Modeling Dual-Task Concurrency and Effort in QN-ACTR and IMPRINT

by

Christopher Jason Best

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in The University of Michigan
2013

Doctoral Committee:

Professor Yili Liu, Chair
Assistant Professor Victoria Booth
John F. Locket III, US Army Research Laboratory
Professor Nadine B. Sarter

It is enough for a Psychohistorian, as
such, to know his Biostatistics and his
Neurochemical Electromathematics.

—Asimov, *Second Foundation*

for the best family

ACKNOWLEDGEMENTS

I would like to thank Yili Liu for his advice, guidance, and enduring patience; Victoria Booth, Bernard Martin, and Nadine Sarter for their invaluable feedback, comments, and questions; likewise John Lockett and for his professional mentorship and dietary, fitness, and gardening advice; Chuck Woolley for being chiefly responsible for my professional and academic career since 2006; Chris Konrad, Eyvind Claxton, Mint, and Candy Ellis for their expert and swift technical and non-technical support; mom for flashcards and frozen leftovers; dad for science magazines and making cognitive ergonomics sound impressive; kin and lliby for being the objects of looking up; the jokers for facilitating both academic and leisurely pursuits; tilly, Double C, Ganish and the Sunshine Colts, Fred, Denny, Ralu, and BDC for something else to do; the University of Michigan and the city of Ann Arbor for making me feel big and small at the same time;

and KL and the monster for Everything

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER I

Introduction

Because of the exceedingly complex nature of human cognition, a unified model explaining all aspects of cognition remains a distant goal. However, many cognitive architectures have been developed to successfully predict observable phenomena and to model cognitive mechanisms at many levels: from the physical laws, anatomical organization, and differential equations that predict action potentials in neurons, to high-level psychological models describing the effects of affect on decision making. Increasingly, modern engineering methods and computer technology are providing opportunities to develop functional simulations of cognition based on psychological and ergonomic literature. Along with the software implementation of the types of models that, historically, were purely mathematical and descriptive, comes the opportunity and challenge to integrate the theories at several levels. A particular difficulty is the requirement to understand both the psychological aspects of the cognitive theories themselves and the technical details of their software implementations. Fluency in programming languages is becoming as vital to academic study of cognitive ergonomics as the scientific language of the psychologist, and the task of maintaining a quality code base is as necessary to the continued development of a cognitive model as documenting the theory and rigorous validation of the model itself. Our research

is based on the integration of several existing cognitive architectures and their use in modeling concurrency in multi-task performance. The integration of these systems required the development of a task management theory and the engineering task of establishing communication between the systems.

We investigated a widely used task performance simulation tool to identify potential improvements and surveyed cognitive architectures and theories that could be used to enhance the software. We found that a feature which suppressed task component execution in high-workload conditions was insufficient to accurately model subtask conflict. We formalize a theoretical architecture for moderating task execution based on cognitive state and implement the architecture in a software extension to the simulation tool. The architecture is divided into one component which enumerates the possible actions that can be taken at decision points during the simulation and another which selects an action from among those provided based on the system state. The latter component can be easily modified to implement and test theories of task management during multi-task performance. We validate a basic, workload threshold-based implementation by comparing simulated performance to an empirical study that identified qualitatively distinct load-shedding strategies adopted by Unmanned Aerial Vehicle (UAV) operators (chapter II).

In another vein of our research, we model dual-task performance using a recently developed combination of two architectures, and analyze simulations in detail to examine how well the models predict task concurrency. Our primary view of concurrency is a quantification in terms of the execution times of the individual tasks that make up the dual-task. Given two distinct tasks that are presented to a participant simultaneously, we call the dual-task execution time the time between the start of the tasks and the completion of the task that is finished last. If two tasks are performed

serially, that is, with no concurrency, the dual-task execution time is equal to the sum of the execution times of the two tasks. At the opposite end of the spectrum, the two tasks have no conflict and are performed with perfect concurrency. In this case, the dual-task execution time will be equal to the larger of the two single-task execution times.

Based on these basic assumptions of concurrency, we quantify dual-task concurrency and study the empirical results of human performance on a dual-task consisting of mental addition and a targeting task. We develop models of the two tasks, simulate performance of the single and dual tasks, and validate the predictions using the empirical data. We examine the model's predictions in terms of a numeric concurrency metric as well as the details of subtask interleaving and conflict. In our first experiment we vary the difficulty of the tasks with three independent variables and discuss their effects on task execution time, accuracy, and concurrency in both empirical and modeled data. We also simulate performance on the experimental tasks using the task management theory described above and compare the execution time and concurrency predictions with the empirical data as well as data simulated with the tool's built-in workload management feature (chapter III).

We generalize the concurrency metric to define concurrency for repeated task iterations of two tasks without synchronized start times and, again, compare QN-ACTR predicted execution time and concurrency values with empirically collected data. In this experiment we remove two task-related independent variables and introduce an incentive variable and offer an empirical result concerning the source of dual-task performance improvement under incentivization. We model the effect of effort by modifying the QN-ACTR models and compare the predicted effects with the empirical data (chapter IV).

In this chapter, we introduce the architectures that we use in our work either in their entirety, with modifications, or as theoretic inspiration.

1.1 QN

The Queueing Networks (QN) system was developed by Liu (1996) as a mathematical model of the cognitive processes that occur during reaction-time tasks. It was integrated with the Model Human Processor (MHP) (Card et al., 1986) and implemented as a computation model capable of simulating task performance in the discrete-event simulation software ProModel (Liu et al., 2006). The resulting QN-MHP system adopted the basic assumptions and parameters of the MHP model and further divided the processing stages based on neuroscience and psychological findings. The software implementation of QN-MHP introduced the capability to visually display the flow of information in real time as a simulation runs. QN-MHP has been used to study the workload of many tasks including driving (Wu and Liu, 2007), transcription typing (Wu and Liu, 2008), and spatial and verbal tasks (Liu, 1997). The QN modeling approach assumes that cognitive resources and regional brain functions can be represented by a network structure of servers. The structure is biologically inspired and determined by the Psychology and Cognitive Modeling literature. The network structure and server tasks as implemented in QN-MHP are shown in Figure 1.1. The network consists of three major subnetworks: perceptual, cognitive, and motor. The perceptual subnetwork models the reception and processing of information through the visual and auditory sensory modalities. Four servers represent each modality: one for common processing; two for location and recognition, which process in parallel; and one to integrate the resulting information. The cognitive subnetwork is responsible for receiving information from the percep-

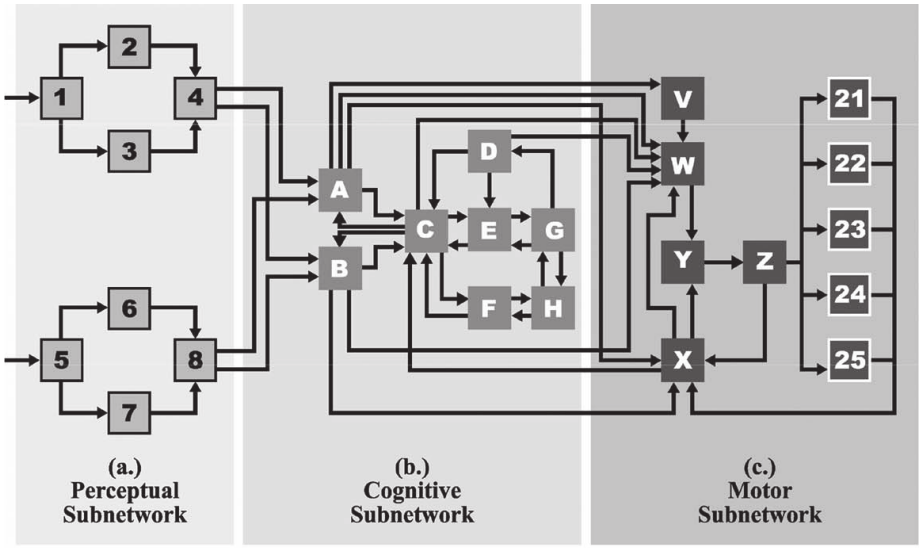
tual subnetwork and processing it in the context of the agent's goals and memory. Faculties represented by cognitive servers include long-term declarative and spatial memory, which models the storage of facts and other learned information; procedural memory, which stores information about performing learned tasks; a goal initiation server, and others. The motor subnetwork models the transformation of cognitive intent to motor commands to the agent's body parts. The motor subnetwork receives information almost directly from the perceptual subnetwork, as well as from the procedural memory server and others from the cognitive subnetwork. In addition to sending information to end effectors to produce movement, the motor subnetwork relays information back to the cognitive subnetwork as motor feedback.

To simulate a task in QN-MHP, a task analysis must be performed and an NGOMSL-style (Kieras, 1996) description of the task produced and converted to a task description sheet that can be read by the software. This type of task description consists of declaring a goal and a sequence of actions that accomplish the goal. Kieras (1996) provides an example of a description of deleting a file on a computer:

```
Method for goal: delete a file
  Step 1. Recall that command verb is "ERASE"
  Step 2. Think of directory name and file name and
     retain as first filespec.
  Step 4. Accomplish goal: enter and execute a command.
  Step 6. Return with goal accomplished.
```

For a QN-MHP model, the task must be specified using a predefined set of operators (Liu et al., 2006).

The simulation of a task often requires interaction with an external environment from which perceptual information can be extracted, and to which motor commands can be applied. For example, to model a driving task, a driving simulator would be connected to the model. Visual information about lane deviation and other cars



Perceptual Subnetwork	Cognitive Subnetwork	Motor Subnetwork
1. Common visual processing	A. Visuospatial sketchpad	V. Sensorimotor integration
2. Visual recognition	B. Phonological loop	W. Motor program retrieval
3. Visual location	C. Central executive	X. Feedback information collection
4. Visual recognition and location integration	D. Long-term procedural memory	Y. Motor program assembling and error detecting
5. Common auditory processing	F. Complex cognitive function	Z. Sending information to body parts
6. Auditory recognition	G. Goal initiation	21-25: Body parts: eye, mouth, left hand, right hand, foot
7. Auditory location	H. Long-term declarative & spatial memory	
8. Auditory recognition and location integration		

Figure 1.1: QN-MHP network structure. Reproduced from Liu et al. (2006)

would be synthesized based on the simulators screen and sent into the perceptual subnetwork. Motor commands such as rotating the steering wheel, applying pressure to a pedal, or even setting the agent's gaze would be sent from the motor subnetwork and translated into commands to be sent as input to the simulator.

The modeler can view the model in simulation-time as information flows through the network. This allows a subjective assessment of cognitive workload as the agent performs a task. QN-MHP supports the study of multiple-task performance by allowing the definition of multiple tasks, which will send information through the network concurrently. For any task, single or multiple, we can study reaction times by calculating the time between the entrance of information through the perceptual subnetwork and the production of corresponding motor commands. To validate a model, we compare the distribution of reaction times exhibited by the simulation and by a participant performing the same task in a laboratory.

The nature of QN is to allow us to assess reaction times and cognitive workload of a given cognitive process. As a functional simulation model, it retains the robust mathematical analysis of reaction times, which characterized its original form as a descriptive model. However, this analysis depends on the manual description of the cognitive process that it analyzes. It does not attempt to produce or even model the production of the cognition required to perform a task or to solve a problem.

1.2 ACT-R

Adaptive Control of Thought - Rational (ACT-R) is a cognitive architecture and theory developed by (Anderson and Lebiere, 1998). The ACT-R theory describes a collection of modules representing human cognitive, perceptual, and motor faculties, that interact to produce detailed models of task performance. ACT-R models can

accurately predict task metrics such as response time and error. ACT-R is implemented as a free software package in the Lisp programming language and is a heavily researched topic in the cognitive modeling field.

The ACT-R theory defines a set of modules representing the perceptual, motor, and cognitive resources used during human task performance. This section introduces the modules used by our task models.

1.2.1 Declarative Module

The declarative module stores information in units called chunks to represent declarative memory (DM). Chunks are added to DM either explicitly by a modeler or automatically when they are cleared from another buffer, and are retrieved by making requests to the retrieval buffer which specify values of chunk properties or "slots" which the retrieved chunk must match. The time required to retrieve a chunk is a function of the chunk activation and is described in chapter III. Like the vision module, only one request can be made to the declarative module at a time.

1.2.2 Procedural Module

The procedural module manages the definition and execution of the productions that modelers write to determine behavior based on the state of other modules. Productions are defined using the `(p production-definition)` command. Below is a production definition taken from the high-difficulty targeting model.

```
(P detect-target-color
  =goal>
    ISA             targeting
    state           distinguish-target

  =visual-location>
    ISA             visual-location
    kind            OVAL
    - color         black
    color           =color
```

```
  ?retrieval>
    state          free
    buffer         empty
==>
  +retrieval>
    ISA            response
    color          =color

  =goal>
    state          decide-whether-to-shoot
)
```

The production definition consists of a name (`detect-target-color`), a set of conditions which appear before the `==>`, and a set of actions which appear at the end. The conditions declare a system state which must be met for the rule to match. In this example, the contents of the goal buffer (indicated by `=goal>`) must be a chunk which is of type `targeting`, and the value of the `state` slot must be `distinguish-target`. Many ACT-R models maintain the state of the task in a slot on the goal chunk. The visual location buffer (`visual-location`) must hold a chunk of type `visual-location` and represents an oval. This condition uses a negative slot test (`- color black`) to state that the color of the oval cannot be black. It also uses a variable test to capture the value of the color slot, whatever it may be (`color =color`). Finally the production tests that the retrieval buffer is available (`state free`) and does not already contain a chunk (`buffer empty`). This test enforces the "greedy-polite" policy of the threaded cognition theory described in chapter III.

If these conditions are met during the production matching phase of the ACT-R algorithm, the production is a candidate for selection. If the rule is selected due to being the only matching rule, or through the conflict resolution process, the actions will be applied during the execution phase. In this example, a request is made to

retrieve a chunk from declarative memory (`+retrieval>`). The request specifies that the chunk should be of the type `response` and the value of the color slot should match the color of the chunk in the visual location buffer. This is achieved by providing the `=color` variable matched in the visual location buffer condition to the color slot of the request. The rule also specifies that the chunk in the goal buffer (`=goal>`) should be modified so that the `state` slot contains the value `decide-whether-to-shoot`.

1.2.3 Goal Module

The goal module represents the intent of the simulated operator and the associated buffer typically contains a chunk describing the state of task execution. In our models we store the stage of processing in the state slot of the goal buffer chunk and use other slots to store task-specific information such as intermediate results of an addition problem. The standard ACT-R goal module defines one goal buffer and is therefore not well suited for easily modeling multi-task behavior. However, extensions to ACT-R implementing Salvucci's threaded cognition theory (Salvucci and Taatgen, 2008) introduce addition goal buffers to manage the state of subtasks separately.

1.2.4 Vision Module

The vision module represents the human visual modality and defines two buffers corresponding to two distinct visual subsystems. The visual location buffer supports requests for information about where objects are in the visual plane. Requests to the buffer describe visual properties of an object to constrain a search for an object that matches. Searches can be constrained by features conducive to preattentive search (Anderson and Lebiere, 1998). For example, the `detect-target-color` production above could match a chunk in the visual location buffer that was found in response

to the following request which searches for an oval with a pixel location with an x value of 100:

```
+visual-location>
  ISA      visual-location
  kind     OVAL
  screen-x  100
```

If a visual object exists that matches the request it is placed in the visual-location buffer, replacing any chunk already there. If no match is found, the buffer remains unchanged and the buffer will be in an error state until a new request is made and succeeds.

Some details of a visual object are not stored in the visual location buffer and are not available for inspection unless a request is made to the visual buffer, the other buffer defined by the vision module. A request to the visual location buffer represents a preattentive search and a request to the visual buffer typically represents an allocation of visual attention, although it does not correspond directly to eye movements (Anderson et al., 2004). The primary request made to the visual buffer is of the type `move-attention`. The request specifies a `screen-location` slot whose value must be a subtype of the `visual-location` chunk-type. Typically, visual buffer requests are made using the results of a visual-location request in the `screen-location` slot. Visual buffer requests have a default latency of 85 ms before the request results are available. Requests made while the visual module is busy will fail. Details of the object such as textual content are stored in the visual buffer chunk. When visual attention is allocated to a location, the location is said to be attended and the attended state can be used to constrain future visual-location searches. The number of locations that can be attended at a given time and the time that a location remains attended are limited and can be specified as model parameters. The visual buffer can

be explicitly emptied and the error flags reset with a clear command. Making a clear request also prevents the module from re-encoding visual objects, which normally occurs when there is a change in the visual scene at the location of visual attention.

1.2.5 Device Module

The external interfaces that the model interacts with are represented by the device module. The module defines a basic set of devices such as a keyboard, mouse, and screen and allows for the definition of custom devices as lisp objects. Parameters such as the screen pixel density and distance can be defined using the device module.

1.2.6 Motor Module

The motor module represents the model's hands and supports basic commands to operate the default devices. Just as custom devices can be defined, custom motor commands can be defined to interact with them. The default motor actions are based on EPIC's Motor Processor (Meyer and Kieras, 1997a) and allow models to perform movements such as striking keys, and moving a mouse. Movement time for large movements such as mouse control is governed by Fitts' Law while smaller movements such as keystrokes have times calculated based on the movement parameters and have a minimum execution time. Some parameters such as the Fitts' Law coefficient for peck motions and noise in cursor movements can be overridden. Each motor action consists of three phases: preparation, processing, and execution. The availability of each phase can be queried by model productions. In the motor module, unlike the visual module, a new request can be made while the processing or execution phase are busy as long as the preparation stage is free.

1.2.7 Speech Module

The speech module provides basic support for simulating speech and subvocalization and is similar to the motor module. Requests of type `speak` or `subvocalize` must contain a `string` slot whose value is the text that should be spoken or subvocalized. The time required to complete the action is 150 ms per syllable, and the module assumes one syllable for every three characters in the string. These parameters can be overridden.

1.2.8 Temporal Module

The temporal module implements a basic capability to estimate time intervals. The module acts as a timer and is started by making a request of chunktype `time`. The request sets or resets a tick counter maintained by the module to 0 and the module begins the process of updating the tick count. The current tick count is available in the `ticks` slot of the chunk in the temporal buffer. The time between tick count increments is defined by a noisy, autoregressive formula that models prediction accuracy that decreases with time (Taatgen et al., 2007).

1.3 QN-ACTR

Cao and Liu (2011a) developed a queueing network based implementation of most of the ACT-R system in the discrete-event simulation software Micro Saint Sharp (MSS). Just as QN-MHP represented components of MHP as servers in ProModel, each ACT-R module and buffer is implemented as one or more tasks[1] in MSS. For example, the portion of the task network that implements the motor module is shown in figure 1.2. Requests to the motor module are passed into the Motor Module

[1]MSS and IMPRINT use the term task to refer to the atomic unit in a network, while ProModel uses the term location for the same concept. QN-ACTR and QN-MHP literature uses the term server to refer to a stage of processing that is implemented as a location or task in ProModel and MSS respectively. When discussing the implementation of QN-ACTR in MSS we use the term task, which is not to be confused with an experimental task being modeled using QN-ACTR.

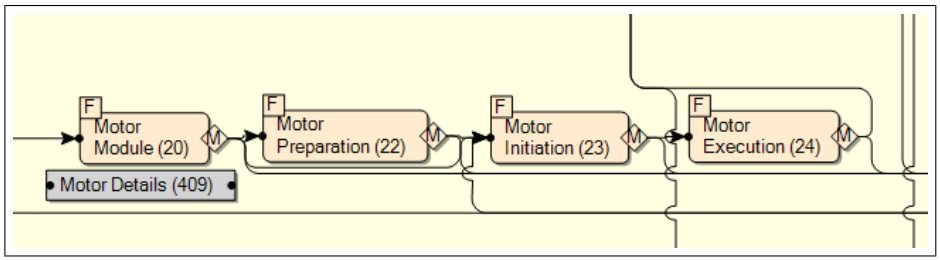


Figure 1.2: Motor module subnetwork in QN-ACTR

task from a subnetwork that is responsible for parsing an ACT-R production and delegating the actions to the appropriate module tasks. The module computes the duration of each stage (preparation, initiation, and execution) and stores the times in the entity. The entity then continues through each stage of processing to simulate the execution of the motor action.

Cao and Liu (2011a) verified their ACT-R implementation by comparing the simulations of 20 different models from the ACT-R tutorials and a dual-task study (Schumacher et al., 2001). QN-ACTR gains the advantages of queueing network simulations while maintaining the power and accuracy of the ACT-R system. The network activity can be viewed as a simulation runs in the MSS graphical interface and utilization-based workload values of a single task or a subnetwork can be automatically computed (Cao and Liu, 2011b). Furthermore, Cao has developed an interface which allows modelers to describe an experimental task design which QN-ACTR can present to both a human participant and an ACT-R model. Further details of the QN-ACTR system are described in chapter III.

1.4 IMPRINT

The Improved Performance Research Integration Tool (IMPRINT) is a software package developed at the Army Research Laboratory (ARL) Human Research and Engineering Directorate MANPRINT Methods and Analysis Branch (HRED MMAB)

(`http://www.arl.army.mil/imprint`). It is a performance modeling application based on the Micro Saint Sharp discrete-event simulation software. IMPRINT provides tools for modeling mission-level logistics as well as operator-level task and cognitive workload modeling. Our research considers the latter feature set and identifies potential improvements. We use version 3.5 of the IMPRINT software.

IMPRINT can be used at any stage of the interface or product design process. It can be used to evaluate an existing system or to assess a system during development, and can reduce development costs by allowing designers to constrain a system from a human factors perspective before producing prototypes (Mitchell and Samms, 2009).

1.4.1 IMPRINT model structure

An IMPRINT task model is a network of task components or groups of task components call functions. In IMPRINT the atomic unit is referred to as simply a task, but we will continue to use the term task component to distinguish from an overall task. An example of an IMPRINT model is shown in figure 1.3. The model represents a simple stimulus-response task in which a character is shown on screen and a user strikes a corresponding key on a keyboard. The task is divided into three task components. First the character on the screen is perceived visually in the perceive-character task component. Second, the location of the corresponding key on the keyboard is recalled in the lookup-key task component. Finally, the stroke of the key is modeled in the strike-key task component. For such a simple task we would not need to use functions. However, functions are useful for representing task hierarchies in which subtasks are further subdivided into their own components. They are also useful for developing modular task definitions such as dual task scenarios. Both tasks could be defined in functions, which in turn are arranged in parallel at the top level of the network.

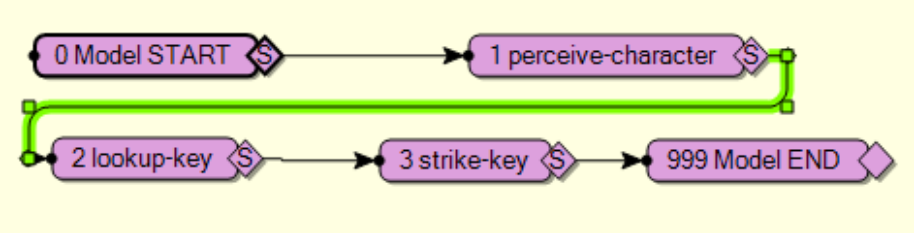


Figure 1.3: Example IMPRINT task network

At a global level, a task model can include variables and macros which are defined in the C# programming language. Variables can be used to store global state and macros are functions in the C# sense: a block of code which can take parameters, execute, and return a value.

Each component contains information relating to task performance. This includes:

1. Duration: Specified as a constant value, a random distribution of times, or an arbitrary expression which can reference model variables and macros.

2. Workload values: IMPRINT defines several resources including cognitive, visual, speech, and motor. A task component can have a workload value specified for each resource individually. The IMPRINT workload value scale is based on the VACP workload scale (McCracken and Aldrich, 1984; Bierbaum et al., 1989). A reference table is provided for each resource for common tasks such as reading text (visual), announcing a single word (speech), and pressing a button (motor). IMPRINT includes a method called the Advanced Workload System for combining the workload values for individual resources in separate but concurrent tasks into a single value using a matrix which specifies the extent to which two resources conflict.

3. Failure probabilities, Taxons, and Crew members: These allow analysts to define constraints on tasks but we will not make use of them in our work.

Task components also have events which fire as an entity progresses through the stages of its execution. An event is a C# function that an analyst can define that will be called at the appropriate times during the task simulation. In IMPRINT these events are referred to as effects. Task events consist of:

1. Release Effect: Called when an entity is scheduled to enter a task. The release effect returns a boolean value which determines if the entity is allowed to enter the task.

2. Beginning Effect: Called after an entity is allowed to enter a task and the task is beginning. This effect can, for example, be used to set global state used to communicate with other tasks or indicate that the task is running.

3. Launch Effect: Called when the entity is beginning but after the duration of the task has been computed. The Beginning Effect executes before the task duration is computed. This can be useful for starting graphical animations that depend on the task duration time.

4. Ending Effect: Called after an entity has finished its duration of task execution.

Task events are powerful tools for guiding and modifying task execution and we will describe how we use them in section 2.2

1.4.2 Workload management

Behavior under high workload conditions can be of particular interest to analysts. When an operator attempts to perform tasks under high workload, the incidence of errors can increase, some task components may be omitted, or the task may have a longer duration. We expect a model of human performance and workload to reflect these errors during task simulation.

In IMPRINT, analysts can use a Workload Management Strategies feature to specify a workload threshold that determines when a simulated operator begins committing errors. If a new task component is scheduled to begin when the current workload is above the predefined threshold, the simulated operator can use one of five strategies:

A. Perform all tasks, regardless of workload.

B. Do not begin the new task, and do not assign it to any other operator.

C. Perform tasks sequentially, beginning with the current task.

D. Interrupt current task to start new task. Current task resumes given opportunity.

E. Assign new task to a contingency or secondary operator.

The Workload Management Strategies feature in IMPRINT has several limitations. First, the analyst must choose only one strategy that will always be applied by the operator in high workload conditions. The strategy, therefore, does not depend on the total workload value, the workload values of individual components, task priority, task duration, or any other state information. Second, strategy applications can produce invalid task sequences. For example, under strategy B, if a task is not performed due to high workload, a subsequent and dependent task may be performed when the workload value decreases despite the omission of the previous task component.

1.5 Soar

Soar, like ACT-R, is a production system. However, unlike ACT-R, which is designed specifically to simulate human task performance accurately and in detail,

Soar is designed as an artificial intelligence (AI) system and is only loosely inspired by human cognition to facilitate complex problem-solving. The structure of ACT-R declarative storage or buffers is strictly specified and the rule actions are constrained in the way they modify the buffers. Soar, on the other hand, defines no relation between the contents of its working memory and entities in the real world, and productions can freely modify any stored information. Our work uses version 9.3.3 of the Soar software.

1.5.1 Problem state representation

Information in Soar is represented by working memory elements (WME) that are stored in a directed graph. A WME is a triple consisting of an identifier, an attribute name, and a value. In Soar syntax a WME is written `(<id> ^attribute value)`. We use this syntax when describing contents of working memory in Soar. The value of a WME can itself be an identifier, allowing for the construction of a graph of WMEs including loops. WMEs can be added by productions and have two types of persistence corresponding to two types of productions.

1.5.2 Productions

Soar is not a simple production system that identifies matching rules, selects one using a conflict resolution strategy, and applies the rule. Instead, Soar defines the concept of an operator, which is some action that causes a transition in the problem-solving space. The basic Soar algorithm is an infinite loop of operator proposals, the selection of an operator from the set of proposed operators using preferences, and the application of the operator.

Elaboration rules

There are two types of productions in Soar: elaboration rules and operator application rules. Elaboration rules, or simply elaborations, are fired whenever their conditions match the current state of working memory. Their actions describe changes to working memory that are applied when they fire. These changes persist only as long as the rule that caused them still matches. Elaboration rules are used to propose operators, declare preferences between operators, and to assert addition working memory state based on current conditions. As an example, an object in working memory may have two associated WMEs that describe an addition problem: `(<problem> ^addend1 4)(<problem> ^addend2 8)`. An elaboration rule could be written to match this state and compute the sum of the values:

```
(<p> ^addend1 <value1>)
(<p> ^addend2 <value2>)
-->
(<p> ^sum (+ <value1> <value2>))
```

As long as the object retains the two addend attributes, the rule will match and the object will be elaborated with the sum of the values. If either of the addend attributes is removed, the rule is retracted and the sum WME is immediately removed.

All matching elaboration rules fire and have their changes applied in parallel. If additional elaborations match as a result of the changes, they are fired in turn. This process continues in waves until quiescence, that is, until no more additional elaboration rules match.

Operators for a problem-solving state are proposed by adding `^operator <o>` to the state object using elaboration rules. Preferences between operators are also declared with elaboration rules. Operators can be rejected, required, declared best or worst, or better or worse than another operator. Details on operator preference

semantics can be found in Lehman et al. (2006).

Operator application rules

A production is an operator application production if it tests the selected operator of a state and modifies the state. All other productions are elaboration rules, described above. Because an operator transitions between states in the problem-solving space, the changes to working memory made by operator application rules persist after the rule no longer matches. A single operator may have multiple application rules.

Impasses

The Soar algorithm selects and applies operators to traverse a graph in the problem state space. The problem state is held in working memory and is defined by the WME graph constructed by operator application rules. There are some cases in which an operator cannot be selected or applied and so no traversal between states in the problem space occurs. This is referred to as an impasse. There are several reasons an impasse may occur (for details, see Lehman et al. (2006)), but each impasse is handled the same way. A new WME is created to represent a new problem space: to resolve the impasse that occurred in the original problem space. The Soar software distribution provides rules to aid in the resolution of certain impasse types. For example, if two or more operators are proposed but the preferences are insufficient to select one, Soar provides a set of rules to select and apply each eligible operator in the substate in order to evaluate their effect. Based on the evaluations, one of the operators is selected in the original state and the impasse is resolved. In order to resolve impasses in general, productions should be written to detect them and resolve the cause of the impasse.

1.5.3 Input and Output (IO)

Soar defines a special entry in working memory which allows a Soar agent to interact with an external environment. The `^io` link is established on the top state of working memory and has two sub-links: `input` and `output`, which we will refer to as the input link and output link, respectively. The Soar software supports the development of external environments in several programming languages including Java, C++, C#, and Python that communicate with a Soar agent via the Soar Markup Language (SML), the details of which we omit. These environments can place arbitrary WMEs on the input link for the agent to test in production conditions. The agent can place WMEs on the output link, which can then be read by the environment. A Soar agent typically places WMEs that describe an action it wishes to take, and the SML client updates the state of the environment accordingly, placing information describing the updated state on the input link for the agent.

1.5.4 Learning

Soar supports two types of learning called chunking (Laird et al., 1986) and reinforcement learning (Nason and Laird, 2005), and recent versions of Soar include implementations of episodic (Derbinsky and Laird, 2009) and semantic memories (Wang and Laird, 2007). However, we do not use these features in our work.

1.5.5 Using Soar in cognitive models

We initially investigated the use of Soar to model aspects of problem-solving cognition, much in the same way that ACT-R is used to model task performance. However, we decided against this for several reasons. Although Soar takes inspiration from human cognition, the architectural assumptions are too distant from limitations of human problem-solving to produce realistic models using standard Soar agents.

It may be possible to develop a base set of productions that implement constraints on the Soar architecture to more closely represent human cognitive faculties in the same way that ACT-R constrains a general production system to accurately portray human performance, however, our literature search did not find sufficient evidence that the task is tractable at this time.

Instead of using Soar directly as a cognitive model, we exploited its strengths as a rule-based reasoning system to implement a multi-task management framework. IMPRINT, as described in section 1.4, is used to simulate human performance using task-network models and features a workload management system designed to restrict task execution under high-workload conditions. The feature is, however, very limited: it contains a strict set of actions to take when it is activated; activation is based only on the current workload, ignoring all other state; and the workload is compared to a single, constant threshold value defined by the analyst to activate the actions. We implemented an IMPRINT plugin that allows an IMPRINT simulation to communicate with a Soar kernel instance. The plugin maintains a representation of the simulation state on the Soar input-link and manipulates the execution of tasks in the simulation based on commands it reads from the output-link.

This allows us to use Soar's powerful reasoning system to develop models of task management during multitask behavior based on operator and task state. We implemented these models by separating the agent into two components. The first is responsible for enumerating the possible task management actions based on the current simulation state and conforms to the theory described in section 2.1. This is accomplished by a set of productions that do not need to be modified to change the specific choice as a function of the current state. This choice is the responsibility of the second agent component. Given the set of possible task actions defined by the

management theory, this component selects the one action that should be applied. This reduces the burden on the task modeler to defining the function from state to action selection and implementing the function as operator preferences in the Soar agent. The task management models we have implemented are detailed in sections 2.4 and 3.7.5.

1.6 EPIC

Executive-process interactive control (EPIC) is a theoretic framework describing the performance of cognitive and perceptual-motor tasks. It supports the modeling of dual-task performance and, similar to Soar and ACT-R, has a computation implementation as a production system. EPIC was developed in the context of early iterations of ACT-R, Soar, and MHP, and is similar to the current ACT-R implementation in its detailed models of cognitive and perceptual-motor subsystems. It is more detailed than ACT-R in some aspects. For example, an efference copy of motor actions is available to the cognitive system via working memory, a detail that ACT-R lacks. EPIC also deviates fundamentally from the serial nature of ACT-R's production system which allows only one production to fire at a time. EPIC has no inherent limit on parallelism in the production system, testing all rules for matching conditions and executing all that match. Consistency of the model is enforced in production definitions and by EPIC's important concept of executive processes. These are defined with sets of productions that are matched and executed along with those that model specific tasks, but do not pertain to executing the tasks themselves. Instead they are uniform across task models and coordinate task execution by manipulating simulation state such as goals and working memory. Executive processes represent the human cognitive mechanisms that moderate task execution includ-

ing central task scheduling, task priority, operator strategies, and resource conflict (Meyer and Kieras, 1997a,b).

The concept of isolation and explicit implementation of these processes is in drastic contrast to the ACT-R approach, in which they emerge from the structure of task simulation using resource modules with availability constraints and a serial limitation of the central production system. Although we do not integrate the EPIC system with the other architectures described in this chapter, we adopt EPIC's model of the executive process in our extension to the IMPRINT software introduced in section 1.5.5 and detailed in chapter II. Much like EPIC's executive processes, our IMPRINT extension is a production system that interferes with the execution of components of task-specific models in order to enforce the constraints of human cognition. IMPRINT's task models, however, are hierarchical task networks as opposed to sets of productions in EPIC models. Therefore we define a theory formalizing the actions taken in response to task network events to bridge the discrete-event simulation nature of IMPRINT models with the production system nature of the Soar agent moderating task execution (section II.

CHAPTER II

IMPRINT and Workload Management with Soar

IMPRINT, described in section 1.4, is a widely used cognitive modeling tool in which tasks are modeled as hierarchical network of task components. Tasks are simulated by the passage of entities through the network, executing arbitrary code written by the modeler in the C# language when tasks begin and end. Task components have property attributes such as workload values and execution times which are used to analyze the difficulty and duration of the overall task. The Advanced Workload System features a method to automatically hamper the execution of task components in high workload conditions (Mitchell, 2000). However, as we discuss in this chapter, this feature is limited and inflexible. The goal of our work is to improve the management of task execution based on cumulative workload values. Parasuraman and Rovira (2005), in their review of potential improvements to IMPRINT, proposed similar efforts, in particular, using a feature which allowed modelers to define task priorities as a function of task properties. This feature has since been removed from the IMPRINT program.

We develop a theory of task management similar to EPIC's executive processes (Meyer and Kieras, 1997a) and an implementation of the theory as an extension to the IMPRINT software. The extension was developed using the Soar cognitive

architecture and is more powerful and flexible than the built-in workload management system. We show that a task simulation of UAV operators moderated with the extension exhibits load-shedding strategies observed in the behavior of human UAV operators (Schulte and Donath, 2011).

The scope of the resulting task management model is limited to endogenous attention allocations. We do not, for example, theorize about how operators react to interruptions concurrently with task performance. Similarly, UAV task described in this chapter, and the experimental tasks described in chapters III and IV contain few, if any interruptions. However, we do suggest that the architecture presented here could integrate interruption management findings (Sarter, 2013; Lu et al., 2013; Nikolic et al., 2001) in an effort dedicated to modeling exogenous attention allocation.

2.1 Workload management extension theory

To formalize our extension of task management in IMPRINT we defined the set of decisions that must be made about whether a task should execute and what those decisions can be. A diagram of the decisions is shown in figure 2.1. The first decision type is whether a task component is allowed to begin after the previous component ends. This is called the release decision and corresponds to the release effect in an IMPRINT task component. The decision is made in response to a single task component eligible for release in the context of the state of other concurrent tasks. The decisions are similar to IMPRINT's strategies except that we are only interested in the task performance of a single operator, so we do not have a decision corresponding to the (E) IMPRINT strategy. The four actions that do result from the decision correspond to strategies A through D and are shown in figure 2.1:

A. Perform task: Allow the new task component to begin without affecting other

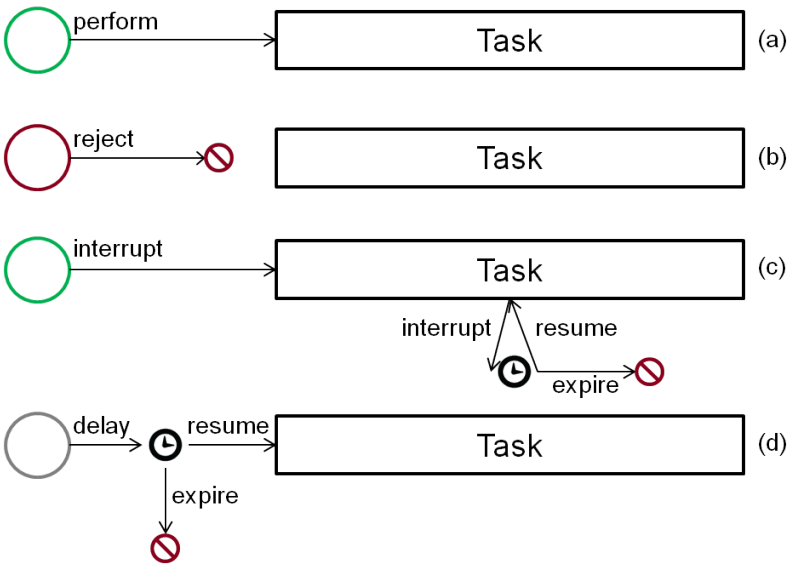


Figure 2.1: Possible actions taken for a scheduled task component

concurrent tasks.

B. Ignore task: Do not allow the new task to begin. In this case the task is never executed, nor are subsequent tasks that would have been executed after this task was complete.

C. Delay task: Do not allow the new task to begin, but consider beginning the task at a later time.

D. Interrupt task: Allow the new task to begin and interrupt a currently executing task. Consider resuming the interrupted task at a later time.

The Delay and Interrupt actions result in a task component being put in a suspended state and incur a new decision: if and when to resume a suspended task. We call this the Resume decision. Because any task in a suspended state was delayed or interrupted in favor of the execution of other tasks, the resume decision is made each time an active task component ends. The action taken in a resume decision is to resume zero or more suspended tasks. The action specifies which of the currently

suspended tasks should resume. There is no distinction made between tasks which are suspended due to an interruption or a delay action.

The third and final decision type is whether to expire a suspended task. This can correspond to a task which is interrupted for a long enough duration that a higher level task in a hierarchy must be restarted.

2.2 Extending IMPRINT

To implement the IMPRINT workload management extension we developed two primary software components: a plugin to establish communication between a Soar kernel and an IMPRINT simulation, and a Soar agent to make decisions based on the current execution state.

2.2.1 Plugin capability

The IMPRINT program is written in C# and exposes an interface that can be used by third-party code that is compiled to a dynamic link library (dll) called a plugin. A plugin can subscribe to events that occur during the course of running the IMPRINT program itself and individual runs of an IMPRINT simulation. Some events are the same as those that are available within IMPRINT such as the Beginning, Release, Launch, and End effects. A plugin can even modify the value returned by the Release effect, allowing or disallowing an entity from starting a task. Other events include variable initialization, simulation beginning, simulation ending, and application ending. All of the IMPRINT features available to code written in effect handlers are also available to plugin code. For example, plugin code can search for, suspend, resume, and modify entities, print to the console window, and even halt the simulation.

2.2.2 Soar plugin

In order to use Soar to make decisions governing an IMPRINT simulation, we developed an IMPRINT plugin that a ran a Soar kernel instance using its C# interface. The full source code of the plugin is available at `https://github.com/sirctseb/Scope`.

When the plugin is initialized, it creates a Soar kernel which will run the agent used to make decisions. The plugin also adds event handlers to IMPRINT events, each of which is described below.

Initialize variable

An event is fired for each variable that the analyst has defined when a simulation begins. We use this event to enable the workload management extension. To allow the soar agent to make decisions and modify the simulation, a variable named EnableScope must be defined in the IMPRINT model. When the plugin receives a variable initialization event corresponding to the variable, it sets an internal flag to enable decision making.

Simulation begin

This event is fired when an IMPRINT simulation starts running. When the plugin receives the event, it creates and initializes the agent by loading the soar files from disk and setting the agent to initial state.

Simulation end

When the IMPRINT simulation ends, the plugin destroys the agent object and writes statistics about the decisions made to log files.

Application close

The plugin shuts down the Soar kernel when the IMPRINT program is closing.

After release condition

This event is fired when an entity is being evaluated for release into a task, after the Release effect code define in the IMPRINT interface has run. The handler can return a boolean value that overrides the value returned in the Release effect code.

Release effect precedence The handler is passed a boolean value that indicates whether the Release Effect code defined by the analyst in the IMPRINT interface returned true or false. If the value is false then there is some domain logic that indicates that the entity should not be allowed to enter the task. The Soar plugin respects this decision and returns false as well.

Resume task precedence The handler then checks if the task was delayed and the agent has made the decision that it should resume. If it is, the handler returns true, indicating that the entity should now be allowed to begin the task. If it is not delayed task but there exist delayed tasks that are waiting to resume, the handler returns false. This allows delayed tasks that the agent has decided should resume to begin executing before making decisions about whether new tasks should start.

Deferred decisions There is no guarantee that, if the release condition for a given entity waiting to enter a task returns true, then the entity will, in fact, immediately enter the task. Multiple entities waiting to enter tasks may have their release conditions evaluated before any of them actually enters. After evaluating the release conditions, IMPRINT may allow any one entity whose condition was true to enter

its task. The remaining entities will have their release conditions evaluated again before allowing them to start.

For this reason, the Soar plugin can not know that a task will start if the release condition handler returns true. Therefore, the decision to allow a task to resume is stored in a list of decisions pending confirmation from IMPRINT that the decisions took effect. In this case, we must wait until the beginning effect event (section 2.2.2) fires for an entity before we can remove the flag that indicates the agent has decided the task should resume.

Delayed tasks Next, the handler checks if the entity is in a suspended state due to a decision to delay the task. If so, the handler returns false, preventing the task from starting. All delayed entities will be prevented from beginning tasks this way until the agent makes a decision to resume them (section 2.2.2).

Deferred decision replacement As described in section 2.2.2, decisions made by the Soar agent do not always incur immediate action in the IMPRINT simulation. An entity will likely have its release condition evaluated several times before it begins a task. As described in section 2.2.2, information about the Soar agent's decision in the release condition is stored in a deferred decision object. Therefore, in the release condition handler, the plugin checks for a deferred decision strategy associated with the current entity. If it exists, it is destroyed so that a new release decision can be made for the entity.

Release decision If none of the special cases described above apply, then the Soar agent must make a decision about whether the current entity should begin its task. The plugin creates a new entry on the agent's input-link to represent the task. The

entry contains a unique identifier for the task, the time the entity first tried to enter the task, and the task priority, salience, and workload values. Finally a flag is added to the entry to indicate that the agent must make a decision about whether the task can begin. The plugin then runs the Soar agent until it produces a release decision, which the plugin reads from the output link and carries out the decision in the IMPRINT simulation.

First, the decision made by the Soar agent is stored in a deferred decision and the task is removed from the Soar input-link. If the decision is to delay or ignore the task, the entity is flagged for later processing and the handler returns false so that the task cannot begin. If the decision is Perform task or Interrupt task the handler returns true.

Beginning effect

When a task is allowed to begin a task, it is because the Soar agent chose Perform task or Interrupt task in a Release decision, or it chose to resume a delayed task in the Resume decision. When the entity enters a task, the Beginning effect event fires. The plugin implements a handler for this event to process the task that is starting.

The handler first removes the flag that indicates that the entity should be allowed to resume if it is set. If the last decision made was an interrupt decision the handler suspends the entity that the Soar agent determined should be interrupted and a flag is set on the entity to indicate that it has been interrupted. Finally, the Soar agent is notified of the new simulation state by adding the task to the input-link.

Ending effect

When a task is complete the Ending effect event fires. The plugin handler for this event removes the task from the Soar input-link to indicate to the agent that the

task is no longer running.

Resume decision The ending of a task indicates a change in system state that may permit a delayed or interrupted task to resume. Therefore, if there are any delayed or interrupted tasks when an ending effect event fires, the handler runs the Soar agent until it makes a Resume decision. If the agent determines that a delayed task should be resumed, the plugin removes the delayed flag from the corresponding entity, adds a resume flag, and removes the task from the Soar input-link. The task will be added back to the input-link in the beginning effect event handler.

If the agent determines that an interrupted task should be resumed, the plugin removes the interrupted flag resumes the corresponding entity, which was suspended in the IMPRINT simulation. The plugin does not need to add a resume flag to the entity like it does for a delayed task because the entity had already started its task and it is able to immediately resume.

Clock advance

When the simulated time in IMPRINT discrete event simulation advances, a Clock advance event is fired. The plugin handles this event to update the state of the Soar agent. When the event fires, the handler adds the clock value to the Soar input-link. It then checks for any deferred decisions to ignore or delay tasks. These decisions, which are made in the release condition event handler, can only be made when the clock changes because IMPRINT evaluates release conditions multiple times for the same task. The plugin can not know which will be the last evaluation until the clock changes.

If the plugin finds a deferred ignore decision, the corresponding entity is aborted. If the plugin finds a deferred delay decision, a delayed flag is added to the entity and

the task is added to the Soar input-link with a `^delayed yes` property.

Expire decision The handler also uses the clock advance effect to request an Expire decision from the Soar agent. The handler places a `^decision-request expire` entry on the Soar input-link and runs the agent until a decision is returned. If the decision states that an task should expire, the corresponding entity is aborted in the IMPRINT simulation and the `decision-request` entry is removed from the input-link.

After entities aborted

This event occurs whenever an entity is aborted for any reason. The plugin handles this event to remove tasks from the Soar input-link for each aborted entity to keep the agent state synchronized with the IMPRINT simulation.

After entities suspended

When the Soar agent decides to interrupt a task, the corresponding entity is suspended and this event fires. The plugin handler for this event removes the `^active yes` property that indicates to the Soar agent that the task is running and adds a `^delayed yes` property to show that the task is suspended.

After entities resumed

When an interrupted task is resumed, the corresponding entity is resumed in the IMPRINT simulation and this event fires. The handler for this event removes the `^delayed yes` property and replaces it with the `^active yes` property to indicate to the agent that the task is running.

2.3 Soar agent

The IMPRINT plugin described in section 2.2.2 listens for events in the IMPRINT simulation, passes state information to the Soar agent, and runs the agent, which uses the information on the input-link to make Release, Resume, and Expire decisions. To organize the processing required to make the decisions, the agent uses a subgoal for each decision type which prepares the possible choices that the agent can make in response to a decision. The selection among these choices is a question of cognitive theory and is independent of the architecture to produce them.

2.3.1 Release decision subgoal

When an IMPRINT entity is scheduled to begin a task, the corresponding release condition is evaluated and the release condition event is fired. The plugin responds to this event by placing the task on the Soar input-link with a `^release yes` property attached. All other tasks that are executing in parallel are already on the Soar input-link with a `^active yes` property. The agent tests for the existence of a task with a `^release yes` property and proposes the `release-decision` operator which becomes a subgoal when it is selected.

Within the `release-decision` subgoal the agent proposes the following operators: one `perform-task`, one `ignore-task`, one `delay-task`, and an `interrupt-task` operator for each active task on the input-link. This set of operators represents the entire set of responses to a Release decision for the current state. The decision, therefore, reduces to choosing among the proposed operators and the implementation of the choice depends on multitasking theory and is described in section 2.3.4. The Soar agent contains rules to apply whichever operator is selected. An application rule places a description of the decision on the output link which is subsequently

read by the IMPRINT plugin.

2.3.2 Expire decision subgoal

The IMPRINT plugin requests an Expire decision by placing a special entry on the input-link: `decision-request expire`. The Soar agent tests for this property and creates an `expire-decision` subgoal. In this subgoal, the agent elaborates each delayed task with the amount of time it has been delayed. The agent proposes an `expire-task` operator for each delayed task and the choice of whether to expire a task is a matter of cognitive theory and described in section 2.3.4. The agent defines an application rule that adds an expire decision to the output link if an `expire-task` operator is selected. If no operator is selected, a state no-change impasse occurs. The agent detects the impasse and proposes an `expire-none` operator that is selected and placed on the output-link upon application.

2.3.3 Resume decision subgoal

If the agent finds a task on the input link with a `^delayed yes` property, it proposes a `resume-decision` operator in order to create a subgoal. This can occur when there is also a task with the `^release yes` property, in which case the agent should be making a release decision. Therefore, the agent defines an elaboration rule which always prefers a `release-decision` operator to a `resume-decision` operator.

In the Release decision subgoal, the agent proposes a `resume-delayed` operator for each delayed task on the input link. The actual decision of whether to resume a task is, again, a theoretical question and the implementation is described in section 2.3.4.

If all `resume-delayed` operators are rejected, a state no-change impasse results and a substate is created. The agent detects the substate and proposes an operator

named `resume-none` similar to the expire decision (section 2.3.2). This operator is selected in the absence of any other eligible operators.

2.3.4 Response selection

The Soar agent infrastructure described above maintains the state of the IMPRINT simulation and enumerates the possible response choices for each decision as operators. The choice among these responses is implemented by defining rules that declare preferences between the proposed operators. By separating the responsibilities we allow for the implementation and extension of response selection without requiring knowledge of the Soar agent architecture or even how to develop a full Soar agent. The barrier to developing custom task-component management models is reduced to understanding simple Soar rule definition syntax and operator preference semantics. We have developed a workload threshold based decision procedure similar to the Advanced Workload System method. We describe the models and the Soar preference rules that implement the response selection below. For details on preference semantics in Soar see Laird (2012) or Lehman et al. (2006).

2.4 UAV Study

Schulte and Donath (2011) performed an empirical study of UAV operators in which they identified load-shedding strategies that the operators applied in high workload conditions. The participants controlled one or three UAVs through three stages of a reconnaissance task. In addition to controlling the UAVs, the participants performed an object identification task, which required that they search the terrain for targets. When a target was identified, they entered the location into a map. They then zoomed in on the target and judged whether the target was friendly or hostile. Finally, they entered the classification into the map. The participants were also

responsible for identifying Surface to Air Missiles (SAM) in the third phase of the task. The participants exhibited five qualitatively distinct load shedding behaviors:

1. Proactive task reduction: In a three-UAV control task, a participant would ignore two of the three UAVs and only use one for the mission.

2. Less exact task performance: A participant would spend less time and would not use full zoom during the target classification subtask, compromising task accuracy.

3. Omission of subtasks: Of the three subtasks in the object identification task, only one or two would be completed before moving on to the next target.

4. Complete neglect of the object identification task: In the third stage of the mission, workload was increased by the addition of a separate task, and the object identification task was not performed.

5. Purposeful delay of task accomplishment: A participant would begin an object identification task for a certain target, but would proceed to start on another target of higher priority before returning and resuming the original task.

We developed an IMPRINT model of the UAV mission task and simulated the task performance using the Soar agent to make task network decisions. Each stage of the task consisted of a small subnetwork representing the flight control task and produced variable workload. The third stage flight subnetwork included an additional component to represent the additional SAM detection task faced by the participants. A single object identification task subnetwork ran in parallel with the flight tasks. In the IMPRINT model, the simulated operator was expected to identify six targets per mission segment.

2.4.1 Release decision

The selection of release decision responses for the UAV model is based on a constant workload threshold. The preferences described below select responses which heuristically prevent the simulated operator from exceeding the workload threshold. Lower level models such as QN-ACTR have functional models of the perceptual, motor, and cognitive mechanisms that simulate task performance, and therefore, natively incur resource conflicts that limit task concurrency. IMPRINT models only describe the resource-specific workload values associated with task components and limitations are not enforced by the software. The workload threshold heuristic approximates the detailed QN-ACTR approach by using these workload values as a proxy indicator of resource conflict.

Our current implementation of action selection is based on the cumulative task workload across resources. However, strict resource conflicts that emerge from ACT-R's modular structure could be implemented in the Soar system by examining workload values at a resource-specific level. For example, ACT-R defines only one declarative module with one associated retrieval buffer, preventing a dual-task production that makes a retrieval buffer request from firing when the buffer is busy. A Soar agent could delay tasks with a non-zero cognitive workload value while another task with a cognitive workload value is active. On the other hand, ACT-R separates the "where" and "what" components of visual perception and allow them to be used concurrently. To intentionally match this behavior a Soar agent could define a visual workload threshold value and allow the concurrent execution of tasks with visual workload values until the threshold is met. This flexibility in modeling an executive process and conflict in a single model applies to all resources, allowing the Soar model to adapt to theoretical changes more easily than ACT-R or QN-ACTR, which would

require architectural modifications.

`elaborate*perform-task*threshold` Each perform-task operator is automatically elaborated with the workload that would result if the task was allowed to begin. This rule matches operators with workload values that exceed the workload threshold and elaborate them with a `^over-threshold yes` property that indicates a workload threshold violation if it was selected.

`preference*perform-task*under-threshold` The primary heuristic in this task management model is to allow all task components to execute as long as the workload threshold is not exceeded. This rule declares a best preference for a perform-task operator as long as it does not have a `^over-threshold` property attached. As long as no other operator has a best preference declared, the perform-task operator will be selected. In this model, no best preferences are declared for ignore, delay, or interrupt task operators. Therefore, all tasks will be performed unless they would exceed the workload threshold.

`prefer*not*perform-task` This rule matches operators with the `^over-threshold` property described above and declares a reject preference for the operator, which prevents the operator from being selected.

The preferences described above cause the perform-task response to be selected as long as the workload threshold is not selected. If the workload threshold is exceeded Soar is unable to choose from all other proposed responses. These responses consist of one ignore-task response, one delay-task response, and one interrupt-task for each task component already executing in the simulation. When this occurs, an operator-tie impasse occurs and a substate is created to resolve the tie.

In the substate, the eligible proposals are evaluated using the selection.soar system, which simulates the selection of each tied operator and evaluates the result of their application. A rule exists for each type of response that computes the workload if the operator was selected. The workload is used as the metric used by selection.soar to evaluate the utility of the operator. The operator with the minimum value is selected. If two or more operators have equal evaluations, one of them is selected randomly. This method of resolving operator ties chooses a task to suspend which results in a minimum resulting workload.

2.4.2 Resume decision

A response for the resume decision is selected using the same heuristic as the release decision. Each resume-task operator proposed in response to the release decision is elaborated with the workload that would result if the task was resumed. A reject preference is declared for any operator that would resume in a workload in excess of the threshold.

If zero proposals remain, a state no-change impasse results and a resume-none operator is proposed and selected as described in section 2.3.2. If exactly one operator is eligible, it is selected by default. If more than one operator can be selected, an operator-tie impasse is created. As for the release decision, the operators are evaluated based on the workload values that result if they were selected. This has the effect of resuming the task with minimum workload.

2.4.3 Expire decision

The model uses a simple time limit for delayed and interrupted tasks. Any expire-task operator whose task is not over the time limit is rejected. Of the operators with tasks that have exceeded the suspension time limit, one is randomly selected as the

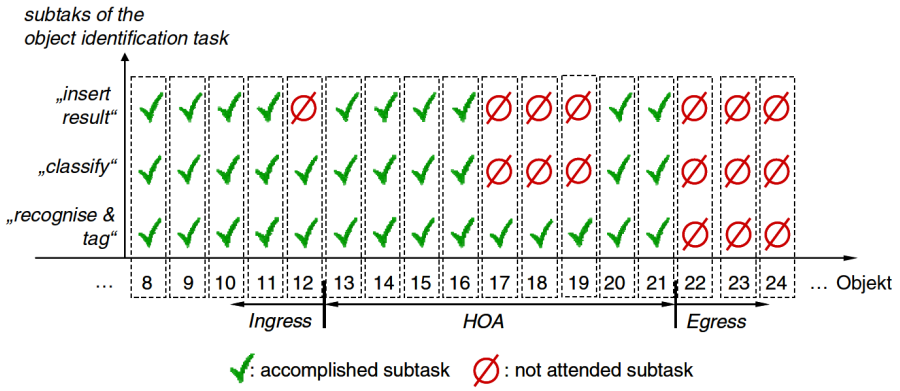


Figure 2.2: Empirical partial and complete task omission. From Schulte and Donath (2011)

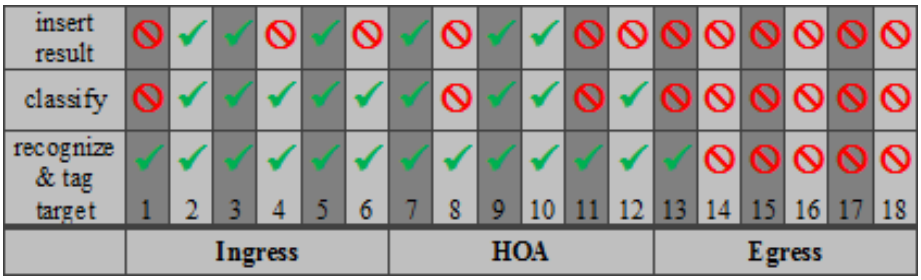


Figure 2.3: Modeled partial and complete task omission

response.

2.5 Results

Figure 2.2 shows the subtask completion and omission for the object identification task during one participant's trial. In the first two stages of the mission there are some instances of partial task omission. Specifically, the participant fails to insert the result for object 12, and fails to classify and insert results for objects 17, 18, and 19. During the third stage, the participant fails to complete any part of the object identification task.

Figure 2.3 shows the subtask completion and omission from one run of the simulated task using the Soar agent. We can see that the model produces partial task completion in the first two stages and complete task omission in the third stage.

In the first two stages of the mission, the variable workload in the flight control tasks combined with the workload of the object identification subtask components exceeded the overall workload threshold. In these cases, the object identification task components were delayed or interrupted in favor of the flight task components, which had lower workload values. Generally, the object identification task components were interrupted and resumed several times before being completed, reflecting the behavior of an operator as he checks in periodically on the flight controls while performing the identification tasks. In some instances, the identification tasks would be suspended for long enough for the Soar agent to determine that they should expire. In these cases, that subtask component and the subsequent components would be omitted. In the final stage, the increased workload due to the additional SAM detection task disrupted the task component that moderated the object identification subtask. In this case, the simulated operator ceased all activity in that subtask and would not restart it even after the workload was reduced to normal levels. This behavior was observed in the empirical UAV operation study.

2.6 Conclusion

We developed an extension to IMPRINT that expands upon and functionally replaces the Advanced Workload System feature. The plugin implements a task management theory that enumerates the actions that can be taken in a hierarchical, discrete-event task performance model when task components are scheduled to begin, end, or resume. The extension monitors the simulation state and passes information to a Soar agent running in parallel, which is responsible for determining which action to take at each decision point. The agent architecture is divided into two primary components. The first enumerates the possible actions according to the

task management theory and does not need to be changed in order to adjust the method by which actions are chosen. The second is the operator preferences which select the action to be taken from those produced by the first. The preferences are simple to define and will allow experts to implement and test theories of task management while modifying a minimal amount of code. We used the applied theory in the IMPRINT software and were able to model results observed in an empirical UAV study in which participants performed two simultaneous tasks and exhibited distinct load-shedding strategies (section 2.4). In chapter III we apply the extension to improve IMPRINT predictions of dual task performance with an addition and targeting task.

CHAPTER III

Modeling concurrency on addition and targeting tasks in QN-ACTR and IMPRINT

In this chapter we develop and validate a QN-ACTR model of the multi-task performance of two tasks. The tasks chosen were a targeting task in which two of three targets were to be identified as enemies by their color and dismissed by moving the cursor over the target and clicking, and an addition task in which a problem of the form A+B was displayed and the sum of the numbers was to be announced orally. A detailed description of each task is given in section 3.2. The performance of the tasks requires visual, memory, motor, and speech resources, all of which are represented in standard ACT-R modules. We describe the experimental tasks (section 3.2), the QN-ACTR models of the tasks (section 3.3), the IMPRINT models of the tasks (section 3.4), and the method and results of the experiment (section 3.5, 3.6).

3.1 QN-ACTR additions

The QN-ACTR system was developed by Cao and Liu (2011a) as an implementation of the ACT-R cognitive architecture in Micro Saint Sharp, a general purpose discrete event simulation program. Cao and Liu verified the ability of QN-ACTR to duplicate ACT-R simulations of 20 different ACT-R models. This work has provided the Queueing Networks (QN) real-time visualization and workload prediction capa-

bilities to ACT-R models. We used QN-ACTR to produce utilization-based workload values during task simulation and to simulate the concurrent execution of the two tasks.

We implemented a number of ACT-R features in QN-ACTR which were needed in the task models described in section 3.2. The primary addition was support for the temporal module which models the internal monitoring of the passage of time (Taatgen et al., 2007). Other additions included support for incremental mouse movements, resolving visual search conflicts by selecting the result nearest to a given location, and preparing a motor request in advance of its execution.

3.2 Tasks

3.2.1 Targeting task

The targeting task begins with the onset of three targets in random locations on the screen. Two of the three targets are enemy targets and one is a friend target. Participants are to dismiss only the enemy targets. A target is dismissed when the participant aims the pointing device cursor within the target area and presses a button on the device. The targeting task is complete when both enemy targets have been dismissed.

The experiment contains two independent variables governing the targeting task. We describe the first as a difficulty variable. In the low-difficulty level, enemy targets appear red and friend targets appear green for the duration of the task. In the high-difficulty level, each target appears black until the participant places the cursor over the target, at which time the target's color is shown. The target's color is only visible while the cursor is over the target, that is, if the cursor is moved away from the target, it will again appear black. The intent of the difficulty variable is to shift the enemy target recognition task component from a highly automatic, preattentive

search to a decision made under time pressure and requiring memory retrieval.

The second independent variable is the speed with which the targets move across the screen. The targets move in a straight line and remain within the screen area for the duration of the iteration. The target speed variable takes values of 0 and 200 pixels per second on a monitor with 96 dots per inch.

The participants stand approximately eight feet from a 24-inch widescreen monitor and the targets are 128x128 pixels. The participants control a blue crosshair cursor by aiming at the screen using a Nintendo Wii remote controller. Pressing the primary trigger button on the controller produces a mouse click event that dismisses a target under the cursor. At the end of each six-second iteration, any target that has not been dismissed is removed from the screen before the next iteration begins.

3.2.2 Addition task

At the beginning of each iteration, an addition problem with two operands is displayed in black text in the center of the monitor with a font size of 100 pixels. The participant mentally adds the two numbers and responds orally with the resulting value. When the participant announces the correct sum, the color of the text is changed to green to provide feedback to the participant. No feedback is provided if the participant announces and incorrect sum and they are responsible for recomputing the sum and responding again. There is no limit on the number of responses the participant can provide.

The difficulty of the addition task is an independent variable controlled by the values of the operands in the problem. In the low-difficulty case, the operands are between 1 and 12. In the high-difficulty case, the operands are between 13 and 25.

3.3 QN-ACTR task models

3.3.1 Addition model

Mental arithmetic is widely studied with experiments conducted to examine nearly every aspect of the process from the encoding of numeric and arithmetic information to the variation of strategies used as a function of addend size. Early theories of single-digit addition proposed that both children and adults solved problems using a counting algorithm with adults having a lower incrementing time (Groen and Parkman, 1972), while more recent studies show that the counting strategy is almost entirely replaced by direct retrieval by adulthood (Barrouillet and Fayol, 1998; Svenson, 1985). Studies have presented evidence that the convergence to a single retrieval approach does not hold for more complex (e.g. two-digit) addition (Thevenot et al., 2001). Instead, a more complex process is required in which the addends are parsed into tens and units values which are added individually and incremented in the case of a carry (Thevenot et al., 2007). Our model attempts to capture these basic assumptions, that mental addition in adults is primarily retrieval-based, that single-digit problems are accomplished by direct retrieval, and that two-digit addition requires a multi-step process in which values are separated into tens and units places and added sequentially.

Many aspects of mental addition are still under debate, including whether numbers and arithmetic facts in memory are encoded in a single format (Blankenberger and Vorberg, 1997) or multiple, for example, analogically, verbally, or visually, which are used in combination depending on the task being performed (Thevenot and Barrouillet, 2006); and the source of the tie effect, in which problems with repeated operands are solved more quickly (Blankenberger, 2001; Campbell and Gunter, 2002; LeFevre et al., 2004). As the purpose of our model is to study the interaction of the addition

task with a concurrent targeting task, we do not attempt to capture these effects or model one among competing theories. Instead we focus on identifying and modeling the resource requirements that drive the performance time and conflict with the concurrent task. For example, we do not subscribe to a single theory of the encoding format of number and arithmetic facts in long term memory, but we do assume that the recognition of a number after the visual perception requires a retrieval before the representation of the addition with another value can be retrieved. Furthermore, we do not model errors in retrieval or other parts of the addition process, although empirical studies have found error rates as high as 5% in problems with low numbers (0-9), and 10% in problems with high numbers (13-39) (Thevenot et al., 2007). We define the addition task to be complete when the correct sum is announced and therefore absorb any increase in time due to error into the performance time. We also assume that visual attention is allocated to each addend exactly once during a given problem, that is, that each number is retrieved successfully upon first examination. There is empirical evidence for this assumption in single-task conditions with single-digit numbers (Zhou et al., 2011).

We produced one ACT-R model for both conditions of the addition task because the independent variable is purely quantitative. To perform the addition task, the model searches for each addend in turn, separating the perceived text into internal representations of the ones and tens digits. The sum of the unit digits is recalled from memory and stored along with whether a carry digit was produced. Next the sum of the tens digits is recalled and optionally incremented in the case of a carry. Finally the resulting tens and unit digits are spoken in order. In the low addend range condition, the algorithm reduces to computation by one retrieval for the recognition of each addend from visual perception and a subsequent retrieval to

produce the sum of the two values. In high addend range conditions, each two-digit number is visually perceived and a retrieval is required to parse the value into tens and unit places and two more retrievals are required to produce the sum of the tens and ones values. Finally, if the ones values sum to ten or greater, a final retrieval is required to increment the tens place following evidence that memory is involved in carry operations (Ashcraft and Kirk, 2001).

Declarative memory The addition model uses three types of memory elements. The first represents the concept of a number. The model is initialized with a memory chunk for the numbers 1 through 24. Each of these number chunks contains a value slot that contains the text of the number, a ones slot containing the unit digit of the value, and a tens slot containing the tens digit. The model retrieves a chunk corresponding to a given number in order to obtain the ones and tens places so they can be added separately.

The second type represents the sum of single-digit numbers. There is an addition-fact chunk for each permutation of two single-digit numbers. Each addition-fact has first and second slots that hold the addends, a result slot that contains the sum of the addends, a ones slot that contains the ones value of result, and a carry slot that contains the tens value of result. The model retrieves an addition-fact to compute the sum of single-digit numbers. The model assumes that direct memory retrievals of sums are limited to single-digit numbers and that the ones and tens places of two-digit numbers are added separately.

Finally, a successor chunk represents consecutive pairs of natural numbers. A chunk of this type is retrieved to compute a new tens place value when there is a carry.

In ACT-R and QN-ACTR, the retrieval time of a chunk from declarative memory is a function of the activation of the chunk:

$$RT = Fe^{-(f*A_i)}$$

The activation, A_i is defined as the sum of three parameters and a noise term. The three parameters are the base-level activation, the spreading activation, and the partial matching value.

$$A_i = B_i + S_i + P_i + \epsilon_i$$

The latter two parameters are 0 under the default ACT-R configuration, which we do not override. The base-level activation for a chunk in the model can be set using the `(set-base-levels chunk-name level)` function. We defined separate base levels for four groups of chunks: low number chunks for numbers between 1 and 12, high number chunks for numbers between 13 and 24, addition chunks which represent results of single-digit additions, and successor chunks that store the next integer number greater than a given number (e.g. `(s01 isa successor value 1 successor 2)`). This final chunk type is used to implement carry operations.

All ACT-R parameters were kept at their default values except the parameter to enable subsymbolic computation (:esc) necessary to model memory retrieval times with varying activation, the activation noise parameter (:ans) which was set to 0.5 in accordance with previous studies (Cao and Liu, 2011b), and the base level activation values of the chunks representing numbers and arithmetic facts. We constrained the relative magnitude of the base level values according to the complexity of the information and the frequency of use. For example, chunks representing the low numbers were larger than the high numbers and addition facts. The high number activation level was lower than the addition facts to represent the complexity of

separating the number into ones and tens places. We selected the values of the base levels within these constraints by fitting the model-simulated execution times to empirical data in single-task conditions. The base level value for each type is listed in table 3.1.

Table 3.1: Base levels for addition task declarative memory chunks

Chunk type	Base level
Low numbers	3.5
High numbers	2
Addition facts	2.7
Successor facts	1

Goal chunk The goal chunk in the addition model is used to store the current state of processing as well as intermediate results. As is common in ACT-R models, the state slot of the goal is tested in the conditions of all productions and often contains the name of the production that should fire next.

The goal chunk also contains slots where the ones and tens digits of each addend are stored after they are retrieved from memory, slots to contain the ones and tens values of the result of the addition, and a slot to record whether a carry will occur.

Reading the first addend The find-first production checks that the visual-location buffer is empty and requests a visual search for the leftmost text object on the screen, which is always the first addend. Next the attend-first production harvests the visual-location buffer, which only provides basic information about the object that is subject to preattentive search. To get the content of the text, the production checks that the visual buffer is free and empty and requests to move visual attention to the location of the text. The encode-first production harvests the text content from the visual buffer, checks that the retrieval buffer is free and empty, and requests a chunk from memory representing the number.

When the retrieval is complete, one of two productions will harvest the separate digits from the memory chunk and store them in the goal chunk. This will be the store-first production if the number is 10 or greater, and the store-first-nil-tens production otherwise. Two productions are necessary because if the number is less than 10, the tens place in the memory chunk will have the special value nil and a production will not match if a variable in the conditions is nil. store-first-nil-tens explicitly checks for the nil value, allowing the production to match. These two productions do not check the goal state and will file sometime after the retrieval is complete between subsequent productions that visually process the second addend. This allows visual processing to continue while the memory chunk is retrieved.

Reading the second addend The two productions that perform the visual search for, and move the visual attention to the second addend are named find-second and attend-second and are exactly equivalent to find-first and attend-first except that they search for the rightmost text on the screen. encode-second is equivalent to encode-first except with the additional condition that the first-ones slot of the goal chunk is not nil. This forces the production to wait until the result of the retrieval of the first addend has been harvested and stored in the goal before requesting the retrieval of the second addend The store-second and store-second-nil-tens productions are similar to store-first and store-first-nil-tens in that they harvest the number chunk from the retrieval buffer and store the digits in the goal. However, in this case, they also initiate a new retrieval request for a memory chunk describing the sum of the unit digits of the two addends to begin the addition computation.

Computing the sum The finish-retrieve-ones production harvests the result of the retrieval requested by the store-second* productions. This memory chunk has a slot

for the units place of the sum of the numbers and a slot that indicates if a carry is needed. finish-retrieve-ones stores these two values in the goal chunk and changes the goal state to add-tens.

At this point in the simulation, processing splits into seven cases based on whether the tens places must be added and whether there is a carry. First, both original addends were less than 10 and their sum is less than ten. In this case, the first-tens and second-tens slots of the goal are nil, and the carry slot is 0. The add-tens-nil-nil-no-carry production checks for these conditions and changes the goal state to response to immediate start responding with the one-digit sum.

Second, both original addends were less than ten, but their sum is at least 10, so the carry slot of the goal chunk is 1. This case is handled by the add-tens-nil-nil-carry production which sets the tens slot of the goal chunk to 1 and changes the goal state to response to begin the oral response.

Third, the first addend was at least 10, the second was less than 10, and the sum of the unit places is less than 10. The add-tens-first-nil-no-carry production checks for this case by matching the first-tens slot of the goal to a variable, the second-tens slot to nil, and the carry slot to 0. No addition of the tens places is necessary because only one addend has a tens value, so the first-tens value is stored in the tens slot the goal chunk and the goal state is set to response.

The fourth case is the same as the third case except the sum of the units places is at least ten. The add-tens-first-nil-carry is the same as add-tens-first-nil-no-carry except the carry slot is checked to be non-zero and the goal state is changed to check-carry to increment the tens place before responding.

The fifth and sixth cases are equivalent to the third and fourth cases, respectively, except that the first addend is less than 10 and the second addend is at least 10. The

add-tens-nil-second-no-carry and add-tens-nil-second-carry handle these cases in the same way as the productions described for the third and fourth cases.

The seventh and final case is that both addends are 10 or greater. In this case another summation retrieval is required to add the tens places. The add-tens production checks for this case and sets the goal state to retrieve-addition-tens.

Adding the tens place In the event that both addends are 10 or greater, the state is set to retrieve-addition-tens, and the retrieve-addition-tens will be selected when its condition that the retrieval buffer is empty and free is met. The production initiates requests a chunk representing the sum of the tens digits from memory.

When the tens digit sum is retrieved and there is no carry, the finish-retrieve-tens-no-carry production harvests the buffer contents and stores the sum variable in the tens slot of the goal chunk and sets the goal state to response. If there is a carry, the finish-retrieve-tens-carry production will do the same but set the goal state to check-carry to increment the tens place.

Carry the one If the sum of the unit digits of the two addends is 10 or greater, the tens place of the overall sum must be incremented. This is done by the check-carry production after a previous production sets the goal state to be check-carry. The production checks that the retrieval buffer is empty and free and requests a memory retrieval to find the value that is one greater than the current tens place. The increment-tens production harvests the result of the memory retrieval, stores the incremented tens value in the goal chunk, and changes the state to response to begin announcing the result.

Response The verbal response begins in the response state and is split into two cases: one for two digit sums and one digit sums. If the sum is less than ten, then the respond-ones-no-tens production is selected. It checks that the vocal system is free and makes a request to announce the unit digit which, in this case, constitutes the entire sum.

If there is a tens digit, the respond-tens production is selected and makes a request for the tens digit of the sum to be spoken. It also sets the goal state to response-ones. As soon as the vocal buffer is ready for the preparation of a new request, the respond-ones production is selected and make the final request for the unit digit to be spoken.

3.3.2 Targeting models

In contrast with the addition task, it is not feasible or useful to produce a single model to capture the performance in all task conditions. Instead, we created a separate model for each condition. The models could be unified in a trivial way by adding a simple set of productions that chose among the four sets of productions after receiving the information about which task condition was beginning at the beginning of the trial. In this section we describe the ACT-R models developed for each task condition. They all share an overall procedure of identifying a target, moving to it, and clicking to dismiss it. Individually, they vary to account for the movement of the targets and monitoring and decision required in the high-difficulty case.

In all four models, the right hand is initially placed on the mouse and the cursor is positioned at the center of the screen. During cursor movements, the cursor location is updated every 10 ms and the cursor-noise parameter is enabled, which introduces random variability in the mouse movements.

Low speed, low difficulty

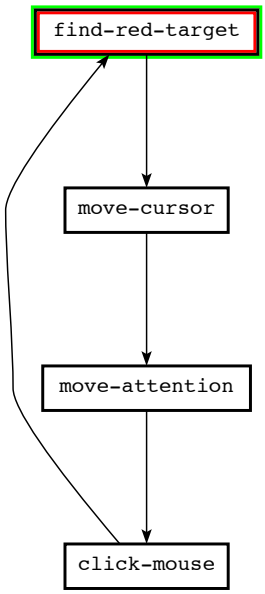


Figure 3.1: Production graph of the low-difficulty, low-speed targeting model

Goal chunk All four targeting models use the state slot of the goal chunk to coordinate the task execution. The goal chunk for the low-speed, low-difficulty model contains a target-location slot to temporarily store the visual location of a target between productions and a check-miss slot to correctly handle misses as described in section 3.3.2.

Finding a target The model for the low-speed and low-difficulty case is the simplest of the four and consists of only five productions. The first, find-red-target, requests the visual location of a red object that has not yet been attended to: an enemy target. This type of visual search based on simple properties such as color is modeled

as a preattentive search in ACT-R (Anderson and Lebiere, 1998) and consumes no simulated time on its own. The production itself lasts 50 ms as with all productions. The production requires that the visual location buffer be empty in order to be selected in keeping with the "greedy-polite" guidelines (Salvucci and Taatgen, 2008). The production also requires that the state of the goal is "find-red-target" and changes the goal state to "move-cursor" after it fires.

Moving the cursor The second production, move-cursor, checks for an oval in the visual location buffer. The simulated targets are represented by buttons which appear in the visual location buffer as ovals. If the task had other types of ovals in the display or if a dual-task also used ovals, the check would have to be made more specific to ensure the correct object was found. However, the addition model only requires text objects that would be ruled out by the oval constraint. The production also checks that the manual system is free and clears a flag called check-miss that will be explained in section 3.3.2. When the production fires, it makes a request to the manual system to move the cursor to the location in the visual location buffer and sets the goal state to move-attention. This causes the move-attention production to fire and a request to move visual attention to the same location. The time to move the visual attention to the location has mean 85 ms but the time until the next production is dominated by the mouse movement which is based on Fitts' law and therefore depends on the distance to the target and the size of the target.

Dismissing the target The third production, click-mouse, requires that the manual system be free and harvests the contents of the visual buffer, which by then contains the target object. The production requests a click-mouse action from the manual system and clears the visual buffer to prevent changes in the visual scene from filling

the visual buffer automatically. This is not strictly necessary for the function of the model but prevents some conflicts with dual-tasks that can arise when the visual buffer is left full. The production sets the goal state to find-red-target so that the model will seek out any remaining enemy target. The mouse click takes 150 ms but the succeeding production does not require a free manual system so it can fire before the click has finished. After the click-mouse production fires, the find-red-target is selected, completing the cycle.

Missing ACT-R supports modeling manual inaccuracy with the :cursor-noise parameter. When set to true, a random vector is added to the final location of the cursor during move-cursor requests such that the cursor lies over the target area along the approach vector 96% of the time. When these misses occur the target is not removed. However, the target is no longer unattended so the visual-location search in the find-red-target production will fail to find the target, none of the first three productions match, and the task remains incomplete. We solve this problem using the fourth production, no-unattended-red. This production is selected when the goal state is move-cursor and checks for an error state in the visual-location buffer that occurs when a visual location search fails. The production sets the goal state to no-unattended-red-search and the corresponding production performs a new visual search for a red oval but does not require the object to be unattended, allowing any target that was missed to be found. The goal state is set back to move-cursor so that the move-cursor production can fire if a previously missed target is found.

The no-unattended-red also sets the check-miss flag to avoid an infinite loop upon task completion. Without the flag, the production would be selected again if there were no red targets remaining, as the visual buffer would again be in an error state

and the model would continue firing the production forever. To prevent this, the production checks that the check-miss flag is not set before firing, and sets the flag when it does fire. If there are no remaining red targets then the visual location search fails and the fifth production, no-red, is selected and puts the goal to an end state. To prevent the model from ending early when two targets are missed, the move-cursor production clears the check-miss flag.

High speed, low difficulty

The model of the high-speed, low-difficulty task is similar to the low-speed, low-difficulty model with some additional productions to account for the motion of the targets. The model cannot simply move the cursor to the location of a target it finds on the screen because the target will have moved by the time the cursor arrives. To solve this problem we add productions which monitor a target briefly to gauge its motion and predict a location where the target is expected to be when a mouse movement is complete.

Goal chunk In addition to the target-location slot, the goal chunk for the high-speed, low-difficulty model contains target-x and target-y slots to store the component values of a target location so that the model can perform the projection computation described in section 3.3.2.

Finding a target The model begins the task by seeking a red target with the find-red-target production. The production is the same as in the non-moving model in that it checks for an empty visual-location buffer and requests a visual search for a red oval. However, the production also checks that the manual system is also free for technical reasons explained in 3.3.2.

Predicting target movement In contrast with the non-moving case, in which the model could simply move the mouse cursor to the location found by find-red-target, the model must account for the movement of the target. The cap-first-location production harvests the visual-location information requested by find-red-target and stores the screen position of the object in the goal chunk. The ensuing production, cap-first-location-search makes another visual search request for a red oval. The request specifies that the object should be the one nearest to the location just found: the exact same object at its new location.

The next production to fire is lead-target, which again harvests the result of the visual-location search. This production fires 50 ms after cap-first-location in a single-task simulation and possibly more in a dual-task simulation when one or more productions from the other task are selected between the two. The non-zero time span allows the object to move slightly between the two visual-location requests. The lead-target production matches the screen position values from the first search request that were stored in the goal chunk and calculates the predicted target location. The target projection is implemented by producing a unit vector in the direction from the first matched location to the second, multiplying by a constant, and adding the result to the more recently matched location. The value of the constant is 139 and was selected to maximize accuracy of the projection. In order to move the cursor to the predicted location in a subsequent production, the predicted location values are stored back into the visual-location buffer.

Moving the cursor The move-cursor production is equivalent to that in the non-moving case: it verifies that the visual-location buffer contains a target object and checks that the manual system is free before requesting a move-cursor action from

the manual system. The following production, move-attention checks that the visual buffer is free and empty and requests for visual attention to be moved to the projected target location.

Dismissing the target The moving condition model uses the feature in ACT-R's motor module that allows a motor request to be prepared before it is executed. The prepare-click production fires while the cursor is being moved to the target as soon as the preparation stage of the manual system is available and makes a manual request to prepare a mouse click. The click is not executed until the click-mouse production fires, which tests that the manual system is once again ready to prepare an action. The early preparation allows the click to occur faster when the cursor moves over the target and reduce target misses. The production also clears the visual buffer.

Missing Unlike the non-moving model, the find-red-target production does not constrain its visual search to unattended targets. Due to the design of the manual system in ACT-R, objects that are moved are considered new and are therefore unattended. The model cannot use the attended state of objects to process which were the focus of previous movements. This simplifies the handling of misses as any red target can be found by the find-red-target visual search and we do not need to have a production to check for attended but extant targets if it fails. Instead, a production named fail-find detects a failed visual-search from find-red-target and moves into a final state.

Because find-red-target cannot filter targets that are currently being clicked from its visual search, it would occasionally find them and restart the process of dismissing one even though it will likely be dismissed in a few milliseconds. In this case, the simulation would fail somewhere in the ensuing production sequence. To avoid this

problem, we require that the manual system be free before selecting the find-red-target production. This requirement implies that, as long as the click doesn't miss the target, the target will be dismissed and removed before the visual search occurs.

Low speed, high difficulty

Each target in the high difficulty task condition is shown in black until the cursor is positioned over it, whereupon the target is shown to be either red or green. Modeling the performance of this task condition is more complex and requires three major additions:

1. Productions to store the location of the green target when it is found and check against it after subsequent visual searches to avoid moving to it more than once (section 3.3.2, 3.3.2).

2. A sequence of productions required to monitor for the change in color while the cursor moves over a target (section 3.3.2).

3. A memory retrieval to determine the appropriate action based on the target color (section 3.3.2).

Goal chunk The goal chunk in this model contains all of the slots of the high-speed, low-difficulty model as well as friend-x and friend-y slots that contain the location components of the green target.

Finding a target The high difficulty model starts by performing a visual search for an unattended black target. The find-black-target production is exactly the same as in the low difficulty, low speed condition except for the color constraint in the visual search.

Avoiding repeated green target checks The process of storing the location of the green target is described in section 3.3.2. After find-black-target fires, the goal state is set to move-cursor, and one of three cases will be true:

1. The cursor has not yet been moved over the green target.
2. The cursor has been moved over the green target, which is not the object found by find-black-target.
3. The cursor has been moved over the green target, which is the object found by find-black-target.

There is one production for each of these cases: move-cursor-no-friend-info (section 3.3.2), move-cursor-not-friend (section 3.3.2), and avoid-friend (this section) respectively. The avoid-friend production harvests the visual-location buffer and checks that the x,y location values are the same as the green target x,y, location values stored in the goal chunk. This condition verifies the case that the object found in the visual search is the green target. The production sets the goal state back to find-black-target for the visual search to be repeated to find a different target.

Moving the cursor The move-cursor-no-friend-info production checks that the goal chunk does not contain any the location of the green target, verifying that it has not yet been found. The production checks that the manual system preparation phase is free and makes a manual request to move the cursor to the result of the visual search. Finally, both the visual-location object and the x,y values of the object are stored in the goal chunk. The former so that the model can refer to the location when performing subsequent visual searches and the latter so the values can be stored as the green target location if this target is found to be green. The goal state is set to

move-attention which causes the move-attention production to fire next, making a visual request to move attention to the target. The visual buffer is then cleared by the harvest-visual production

The move-cursor-not-friend differs from move-cursor-no-friend-info only in its conditions. It checks for the remembered green target location in the goal buffer and compares it to the visual search result to ensure they are different.

Monitoring for target color After previous productions begin moving the cursor to the location of a black target, the check-target production matches the visual-location object from the goal chunk that refers to the target of focus. When it fires a visual search is requested for an oval at the exact location of the target found by the initial search. The state is changed to distinguish-target.

The visual search made by check-target starts a production loop in which visual searches are repeated requested for the original object until it is no longer black. The loop is implemented in the distinguish-target-black production, which checks for a busy manual system and a black oval in the visual-location buffer and requests a new visual search for the object without changing the goal state. This production will be repeatedly selected and fired as long as the target remains black and the cursor is moving.

When the cursor enters the target bounds the target will become red or green. The detect-target-color checks for an oval in the visual-location buffer that has any color but black and is selected as soon as the target becomes colored and its other requirement, that the retrieval buffer is free and empty, is satisfied. The production's actions are described in the next section 3.3.2.

One more production is required to prevent an infinite loop when the cursor misses

the target and the target never changes color. The distinguish-whiff production checks that the result of the most recent visual search is a black target, that the manual system is free, meaning the mouse movement has completed, and that the goal state is distinguish-target. When the production fires, a new manual request is made to move the cursor to the same target and a new visual search is made to restart the color monitoring loop.

Note that this second cursor movement is not within the closed-loop feedback control system that is suggested to enforce Fitts' law (Gawthrop et al., 2008). Fitts' law is used within ACT-R and governs the timing of each discrete cursor movement. The total time and trajectory of a motor action in ACT-R is calculated before the motion starts and cannot be altered during the movement. The second cursor movement will be small but distinct from the original, requiring its own motor preparation and initiation time. This introduces some difficulties in modeling the targeting task which emerge during simulations of dual-task conditions as discussed in section 3.7.

Decision making When the detect-target-color production fires, a declarative memory chunk is requested for a response whose color slot matches the color of the visual-location object. This retrieval models a human's decision based on target color. The goal state is changed to decide-whether-to-shoot.

When the appropriate action is retrieved from memory, one of two productions will be selected to carry out the decision. The decide-to-shoot production matches a chunk in the retrieval buffer when the action slot contains the value shoot, and changes the goal state to click-mouse to begin the process of dismissing the target. The decide-not-to-shoot production matches when the action is dont-shoot and retrieves the x,y values of the target that were stored by the production that moved

the cursor. When it fires, these values are stored as the green target x,y location values in the goal chunk and the goal state is changed to find-black-target to begin searching for a new target.

Dismissing the target The click-mouse production checks that the manual system is free, requests a mouse click and sets the goal state back to find-black-target.

Special cases A model with the productions described above can fail if two targets overlap. If the cursor is moved into the area where the targets overlap, both will be shown in color. If one target has already been dismissed, there will be no remaining black targets so the visual search requested by find-black-target will fail. Therefore, we include a production, fail-find, which detects the failed visual search and performs a new search without the black color constraint.

High speed, high difficulty

The high speed, high difficulty model is the most complex of the four as it includes the extra considerations of both the high speed, low difficulty model and the low speed, high difficulty model.

Goal chunk The model introduces five additional slots to the goal chunk. The first two are friend-x-diff and friend-y-diff, which store the motion information about the green target when it is found. These are required to achieve the same green-target avoidance system implemented in the low-speed, high-difficulty model which only needs the location information. The next two are cur-x-diff and cur-y-diff, which store the motion information of the target currently being processed. Finally, the heuristic slot contains information used to reduce the number of times the

avoid-friend production fires. The heuristic initially contains the value lowest and is explained in section 3.3.2.

Finding a target The find-black-target production checks that the visual-location buffer is empty and matches the heuristic slot of the goal chunk which contains either lowest or highest. The visual-location buffer allows searches constrained by x or y value. A search can specify an absolute value, a relative value (e.g. >4), or can use the special values lowest and highest to select the match with the lowest and highest values for the slot, respectively. In this case, the production provides the values contained in the heuristic slot to the screen-x slot of the search, which causes the search to return the leftmost or rightmost matching location, depending on the heuristic value. Initially, the heuristic slot contains lowest, so the leftmost black target will be found by the search.

Avoiding repeated green target checks The method used to avoid repeatedly moving the cursor over the friend target in the high-speed, high-difficulty model is similar to that in the low-speed, high-difficulty model. However, there are two additional complications. The first is that the search heuristic should be toggled whenever the friend target is avoided. For example, if the find-black-target searched for the leftmost target and that target is the friend target, then the next firing of find-black-target should be for the rightmost target, so that it is less likely to find the friend target again. Therefore we have two productions to detect the friend target: avoid-friend-lowest and avoid-friend-highest. Both check that the target found by the find-black-target search is the friend target and set the goal state back to find-black-target to begin a new search. The first checks that the heuristic slot contains lowest and sets updates the value to be highest. The second does the opposite. The second

consideration is that a target can not be identified only by a location value. Instead, information about a target's movement is stored in the goal state as described in section 3.3.2. The avoid-friend* productions test if the object returned by the visual search lies on a line constructed from the stored information. This is accomplished using an ACT-R !bind! statement which stores the result of the is-on-line function into a condition variable =on-line. If the current object does not lie on the motion path of the friend object, the function returns nil, which prevents the production from matching. The production will also fail to match if no motion information is stored in the goal chunk.

Predicting target location The low speed, high difficulty model required two move-cursor production variants to handle the first two cases described at the beginning of section 3.3.2. This model contains the extra step of obtaining movement information from a target before moving the cursor, and so, contains two cap-first-location variants for the two cases.

The cap-first-location production is similar to that in the high speed, low difficulty model. It harvests the visual location buffer, stores the x,y values in the goal in case they need to be stored to identify the green target later, and requests a new visual search for the same target so that the next production can predict the target motion based on two locations. The production also checks that no friend location information is stored in the goal chunk.

The cap-first-location-no-friend production harvests the visual location buffer and checks that it is not the green target by binding a condition variable to (not (is-on-line)), that is, the logical inverse of the is-on-line function. The production stores the x,y location of the object and the object reference itself in the goal chunk.

The lead-target production harvests the visual location buffer, calculates a predicted target location, and stores it back to the buffer just as in the high speed, low difficulty model. In this case, the change in locations is also stored in the goal chunk in case the target turns out to be green.

Moving the cursor The move-cursor production is identical to that in the high speed, low difficulty case except that the visual-location object harvested is stored in the goal chunk to that future searches are based on the predicted target location instead of the current location.

Monitoring for target color The method for detecting a change in a target's color is similar to the low speed, high difficulty case. Since the targets are moving, we can't request visual searches based on an exact location. Instead the check-target production performs an initial search for an oval nearest to the predicted target location and moves to the distinguish-target state. As long as the target remains black and the cursor is still moving, distinguish-target-black will match and fire repeatedly. The detect-target-color production is the same as in the low speed, high difficulty model.

This model also accounts for the case in which the cursor movement has enough error that it does not move over the target, which would normally cause an infinite loop. The distinguish-whiff production does not, however, immediately request a new motor movement for two reasons:

1. the target is moving and a new predicted location would need to be computed first, and

2. the originally predicted target location is computed such that nearly all whiffs

occur when the cursor lies in a place where the target will soon move into.

Therefore, instead of making a new motor request, two productions are introduced to wait a short period of time in case the target moves to the final cursor location, at which time the model can react to its color.

The distinguish-whiff production begins initiates a timer from the temporal module and requests a new visual search for the target. The whiff-spin production checks that a timer is running and that the total number of ticks is less than a constant number, 16, and that the result of the visual search is a black target. The details of the temporal module are described in Taatgen et al. (2007) and the ACT-R 6.0 Reference Manual. It makes a new visual search request, forming loop as long as the target remains black and 16 ticks have not elapsed. During this time, the goal state keeps the value distinguish-target, so that detect-target-color will be selected if the target moves under the cursor and becomes colored. After 16 ticks elapse, the move-after-whiff production will match and change the goal state to find-black-target, starting a brand new target search.

Decision making The decide-to-shoot production is similar to that in the low speed, high difficulty model except that it includes a request to the manual system to click the mouse. During the cursor movement, a production called prepare-click makes a request to the manual system to prepare a mouse click action, which is executed when decide-to-shoot fires. This is designed to reduce the time required to click the mouse button, which helps reduce misses when targets are moving.

The decide-not-to-shoot is also similar to the low speed equivalent but it must also record the difference between the two object locations in addition to the the first object location found.

3.3.3 Concurrency model

The concurrency model implemented in QN-ACTR is based on the threaded cognition model proposed and implemented in ACT-R by Salvucci and Taatgen (2008). The theoretical single-task performance assumptions made by the model essentially describe the architecture of the ACT-R system: that a number of resources such as declarative memory, perceptual, and motor systems are available to perform a task and that they function as buffers that can be read and written and have their states queried. Additional assumptions describe multi-task behavior. First, multiple tasks are maintained as separate goals whose processing is threaded. Second, the resources used to accomplish tasks process requests in serial, essentially abiding by a strict version of Multiple Resource Theory (MRT) (Wickens, 1984). Third, the individual goal threads use resources in a "greedy, polite manner" (Salvucci and Taatgen, 2008). A greedy-polite thread checks that the resources it requires are available, requests them for processing, and ensures they are free and available when it is finished using them. This assumption has the largest effect on task model development while the other three specify the implementation of the theory. Finally, preference is given to the least recently processed thread when two or more threads have productions available for processing.

To use the QN-ACTR instantiation of threaded cognition, one model must first have its references to the goal module changed to a second goal module. For example `=goal>` is replaced by `=goal-2>`. The new name of the goal module does not imply that the task is a secondary task, only that it is a conceptually separate task in the mind of the modeled participant. The second and more demanding modeling task is to ensure that the productions of each model abide by the greedy-polite method of resource acquisition. There is no computational requirement that the

models exhibit the greedy-polite property, that is, any valid ACT-R models can be run in a multi-task context with threaded cognition. However, if the models require the same resources, they can very easily fail to accomplish their tasks during multi-task simulation even if they operate correctly independently. We found in particular that extra care must be taken when two related resources are used in sequence. For example, a common pattern in ACT-R is to request a search from the visual-location buffer and request a move-attention action from the visual buffer using the result of the search. A fatal conflict can occur if a dual task makes a visual request concurrent with the first goal's visual-location request and a visual-location request immediately thereafter. In this case it is insufficient for productions to maintain greedy-politeness in the two buffers independently but instead must check for the availability of the second resource before requesting the first. With the greedy-polite property in place in the individual models, no further considerations must be made to run them in a dual-task context in QN-ACTR.

3.4 IMPRINT task models

Unlike ACT-R which is a production system, IMPRINT is based on a general purpose discrete event simulation tool called Micro Saint Sharp. IMPRINT analyses are developed by dividing a task into components and arranging them in a network through which entities flow. Each task network has a single beginning but may branch and loop within the network. Each task component is assigned workload values for a number of cognitive, perceptual, and psychomotor resources. The time required for an entity to pass through a task component can be defined as a constant, a random distribution, or a mathematical expression defined by an analyst in C#.

ACT-R models define the conditions under which productions fire and the pro-

cesses that begin when productions fire. IMPRINT task components, however, correspond to the processes themselves and the conditions are contained in the paths between components. Therefore, IMPRINT task components correspond to the time between productions instead of the productions themselves. We have developed IMPRINT models of each of the ACT-R models of the experimental tasks. The task components in the IMPRINT models generally have a one-to-one correspondence between pairs of successive ACT-R productions. For example, figure 3.1 shows the ACT-R production graph, and the IMPRINT task network for the low-speed, low-difficulty targeting task is shown in figure 3.2. The name of each task component in the IMPRINT model is a combination of the two corresponding ACT-R productions.

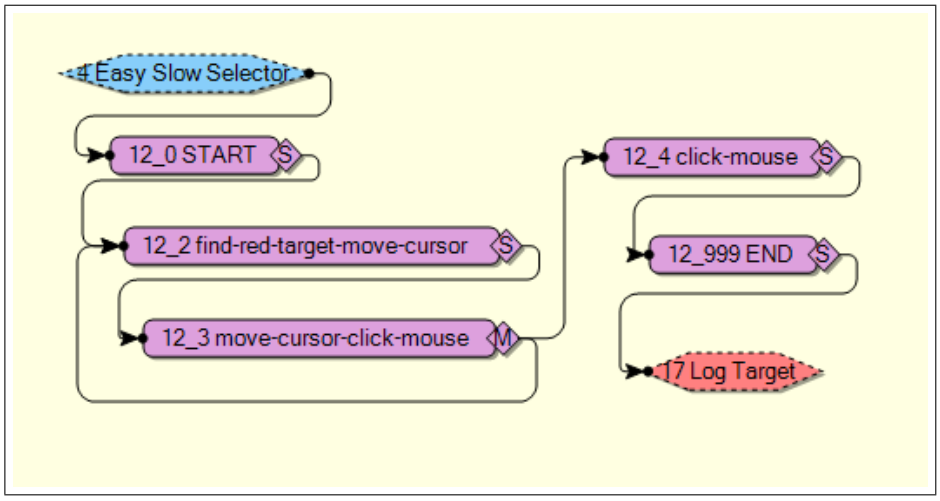


Figure 3.2: Low-speed, low-difficulty targeting task network in IMPRINT

To assign durations to the task components, we identified the time between production firings in the ACT-R model. For example, if a production makes no requests in its conditions and the next production can fire immediately, the corresponding task component has a duration of 50 ms, which is the time between a production being selected and firing. If a production requests to move visual attention to a location, a random time with mean 85 ms is added. For ACT-R resource requests that

result in stochastic execution times, we use identical execution time distributions in the corresponding IMPRINT task components. As a result, we can expect the IMPRINT and QN-ACTR models to predict nearly identical completion time values in single-task conditions.

Finally, we assigned workload values to each task component using values in the IMPRINT workload reference tables that correspond to the task activity. The IMPRINT tables include values for visual search, monitoring, motor movements, speech, and cognitive tasks such as simple association and calculation.

3.5 Method

20 participants were recruited from a population of undergraduate and graduate students at the University of Michigan. Half of the participants were male and half female to control for a potentially confounding gender variable. Participants completed trials in which they attempted to complete one or more tasks quickly and accurately. Each trial consisted of 12 task replications in which the participant had six seconds to complete the tasks presented. The participant's goal was to complete the tasks while minimizing execution time and errors. The participants were scored based on both execution time and errors and the score was visible to the participants.

All independent variables are within-participant factors and the order of conditions was randomized. In addition to each block condition of the dual-task, participants completed trials of the addition task at each difficulty level without the targeting task, and the targeting task at each combination of the speed and difficulty independent variables without the addition task. A table of independent variables is shown in Table 3.2.

Table 3.2: Experiment 1 Design

Independent variables		
Targeting difficulty	Low	High
Target speed (pixels / second)	0	200
Addition operands	[1,12]	[13,24]

3.5.1 Scoring

Participants received a score on each task that was visible to them to reward fast and accurate task completion. The participants received 100 points for each enemy target dismissed and 100 points for a correct answer to the addition problem. 100 points were deducted for each friend target dismissed, and 20 points were deducted for each button click that did not hit any target. In order to encourage fast performance, 100 points were added to the score for each second remaining second in the iteration after the overall task is completed. The participants were informed of the scoring policy before the experiment began.

3.5.2 Procedure

Before beginning the trials, we described the addition and targeting tasks and the parameter levels to the participant. We also described the scoring system in detail to ensure participants attempted to complete the tasks quickly and accurately. Participants first completed a set of practice trials that consisted of one trial of each single and dual-task condition. Between each trial, participants were permitted to take time to rest in order to avoid the effects of fatigue.

After verifying that the participant was comfortable with each task condition and was capable of performing the tasks, the participant began experimental trials. The participant performed two experimental trials of each dual and single-task condition. After the two trials in each block condition, the participant responded to the NASA-TLX survey that was presented on-screen (Hart and Staveland, 1988). After all

block conditions were completed, participants weighted the sources of each workload subscale according to the NASA-TLX procedure.

3.5.3 Apparatus

We implemented the experiment as a web browser application using HTML, CSS and the Dart language. The browser window occupied the entirety of a 24-inch widescreen monitor. The Nintendo Wii remote controller senses four infrared LEDs that were affixed to the corners of the monitor and were powered by a 5V USB connection. We developed Java software to compute the screen location where the remote was pointing and produce corresponding system events for mouse movements and clicks, which were received by the browser. Software for logging and analyzing data is written in Dart, Go, perl, and R. A screenshot of the experiment task interface is shown in figure 3.3.

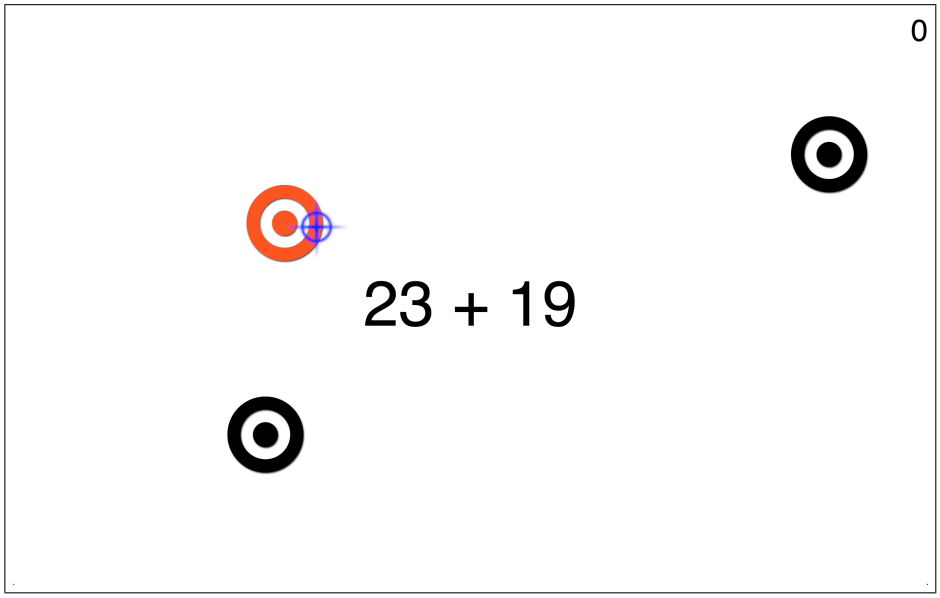


Figure 3.3: Experiment task interface

3.5.4 Data collection

After each block, the participant responded to the NASA-TLX survey to measure the subjective workload of each task condition. The survey asks respondents to assess the workload of a task on six subscales: mental demand, physical demand, temporal demand, performance, effort, and frustration. During the experiment, mouse events including movements and clicks were recorded. Target hits and misses were recorded to examine the error rate and execution time of the targeting task. During trials with an addition task, an audio recording was taken of the participant's responses. 3 tones are played prior to the start of the trial to synchronize the audio data.

3.6 Results

3.6.1 Empirical results

Single tasks In the single addition task scenario, the addend value range variable had a significant effect on the execution time ($\alpha = 0.05$). The mean completion time for the low addend range was ≈ 1.27 and for the high range was ≈ 2.05.

In the targeting task, the difficulty variable and the interaction between difficulty and speed had significant effects on the execution time while the speed variable itself did not. The difficulty variable increased on the execution time ($\alpha = 0.05$) from mean ≈ 1.84 to mean ≈ 2.39 across speed levels and was similar with the speed variable fixed at either level. The speed variable itself had a significant effect on execution time only with the difficulty variable fixed at the low value ($\alpha = 0.05$). At low difficulty, an increase in target speed reduced the mean of the execution time from ≈ 1.87 to ≈ 1.80.

Dual tasks The targeting difficulty and addition difficulty variables had significant increases on task execution time in the dual-task conditions while the target speed

variable had a small but significant decrease in execution time ($\alpha = 0.05$). The increase in speed decreased the mean execution time by ≈ 0.12 when the addend range was low and was not significant in the high addend range cases. The variable interactions had no significant effects.

Concurrency To quantify concurrency we defined a metric based on the relationship between the execution time on the dual task with those on the single tasks of the same condition. For a given pair of single task conditions A and T, let the average independent execution time of each be ET_A and ET_T respectively. For the dual-task condition AT where tasks A and T are performed simultaneously, the execution time is $ET(AT)$. Then we define concurrency as

$$C = \frac{ET_A + ET_T - ET(AT)}{\min(ET_A, ET_T)}$$

The properties of this definition of concurrency are described in section 3.7.

Each of the three independent variables as well as the interaction between addition difficulty and target difficulty had significant effects on concurrency ($\alpha = 0.05$), with the addition difficulty variable having the largest reduction in concurrency (mean ≈ 0.78 to mean ≈ 0.58). The target difficulty variable reduced the concurrency value while the speed variable increased concurrency.

3.6.2 QN-ACTR Model validity

Single tasks Bar plots of the empirical and modeled execution times on each single-task condition are shown in figures 3.4 and 3.5 for the addition and targeting tasks respectively. The difference in completion times between the empirical and modeled data is significant only for the low-difficulty, high-speed targeting task.

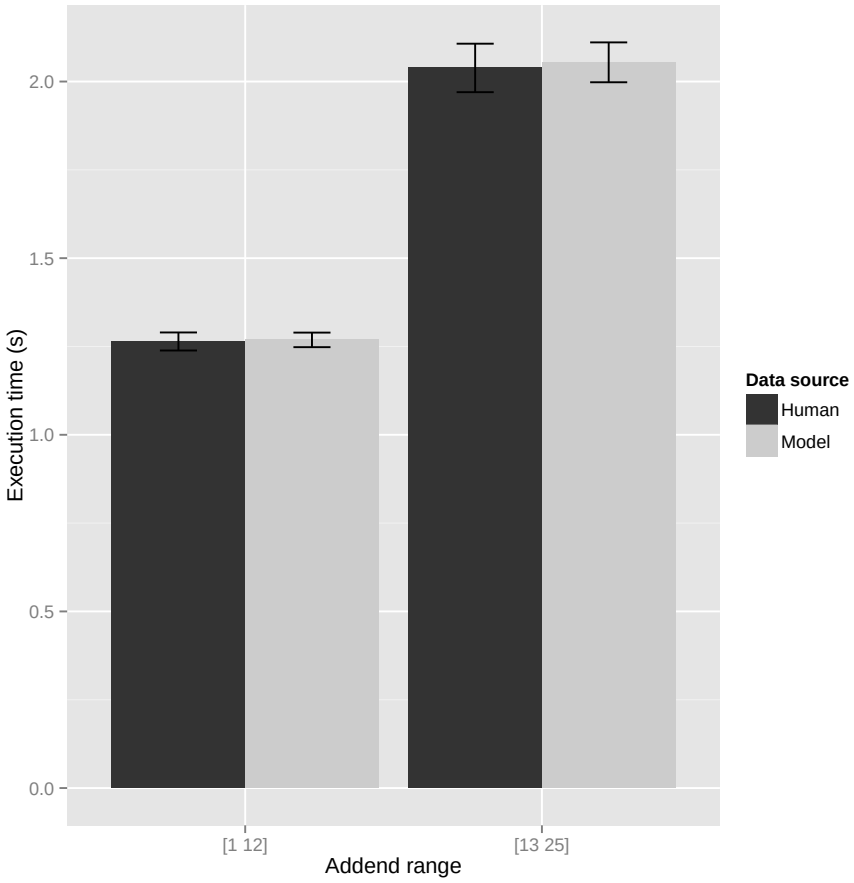


Figure 3.4: Execution times by difficulty in single-task addition problems

Dual tasks Dual task empirical and modeled execution time boxplots are shown in figure 3.6. There are significant differences in completion times between the model and empirical data for two cases: in the high-difficulty, high-speed targeting conditions.

Accuracy During experimental trials, each button click is logged and classified as a hit or miss. We define the targeting error rate as

$$\frac{misses}{shots}$$

The overall error rate across targeting tasks in the empirical data is ≈ 0.067 while

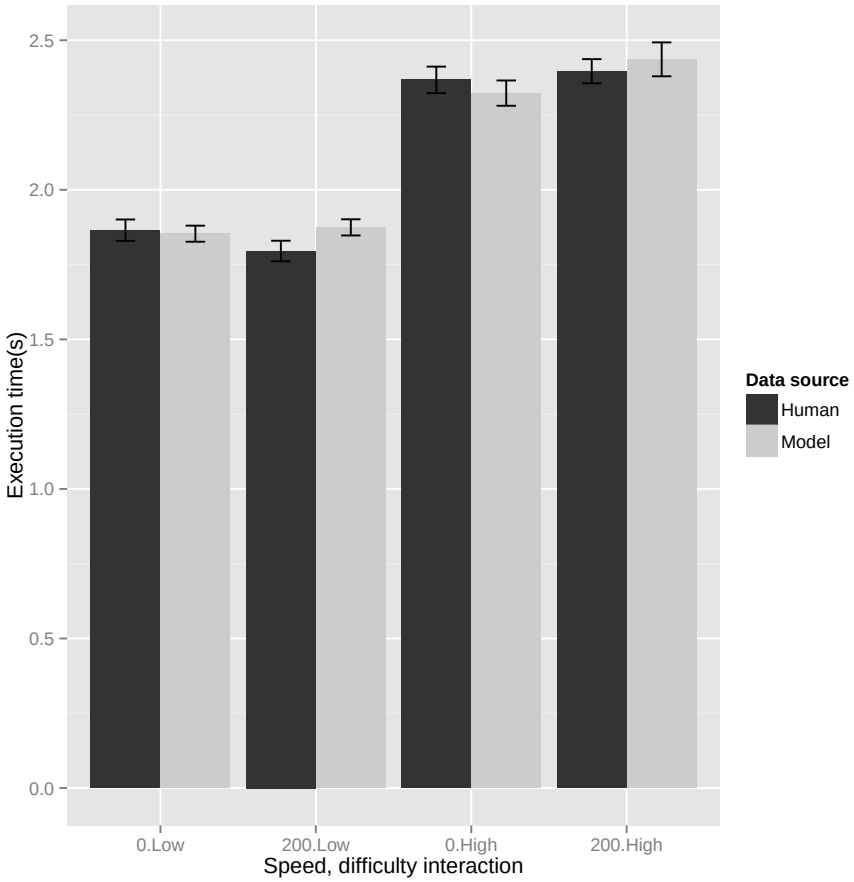


Figure 3.5: Execution times by difficulty and speed in single-task targeting condition

the modeled error rate is ≈ 0.066. Bar plots of the error rate across the single task targeting conditions is shown in figure 3.7, and across dual task targeting conditions in figure 3.8. The model notably overestimates the error rate in the high difficulty, high speed conditions in the dual task.

Concurrency The bar plot in figure 3.9 compares the means of the currency values for each condition. The modeled concurrency values are significantly different from empirical values only in the high-diffculty, high-speed conditions ($\alpha = 0.05$).

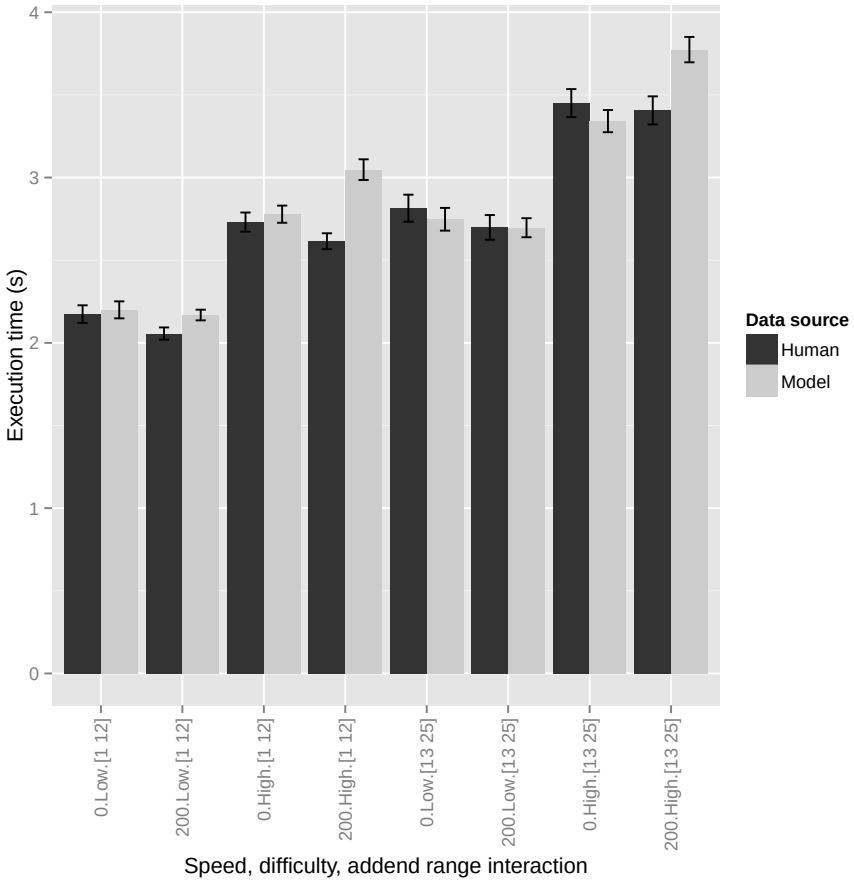


Figure 3.6: Execution times by addition and targeting difficulty in dual-task performance

3.6.3 IMPRINT

The single-task execution times for the IMPRINT-simulated addition and targeting tasks in each condition are shown in tables 3.3 and 3.4. The values are nearly identical to the QN-ACTR simulation values due to the fact that the IMPRINT models were designed to duplicate the QN-ACTR versions. As a result, the values are significantly different from the empirical values in exactly the same cases as the QN-ACTR values: the high-speed, low-difficulty targeting condition.

Table 3.3: IMPRINT simulated addition task completion times

Addend range	Completion time
[1 12]	1.26 ± 0.02s
[13 24]	1.99 ± 0.05s

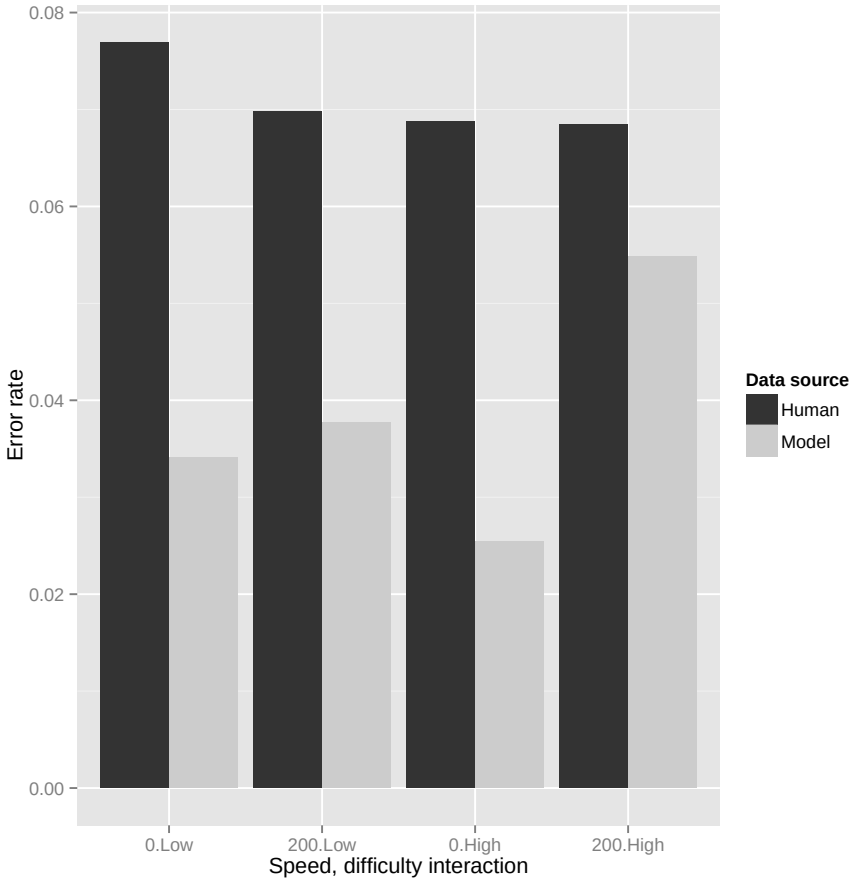


Figure 3.7: Targeting error rate in single task conditions

We used IMPRINT to simulate task performance in each dual-task condition with and without the extension described in chapter II. Without the extension, there was no limit on task concurrency and the dual-task completion times were equal to the greater of the completion times of the two primary subtasks. The simulated completion times when run with the extension are shown in figure 3.10. The empirical and QN-ACTR simulated values are included for comparison. The dual-task completion time predicted by IMPRINT differs significantly from the empirical data in three of the eight conditions, and differs from the QN-ACTR predictions in two.

The concurrency metric calculated from the IMPRINT simulations is shown along with the QN-ACTR prediction and the empirical value for each dual-task condition

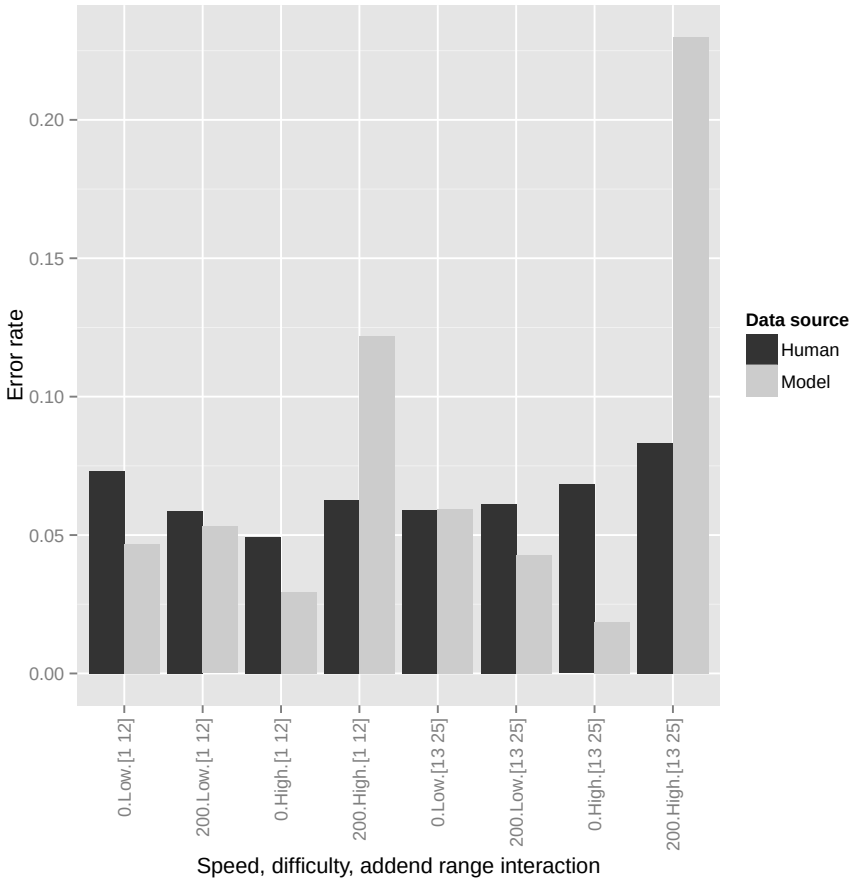


Figure 3.8: Targeting error rate in dual task conditions

in figure 3.11. The IMPRINT value differs significant from the empirical and QN-ACTR values in two conditions each.

3.7 Discussion

3.7.1 Effect of speed

In the cases when the speed variable was significant, increasing the speed of the targets had the unintuitive effect of decreasing performance time or increasing concurrency. We had expected a speed increase to increase the complexity of the task and increase execution times and the QN-ACTR models predicted an increase in execution time in some conditions.

This effect is likely caused by a reduced mean distance to the targets in the high-

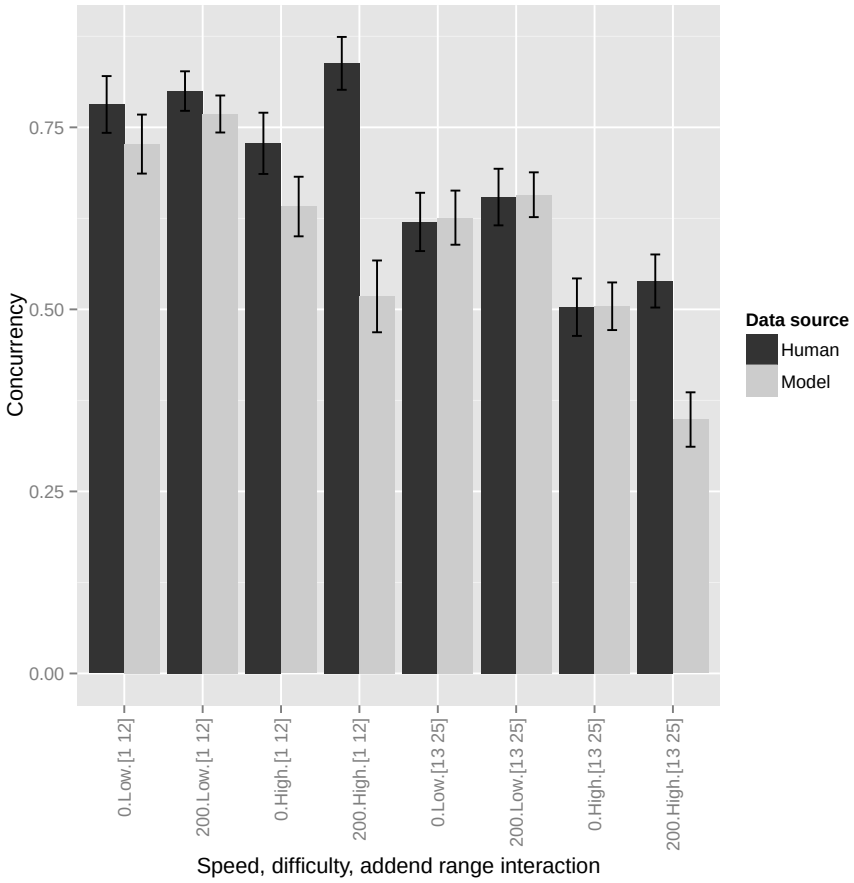


Figure 3.9: Concurrency by speed, difficulty, and addend range

speed case due to the experiment design. In the high-speed condition, the targets move at 200 pixels per second for 6 seconds, so they will each move 1200 pixels. This traversal is a large portion of the monitor, which has a resolution of 1920x1200 pixels. As a result, the target trajectories tend to pass near the center of the screen, reducing the distance between them and therefore the time to move from one to another. The participants also tended to position the cursor in the center of the screen before the start of each iteration, so the targets would be moving toward the cursor itself, compounding the effect.

If the advantage gained by human participants from the centripetal target motion causes the reduction in targeting execution time, then the model is failing to capital-

Table 3.4: IMPRINT simulated target task completion times

Speed	Difficulty	Completion time
0px/s	Low	1.85 ± 0.01s
200px/s	Low	1.89 ± 0.01s
0px/s	High	2.32 ± 0.02s
200px/s	High	2.43 ± 0.03s

ize on it. This could be because the model randomly selects which target to process first instead of selecting based on a heuristic, while human participants tended to select the target nearest to the cursor first.

The high-speed models also require two extra productions between identifying a target and moving the cursor to calculate a final cursor location. These productions take 50 ms for a total increase of 200 ms to move to both targets in the low-difficulty case. This difference accounts for the combination of the 40 ms increase in predicted execution time and the 122 ms decrease in empirical execution time. This can explain the model inaccuracy if the human control system begins moving the cursor toward a target immediately and accounts for target motion during the motion.

3.7.2 Execution time

Histograms comparing modeled and empirical execution times in the single-task conditions are shown in figures 3.12 and 3.13. The figures show that the effects of targeting and addition difficulty are correctly predicted by the model and that predicted distributions have similar means and lower variance than the empirical distributions. The lower model variance is expected as the model simulates a single person performing the tasks and does not reflect individual differences.

Another notable difference in the distributions can be seen in the targeting task conditions with high difficulty. We can see from figure 3.12 that the modeled execution time distributions are bimodal while the distributions from the empirical data are not. The left peak in the model histograms correspond to trials in which both

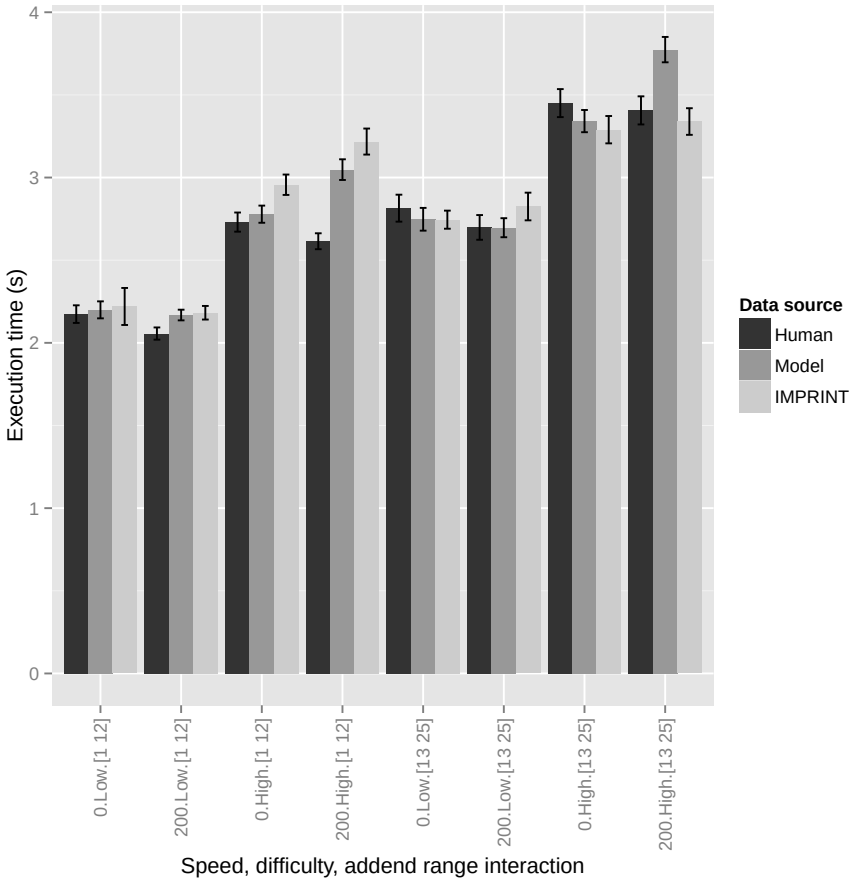


Figure 3.10: IMPRINT dual task completion times by speed, difficulty, and addend range

enemy targets were identified before the friend target. In these cases the task is complete after checking only two targets as opposed to three. There is a significant difference in completion time between these two cases in the empirical data as well but the higher variance prevents bimodality. We also see an an exaggerated effect of misses. The model distributions have a third peak corresponding to trials when a target is missed. The effect is weaker in the empirical data both because of the higher variance and because the increase in completion time due to a missed target is significantly higher in the model.

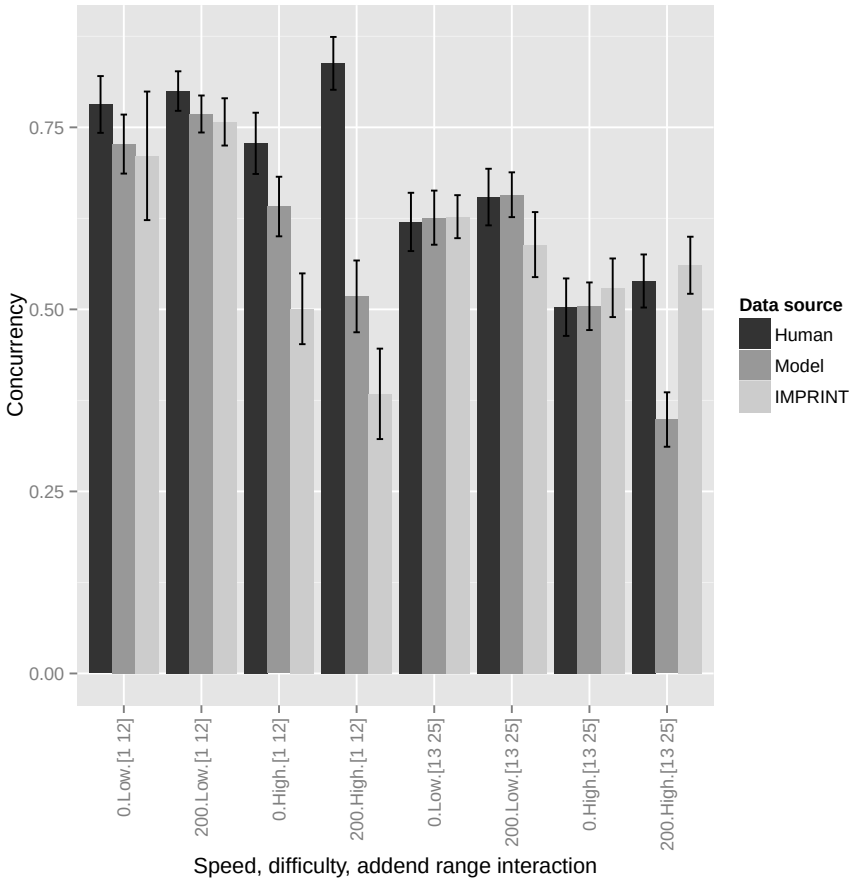


Figure 3.11: IMPRINT concurrency by speed, difficulty, and addend range

3.7.3 Concurrency

Metric properties The concurrency metric was based on two basic assumptions about dual task execution. Consider two tasks, A and B, that have execution times T_A and T_B respectively. The first assumption is that no concurrency occurs if the time required to perform both is $T_A + T_B$, as this is the time required to perform the two tasks in serial. Second, we assume perfect concurrency occurs if the time required to perform both is $\max(T_A, T_B)$. In this case, the shorter of the two tasks is completed within time required to perform the longer of the two tasks. Our concurrency metric linearly interpolates these two points, mapping execution time $\max(T_A, T_B)$ to concurrency $C = 1$ and execution time $T_A + T_B$ to concurrency

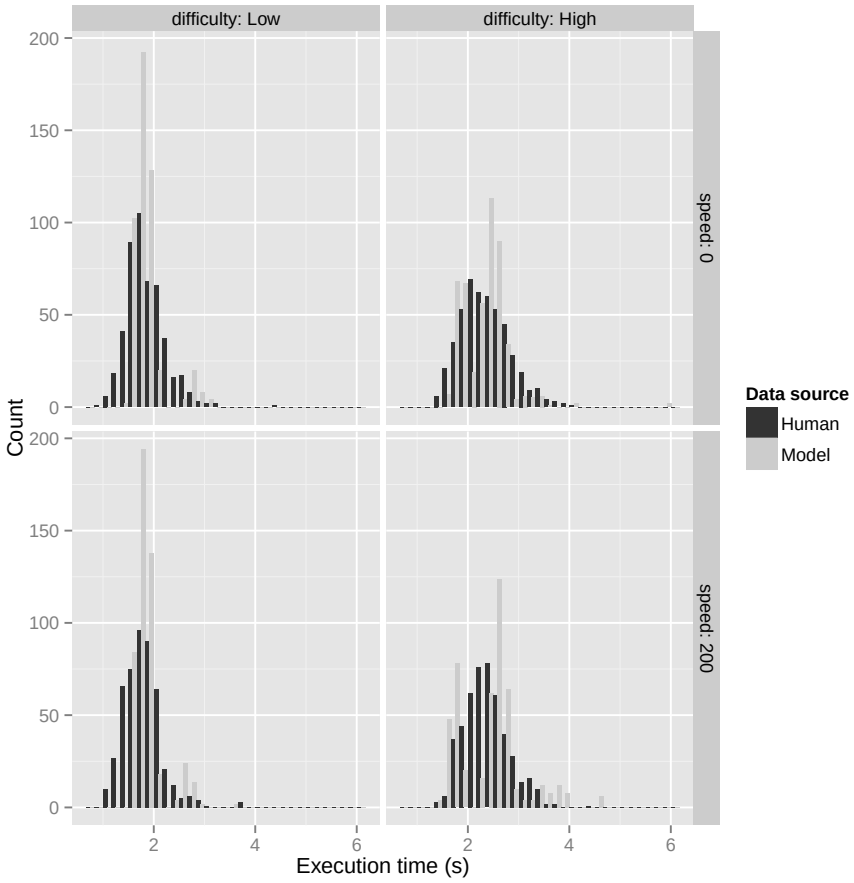


Figure 3.12: Targeting task execution time distributions

$C = 0$. When computing the concurrency metric from the data, we use the mean execution time from single-task trials of the addition and targeting tasks as T_A and T_B.

Model accuracy The predicted concurrency values are significantly different from empirical values in 2 of the 8 conditions: the high-speed, high-difficulty targeting conditions. These two underestimates are caused by the model significantly overestimating the number of target misses. In this targeting condition, the model moves the cursor to a target and performs a memory retrieval to decide whether to click a target based on the color. In the single task condition, the model is able to make the

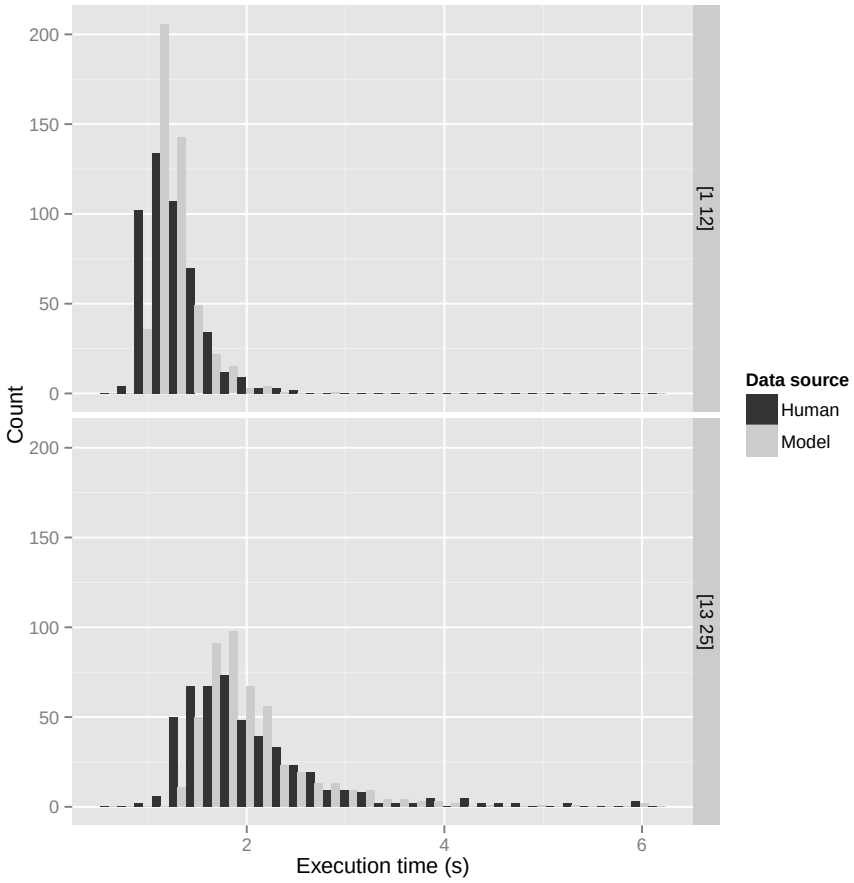


Figure 3.13: Addition task execution time distributions

decision and execute a click before the target moves away from the cursor. However, when the addition task is performed concurrently, the targeting task processing is often delayed enough to allow the target to move away before the click is executed.

There are two underlying modeling failures that explain this increase in misses. First, the models use ACT-R's motor module to simulate cursor movement. Cursor movements in ACT-R begin with a request and can not be modified or interrupted until they are complete. This prevents the model from tracking a target with the cursor as it moves as human participants do. The model may be improved by reducing the motor initiation and preparation times and introducing additional productions to simulate target tracking.

The second related modeling failure is that the model predicts longer completion times for the addition subtask in dual-task conditions than empirical data show. Figure 3.14 shows the addition subtask completion times across all conditions from the modeled and empirical data. We can see that the model systematically overestimates the completion time of the addition subtask but not the overall task completion time. This suggests that while the overall completion time and concurrency estimates may be accurate, the policy of alternately selecting productions from the two goals does not represent actual behavior. We could attempt to correct this in the model itself by violating the polite component of the greedy-polite policy of the threaded cognition theory. The addition task model could, for example, retain the use of the visual and visual-location across several productions to prevent interruption by the targeting task.

3.7.4 Behavioral modeling

In addition to overall execution time and concurrency, the dual-task model generally matches the behavioral aspects of human performance. Figure 3.15 which shows the subtask and overall completion times of the empirical and modeled data in one condition. We can see that the model correctly predicts that the addition subtask is completed before the targeting subtask. This behavior and prediction holds for all other conditions.

Although the order of subtask completion is predicted correctly, the completion time for the addition subtask are often overestimated in the model, suggesting that it fails to identify a resource conflict between the tasks. Empirical studies have shown that the phonological and visual-spatial memory play a role in mental arithmetic problems with medium to large numbers that are not solved by direct retrieval (DeStefano and LeFevre, 2004; Trbovich and LeFevre, 2003; Heathcote, 1994). We

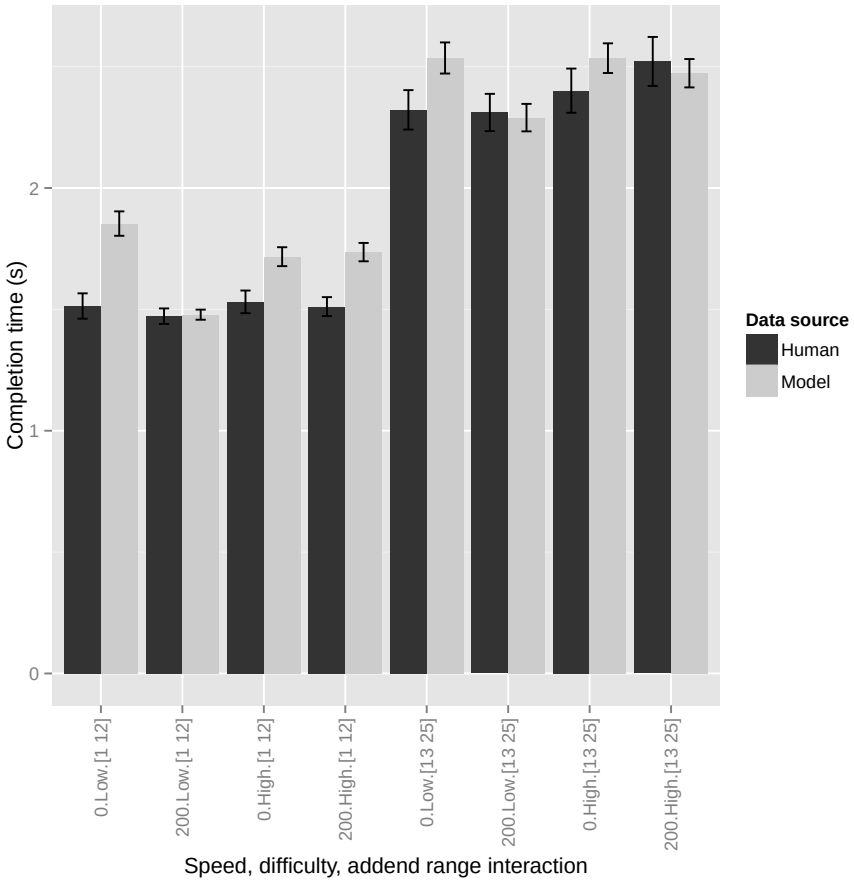


Figure 3.14: Addition subtask execution times in dual-task conditions

found anecdotal evidence for a resource conflict due to this visual-spatial memory requirement during empirical trial observation. In high-difficulty conditions, participants would often move the cursor over a target but would not click to dismiss the target until they finished calculating the sum of the addition problem. These visual-spatial memory requirements are neglected by our addition model, which uses only ACT-R's declarative memory module once the two numbers have been visually attended once. Figure 3.16 plots the addition subtask completion time against the time until the first target is hit, by targeting and addition task difficulty. In trials with low targeting difficulty, the model-predicted target hit times are a near constant 1s while the empirical data show that target hit times increase with the addition task

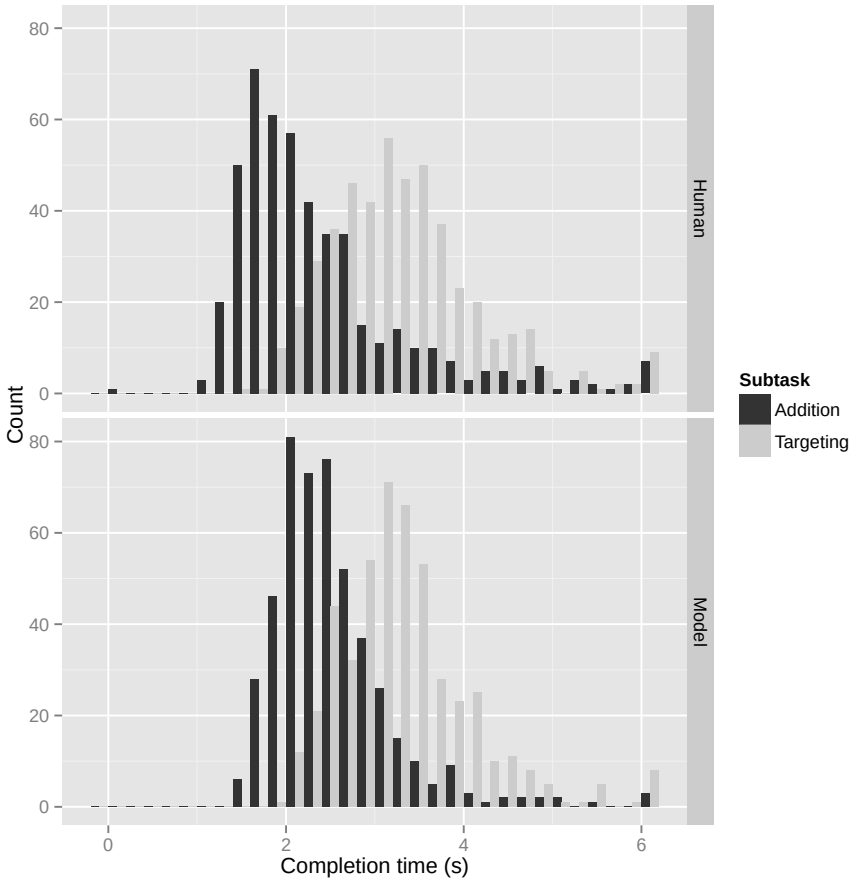


Figure 3.15: Subtask completion order in a dual-task condition

time. The model inaccuracy is alleviated in conditions with high targeting difficulty because the model requires both a visual perception and a retrieval to determine whether the target should be dismissed. The relationship between addition completion times and target hit times and improved model accuracy in high-difficulty conditions provides more evidence for the claim that visual-spatial memory is used during mental arithmetic.

3.7.5 Application to IMPRINT

Using the Soar-based task management extension described in chapter II we were able to accurately simulate task concurrency in IMPRINT. Figures 3.10 and 3.11 show that IMPRINT simulations with the extension matched or improved the com-

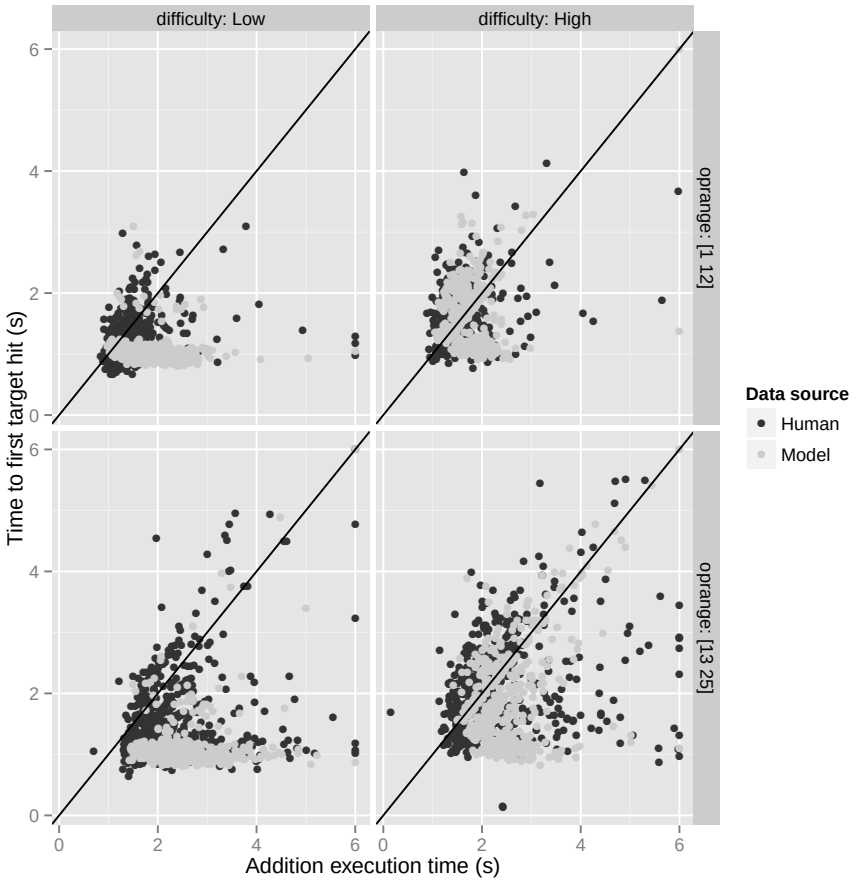


Figure 3.16: First target hit time vs. addition task completion time in dual-task conditions

pletion and concurrency predictions in most experimental conditions. In one condition (low-speed, high-difficulty, low-addend range) the completion time and concurrency predictions made by IMPRINT were significantly different from the empirical values while the QN-ACTR prediction was not. Similarly, there was one condition (high-speed, high-difficulty, high-addend range) in which the QN-ACTR values showed a significant difference from the empirical values while the IMPRINT values did not.

Figure 3.17 shows the dual-task completion times predicted by IMPRINT with the Soar-based task management extension and with the built-in Advanced Workload System using strategy A (section 1.4.2). The empirical and QN-ACTR pre-

dicted values are also included for comparison. The IMPRINT predictions using AWS consistently overestimated the dual-task completion times by 10 to 50%. The concurrency values, shown in figure 3.18 are greatly underestimated in all conditions, including three in which the predicted concurrency is negative.

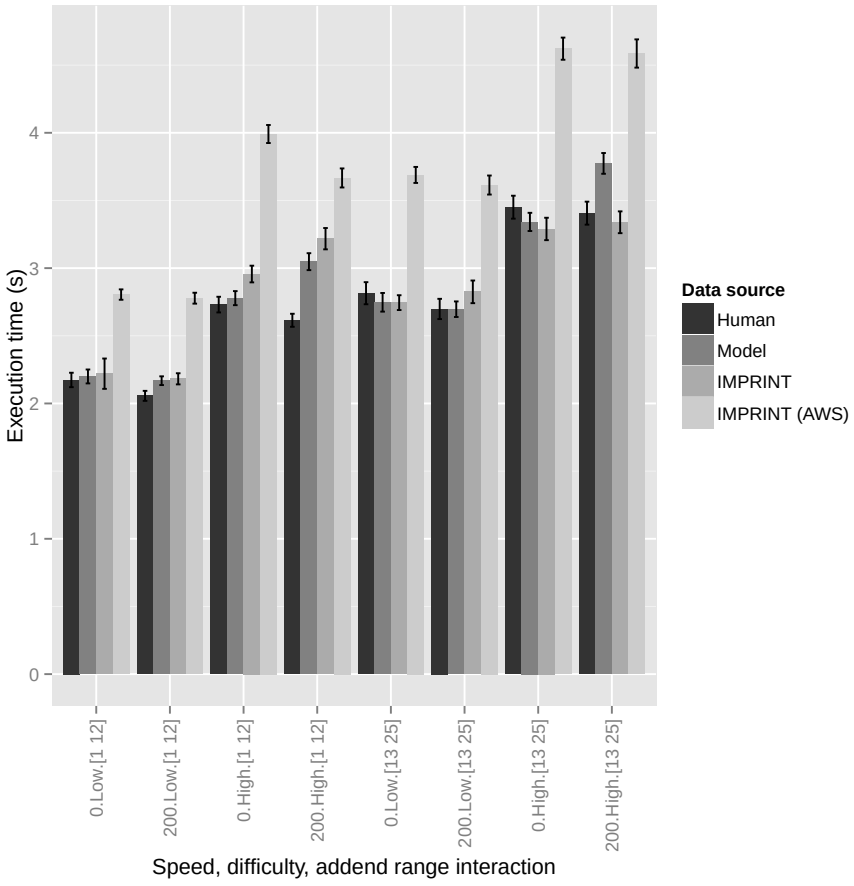


Figure 3.17: Comparison of IMPRINT predicted dual task completion times using AWS and the Soar-based extension in dual-task conditions

3.8 Conclusion

We developed QN-ACTR and IMPRINT models of multi-task performance consisting of an addition and targeting task. We collected empirical data of human performance in each single and dual-task condition from 20 participants in order to validate the completion time, accuracy, and concurrency predictions made by

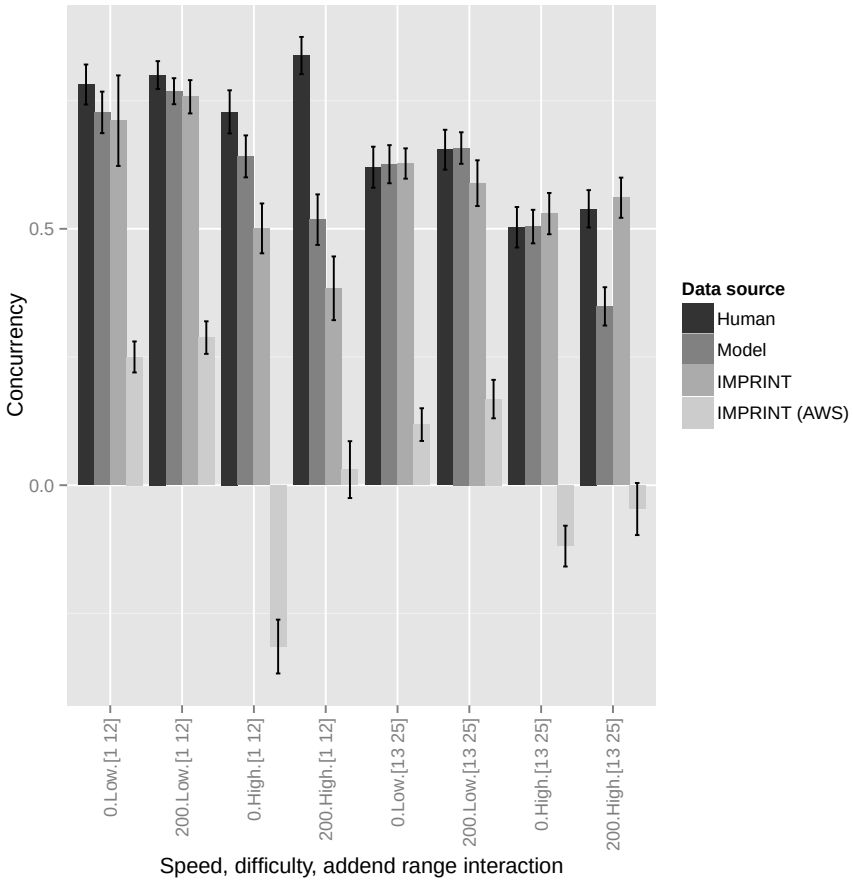


Figure 3.18: Comparison of IMPRINT predicted concurrency values using AWS and the Soar-based extension in dual-task conditions

QN-ACTR. The IMPRINT models were simulated using both the Soar-based task management extension described in chapter II and with the built-in AWS feature. We found that the model with the Soar-based extension produced significantly more accurate completion time and concurrency predictions than with AWS.

The QN-ACTR models predict task performance in terms of execution time and concurrency well, but fail to capture two aspects of dual-task performance. First, while the predicted completion times are accurate, execution times for the addition task are overestimated while the target task times are underestimated. This suggests that a strict least-recently processed goal selection algorithm for threaded cognition may not reflect actual task performance strategies. We propose that a more sophisti-

cated algorithm could be used, which favors the execution of a single task for longer periods of time. A system such as the Soar-based task management extension to IMPRINT may be used for this purpose. The system could be applied at the goal level, selecting whether a production should be allowed to fire based on current resource usage, or to the task network implementation of the ACT-R algorithm in QN-ACTR itself, as it is in IMPRINT. However, network components in IMPRINT represent task components themselves while QN-ACTR network components represent the mechanisms simulating task performance. Applying the Soar-based extension to the current QN-ACTR implementation would correspond to interfering with individual actions and conditions in ACT-R model productions leading to complex interactions in the simulation. Applying the system at the goal-threading level is a more promising option.

Second, the model fails to capture a resource conflict in the latter stages of task performance when empirical data suggests the use of visual-spatial memory interferes with the visual requirements of the target task. This conflict is not predicted in ACT-R without explicitly introducing visual resource usage in the addition model. This is more easily remedied in the IMPRINT simulation with the Soar-based extension where a visual workload value can be introduced in the appropriate network component, increasing the likelihood of a task delay.

CHAPTER IV

The effect of effort on dual-task performance and concurrency

The study of the effects of incentive and effort covers many aspects of task performance: manual tasks (Toppen, 1965; Kuhl and Koch, 1984), decision making (Bettman et al., 1990; Hogarth et al., 1991), problem solving (Wieth and Burns, 2006), pain tolerance (Baker and Kirsch, 1991), and job performance (Dillard and Fisher, 1990). Bonner and Sprinkle (2002) and Bonner et al. (2000) provide an excellent overview of research on financial incentives, effort, and task performance, and review the effectiveness of incentives based on reward scheme and task type in laboratory settings. Few studies have been conducted on the effect of effort on dual-task performance (Erez et al., 1990; Schmidt et al., 1984) and these examined performance only in terms of speed or accuracy. The focus of the present study is on the source of improvement and whether time-sharing between dual-tasks improves.

We conducted an experiment using the experimental tasks from chapter III with modifications to require dual-task performance for the duration of the task and without the independent variables relating to targeting task difficulty and speed. A monetary reward was introduced as a within-participants variable: each single and dual-task condition was performed by a participant with and without the incentive. Pursuant to the results of Dambacher et al. (2011) and Bonner et al. (2000), the

incentive was linear with the number of tasks completed within each 90 second trial. The reward primarily incentivized speedy performance: no deductions were made for incorrect responses to the addition task and the deduction for hitting a green target was used only to prevent participants from dismissing all targets regardless of color, ensuring the appropriate visual and cognitive task requirements.

4.1 Tasks

The experimental tasks were similar to those in the first experiment with a few changes. In the first experiment the tasks were presented to the participants at the beginning of each 6 second interval, allowing us to easily define the start and end time of the dual-task. However, this produced a period of time in which a participant is performing only one task. After one of the two tasks was complete, the participant completed the remainder of the remaining task without the interaction of the first task. It also limited the fraction of the participation time that participants spent actively performing tasks and the total number of tasks performed.

To ensure that participants were exhibiting truly dual-task performance during the dual-task trials we modified the task design. In this experiment, a new task was presented as soon as the participant completed the task presented. In the addition task, when the participant announced the correct answer, the display would be refreshed with a new problem. In the targeting task, a new set of three targets would appear as soon as a participant dismissed the two red targets. This was true for the dual-task conditions so that there would be two active tasks at all times. To accommodate for the new task design, we modified the concurrency definition as described in section 4.4.

4.1.1 Targeting task

The targeting task experiment interface is identical to the first experiment. However, unlike the first experiment, there are no independent variables governing the targeting task. The task only takes the high-difficulty, low-speed targeting condition from the first experiment: each target is stationary and is displayed in black until the cursor is placed over it, whereupon the red or green color is shown.

4.1.2 Addition task

The addition task is identical to the first experiment except that a new task is shown immediately when a task is complete. The difficulty of the task is an independent variable with two settings: addends between 1 and 12, and between 13 and 24.

4.2 QN-ACTR models

4.2.1 Addition model

In order to model the new task design, we made minor changes to the addition model. The changes to the model are described here. For a full description of the model, see section 3.3.1.

Instead of moving to a final state after announcing the sum in the `respond-ones` and `respond-ones-no-tens` productions, the model sets the goal state to `reset` and a production with the same name fires. The production resets the goal chunk slots to `nil` and sets the goal state to `find-first` to begin the next addition problem.

In both the experiment and model, there is a brief delay between the response and the display of the new addition problem. Without any changes to the `find-first` production, the result of the visual-location request would be the plus sign, which is the only remaining unattended text item while the completed problem remains on

screen. We added a constraint to the visual-location request: `< screen-x 960` which limits the search to the left side of the screen, preventing the plus sign from being found. Before the new problem is displayed, this visual-location request will fail. We therefore added a new production, `fail-find`, which detects the visual-location buffer error state and performs a new search for unattended text on the left side of the screen. This production will fire until the new problem is displayed and normal processing resumes.

To model the increase in effort induced by the incentive payment, we modified the activation base level values in simulations of incentivized trials. Recent neuroscience studies provide support for long-held theories (Kahneman, 1973; Pribram and McGuinness, 1975) and found direct evidence of increased activation levels leading to reduced reaction time (Jansma et al., 2007; Engström et al., 2013). The base level values from the first experiment were increased by a constant fraction and used for incentivized simulations. The fraction was fit using empirical addition task execution times in the low-difficulty, incentivized condition.

4.2.2 Targeting model

Experiment 2 uses only the high-difficulty, low-speed targeting condition from experiment 1. We therefore used the corresponding model from experiment 1 with minor changes described here. For a full description of the model see section 3.3.2.

The model in experiment 1 was already designed to repeatedly search for targets so no change was required to account for the immediate display of new targets upon task completion. However, as in the addition task, there is a brief period of time after the `click-mouse` production when the model continues with the next productions but the new task has not yet been displayed. In this case, the visual-location request performed by `find-black-target-friend` will fail and result in an error state. We

added a production named `fail-find-not-friend` to handle this case. The production detects the error and performs a new search for a black target. This production continue firing until new targets are displayed, causing the visual-location request to succeed.

The `fail-find-not-friend` production only handles the case in which the friend target was encountered before dismissing the second red target. If the friend target was never found, the `find-black-target` production will fire and successfully find the friend target which will momentarily be removed from the display. In this case, processing continues until the `check-target` production performs another visual-location request for the same target. By the time this production fires, the target has been removed from the screen and new targets shown and the visual-location request fails. We added a `distinguish-target-error` production which detects this search failure and changes the state back to `find-black-target` to begin processing the new targets.

4.3 Method

20 participants were recruited from a population of undergraduate and graduate students at the University of Michigan. Half of the participants were male and half female to control for a potentially confounding gender variable. Participants completed trials in which they attempted to complete one or more tasks quickly and accurately. Each trial lasted 90 seconds in which the participant completed as many tasks as possible. The participants were instructed to complete the tasks while minimizing execution time and errors. During incentivized trials, the participant's reward was computed in real-time and displayed in the corner of the screen.

All independent variables are within-participant factors and the order of condi-

Table 4.1: Experiment 2 Design

Independent variables		
Incentive	No	Yes
Addition operands	[1,12]	[13,24]

tions was randomized. In addition to each block condition of the dual-task, participants completed trials of the addition task at each difficulty and incentive level without the targeting task, and the targeting task at each incentive level without the addition task. A table of independent variables is shown in Table 4.1.

4.3.1 Scoring

During incentivized trials, participants received a dollar amount based on task performance that was visible to them to reward fast and accurate task completion. The participants received \$0.05 for each enemy target dismissed and \$0.10 for a correct answer to the addition problem. \$0.05 was deducted for each friend target dismissed, and \$0.02 was deducted for each button click that did not hit any target. The participants were informed of the scoring policy before the experiment began and the task conditions before beginning each trial.

4.3.2 Procedure

Before beginning the trials, we described the addition and targeting tasks and the parameter levels to the participant. We also described the reward system in detail to ensure participants attempted to complete the tasks quickly and accurately. Participants first completed a set of practice trials that consisted of one trial of each single and dual-task condition with no incentive. Between each trial, participants were permitted to take time to rest in order to avoid the fatigue effects.

After verifying that the participant was comfortable with each task condition and was capable of performing the tasks, the participant began experimental trials. The participant performed two experimental trials of each dual and single-task condition.

After the two trials in each block condition, the participant responded to the NASA-TLX survey that was presented on-screen (Hart and Staveland, 1988). After all block conditions were completed, participants ranked the sources of each workload subscale according to the NASA-TLX procedure.

4.3.3 Apparatus

We implemented the experiment as a web browser application using HTML, CSS and the Dart language. The browser window occupied the entirety of a 24-inch widescreen monitor. The Nintendo Wii remote controller senses four infrared LEDs that were affixed to the corners of the monitor and were powered by a 5V USB connection. We developed Java software to compute the screen location where the remote was pointing and produce corresponding system events for mouse movements and clicks, which were received by the browser. Software for logging and analyzing data is written in Dart, Go, and R.

4.3.4 Data collection

After each block the participant responded to the NASA-TLX survey to measure the subjective workload of each task condition. The survey asks respondents to assess the workload of a task on six subscales: mental demand, physical demand, temporal demand, performance, effort, and frustration. During the experiment, mouse events including movements and clicks were recorded. Target hits and misses were recorded to examine the error rate and execution time of the targeting task. During trials with an addition task, an audio recording was taken of the participant's responses. 3 tones are played prior to the start of the trial to synchronize the audio data.

4.4 Results

4.4.1 Single tasks

The incentive and addition difficulty variables both had significant effects on single-task addition completion times ($\alpha = 0.05$). For low-difficulty trials, incentivization reduced the mean completion time from 1.19 to 1.11, and from 2.08 to 1.96 in high-difficulty trials. The data show no significant effect of incentive on single-task targeting completion times.

4.4.2 Dual tasks

In both unincentivized trials, the addition difficulty variable had a significant effect on both addition and target task completion times ($\alpha = 0.05$). The effect of incentivization, however depended on the addition difficulty variable. In low-difficulty trials, the addition completion times showed a significant decrease under incentivization while target task completion times showed a significant increase. In high-difficulty trials, neither addition nor target task completions times showed a significant change under incentivization.

The score awarded to participants was also calculated for unincentivized conditions. In both unincentivized and incentivized trials, the addition difficulty variable had a significant effect on the score ($\alpha = 0.05$). The score significantly increased under incentivization in low-difficulty conditions, but not high-difficulty conditions.

The concurrency metric used in the first experiment assumes that two subtasks are presented simultaneously and defines the dual-task completion time as the time that both subtasks are complete. The tasks in the second experiment, however, begin and end independently of each other, so the original concurrency metric cannot be directly applied. Instead we define a generalized concurrency metric based on the subtask

completion times in single and dual-task conditions. Let A be a single-task condition of the addition task, T be the single-task condition of the targeting task, and AT be the dual-task condition in which A and T are performed simultaneously. Then let $ET_A(X)$ and $ET_T(X)$ be the mean execution time of the addition and targeting task in condition X respectively. For example, $ET_A(A)$ is the mean execution time of the addition task in a single-task condition, while $ET_A(AT)$ is the mean addition task execution time in the corresponding dual-task condition. We define concurrency as

$$1 - \frac{ET_A(AT) + ET_T(AT) - (ET_A(A) + ET_T(T))}{ET_A(A) + ET_T(T)}$$

The right side measures the fractional increase in the sum of the mean completion times of the two tasks when performed simultaneously. If there is no increase in the completion times in the dual-task scenario, we have $ET_A(AT) = ET_A(A)$ and $ET_T(AT) = ET_T(T)$. The fraction reduces to 0 and the concurrency metric is 1. If the two tasks were performed alternately and in serial, we have $ET_A(AT) = ET_T(AT) = ET_A(A) + ET_T(T)$. The fraction reduces to 1 and the concurrency metric is 0.

As in experiment 1, the addition difficulty variable had a significant effect on concurrency regardless of incentive ($\alpha = 0.05$). However, incentive did not have a significant on concurrency in either the high or low difficulty dual-task conditions.

4.4.3 Model validity

We found no significant differences between empirical and model-predicted execution times in the single-task conditions of the addition and targeting tasks. Furthermore the model showed a significant decrease in completion times of the addition tasks in incentivized trials in both the low and high-difficulty conditions, and no

significant difference in targeting times based on incentive. Bar plots comparing the empirical and modeled times in the addition and targeting tasks are shown in figures 4.1 and 4.2 respectively.

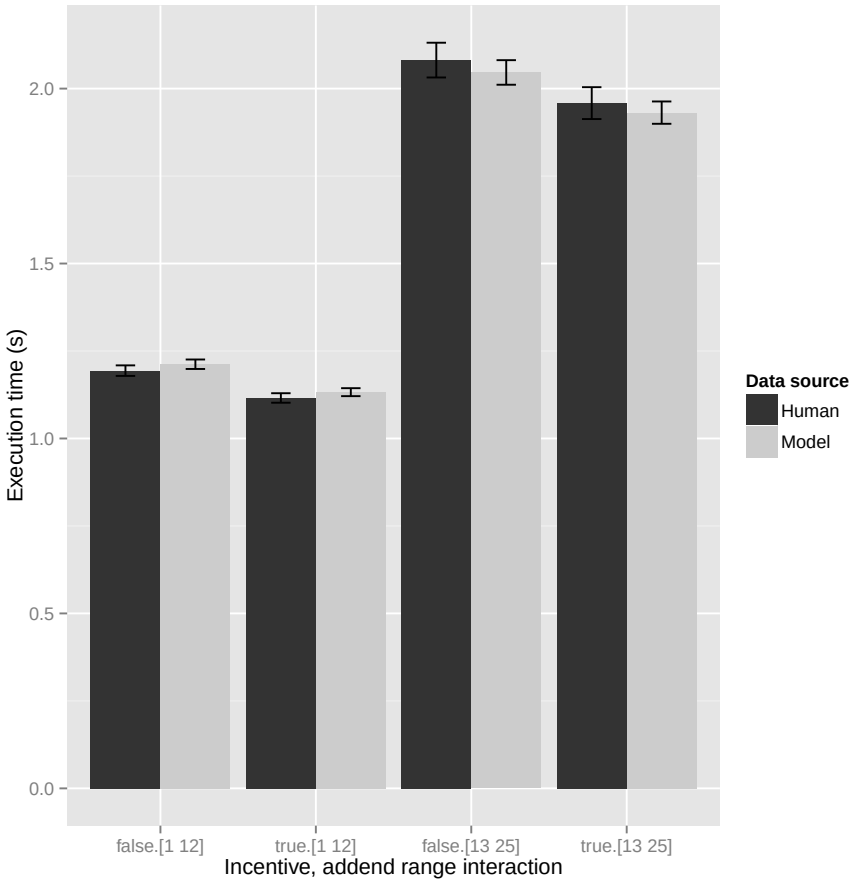


Figure 4.1: Execution times by difficulty and incentive in single-task addition problems

A bar plot comparing the empirical and model-predicted addition task completion times in each dual-task condition is shown in figure 4.3. As in the empirical data, the model data shows a significant decrease in addition task completion times based on incentive only in the low-difficulty conditions. We also find no significant difference between empirical and modeled completion times for the addition task in the high-difficulty condition. However, the model significantly overestimates the completion times in the low-difficulty conditions.

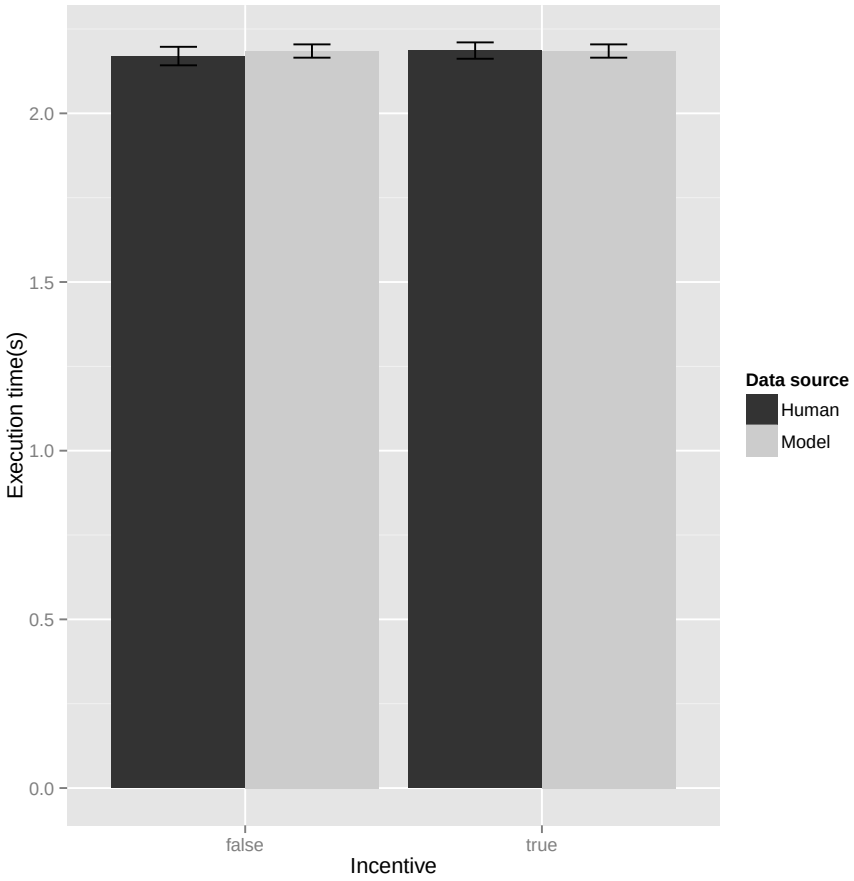


Figure 4.2: Execution times by incentive in the single-task targeting problem

The model also significantly overestimates the targeting task completion times in the low-difficulty conditions as shown in figure 4.4. Furthermore, the model-predicted targeting task completion times in the low-difficulty conditions show no significant difference between incentivization levels. In the high-difficulty conditions the model significantly underestimates the task completion times and shows not significant difference based on the incentive variable.

Due to the task design, there is no dual-task completion time metric to examine directly. To compare the performance in dual-task conditions, we computed the incentivization score in units of dollars without deductions for targeting-task errors. A bar plot showing the mean score for a single trial in each condition is shown in

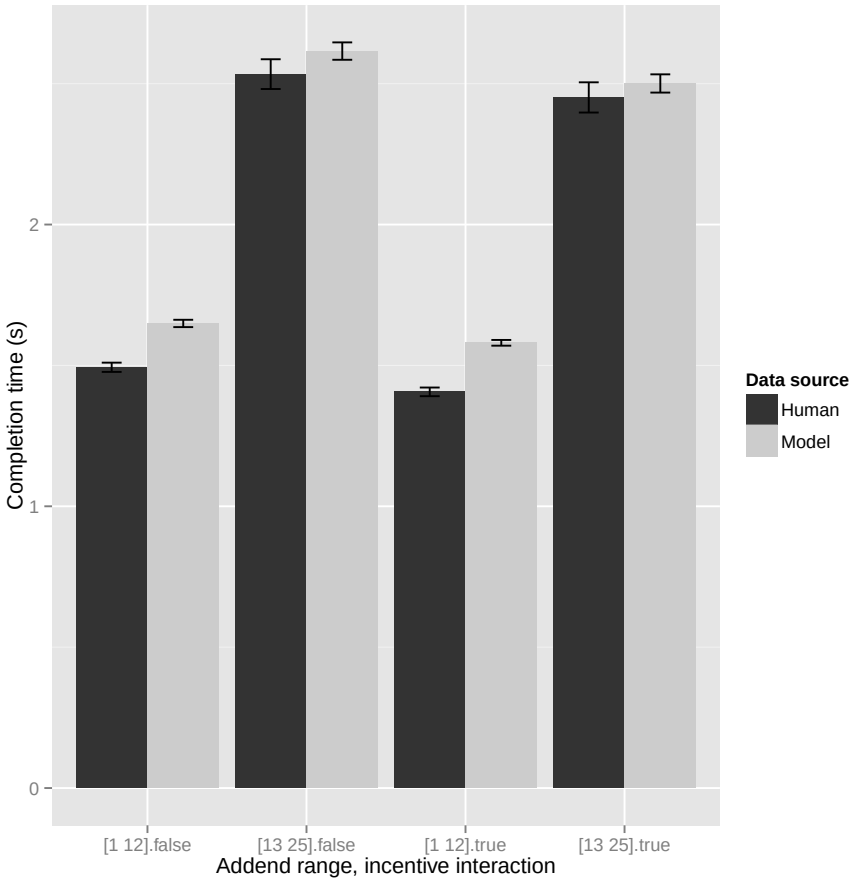


Figure 4.3: Addition task execution times by difficulty and incentive in dual-task conditions

figure 4.5. Due to overestimating both the addition and targeting-task completion times in the low-difficulty conditions, the model significantly underestimates the score awarded in these conditions. In the high-difficulty conditions, the data show no significant difference between empirical and modeled scores. There is no significant difference in model-predicted scores based on incentive.

The error of overestimating the addition and targeting is also reflected in a significant modeled underestimate of concurrency in the low-difficulty conditions. The concurrency bar plot in figure 4.6 shows that the model underestimates concurrency in the low-difficulty, but shows no significant difference in concurrency in the high-difficulty conditions.

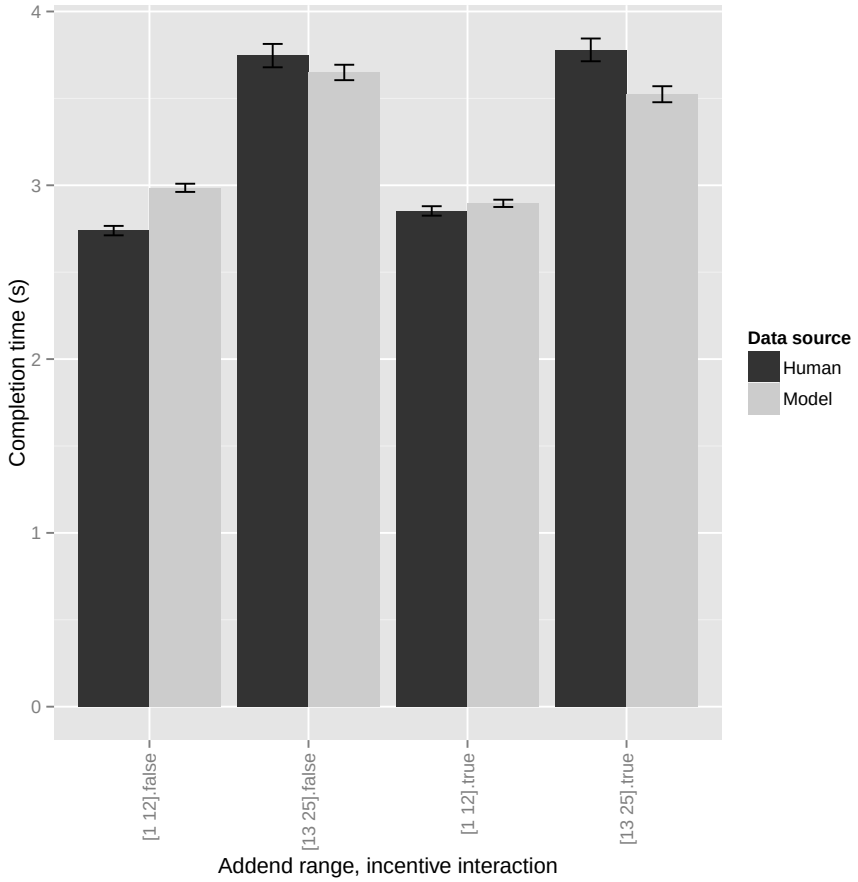


Figure 4.4: Targeting task execution times by difficulty and incentive in dual-task conditions

4.5 Discussion

4.5.1 Single task incentivization

In regard to task completion times, our findings are consistent with previous studies of incentivization and performance (Bonner and Sprinkle, 2002). As shown in figure 4.1, addition completion times in single-task conditions were significantly improved under incentivization. We noted, anecdotally, indicators of increased arousal in speech including pitch, volume and speed during incentivized trials (Burgoon et al., 1989, 1992). Targeting task completion times, however, were not effected by incentivization in the single-task conditions as shown in figure 4.2. The QN-ACTR models of performance under incentivization correctly predicted both the effect on

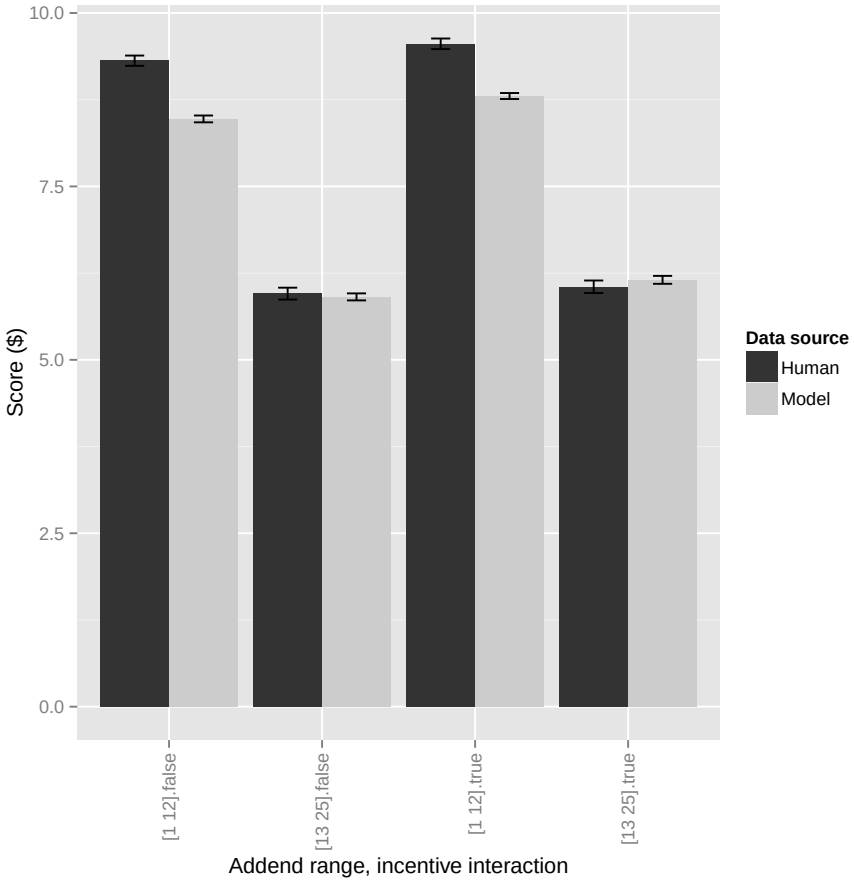


Figure 4.5: Score by difficulty and incentive in dual-task conditions

the addition tasks and the lack of effect on the targeting task.

4.5.2 Dual task performance

Incentivization was also found to affect performance in dual-task trials. The score metric, in units of dollars awarded to the participant, showed a significant increase with incentivization. Addition completion times showed a significant decrease under incentivization in both low and high difficult trials as shown in figure 4.3. Targeting task completion times, as shown in figure 4.4, showed no difference under incentivization in high-difficulty conditions. In low-difficulty conditions, however, targeting task completion times showed a significant increase in incentivized trials.

The effects of incentivization on task performance were complex: in low-difficulty

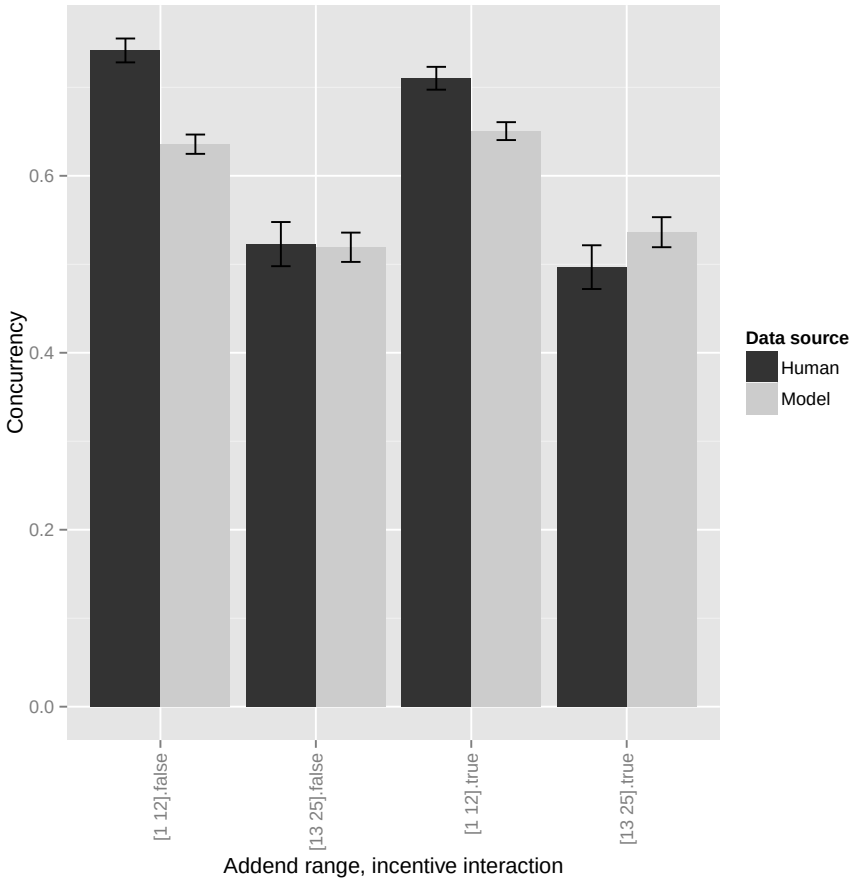


Figure 4.6: Concurrency by difficulty and incentive in dual-task conditions

trials, addition times decreased, targeting times increased, and the score increased. A pure effort effect would suggest that both addition and target times would decrease or at least remain constant under incentivization. That target times increased shows that participants not only increased effort but also adjusted their strategy to attain higher reward. The QN-ACTR model fails to capture the effect of the strategy allocation of attention to the addition task, predicting a significant decrease in targeting task completion times under incentivization in both low and high difficulty conditions. Figure 4.5 shows the model-predicted effects of difficulty and incentive on score in dual-task conditions.

The increase in scores in dual-task conditions can derive from two distinct sources

of improvement. Improvement on the performance of the individual tasks, and improvement on performing the tasks concurrently. Results from the single-task trials show that participants improve their performance on the addition task when given an incentive. To isolate the effect of true concurrency improvement, we calculate concurrency as a function of single-task execution times separately for incentivized and unincentivized trials. Despite evidence of increased effort and strategy adjustment, the data do not show an increase in concurrency in incentivized trials. Figure 4.6 shows the empirical and model-predicted effects of task difficulty and incentive on task concurrency. Incentive, in both empirical and modeled-data, has no significant effect on concurrency at either difficulty level.

One explanation for this lack of effect is that increased effort and arousal induced by the incentive does not reduce the conflict between cognitive resources required to perform each of the tasks. We did not make any modifications to the model to attempt to improve resource concurrency under incentivization. ACT-R itself provides only minimal mechanisms for achieving increased resource sharing. Only one of each type of buffer exists and each can contain only one chunk at a time. The motor and speech modules allow new requests to be made before the previous action is complete. Each action is separated into three stages: preparation, processor, and execution. A new request will be accepted if the latter two stages are busy but the preparation stage is free. Other modules including visual and declarative will only accept new requests when all three stages are not busy. QN-ACTR introduces concurrency accommodations notably by the inclusion of multiple goal buffers. Due to its server-network model, QN-ACTR could parallelize the usage of the single resource buffers, however we did not investigate this possibility.

It is also possible, however, that increased effort can increase resource concur-

rency but that our particular experimental tasks are unsuitable for observing such an effect. The two experiment tasks are visually intensive and require memory retrieval, although the targeting task execution time is dominated by the movement of the cursor between targets. While effort does not appear to improve concurrency in or between the visual and memory resources, it is possible that we would observe improvement in tasks that use, for example, auditory cues instead of visual cues. Further experiments with alternate resource requirements would allow us to either generalize the claim that effort does not improve concurrency or identify resources whose conflicts are sensitive to effort and arousal.

Finally, we can hypothesize that the arousal mechanisms that produce single-task performance improvement themselves prevent an increase in concurrency. The data show that addition task completion times in both single-task and dual-task conditions improve under incentivization, likely due to increase allocation of attention and reduced memory retrieval times. Tasks performed with higher effort and with increased attention would naturally be more resistant to concurrent performance with another task. We can see evidence of this in the low-difficulty dual-task conditions under incentivization when addition task completion times decrease but targeting task completion times increase.

4.5.3 Concurrency predictions

Although the model correctly predicts a significant increase in score under incentive in low-difficulty dual-task conditions, it significantly underestimates the magnitude of the score and concurrency in these cases. This modeling inaccuracy was not observed in the first experiment which featured a slightly different task design. In the first experiment, the two tasks started simultaneously every 6 seconds. In this experiment, however, a new task of a given type would begin as soon as the

previous was complete. That is, when an addition problem was answered, a new set of numbers was shown immediately, regardless of the participant's progress on the targeting task, and vice versa. The ACT-R models of the tasks had to be modified to restart when a task was finished. A production was added to the addition task model to reset the goal state and start again at the first production after the response was spoken. Because there is a delay between a speech request and its execution, the production to find the location of the first addend would fire before the task display was updated with the new values. The visual-location request checks that the object is unattended and therefore fails. A production was added to detect this search failure and perform a new search for the first addend. This production would fire cyclically until the new problem was shown. Until the new values are displayed and the request succeeds, the visual-location buffer is in an error state, preventing any production in the targeting model from firing if it requires the visual-location buffer. Several productions in the targeting model use the visual-location buffer as the targeting task is largely visual.

The targeting task model was modified in a similar way. There is a brief delay between the time the production that requests the last click and the time the new targets are displayed on screen. During the delay, processing in the targeting model proceeds and it attempts to locate a target to move to. The model continues searching until the display refreshes and the new targets are available. Productions in the addition model that use the visual-location buffer are prevented from firing during this time.

These conflicts do not affect single-task execution time but do increase execution time in dual-task simulations. The increase has a particularly strong effect on the low-difficulty addition task when the time increase is a larger fraction of the overall

execution time. This is a technical and not theoretical problem, and could potentially be resolved by modifying the task models. For example, a clear request could be made to the visual module which would clear the error state of the visual-location buffer. However, the request would also empty the visual buffer and reset the visual buffer state, fatally interrupting processing in the opposite task when it is using the visual buffer. This option is not feasible without implementing the greedy-polite policy over the visual and visual-location buffer as a single resource instead of separately.

The effect of the conflict is also weaker in the score and concurrency metrics in high-difficulty conditions because the addition task execution time overestimate is offset by an underestimate of the targeting task execution times. This is an artifact of the alternate-goal threading policy used in QN-ACTR as discussed in section 3.7.3.

4.6 Conclusion

We modified the experimental design from chapter III to increase dual-task requirements and study the effect of incentive and effort on multi-task performance. The most striking result from this study is the lack of evidence that increased effort improves concurrency in dual task performance even when execution time improves. This suggests that resource sharing and the executive processes responsible for subtask scheduling are insensitive to arousal. We also validated the QN-ACTR model of task performance and its ability to capture the effect of effort. Technical problems in the ACT-R models introduced some error in the model predictions in low-difficulty conditions.

CHAPTER V

Conclusion

In chapter II we described an architecture for governing task execution in models of dual-task performance and its integration with the IMPRINT modeling software. The IMPRINT extension fills a role similar to EPIC's executive processes: it is independent of the task model itself and enforces cognitive constraints on otherwise uninhibited concurrent task execution.

We implemented the architecture as a Soar agent in two components: one that enumerates the possible actions that can be taken at each decision point in a task performance simulation, and another that selects one from among those actions based on the simulation state. The selection procedure uses heuristics to limit task execution based on a workload-threshold. Using the extension, an IMPRINT simulation of UAV operator tasks from a study by Schulte and Donath (2011) produced several qualitatively distinct load-shedding strategies exhibited by experiment participants.

In chapter III we describe our use of QN-ACTR to model concurrent task execution on a mental arithmetic task and a targeting task. We are particularly interested in the prediction of a quantitative concurrency metric that describes the extent of time sharing between the two tasks. We validated the model by collecting experimental data and discussed various aspects of task performance and how well they

were predicted by the model. We found that the magnitude of the addends and the complexity of the targeting task but not the speed of the targets had significant effects on execution time and concurrency. The model predicted both well in most cases, but failed notably in two dual-task conditions when large overestimates in targeting errors propagated to high execution time and low concurrency predictions.

We also developed IMPRINT models of the experimental tasks and used both the Soar-based task management extension and the built-in workload management system to predict execution time and concurrency, and compared the results with empirical data. The Soar-based system was found to significantly improve the IMPRINT predictions.

Finally, in chapter IV, we collected experimental data concerning the effect of effort on task performance and concurrency. We presented evidence that decreases in dual-task execution times resulting from increased effort are not caused by improvement in the executive processes governing task scheduling or improvements in resource sharing, but by the same mechanisms that decrease the execution times of the single-tasks. We modified the QN-ACTR models from chapter III to model the effort effect and validated the model using the empirical data.

5.1 Summary of models and their roles

Our work makes direct use of, and expands upon, several cognitive architectures for the purpose of further integration of modeling methods and improvement of modeling capabilities (Liu, 2009).

QN-ACTR As an implementation of ACT-R in discrete-event simulation software, QN-ACTR combines the powerful and accurate model capabilities with the visualization and utilization-based workload features of the Queueing Networks modeling

approach. We used QN-ACTR to model single and dual-task performance of an addition and targeting task with variable difficulty parameters and an incentive variable, and validated the models with two experiments described in chapters III and IV.

IMPRINT The IMPRINT software is widely used to evaluate the difficulty of tasks and the usability of interfaces. We identified a limitation in the IMPRINT software, specifically, the modeling of multi-task concurrency under high workload and developed an extension to suppress task execution depending on the state of the simulation.

Soar To implement the IMPRINT extension described above, we developed a Soar agent which examined the simulation state and decided when task components should be delayed, interrupted, or ignored.

5.2 Scientific contributions and Future work

As scientific contributions of this dissertation research, two empirical findings stand out from the work described in chapters III and IV. First, as described in section 3.7.4, our data provide further evidence for the use of the visual-spatial resource during complex addition tasks (DeStefano and LeFevre, 2004; Trbovich and LeFevre, 2003; Heathcote, 1994). We used the declarative and goal module to model the later stages of the addition task in QN-ACTR and the model failed to capture the conflict between the addition and targeting tasks. We view this not only as a limitation of the model, but also of the ACT-R system, as there is no direct representation of visual-spatial memory in ACT-R (Lyon et al., 2004). We propose both further use of QN-ACTR to investigate details of the use of visual-

spatial memory in complex addition, and a concrete representation of the resource in ACT-R or QN-ACTR.

Second, the primary empirical finding of our second experiment was the lack of increase in the concurrency metric under incentive. The empirical data show a small but significant increase in the rewarded dimension in dual-task conditions, but no increase in the concurrency metric. We suggest that further empirical studies in different task domains to verify the result. Data from existing studies can also be analyzed for the same purpose.

The addition of the Soar extension did not reduce the performance of the IMPRINT simulations of the UAV task or the combined addition and targeting task. However, an analysis should be performed to examine whether the extension slows simulations with a large number of task components.

Although the model predicted accurate execution times and concurrency in dual-task performance, we found that the least-recently processed policy for selecting goal productions in the threaded-cognition model caused sensitivity to the number of productions in the individual ACT-R task models in the QN-ACTR simulation. This resulted in slightly overestimated addition subtask execution times and slightly underestimated target subtask execution times. We suggest that the threaded-cognition approach to modeling dual task performance in QN-ACTR should be replaced by a parallel goal processing mechanism, or that the threading policy should be modified to model a more sophisticated executive process that schedules task components with a model of task set inertia and task switching costs (Monsell, 2003; Reynolds et al., 2006; Meiran, 2000; Rubinstein et al., 2001).

Our work on improving the power and accuracy of cognitive models and modeling tools can be applied in diverse and important domains. During an internship at

the Army Research Laboratory, I attended a meeting between an analyst who uses IMPRINT to assess tasks and an operator of a military vehicle. The analyst had produced a model of vehicle interface and the operator's task and interviewed the operator to confirm the model design. The model was used to evaluate the difficulty of the task and suggest improvements to protect the safety of the operator. This application domain, along with many others, illustrates the value of computational task performance simulation and the ability to quickly assess task difficulty and interface usability.

APPENDICES

APPENDIX A

ACT-R models

Below are the ACT-R models used to model the tasks in the experiment described in chapter III. Minor modifications were made for the experiment in chapter IV and so the full models are omitted.

A.1 Addition model

```
(defparameter *seq-base-level* 1)
(defparameter *n-low-base-level* 3.5)
(defparameter *n-high-base-level* 2)
(defparameter *a-base-level* 2.7)

(define-model addition

  (sgp
    :esc t
    :ans 0.5)

  (set-visloc-default isa visual-location color
     does-not-exist)

  (chunk-type arithmetic first second result ones carry)
  (chunk-type arithmetic-problem first-ones second-ones
     first-tens second-tens state ones carry tens)
  (chunk-type successor value successor)
  (chunk-type number ones tens value)

  (add-dm
    (a0+0 ISA ARITHMETIC FIRST "0" SECOND "0" RESULT "0"
       ONES "0" CARRY "0")
```

```
(a0+1 ISA ARITHMETIC FIRST "0" SECOND "1" RESULT "1"
   ONES "1" CARRY "0")
(a0+2 ISA ARITHMETIC FIRST "0" SECOND "2" RESULT "2"
   ONES "2" CARRY "0")
(a0+3 ISA ARITHMETIC FIRST "0" SECOND "3" RESULT "3"
   ONES "3" CARRY "0")
(a0+4 ISA ARITHMETIC FIRST "0" SECOND "4" RESULT "4"
   ONES "4" CARRY "0")
(a0+5 ISA ARITHMETIC FIRST "0" SECOND "5" RESULT "5"
   ONES "5" CARRY "0")
(a0+6 ISA ARITHMETIC FIRST "0" SECOND "6" RESULT "6"
   ONES "6" CARRY "0")
(a0+7 ISA ARITHMETIC FIRST "0" SECOND "7" RESULT "7"
   ONES "7" CARRY "0")
(a0+8 ISA ARITHMETIC FIRST "0" SECOND "8" RESULT "8"
   ONES "8" CARRY "0")
(a0+9 ISA ARITHMETIC FIRST "0" SECOND "9" RESULT "9"
   ONES "9" CARRY "0")
(a1+0 ISA ARITHMETIC FIRST "1" SECOND "0" RESULT "1"
   ONES "1" CARRY "0")
(a1+1 ISA ARITHMETIC FIRST "1" SECOND "1" RESULT "2"
   ONES "2" CARRY "0")
(a1+2 ISA ARITHMETIC FIRST "1" SECOND "2" RESULT "3"
   ONES "3" CARRY "0")
(a1+3 ISA ARITHMETIC FIRST "1" SECOND "3" RESULT "4"
   ONES "4" CARRY "0")
(a1+4 ISA ARITHMETIC FIRST "1" SECOND "4" RESULT "5"
   ONES "5" CARRY "0")
(a1+5 ISA ARITHMETIC FIRST "1" SECOND "5" RESULT "6"
   ONES "6" CARRY "0")
(a1+6 ISA ARITHMETIC FIRST "1" SECOND "6" RESULT "7"
   ONES "7" CARRY "0")
(a1+7 ISA ARITHMETIC FIRST "1" SECOND "7" RESULT "8"
   ONES "8" CARRY "0")
(a1+8 ISA ARITHMETIC FIRST "1" SECOND "8" RESULT "9"
   ONES "9" CARRY "0")
(a1+9 ISA ARITHMETIC FIRST "1" SECOND "9" RESULT "10"
   ONES "0" CARRY "1")
(a2+0 ISA ARITHMETIC FIRST "2" SECOND "0" RESULT "2"
   ONES "2" CARRY "0")
(a2+1 ISA ARITHMETIC FIRST "2" SECOND "1" RESULT "3"
   ONES "3" CARRY "0")
(a2+2 ISA ARITHMETIC FIRST "2" SECOND "2" RESULT "4"
   ONES "4" CARRY "0")
(a2+3 ISA ARITHMETIC FIRST "2" SECOND "3" RESULT "5"
   ONES "5" CARRY "0")
```

```
(a2+4 ISA ARITHMETIC FIRST "2" SECOND "4" RESULT "6"
   ONES "6" CARRY "0")
(a2+5 ISA ARITHMETIC FIRST "2" SECOND "5" RESULT "7"
   ONES "7" CARRY "0")
(a2+6 ISA ARITHMETIC FIRST "2" SECOND "6" RESULT "8"
   ONES "8" CARRY "0")
(a2+7 ISA ARITHMETIC FIRST "2" SECOND "7" RESULT "9"
   ONES "9" CARRY "0")
(a2+8 ISA ARITHMETIC FIRST "2" SECOND "8" RESULT "10"
   ONES "0" CARRY "1")
(a2+9 ISA ARITHMETIC FIRST "2" SECOND "9" RESULT "11"
   ONES "1" CARRY "1")
(a3+0 ISA ARITHMETIC FIRST "3" SECOND "0" RESULT "3"
   ONES "3" CARRY "0")
(a3+1 ISA ARITHMETIC FIRST "3" SECOND "1" RESULT "4"
   ONES "4" CARRY "0")
(a3+2 ISA ARITHMETIC FIRST "3" SECOND "2" RESULT "5"
   ONES "5" CARRY "0")
(a3+3 ISA ARITHMETIC FIRST "3" SECOND "3" RESULT "6"
   ONES "6" CARRY "0")
(a3+4 ISA ARITHMETIC FIRST "3" SECOND "4" RESULT "7"
   ONES "7" CARRY "0")
(a3+5 ISA ARITHMETIC FIRST "3" SECOND "5" RESULT "8"
   ONES "8" CARRY "0")
(a3+6 ISA ARITHMETIC FIRST "3" SECOND "6" RESULT "9"
   ONES "9" CARRY "0")
(a3+7 ISA ARITHMETIC FIRST "3" SECOND "7" RESULT "10"
   ONES "0" CARRY "1")
(a3+8 ISA ARITHMETIC FIRST "3" SECOND "8" RESULT "11"
   ONES "1" CARRY "1")
(a3+9 ISA ARITHMETIC FIRST "3" SECOND "9" RESULT "12"
   ONES "2" CARRY "1")
(a4+0 ISA ARITHMETIC FIRST "4" SECOND "0" RESULT "4"
   ONES "4" CARRY "0")
(a4+1 ISA ARITHMETIC FIRST "4" SECOND "1" RESULT "5"
   ONES "5" CARRY "0")
(a4+2 ISA ARITHMETIC FIRST "4" SECOND "2" RESULT "6"
   ONES "6" CARRY "0")
(a4+3 ISA ARITHMETIC FIRST "4" SECOND "3" RESULT "7"
   ONES "7" CARRY "0")
(a4+4 ISA ARITHMETIC FIRST "4" SECOND "4" RESULT "8"
   ONES "8" CARRY "0")
(a4+5 ISA ARITHMETIC FIRST "4" SECOND "5" RESULT "9"
   ONES "9" CARRY "0")
(a4+6 ISA ARITHMETIC FIRST "4" SECOND "6" RESULT "10"
   ONES "0" CARRY "1")
```

```
(a4+7 ISA ARITHMETIC FIRST "4" SECOND "7" RESULT "11"
   ONES "1" CARRY "1")
(a4+8 ISA ARITHMETIC FIRST "4" SECOND "8" RESULT "12"
   ONES "2" CARRY "1")
(a4+9 ISA ARITHMETIC FIRST "4" SECOND "9" RESULT "13"
   ONES "3" CARRY "1")
(a5+0 ISA ARITHMETIC FIRST "5" SECOND "0" RESULT "5"
   ONES "5" CARRY "0")
(a5+1 ISA ARITHMETIC FIRST "5" SECOND "1" RESULT "6"
   ONES "6" CARRY "0")
(a5+2 ISA ARITHMETIC FIRST "5" SECOND "2" RESULT "7"
   ONES "7" CARRY "0")
(a5+3 ISA ARITHMETIC FIRST "5" SECOND "3" RESULT "8"
   ONES "8" CARRY "0")
(a5+4 ISA ARITHMETIC FIRST "5" SECOND "4" RESULT "9"
   ONES "9" CARRY "0")
(a5+5 ISA ARITHMETIC FIRST "5" SECOND "5" RESULT "10"
   ONES "0" CARRY "1")
(a5+6 ISA ARITHMETIC FIRST "5" SECOND "6" RESULT "11"
   ONES "1" CARRY "1")
(a5+7 ISA ARITHMETIC FIRST "5" SECOND "7" RESULT "12"
   ONES "2" CARRY "1")
(a5+8 ISA ARITHMETIC FIRST "5" SECOND "8" RESULT "13"
   ONES "3" CARRY "1")
(a5+9 ISA ARITHMETIC FIRST "5" SECOND "9" RESULT "14"
   ONES "4" CARRY "1")
(a6+0 ISA ARITHMETIC FIRST "6" SECOND "0" RESULT "6"
   ONES "6" CARRY "0")
(a6+1 ISA ARITHMETIC FIRST "6" SECOND "1" RESULT "7"
   ONES "7" CARRY "0")
(a6+2 ISA ARITHMETIC FIRST "6" SECOND "2" RESULT "8"
   ONES "8" CARRY "0")
(a6+3 ISA ARITHMETIC FIRST "6" SECOND "3" RESULT "9"
   ONES "9" CARRY "0")
(a6+4 ISA ARITHMETIC FIRST "6" SECOND "4" RESULT "10"
   ONES "0" CARRY "1")
(a6+5 ISA ARITHMETIC FIRST "6" SECOND "5" RESULT "11"
   ONES "1" CARRY "1")
(a6+6 ISA ARITHMETIC FIRST "6" SECOND "6" RESULT "12"
   ONES "2" CARRY "1")
(a6+7 ISA ARITHMETIC FIRST "6" SECOND "7" RESULT "13"
   ONES "3" CARRY "1")
(a6+8 ISA ARITHMETIC FIRST "6" SECOND "8" RESULT "14"
   ONES "4" CARRY "1")
(a6+9 ISA ARITHMETIC FIRST "6" SECOND "9" RESULT "15"
   ONES "5" CARRY "1")
```

```
(a7+0 ISA ARITHMETIC FIRST "7" SECOND "0" RESULT "7"
   ONES "7" CARRY "0")
(a7+1 ISA ARITHMETIC FIRST "7" SECOND "1" RESULT "8"
   ONES "8" CARRY "0")
(a7+2 ISA ARITHMETIC FIRST "7" SECOND "2" RESULT "9"
   ONES "9" CARRY "0")
(a7+3 ISA ARITHMETIC FIRST "7" SECOND "3" RESULT "10"
   ONES "0" CARRY "1")
(a7+4 ISA ARITHMETIC FIRST "7" SECOND "4" RESULT "11"
   ONES "1" CARRY "1")
(a7+5 ISA ARITHMETIC FIRST "7" SECOND "5" RESULT "12"
   ONES "2" CARRY "1")
(a7+6 ISA ARITHMETIC FIRST "7" SECOND "6" RESULT "13"
   ONES "3" CARRY "1")
(a7+7 ISA ARITHMETIC FIRST "7" SECOND "7" RESULT "14"
   ONES "4" CARRY "1")
(a7+8 ISA ARITHMETIC FIRST "7" SECOND "8" RESULT "15"
   ONES "5" CARRY "1")
(a7+9 ISA ARITHMETIC FIRST "7" SECOND "9" RESULT "16"
   ONES "6" CARRY "1")
(a8+0 ISA ARITHMETIC FIRST "8" SECOND "0" RESULT "8"
   ONES "8" CARRY "0")
(a8+1 ISA ARITHMETIC FIRST "8" SECOND "1" RESULT "9"
   ONES "9" CARRY "0")
(a8+2 ISA ARITHMETIC FIRST "8" SECOND "2" RESULT "10"
   ONES "0" CARRY "1")
(a8+3 ISA ARITHMETIC FIRST "8" SECOND "3" RESULT "11"
   ONES "1" CARRY "1")
(a8+4 ISA ARITHMETIC FIRST "8" SECOND "4" RESULT "12"
   ONES "2" CARRY "1")
(a8+5 ISA ARITHMETIC FIRST "8" SECOND "5" RESULT "13"
   ONES "3" CARRY "1")
(a8+6 ISA ARITHMETIC FIRST "8" SECOND "6" RESULT "14"
   ONES "4" CARRY "1")
(a8+7 ISA ARITHMETIC FIRST "8" SECOND "7" RESULT "15"
   ONES "5" CARRY "1")
(a8+8 ISA ARITHMETIC FIRST "8" SECOND "8" RESULT "16"
   ONES "6" CARRY "1")
(a8+9 ISA ARITHMETIC FIRST "8" SECOND "9" RESULT "17"
   ONES "7" CARRY "1")
(a9+0 ISA ARITHMETIC FIRST "9" SECOND "0" RESULT "9"
   ONES "9" CARRY "0")
(a9+1 ISA ARITHMETIC FIRST "9" SECOND "1" RESULT "10"
   ONES "0" CARRY "1")
(a9+2 ISA ARITHMETIC FIRST "9" SECOND "2" RESULT "11"
   ONES "1" CARRY "1")
```

```
(a9+3 ISA ARITHMETIC FIRST "9" SECOND "3" RESULT "12"
   ONES "2" CARRY "1")
(a9+4 ISA ARITHMETIC FIRST "9" SECOND "4" RESULT "13"
   ONES "3" CARRY "1")
(a9+5 ISA ARITHMETIC FIRST "9" SECOND "5" RESULT "14"
   ONES "4" CARRY "1")
(a9+6 ISA ARITHMETIC FIRST "9" SECOND "6" RESULT "15"
   ONES "5" CARRY "1")
(a9+7 ISA ARITHMETIC FIRST "9" SECOND "7" RESULT "16"
   ONES "6" CARRY "1")
(a9+8 ISA ARITHMETIC FIRST "9" SECOND "8" RESULT "17"
   ONES "7" CARRY "1")
(a9+9 ISA ARITHMETIC FIRST "9" SECOND "9" RESULT "18"
   ONES "8" CARRY "1")

(n00 ISA NUMBER VALUE "0" ONES "0" TENS nil)
(n01 ISA NUMBER VALUE "1" ONES "1" TENS nil)
(n02 ISA NUMBER VALUE "2" ONES "2" TENS nil)
(n03 ISA NUMBER VALUE "3" ONES "3" TENS nil)
(n04 ISA NUMBER VALUE "4" ONES "4" TENS nil)
(n05 ISA NUMBER VALUE "5" ONES "5" TENS nil)
(n06 ISA NUMBER VALUE "6" ONES "6" TENS nil)
(n07 ISA NUMBER VALUE "7" ONES "7" TENS nil)
(n08 ISA NUMBER VALUE "8" ONES "8" TENS nil)
(n09 ISA NUMBER VALUE "9" ONES "9" TENS nil)
(n10 ISA NUMBER VALUE "10" ONES "0" TENS "1")
(n11 ISA NUMBER VALUE "11" ONES "1" TENS "1")
(n12 ISA NUMBER VALUE "12" ONES "2" TENS "1")
(n13 ISA NUMBER VALUE "13" ONES "3" TENS "1")
(n14 ISA NUMBER VALUE "14" ONES "4" TENS "1")
(n15 ISA NUMBER VALUE "15" ONES "5" TENS "1")
(n16 ISA NUMBER VALUE "16" ONES "6" TENS "1")
(n17 ISA NUMBER VALUE "17" ONES "7" TENS "1")
(n18 ISA NUMBER VALUE "18" ONES "8" TENS "1")
(n19 ISA NUMBER VALUE "19" ONES "9" TENS "1")
(n20 ISA NUMBER VALUE "20" ONES "0" TENS "2")
(n21 ISA NUMBER VALUE "21" ONES "1" TENS "2")
(n22 ISA NUMBER VALUE "22" ONES "2" TENS "2")
(n23 ISA NUMBER VALUE "23" ONES "3" TENS "2")
(n24 ISA NUMBER VALUE "24" ONES "4" TENS "2")

(s01 ISA SUCCESSOR VALUE "0" SUCCESSOR "1")
(s12 ISA SUCCESSOR VALUE "1" SUCCESSOR "2")
(s23 ISA SUCCESSOR VALUE "2" SUCCESSOR "3")
(s34 ISA SUCCESSOR VALUE "3" SUCCESSOR "4")
```

```
  (addition-goal ISA arithmetic-problem state
    find-first))

(goal-focus addition-goal)

(set-base-levels
  (s01 *seq-base-level*) (s12 *seq-base-level*) (s23
    *seq-base-level*)
  (s34 *seq-base-level*) (s45 *seq-base-level*) (s56
    *seq-base-level*)
  (s67 *seq-base-level*) (s78 *seq-base-level*) (s89
    *seq-base-level*)

  (n00 *n-low-base-level*) (n01 *n-low-base-level*)
    (n02 *n-low-base-level*)
  (n03 *n-low-base-level*) (n04 *n-low-base-level*)
    (n05 *n-low-base-level*) (n06 *n-low-base-level*)
  (n07 *n-low-base-level*) (n08 *n-low-base-level*)
    (n09 *n-low-base-level*)
  (n10 *n-low-base-level*) (n11 *n-low-base-level*)
    (n12 *n-low-base-level*)

  (n13 *n-high-base-level*) (n14 *n-high-base-level*)
    (n15 *n-high-base-level*)
  (n16 *n-high-base-level*) (n17 *n-high-base-level*)
    (n18 *n-high-base-level*)
  (n19 *n-high-base-level*) (n20 *n-high-base-level*)
    (n21 *n-high-base-level*)
  (n22 *n-high-base-level*) (n23 *n-high-base-level*)
    (n24 *n-high-base-level*) (n25 *n-high-base-level*)
)

(set-base-levels
  (a0+0 *a-base-level*) (a0+1 *a-base-level*) (a0+2
    *a-base-level*) (a0+3 *a-base-level*) (a0+4
    *a-base-level*)
  (a0+5 *a-base-level*) (a0+6 *a-base-level*) (a0+7
    *a-base-level*) (a0+8 *a-base-level*) (a0+9
    *a-base-level*)
  (a1+0 *a-base-level*) (a1+1 *a-base-level*) (a1+2
    *a-base-level*) (a1+3 *a-base-level*) (a1+4
    *a-base-level*)
  (a1+5 *a-base-level*) (a1+6 *a-base-level*) (a1+7
    *a-base-level*) (a1+8 *a-base-level*) (a1+9
    *a-base-level*)
  (a2+0 *a-base-level*) (a2+1 *a-base-level*) (a2+2
```

```
   *a-base-level*) (a2+3 *a-base-level*) (a2+4
   *a-base-level*)
(a2+5 *a-base-level*) (a2+6 *a-base-level*) (a2+7
   *a-base-level*) (a2+8 *a-base-level*) (a2+9
   *a-base-level*)
(a3+0 *a-base-level*) (a3+1 *a-base-level*) (a3+2
   *a-base-level*) (a3+3 *a-base-level*) (a3+4
   *a-base-level*)
(a3+5 *a-base-level*) (a3+6 *a-base-level*) (a3+7
   *a-base-level*) (a3+8 *a-base-level*) (a3+9
   *a-base-level*)
(a4+0 *a-base-level*) (a4+1 *a-base-level*) (a4+2
   *a-base-level*) (a4+3 *a-base-level*) (a4+4
   *a-base-level*)
(a4+5 *a-base-level*) (a4+6 *a-base-level*) (a4+7
   *a-base-level*) (a4+8 *a-base-level*) (a4+9
   *a-base-level*)
(a5+0 *a-base-level*) (a5+1 *a-base-level*) (a5+2
   *a-base-level*) (a5+3 *a-base-level*) (a5+4
   *a-base-level*)
(a5+5 *a-base-level*) (a5+6 *a-base-level*) (a5+7
   *a-base-level*) (a5+8 *a-base-level*) (a5+9
   *a-base-level*)
(a6+0 *a-base-level*) (a6+1 *a-base-level*) (a6+2
   *a-base-level*) (a6+3 *a-base-level*) (a6+4
   *a-base-level*)
(a6+5 *a-base-level*) (a6+6 *a-base-level*) (a6+7
   *a-base-level*) (a6+8 *a-base-level*) (a6+9
   *a-base-level*)
(a7+0 *a-base-level*) (a7+1 *a-base-level*) (a7+2
   *a-base-level*) (a7+3 *a-base-level*) (a7+4
   *a-base-level*)
(a7+5 *a-base-level*) (a7+6 *a-base-level*) (a7+7
   *a-base-level*) (a7+8 *a-base-level*) (a7+9
   *a-base-level*)
(a8+0 *a-base-level*) (a8+1 *a-base-level*) (a8+2
   *a-base-level*) (a8+3 *a-base-level*) (a8+4
   *a-base-level*)
(a8+5 *a-base-level*) (a8+6 *a-base-level*) (a8+7
   *a-base-level*) (a8+8 *a-base-level*) (a8+9
   *a-base-level*)
(a9+0 *a-base-level*) (a9+1 *a-base-level*) (a9+2
   *a-base-level*) (a9+3 *a-base-level*) (a9+4
   *a-base-level*)
(a9+5 *a-base-level*) (a9+6 *a-base-level*) (a9+7
   *a-base-level*) (a9+8 *a-base-level*) (a9+9
```

```
     *a-base-level*)
)

(P find-first
  =goal>
    ISA          arithmetic-problem
    state        find-first

  ?visual-location>
    buffer       empty
==>
  +visual-location>
    ISA          visual-location
    :attended    nil
    kind         text
    screen-x     lowest

  =goal>
    state        attend-first
  )

(P attend-first
  =goal>
    ISA          arithmetic-problem
    state        attend-first

  =visual-location>
    ISA          visual-location
    kind text

  ?visual>
    state        free
    buffer       empty
==>
  +visual>
    ISA          move-attention
    screen-pos   =visual-location

  =goal>
    state        encode-first
)

(P encode-first
  =goal>
    ISA          arithmetic-problem
    state        encode-first
```

```
  ?visual>
    state       free
  =visual>
    ISA         text
    value       =value

  ?retrieval>
    state       free
    buffer      empty

==>
  +retrieval>
    ISA         number
    value       =value

  +visual>
    ISA         clear

  =goal>
    state       find-second
)

(P store-first
  =goal>
    ISA         arithmetic-problem
    first-ones nil

  =retrieval>
    ISA         number
    ones        =ones
    tens        =tens
==>
  =goal>
    first-ones =ones
    first-tens =tens
  )

(P store-first-nil-tens
  =goal>
    ISA         arithmetic-problem
    first-ones  nil

  =retrieval>
    ISA         number
    ones        =ones
```

```
    tens         nil
==>
  =goal>
    first-ones =ones
)

(P find-second
  =goal>
    ISA arithmetic-problem
    state find-second

  ?visual-location>
    buffer  empty
  ?visual>
    state free
    buffer empty
==>
  +visual-location>
    ISA          visual-location
    kind         text
    screen-x     highest

  =goal>
    state attend-second
)

(P attend-second
  =goal>
    ISA arithmetic-problem
    state attend-second

  =visual-location>
    ISA visual-location
    kind text

  ?visual>
    buffer empty
    state free
==>
  +visual>
    ISA          move-attention
    screen-pos   =visual-location

  =goal>
    state encode-second
)
```

```
(P encode-second-ones
  =goal>
    ISA          arithmetic-problem
    state        encode-second
    - first-ones nil

  =visual>
    ISA          text
    value        =value

  ?visual>
    state free

  ?retrieval>
    state        free
    buffer       empty
==>
  +retrieval>
    ISA          number
    value        =value

  +visual>
    ISA          clear

  =goal>
    state        store-second
)

(P store-second
  =goal>
    ISA          arithmetic-problem
    state        store-second
    first-ones =first-ones

  =retrieval>
    ISA          number
    ones         =second-ones
    tens         =second-tens

  ?retrieval>
    state        free
==>
  =goal>
    state         finish-retrieve-ones
    second-ones  =second-ones
```

```
    second-tens  =second-tens

  +retrieval>
    ISA          arithmetic
    first        =first-ones
    second       =second-ones
  )

(P store-second-tens-nil
  =goal>
    ISA          arithmetic-problem
    state        store-second
    first-ones =first-ones

  =retrieval>
    ISA          number
    ones         =second-ones
    tens         nil

  ?retrieval>
    state        free
==>
  +retrieval>
    ISA          arithmetic
    first        =first-ones
    second       =second-ones

  =goal>
    state        finish-retrieve-ones
    second-ones =second-ones
)

(P finish-retrieve-ones
  =goal>
    ISA          arithmetic-problem
    state        finish-retrieve-ones
    first-ones   =first
    second-ones =second

  =retrieval>
    ISA          arithmetic
    first        =first
    second       =second
    ones         =ones
    carry        =carry
==>
```

```
  =goal>
    ones         =ones
    carry        =carry
    state        add-tens
)

(P add-tens-nil-nil-no-carry
  =goal>
    ISA          arithmetic-problem
    state        add-tens
    first-tens   nil
    second-tens  nil
    carry        "0"
==>
  =goal>
    tens         "0"
    state        response
)

(P add-tens-nil-nil-carry
  =goal>
    ISA          arithmetic-problem
    state        add-tens
    first-tens   nil
    second-tens  nil
    - carry      "0"
==>
  =goal>
    tens         "1"
    state        response
)

(P add-tens-first-nil-no-carry
  =goal>
    ISA          arithmetic-problem
    state        add-tens
    first-tens   =first-tens
    second-tens  nil
    carry        "0"
==>
  =goal>
    tens         =first-tens
    state        response
)

(P add-tens-first-nil-carry
```

```
  =goal>
    ISA          arithmetic-problem
    state        add-tens
    first-tens   =first-tens
    second-tens nil
    - carry        "0"
==>
  =goal>
    tens         =first-tens
    state        check-carry
)

(P add-tens-nil-second-no-carry
  =goal>
    ISA          arithmetic-problem
    state        add-tens
    first-tens   nil
    second-tens =second-tens
    carry        "0"
==>
  =goal>
    tens         =second-tens
    state        response
)

(P add-tens-nil-second-carry
  =goal>
    ISA          arithmetic-problem
    state        add-tens
    first-tens   nil
    second-tens =second-tens
    - carry        "0"
==>
  =goal>
    tens         =second-tens
    state        check-carry
)

(P add-tens
  =goal>
    ISA          arithmetic-problem
    state        add-tens
    first-tens   =first-tens
    second-tens =second-tens
==>
  =goal>
```

```
    state       retrieve-addition-tens
)

(P retrieve-addition-tens
  =goal>
    ISA         arithmetic-problem
    state       retrieve-addition-tens
    first-tens  =first
    second-tens =second

  ?retrieval>
    state       free
    buffer      empty
==>
  =goal>
    state    finish-retrieve-tens

  +retrieval>
    ISA         arithmetic
    first       =first
    second      =second
)

(P finish-retrieve-tens-no-carry
  =goal>
    ISA         arithmetic-problem
    state       finish-retrieve-tens
    first-tens  =first
    second-tens =second
    carry       "0"

  =retrieval>
    ISA         arithmetic
    first       =first
    second      =second
    ones        =ones
==>
  =goal>
    tens        =ones
    state       response
)

(P finish-retrieve-tens-carry
  =goal>
    ISA         arithmetic-problem
    state       finish-retrieve-tens
```

```
    first-tens  =first
    second-tens =second
    - carry       "0"

  =retrieval>
    ISA         arithmetic
    first       =first
    second      =second
    ones        =ones
==>
  =goal>
    tens        =ones
    state       check-carry
)

(P check-carry
  =goal>
    ISA         arithmetic-problem
    state       check-carry
    carry       "1"
    tens        =tens

  ?retrieval>
    state       free
    buffer      empty
==>
  +retrieval>
    ISA         SUCCESSOR
    value       =tens

  =goal>
    state       increment-tens
  )

(P increment-tens
  =goal>
    ISA         arithmetic-problem
    state       increment-tens

  =retrieval>
    ISA         SUCCESSOR
    SUCCESSOR   =value
==>
  =goal>
    tens        =value
    state       response
```

```
  )

(P respond-tens
  =goal>
    ISA          arithmetic-problem
    state        response
    tens         =tens
    - tens       "0"

  ?vocal>
    preparation free
==>
  +vocal>
    ISA          speak
    string       =tens

  =goal>
    state        response-ones
  )

(P respond-ones-no-tens
  =goal>
    ISA          arithmetic-problem
    state        response
    tens         "0"
    ones         =ones

  ?vocal>
    preparation free
==>
  +vocal>
    ISA          speak
    string       =ones

  =goal>
    state        done
  )

(P respond-ones
  =goal>
    ISA          arithmetic-problem
    state        response-ones
    ones         =ones

  ?vocal>
    preparation free
```

```
  ==>
    +vocal>
      ISA          speak
      string       =ones

    =goal>
      state        done
    )
)
```

A.2 Targeting models

A.2.1 Low speed, low difficulty

```
(define-model targeting-easy-slow

  (sgp
    :cursor-noise t
    :incremental-mouse-moves 0.01
    :esc t)

  (set-visloc-default isa visual-location color
     does-not-exist)
  (start-hand-at-mouse)
  (set-cursor-position 960 600)

  (chunk-type targeting state target-x target-y
     target-location check-miss)
  (chunk-type response color action)

  (add-dm
    (enemy-response isa response color red action shoot)
    (friend-response isa response color green action
       dont-shoot)
    (goal isa targeting state find-red-target))

  (goal-focus goal)

  (P find-red-target
    =goal>
      ISA          targeting
      state        find-red-target

    ?visual-location>
      buffer       empty

  ==>
```

```
  +visual-location>
    ISA          visual-location
    :attended    nil
    kind         OVAL
    color        red

  =goal>
    state        move-cursor
)

(P no-unattended-red
  =goal>
    ISA            targeting
    state          move-cursor
    check-miss     nil

  ?visual-location>
    state          error

  ?manual>
    state          free
==>
  =goal>
    state no-unattended-red-search
    check-miss     t
)

(P no-unattended-red-search
  =goal>
    ISA targeting
    state no-unattended-red-search

  ?visual-location>
    buffer empty
==>
  +visual-location>
    ISA visual-location
    kind OVAL
    color red

  =goal>
    state move-cursor
)

(P no-red
  =goal>
```

```
    ISA             targeting
    state           move-cursor
    check-miss      t

  ?visual-location>
    state           error
==>
  =goal>
    state           end
)

(P move-cursor
  =goal>
    ISA             targeting
    state           move-cursor

  =visual-location>
    ISA             visual-location
    kind            OVAL

  ?manual>
    preparation     free
==>

  +manual>
    ISA             move-cursor
    loc             =visual-location

  =goal>
    state           move-attention
    target-location =visual-location
    check-miss      nil
)

(P move-attention
  =goal>
    ISA targeting
    state move-attention
    target-location =target-location

  ?visual>
    state           free
    buffer          empty
==>

  +visual>
```

```
      ISA             move-attention
      screen-pos      =target-location

    =goal>
      state click-mouse
  )

  (P click-mouse
    =goal>
      ISA             targeting
      state           click-mouse

    ?manual>
      state           free

    =visual>
      ISA             OVAL
  ==>

    +manual>
      ISA             click-mouse

    +visual>
      ISA             clear

    =goal>
      state           find-red-target
  )
)
```

A.2.2 High speed, low difficulty

```
(define-model targeting-easy-fast

  (sgp
    :cursor-noise t
    :incremental-mouse-moves 0.01)

  (set-visloc-default isa visual-location color
     does-not-exist)
  (start-hand-at-mouse)
  (set-cursor-position 960 600)

  (chunk-type targeting state target-location target-x
     target-y projected-x projected-y)

  (add-dm (goal isa targeting state find-red-target))
```

```
(goal-focus goal)

(P find-red-target
  =goal>
    ISA          targeting
    state        find-red-target

  ?manual>
    state        free

  ?visual-location>
    buffer       empty
==>

  +visual-location>
    ISA          visual-location
    :attended    nil
    kind         OVAL
    color        red

  =goal>
    state        cap-first-location

)

(P fail-find
  =goal>
    ISA          targeting
    state        cap-first-location

  ?visual-location>
    state        error
==>
  =goal>
    state        fail
)

(P cap-first-location
  =goal>
    ISA            targeting
    state          cap-first-location

  =visual-location>
    ISA            visual-location
    screen-x       =tx
```

```
    screen-y      =ty

==>
  =goal>
    target-x      =tx
    target-y      =ty
    target-location =visual-location
    state         cap-first-location-search
)

(P cap-first-location-search
  =goal>
    isa targeting
    state cap-first-location-search
    target-location =target-location

  ?visual-location>
    buffer empty

==>
  +visual-location>
    ISA           visual-location
    :nearest      =target-location
    color         red

  =goal>
    state lead-target
)

(P lead-target
  =goal>
    ISA           targeting
    state         lead-target
    target-x      =tx
    target-y      =ty

  =visual-location>
    ISA           visual-location
    screen-x      =sx
    screen-y      =sy

==>
  !bind!          =x-diff (- =sx =tx)
  !bind!          =y-diff (- =sy =ty)
  !bind!          =mag (sqrt (+ (* =x-diff =x-diff) (*
     =y-diff =y-diff)))
```

```
  !bind!          =x-diff-normal (/ =x-diff =mag)
  !bind!          =y-diff-normal (/ =y-diff =mag)
  !bind!          =projected-x (+ =sx (*
     *target-projection* =x-diff-normal))
  !bind!          =projected-y (+ =sy (*
     *target-projection* =y-diff-normal))

  =visual-location>
    screen-x      =projected-x
    screen-y      =projected-y

  =goal>
    state         move-cursor
)

(P move-cursor
  =goal>
    ISA           targeting
    state         move-cursor

  =visual-location>
    ISA           visual-location
    kind          OVAL

  ?manual>
    preparation   free
==>

  +manual>
    ISA           move-cursor
    loc           =visual-location

  =goal>
    state move-attention
    target-location =visual-location
)

(P move-attention
  =goal>
    isa targeting
    state move-attention
    target-location =target-location

  ?visual>
    state         free
    buffer        empty
```

```
==>

  +visual>
    ISA           move-attention
    screen-pos    =target-location

  =goal>
    state prepare-click
)

(P prepare-click
  =goal>
    ISA           targeting
    state         prepare-click

  ?manual>
    last-command  move-cursor
    preparation   free

==>

  +manual>
    ISA           prepare
    style         punch
    hand          right
    finger        index

  =goal>
    state click-mouse
)

(P click-mouse
  =goal>
    ISA           targeting
    state         click-mouse

  ?manual>
    last-command  prepare
    preparation   free

  ?visual>
    state         free
  =visual>
    ISA           OVAL
==>
  +manual>
```

```
      ISA             execute

    +visual>
      ISA             clear

    =goal>
      state           find-red-target
  )
)
```

A.2.3 Low speed, high difficulty

```
(define-model targeting-hard-slow

  (sgp
    :cursor-noise t
    :incremental-mouse-moves 0.01
    :esc t)

  (set-visloc-default isa visual-location color
     does-not-exist)
  (start-hand-at-mouse)
  (set-cursor-position 960 600)

  (chunk-type targeting state target-location target-x
     target-y friend-x friend-y)
  (chunk-type response color action)

  (add-dm
    (enemy-response isa response color red action shoot)
    (friend-response isa response color green action
       dont-shoot)
    (goal isa targeting state find-black-target))

  (set-base-levels (enemy-response 3) (friend-response 3))

  (goal-focus goal)

  (P find-black-target
    =goal>
      ISA         targeting
      state       find-black-target

    ?visual-location>
      buffer      empty
  ==>
    +visual-location>
```

```
    ISA          visual-location
    kind         OVAL
    color        black

  =goal>
    state        move-cursor
)

(P avoid-friend
  =goal>
    ISA            targeting
    state          move-cursor
    friend-x               =fx
    friend-y               =fy

  =visual-location>
    ISA            visual-location
    screen-x       =fx
    screen-y       =fy

==>
  =goal>
    state          find-black-target
)

(P move-cursor-no-friend-info
  =goal>
    ISA            targeting
    state          move-cursor
    friend-x       nil

  =visual-location>
    ISA            visual-location
    kind           OVAL
    screen-x       =x
    screen-y       =y

  ?manual>
    preparation    free
==>

  +manual>
    ISA            move-cursor
    loc            =visual-location

  =goal>
```

```
    state          move-attention
    target-x       =x
    target-y       =y
    target-location =visual-location
)

(P move-cursor-not-friend
  =goal>
    ISA            targeting
    state          move-cursor
    friend-x       =fx
    friend-y       =fy

  =visual-location>
    ISA            visual-location
    kind           OVAL
    - screen-x     =fx
    - screen-y     =fy
    screen-x       =x
    screen-y       =y

  ?manual>
    preparation    free
==>

  +manual>
    ISA            move-cursor
    loc            =visual-location

  =goal>
    state          move-attention
    target-x       =x
    target-y       =y
    target-location =visual-location
)

(P move-attention
  =goal>
    isa targeting
    state move-attention
    target-location =target-location

  ?visual>
    state free
    buffer empty
==>
```

```
  +visual>
    isa move-attention
    screen-pos =target-location

  =goal>
    state check-target
)

(P check-target
  =goal>
    ISA           targeting
    state         check-target
    target-x      =x
    target-y      =y

  ?visual-location>
    buffer        empty
==>
  +visual-location>
    ISA           visual-location
    kind          OVAL
    screen-x      =x
    screen-y      =y

  =goal>
    state         distinguish-target
)

(P harvest-visual
  =goal>
    isa targeting

  ?visual>
    state free

  =visual>
    isa OVAL
==>
  +visual>
    isa clear
)

(P detect-target-color
  =goal>
    ISA           targeting
```

```
    state          distinguish-target
    target-x       =target-x
    target-y       =target-y

  =visual-location>
    ISA            visual-location
    kind           OVAL
    - color        black
    color          =color

  ?retrieval>
    state          free
    buffer         empty
==>
  +retrieval>
    ISA            response
    color          =color

  =goal>
    state          decide-whether-to-shoot
)

(P decide-to-shoot
  =goal>
    ISA            targeting
    state          decide-whether-to-shoot

  =retrieval>
    ISA            response
    action         shoot

==>
  =goal>
    state          click-mouse
)

(P decide-not-to-shoot
  =goal>
    ISA            targeting
    state          decide-whether-to-shoot
    target-x       =sx
    target-y       =sy

  =retrieval>
    ISA            response
    action         dont-shoot
```

```
==>

  =goal>
    state          find-black-target
    friend-x       =sx
    friend-y       =sy
)

(P distinguish-whiff
  =goal>
    ISA            targeting
    state          distinguish-target

  =visual-location>
    ISA            visual-location
    kind           OVAL
    color          black

  ?manual>
    state          free
==>
  =goal>
    state distinguish-whiff-search

  +manual>
    isa            move-cursor
    loc            =visual-location
)

(P distinguish-whiff-search
  =goal>
    isa targeting
    state distinguish-whiff-search
    target-x       =target-x
    target-y       =target-y

  ?visual-location>
    buffer empty
==>
  +visual-location>
    ISA            visual-location
    kind           OVAL
    screen-x       =target-x
    screen-y       =target-y

  =goal>
```

```
    state distinguish-target
)

(P distinguish-target-black
  =goal>
    ISA           targeting
    state         distinguish-target

  =visual-location>
    ISA           visual-location
    kind          OVAL
    color         black

  ?manual>
    state         busy
==>
  =goal>
    state         distinguish-target-black-search
)

(P distinguish-target-black-search
  =goal>
    isa targeting
    state distinguish-target-black-search
    target-x      =target-x
    target-y      =target-y

  ?visual-location>
    buffer empty

==>
  +visual-location>
    ISA           visual-location
    kind          OVAL
    screen-x      =target-x
    screen-y      =target-y

  =goal>
    state distinguish-target
)

(P click-mouse
  =goal>
    ISA           targeting
    state         click-mouse
```

```
    ?manual>
      state          free
  ==>

    +manual>
      ISA            click-mouse

    =goal>
      state          find-black-target
  )
)
```

A.2.4 High speed, high difficulty

```
(define-model targeting-hard-fast

  (sgp
    :cursor-noise t
    :incremental-mouse-moves 0.01
    :esc t)

  (set-visloc-default isa visual-location color
     does-not-exist)
  (start-hand-at-mouse)
  (set-cursor-position 960 600)

  (chunk-type targeting state target-x target-y
     target-location
               friend-x friend-y friend-x-diff
                  friend-y-diff cur-x-diff cur-y-diff
                  ticks heuristic)
  (chunk-type response color action)

  (add-dm
    (enemy-response isa response color red action shoot)
    (friend-response isa response color green action
       dont-shoot)
    (goal isa targeting state find-black-target heuristic
       lowest))
  (set-base-levels (enemy-response 3) (friend-response 3))

  (goal-focus goal)

  (P find-black-target
    =goal>
      ISA          targeting
      state        find-black-target
```

```
    heuristic    =heuristic

  ?visual-location>
    buffer       empty
==>
  +visual-location>
    ISA          visual-location
    :attended    nil
    kind         OVAL
    color        black
    screen-x     =heuristic

  =goal>
    state        cap-first-location
)

(P avoid-friend-lowest
  =goal>
    ISA           targeting
    state         cap-first-location
    friend-x      =fx
    friend-y      =fy
    friend-x-diff =x-diff
    friend-y-diff =y-diff
    heuristic     lowest

  =visual-location>
    ISA           visual-location
    screen-x      =tx
    screen-y      =ty

  !bind!          =on-line (is-on-line =tx =ty =fx =fy
     =x-diff =y-diff)

==>
  =goal>
    state         find-black-target
    heuristic     highest
)
(P avoid-friend-highest
  =goal>
    ISA           targeting
    state         cap-first-location
    friend-x      =fx
    friend-y      =fy
    friend-x-diff =x-diff
```

```
    friend-y-diff =y-diff
    heuristic     highest

  =visual-location>
    ISA           visual-location
    screen-x      =tx
    screen-y      =ty

  !bind!          =on-line (is-on-line =tx =ty =fx =fy
     =x-diff =y-diff)

==>
  =goal>
    state         find-black-target
    heuristic     lowest
)

(P cap-first-location
  =goal>
    ISA           targeting
    state         cap-first-location
    friend-x      nil

  =visual-location>
    ISA           visual-location
    screen-x      =tx
    screen-y      =ty
==>
  =goal>
    target-x      =tx
    target-y      =ty
    state         cap-first-location-search
    target-location =visual-location
)
(P cap-first-location-not-friend
  =goal>
    ISA           targeting
    state         cap-first-location
    friend-x      =fx
    friend-y      =fy
    friend-x-diff =x-diff
    friend-y-diff =y-diff

  =visual-location>
    ISA           visual-location
    screen-x      =tx
```

```
    screen-y      =ty

  !bind!          =on-line (not (is-on-line =tx =ty =fx
     =fy =x-diff =y-diff))
==>
  =goal>
    target-x      =tx
    target-y      =ty
    state         cap-first-location-search
    target-location =visual-location
)

(P cap-first-location-search
  =goal>
    isa targeting
    state cap-first-location-search
    target-location =target-location

  ?visual-location>
    buffer empty
==>
  +visual-location>
    ISA visual-location
    kind      OVAL
    :nearest  =target-location

  =goal>
    state lead-target
)

(P lead-target
  =goal>
    ISA           targeting
    state         lead-target
    target-x      =tx
    target-y      =ty

  =visual-location>
    ISA           visual-location
    screen-x      =sx
    screen-y      =sy
==>
  !bind!          =x-diff (- =sx =tx)
  !bind!          =y-diff (- =sy =ty)
  !bind!          =mag (sqrt (+ (* =x-diff =x-diff) (*
     =y-diff =y-diff)))
```

```
  !bind!          =x-diff-normal (/ =x-diff =mag)
  !bind!          =y-diff-normal (/ =y-diff =mag)
  !bind!          =projected-x (+ =sx (*
     *target-projection* =x-diff-normal))
  !bind!          =projected-y (+ =sy (*
     *target-projection* =y-diff-normal))

  =visual-location>
    screen-x      =projected-x
    screen-y      =projected-y

  =goal>
    state         move-cursor
    cur-x-diff    =x-diff-normal
    cur-y-diff    =y-diff-normal
)

(P move-cursor
  =goal>
    ISA           targeting
    state         move-cursor

  =visual-location>
    ISA           visual-location
    kind          OVAL

  ?manual>
    preparation   free
==>

  +manual>
    ISA           move-cursor
    loc           =visual-location

  =goal>
    state         check-target
    target-location =visual-location
)

(P check-target
  =goal>
    ISA           targeting
    state         check-target
    target-location =vis-loc

  ?visual-location>
```

```
      buffer         empty
==>

  +visual-location>
    ISA            visual-location
    kind           OVAL
    :nearest       =vis-loc

  =goal>
    state          distinguish-target
)

(P distinguish-target-black
  =goal>
    ISA            targeting
    state          distinguish-target
    target-location =target-location

  =visual-location>
    ISA            visual-location
    kind           OVAL
    color          black

  ?manual>
    state          busy
    last-command   prepare
==>
  +temporal>
    ISA            clear

  =goal>
    state distinguish-target-black-search
)

(P distinguish-target-black-search
  =goal>
    ISA targeting
    state distinguish-target-black-search
    target-location =target-location

  ?visual-location>
    buffer empty
==>
  =goal>
    state distinguish-target
```

```
  +visual-location>
    ISA           visual-location
    kind          OVAL
    :nearest      =target-location
)

(P prepare-click
  =goal>
    ISA           targeting
    state         distinguish-target

  ?manual>
    last-command  move-cursor
    preparation   free
==>
  +manual>
    ISA           prepare
    style         punch
    hand          right
    finger        index
)

(P detect-target-color
  =goal>
    ISA           targeting
    state         distinguish-target
    target-x      =cx
    target-y      =cy
    cur-x-diff    =x-diff
    cur-y-diff    =y-diff

  =visual-location>
    ISA         visual-location
    kind        OVAL
    - color     black
    color       =color
    screen-x    =sx
    screen-y    =sy

  !bind!          =on-line (is-on-line =sx =sy =cx =cy
     =x-diff =y-diff)

  ?retrieval>
    state       free
    buffer      empty
==>
```

```
  +retrieval>
    ISA          response
    color        =color

  =goal>
    state        decide-whether-to-shoot
)

(P decide-to-shoot
  =goal>
    ISA            targeting
    state          decide-whether-to-shoot

  =retrieval>
    ISA            response
    action         shoot

  ?manual>
    last-command   prepare
    preparation    free
==>
  =goal>
    state          find-black-target

  +manual>
    ISA            execute

  +temporal>
    ISA            clear
)

(P decide-not-to-shoot
  =goal>
    ISA            targeting
    state          decide-whether-to-shoot
    target-x       =sx
    target-y       =sy
    target-location =target-location
    cur-x-diff     =x-diff
    cur-y-diff     =y-diff

  =retrieval>
    ISA            response
    action         dont-shoot
==>
```

```
  +temporal>
    ISA             clear

  =goal>
    state           find-black-target
    friend-x        =sx
    friend-y        =sy
    friend-x-diff   =x-diff
    friend-y-diff   =y-diff
    heuristic       highest
)

(P distinguish-whiff
  =goal>
    ISA             targeting
    state           distinguish-target

  ?temporal>
    buffer          empty

  ?manual>
    state           free

  =visual-location>
    ISA             visual-location
    kind            OVAL
    color           black
==>
  +temporal>
    ISA             time

  =goal>
    state distinguish-whiff-search
)

(P distinguish-whiff-search
  =goal>
    isa targeting
    state distinguish-whiff-search
    target-location =target-location

  ?visual-location>
    buffer empty
==>
  +visual-location>
    ISA             visual-location
```

```
    kind          OVAL
    :nearest      =target-location

  =goal>
    state distinguish-target
)

(P whiff-spin
  =goal>
    ISA           targeting
    state         distinguish-target

  !bind!          =whiff-wait-time *whiff-wait-time*
  =temporal>
    ISA           time
    <= ticks      =whiff-wait-time

  =visual-location>
    ISA           visual-location
    kind          OVAL
    color         black
==>
  =goal>
    state whiff-spin-search
)

(P whiff-spin-search
  =goal>
    isa targeting
    state whiff-spin-search
    target-location =target-location

  ?visual-location>
    buffer empty
==>
  +visual-location>
    ISA           visual-location
    kind          OVAL
    :nearest      =target-location

  =goal>
    state distinguish-target
)

(P move-after-whiff
  =goal>
```

```
      ISA             targeting
      state           distinguish-target

    !bind!            =whiff-wait-time *whiff-wait-time*

    =temporal>
      ISA             time
      > ticks         =whiff-wait-time

    =visual-location>
      ISA             visual-location
  ==>
    =goal>
      state           find-black-target

    +temporal>
      isa             clear
  )
)
```

BIBLIOGRAPHY

BIBLIOGRAPHY

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C. and Qin, Y. An integrated theory of the mind. Technical Report 4, Psychology Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA. ja@cmu.edu., 2004.

Anderson, J. R. and Lebiere, C. *The atomic components of thought*. Lawrence Erlbaum Associates, 1998. ISBN 0805828176.

Ashcraft, M. H. and Kirk, E. P. The relationships among working memory, math anxiety, and performance. *Journal of experimental psychology General*, 130(2):224–237, 2001.

Baker, S. L. and Kirsch, I. Cognitive mediators of pain perception and tolerance. *Journal of Personality and Social Psychology*, 61(3):504–510, 1991.

Barrouillet, P. and Fayol, M. From algorithmic computing to direct retrieval: evidence from number and alphabetic arithmetic in children and adults. *Memory cognition*, 26(2):355–368, 1998.

Bettman, J. R., Johnson, E. J. and Payne, J. W. A componential analysis of cognitive effort in choice. *Organizational Behavior and Human Decision Processes*, 45(1):111–139, 1990. ISSN 07495978. doi:10.1016/0749-5978(90)90007-V.

Bierbaum, C. R., Szabo, S. M. and Aldrich, T. B. Task analysis of the UH-60 mission

and decision rules for developing a UH-60 workload prediction model: Volume 1. Summary report. *US Army Research Institute*, 1989.

Blankenberger, S. The arithmetic tie effect is mainly encoding-based. *Cognition*, 82(1):B15–B24, 2001.

Blankenberger, S. and Vorberg, D. The single-format assumption in arithmetic fact retrieval. *Journal Of Experimental Psychology. Learning Memory And Cognition*, 23(3):721–738, 1997. ISSN 02787393. doi:10.1037/0278-7393.23.3.721.

Bonner, S. E., Hastie, R., Sprinkle, G. B. and Young, S. M. A Review of the Effects of Financial Incentives on Performance in Laboratory Tasks: Implications for Management Accounting. *Journal of Management Accounting Research*, 12(1):19–64, 2000. ISSN 10492127. doi:10.2308/jmar.2000.12.1.19.

Bonner, S. E. and Sprinkle, G. B. The effects of monetary incentives on effort and task performance: theories, evidence, and a framework for research. *Accounting, Organizations and Society*, 27(4-5):303–345, 2002. ISSN 03613682. doi:10.1016/S0361-3682(01)00052-6.

Burgoon, J. K., Kelley, D. L., Newton, D. A. and Keeley-Dyreson, M. P. The Nature of Arousal and Nonverbal Indices. *Human Communication Research*, 16(2):217–255, 1989. ISSN 0360-3989. doi:10.1111/j.1468-2958.1989.tb00210.x.

Burgoon, J. K., Poire, B. A., Beutler, L. E., Bergan, J. and Engle, D. Nonverbal behaviors as indices of arousal: Extension to the psychotherapy context. *Journal of Nonverbal Behavior*, 16(3):159–178, 1992. doi:10.1007/BF00988032.

Campbell, J. I. D. and Gunter, R. Calculation, culture, and the repeated operand effect. *Cognition*, 86(1):71–96, 2002.

Cao, S. and Liu, Y. Integrating Queueing Network and ACT-R Cognitive Architectures. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 55(1):836–840, 2011a. ISSN 1071-1813. doi:10.1177/1071181311551174.

Cao, S. and Liu, Y. Mental Workload Modeling in an Integrated Cognitive Architecture. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 55(1):2083–2087, 2011b. ISSN 1071-1813. doi:10.1177/1071181311551434.

Card, S. K., Moran, T. P. and Newell, A. The model human processor- An engineering model of human performance. *Handbook of perception and human performance*, 2(4):45–1, 1986. doi:10.1037/1076-898X.13.4.224.

Dambacher, M., Hübner, R. and Schlösser, J. Monetary incentives in speeded perceptual decision: effects of penalizing errors versus slow responses. *Frontiers in psychology*, 2(September):248, 2011. ISSN 16641078. doi:10.3389/fpsyg.2011.00248.

Derbinsky, N. and Laird, J. E. Efficiently Implementing Episodic Memory. *Case-Based Reasoning Research and Development*, 1001:403–417, 2009. doi:10.1007/978-3-642-02998-1_29.

DeStefano, D. and LeFevre, J.-A. The role of working memory in mental arithmetic. *European Journal of Cognitive Psychology*, 16(3):353–386, 2004. ISSN 09541446. doi:10.1080/09541440244000328.

Dillard, J. F. and Fisher, J. G. Compensation Schemes, Skill Level, and Task Performance: An Experimental Examination*. *Decision Sciences*, 21(1):121–137, 1990. ISSN 00117315. doi:10.1111/j.1540-5915.1990.tb00320.x.

Engström, M., Landtblom, A.-M. and Karlsson, T. Brain and effort: brain activation and effort-related working memory in healthy participants and patients with

working memory deficits. *Frontiers in human neuroscience*, 7:140, 2013. ISSN 1662-5161. doi:10.3389/fnhum.2013.00140.

Erez, M., Gopher, D. and Arzi, N. Effects of goal difficulty, self-set goals, and monetary rewards on dual task performance. *Organizational Behavior and Human Decision Processes*, 47(2):247–269, 1990.

Gawthrop, P., Lakie, M. and Loram, I. Predictive feedback control and Fitts' law. *Biological Cybernetics*, 98(3):229–238, 2008.

Groen, G. J. and Parkman, J. M. A chronometric analysis of simple addition. *Psychological Review*, 79(4):329–343, 1972. ISSN 0033295X. doi:10.1037/h0032950.

Hart, S. G. and Staveland, L. E. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Human mental workload*, 1:139–183, 1988. doi:10.1016/s0166-4115(08)62386-9.

Heathcote, D. The Role of Visuospatial Working-Memory in the Mental Addition of Multi-Digit Addends. *Cahiers De Psychologie CognitiveCurrent Psychology of Cognition*, 13(2):207–245, 1994. ISSN 02499185.

Hogarth, R. M., Gibbs, B. J., McKenzie, C. R. and Marquis, M. A. Learning from feedback: Exactingness and incentives. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17(4):734–752, 1991. doi:10.1037/0278-7393.17.4.734.

Jansma, J. M., Ramsey, N. F., De Zwart, J. A., Van Gelderen, P. and Duyn, J. H. fMRI study of effort and information processing in a working memory task. *Human Brain Mapping*, 28(5):431–440, 2007.

Kahneman, D. *Attention and Effort*, volume 88 of *Prentice-Hall Series in Experimental Psychology*. Prentice-Hall, 1973. ISBN 0130505188. doi:10.2307/1421603.

Kieras, D. A Guide to GOMS Model Usability Evaluation using NGOMSL. *Handbook of HumanComputer Interaction*, 2(313):49, 1996.

Kuhl, J. and Koch, B. Motivational determinants of motor performance: The hidden second task. *Psychological Research*, 46(1-2):143–153, 1984.

Laird, J. *The Soar Cognitive Architecture*. MIT Press, 2012. ISBN 0262122960.

Laird, J. E., Rosenbloom, P. S. and Newell, A. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 1986. ISSN 08856125. doi:10.1007/BF00116249.

LeFevre, J.-A., Shanahan, T. and DeStefano, D. The tie effect in simple arithmetic: an access-based account. *Memory cognition*, 32(6):1019–1031, 2004.

Lehman, J. F., Laird, J. and Rosenbloom, P. A Gentle Introduction to Soar, an Architecture for Human Cognition: 2006 Update. *Science*, 4(0413013):1–37, 2006.

Liu, Y. Queueing network modeling of elementary mental processes. *Psychological Review*, 103(1):116–136, 1996.

Liu, Y. Queueing network modeling of human performance of concurrent spatial and verbal tasks. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 27(2):195–207, 1997. ISSN 10834427. doi:10.1109/3468.554682.

Liu, Y. QN-ACES: Integrating Queueing Network and ACT-R, CAPS, EPIC, and Soar Architectures for Multitask Cognitive Modeling. *International Journal of*

Human-Computer Interaction, 25(6):554–581, 2009. ISSN 10447318. doi:10.1080/10447310902973182.

Liu, Y., Feyen, R. and Tsimhoni, O. Queueing Network-Model Human Processor (QN-MHP): A computational architecture for multitask performance in human-machine systems. *Engineering*, 13(1):37–70, 2006. ISSN 10730516. doi:10.1145/1143518.1143520.

Lu, S. A., Wickens, C. D., Prinet, J. C., Hutchins, S. D., Sarter, N. and Sebok, A. Supporting Interruption Management and Multimodal Interface Design: Three Meta-Analyses of Task Performance as a Function of Interrupting Task Modality. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, pages 0018720813476298–, 2013. ISSN 0018-7208. doi:10.1177/0018720813476298.

Lyon, D. R., Gunzelmann, G. and Gluck, K. A. Emulating a visuospatial memory field using ACT-R. In *Proceedings of the Sixth International Conference of Cognitive Modeling*. 2004.

McCracken, J. H. and Aldrich, T. B. Analysis of selected LHX mission functions implications for operator workload and system automation goals. *US Army Research Institute*, 1984.

Meiran, N. Modeling cognitive control in task-switching. *Psychological Research*, 63(3-4):234–249, 2000.

Meyer, D. E. and Kieras, D. E. A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, 104(1):3–65, 1997a.

Meyer, D. E. and Kieras, D. E. A computational theory of executive cognitive pro-

cesses and multiple-task performance: Part 2. Accounts of psychological refractory-period phenomena. *Psychological Review*, 104(4):749–791, 1997b. ISSN 0033295X. doi:10.1037//0033-295X.104.4.749.

Mitchell, D. K. Mental workload and ARL workload modeling tools. Technical report, Army Research Laboratory, 2000.

Mitchell, D. K. and Samms, C. Workload Warriors: Lessons Learned from a Decade of Mental Workload Prediction Using Human Performance Modeling. In *Proceedings of the Human Factors and Ergonomics Society 53rd Annual Meeting*, volume 53, pages 819–823. Human Factors and Ergonomics Society, 2009. ISSN 10711813. doi:10.1518/107118109X12524442637868.

Monsell, S. Task switching. *Trends in Cognitive Sciences*, 7(3):134–140, 2003. ISSN 13646613. doi:10.1016/S1364-6613(03)00028-7.

Nason, S. and Laird, J. E. Soar-RL: integrating reinforcement learning with Soar. *Cognitive Systems Research*, 6(1):51–59, 2005. ISSN 13890417. doi:10.1016/j.cogsys.2004.09.006.

Nikolic, M. I., Ho, C.-Y. and Sarter, N. B. Supporting Timesharing and Interruption Management Through Multimodal Information Presentation. 2001. doi:10.1177/154193120104500416.

Parasuraman, R. and Rovira, E. Workload Modeling and Workload Management : Recent Theoretical Developments. (April), 2005.

Pribram, K. H. and McGuinness, D. Arousal, activation, and effort in the control of attention. *Psychological Review*, 82(2):116–149, 1975.

Reynolds, J., Braver, T., Brown, J. and Vanderstigchel, S. Computational and neural mechanisms of task switching. *Neurocomputing*, 69(10-12):1332–1336, 2006. ISSN 09252312. doi:10.1016/j.neucom.2005.12.102.

Rubinstein, J. S., Meyer, D. E. and Evans, J. E. Executive Control of Cognitive Processes in Task Switching. *Journal of Experimental Psychology Human Perception and Performance*, 27(4):763–797, 2001. ISSN 00961523.

Salvucci, D. D. and Taatgen, N. A. Threaded cognition: An integrated theory of concurrent multitasking. *Psychological Review*, 115(1):101–130, 2008. ISSN 0033295X.

Sarter, N. Multimodal Support for Interruption Management: Models, Empirical Findings, and Design Recommendations. *Proceedings of the IEEE*, 101(9):2105–2112, 2013. ISSN 0018-9219. doi:10.1109/JPROC.2013.2245852.

Schmidt, K.-H., Kleinbeck, U. and Brockmann, W. Motivational control of motor performance by goal setting in a dual-task situation. *Psychological Research*, 46(1-2):129–141, 1984. ISSN 0340-0727. doi:10.1007/BF00308598.

Schulte, A. and Donath, D. Measuring self-adaptive UAV operators' load-shedding strategies under high workload. *Engineering Psychology and Cognitive Ergonomics*, pages 342–351, 2011.

Schumacher, E. H., Seymour, T. L., Glass, J. M., Fencsik, D. E., Lauber, E. J., Kieras, D. E. and Meyer, D. E. Virtually perfect time sharing in dual-task performance: uncorking the central cognitive bottleneck. *Psychological Science*, 12(2):101–108, 2001.

Svenson, O. Memory retrieval of answers of simple additions as reflected in response latencies. *Acta Psychologica*, 59(3):285–304, 1985.

Taatgen, N. A., Van Rijn, H. and Anderson, J. An integrated theory of prospective time interval estimation: the role of cognition, attention, and learning. *Psychological Review*, 114(3):577–598, 2007.

Thevenot, C. and Barrouillet, P. Encoding numbers: behavioral evidence for processing-specific representations. *Memory cognition*, 34(4):938–948, 2006.

Thevenot, C., Barrouillet, P. and Fayol, M. Algorithmic solution of arithmetic problems and operands-answer associations in long-term memory. *Quarterly Journal of Experimental Psychology*, 54(2):599–611, 2001.

Thevenot, C., Fanget, M. and Fayol, M. Retrieval or nonretrieval strategies in mental arithmetic? An operand recognition paradigm. *Memory cognition*, 35(6):1344–1352, 2007.

Toppen, J. T. Effect of Size and Frequency of Money Reinforcement on Human Operant (Work) Behavior. *Perceptual and Motor Skills*, 20(1):259–269, 1965. ISSN 0031-5125. doi:10.2466/pms.1965.20.1.259.

Trbovich, P. L. and LeFevre, J.-A. Phonological and visual working memory in mental addition. *Memory cognition*, 31(5):738–45, 2003. ISSN 0090502X.

Wang, Y. and Laird, J. E. Integrating Semantic Memory into a Cognitive Architecture Investigators. Technical report, Center for Cognitive Architectures, University of Michigan, Ann Arbor, 2007.

Wickens, C. D. Processing resources in attention. In Parasuraman, R. and Davies,

D. R., editors, *Varieties of Attention*, 3, pages 63–102. Academic Press, 1984. ISBN 0125449704.

Wieth, M. and Burns, B. D. Incentives improve performance on both incremental and insight problem solving. *The Quarterly Journal of Experimental Psychology (2006)*, 59(8):1378–1394, 2006.

Wu, C. and Liu, Y. Queuing Network Modeling of Driver Workload and Performance. *IEEE Transactions on Intelligent Transportation Systems*, 8(3):528–537, 2007. ISSN 15249050. doi:10.1109/TITS.2007.903443.

Wu, C. and Liu, Y. Queuing network modeling of transcription typing. *ACM Transactions on ComputerHuman Interaction TOCHI*, 15(1):1–45, 2008. ISSN 10730516. doi:10.1145/1352782.1352788.

Zhou, F., Zhao, Q., Chen, C. and Zhou, X. Mental representations of arithmetic facts : Evidence from eye movement recordings supports the preferred operand-order- specific representation hypothesis. *Experimental Psychology*, 26(February 2012):37–41, 2011. ISSN 17470226. doi:10.1080/17470218.2011.616213.