they happen to be computed by running some program code.

7. FURTHER WORK

The present work raises numerous interesting research issues that go beyond the scope of a single paper; for example:

(1) The modeling should be repeated with more devices; it is possible that the results are a quirk of the PVR and meter, though this is thought unlikely, as the devices are so different. Unfortunately there are very few accurate and complete models available to researchers.

(2) The Fitts Law is only one of many functions to estimate timings. We showed that Hick-Hyman can be used within the present approach for user interfaces with soft buttons (and hence a varying number of user choices). More sophisticated approaches, such

\(^8\)As in CogTool [19], one could annotate the states with the types of operators required, or derive the operators from a generic description of states.
as ACT/R, assume a cognitive model [1] and are beyond the present scope of this research.

(3) The Fitts Law only models movement; it does not model perception or cognition. A recent paper [1] suggests that the Fitts Law consistently under-estimates 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.

(4) We assumed the user only moves a single finger throughout the task, and this assumption gets increasingly implausible the longer the sequence of actions needed to perform the task. More sophisticated models would account for two (or more) fingered use.

(5) A graph is readily simulated, and this would allow user experiments, and recording of empirical times, rather than calculated times. Again, this would be easy to integrate into the present graph model approach. A Java framework has been suggested [30] but this does not provide a graph model; while [31] provides a general graph-based simulation framework.

(6) From figure 5, 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 will generally 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.

(7) 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 completed devices, not designs in progress.

The purpose of the case studies was to test the relation between optimal timings and optimal action use of devices, and this they do well. Completed production devices were used as case studies because they were not created specifically to support the claims of this paper, and because they are clearly of realistic complexity—that the approach should be able to, and does, handle. Now that it is established that there is a correlation, this opens the possibility of using optimal 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.

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 about a month 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. Reverse-engineering is just an artifact of the research methodology and is

---

9Many conventional HCI papers build new or revised systems and then show those systems to be improvements (often without double blind evaluation), thus begging many methodological questions. Evaluating a variety of existing commercial systems, as done here, is arguably more rigorous.
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.

8. CONCLUSIONS

Usability depends on efficient use of interactive systems, and to design efficient systems requires analysis of the time complexity of the designs.

The paper has provided an algorithm for obtaining optimal task time estimates from physical data and any state transition model, illustrated by using the Fitts Law for estimating individual user action times. The entire work is placed within a standard complexity framework, using $\Omega$ notation, which has been shown appropriate for user interface time analysis. The algorithm has been applied in case studies demonstrating its value, but its main future use will be in enabling designers and design evaluators to study more systems. For example, identifying tasks where least times take different paths than least actions will help designers focus on optimizing time-critical features of a design.

We showed theoretically that when operating a pushbutton device, the best user time in general may not be the obvious $t = \Omega(\text{optimal count})$. We illustrated these ideas with a contrived device, but for the various real devices studied it is approximately $k \times (\text{optimal count})$, a surprising result (and one more general than previous work with sequential task models, menu selection, etc). The correlation achieved is comparable to typical empirical work, so in appropriate circumstances we are justified in using optimal button press counts rather than timings to optimize design, unless we need estimates of actual times for some reason.

Acknowledgments

This work was supported by Engineering and Physical Sciences Research Council grants [EP/F020031, EP/F059116]. The author is a Royal Society-Leverhulme Trust Senior Research Fellow, and acknowledges this support. Ann Blandford, Paul Cairns, Andy Cockburn, Parisa Eslambolchilar, Jeremy Gow, Michael Harrison, Tony Hoare, Andy Howes, and referees made very useful comments. The present paper is a corrected and greatly expanded version of [33]; see section §1.2.

REFERENCES

Calculating optimal times


