names arise from the quality of learning, i.e., the task specific constraints that professional drivers pass to the task drivers, instead of a specific
ment architecture or processes. We built this into the learning material to account since bid routes experienced professional drivers frequently use when navigating in the city of Liebherr.

Besides the exploration of the task relevant con-
straints, another aim of our simulation study was to predict various phenomena observed in expert
taxi drivers’ performance in a serial recall task. We were especially interested in the phenomena which were assumed to be based on the organisation of environmental knowledge, not on an enhanced role learning of street names. The taxi driver simulation

References

drivers. In H. A. Togas & J. A. Sheth (Eds.), The acquisition of symbolic skills (pp. 301–409).
New York: Plenum Press.

working memory. Psychological Review, 102, 211–
245.

How experts advance in representative task
demands account for the expertise effect in mem-

Kalakouzt, V. & Sarawutka, P. (in press). Taxi
drivers’ perceptual memory of street names: Memory
Capacity.

tance estimation by taxi drivers and the general
general. Journal of Experimental Psychology, 9,
233–239.

Riedman, B.B., Starks, J.J., Simon, H.A.
(1992). Simulation of expert memory using
2, 232–239.

effects in memory recall: Comments on Vicente and
503–505.

ical theory of expertise effects in memory recall.

Vicente, R. (2002). The four-cornered atten-
tion hypothesis: A reply to Ercan, Pe-

Intention superiority effect: A context-sensitivity account

Christian Le clientele (email)
Human Computer Interaction Institute Carnegie Mellon University
Pittsburgh, PA 15213 USA

Frank J. Lee (email)
Department of Psychology Carnegie Mellon University
Pittsburgh, PA 15213 USA

Abstract

Intention superiority effect (Goldclue & Kollar, 1995; Marsh, Hicks, & Bink, 1998) is the finding that the time to retrieve memory items related to completed or partially completed intentions is faster than for those with no associated intentions. However, this relationship reverses when the intended tasks are completed (Marsh, Hicks, & Bink, 1999). Thus, the times to retrieve memory items related to completed intentions are slower than for those with no associated intentions. In this paper, we present a computational account of the intention superiority effect using the ACT-R (Anderson
& Lebiere, 1998) cognitive architecture. Our modeling approach is based on the idea that uncompleted or partially completed intentions are available in context and guide trial-by-trial memory items with relevant context associated with them. We describe an ACT-R model that is able to reproduce all of the effects reported in Marsh, Hicks, and Bink (1998).

Keywords: Perspective memory, Intention superiority effect, ACT-R

Introduction

Perspective memory has recently been receiving a lot of attention among psychologists (Brandimonte, Emslie, 
& McDaniel, 1998). The interest in perspective memory reflects a trend in psychology to investigate "in-the-world" phenomena. For the ACT-R theory (Anderson & Lebiere, 1998), the importance of perspective memory research is new. First, as a unified theory of cognition, especially perspective memory, the ACT-R theory must be able to account for the results from this body of research. Second, as the ACT-R theory is pushed towards more complex and dynamic tasks, an account of perspective memory working in dynamic task environments will be crucial, because it is central to planning and goal maintenance.

To begin our task of understanding perspective memory from the ACT-R theoretical framework, we decided to focus on two phenomena in perspective memory called the intention superiority effect (Goldclue & Kollar, 1995; Marsh, Hicks, & Bink, 1998).
Goschke & Kuhl, 1993). The Zeigarnik effect is the finding that people’s access to the memory of the task after it is completed is poorer compared to their access to the memory of the task while they were performing it. As Marsh et al. noted, Zeigarnik effect seemed very close to the proactive memory effect, and hence they decided to investigate using the same experimental paradigm. In Figure 1d, we report their data from Experiment 4. As can be seen, the results reported those from Experiment 3 and added support for the Zeigarnik effect. Namely, people were quicker to access the memory items related to the prospective script compared to the neutral script during the execution of the intended task, but their access to the memory items related to the prospective script were slower compared to the neutral script after they completed the intended task. This would seem to suggest that the same mechanisms underlie both phenomena.

In the next section, we describe an ACT-R model of the four experiments that we have reviewed above from Marsh et al. (1998).

Model

Symbolic Level

At the symbolic level, the ACT-R model of intention enhancement effect, or more specifically of the Lexical Decision Task, is straightforward. There is only one type of declarative knowledge, contained in chunks of type lexemes. Those chunks contain three slots: word, which holds a word, spelling, which holds its spelling, and context, which holds the context in which this word occurred. The goal is to perform the LDT in the context of a sentence. When a goal is completed, it becomes a memory chunk, or receives an existing one if an identical chunk already exists in long-term memory. Thus past goals then perform lexical decision tasks become long-term memory structures used in performing future ones.

Since Marsh et al. found no significant differences between the prospective and the neutral scripts in the observe condition in their first two experiments, they decided to focus on the perform condition in Experiments 3 and 4. In Experiment 3, the participants were told to perform the prospective script in both blocks of the experiment. In one of the blocks of the experiment, the LDTs were collected after they performed the script, and in the other, the LDTs were collected after they performed the script. Figure 1c, we report Marsh et al.’s data from that Experiment 3. As can be seen, they replicated the basic results from their previous two experiments in a within-subjects design. Namely, people were quicker to access the memory items of the prospective script compared to those of the neutral script before they performed the intended task, but after they completed the intended task, they were slower to access the memory items of the prospective script compared to those of the neutral script.

In Experiment 4, Marsh et al.Basically followed the procedure outlined in the previous three experiments, but in one of the blocks of this experiment, they interrupted the participants while they were performing the intended task and gave the LDTs, and in the other block, the LDTs were given after they completed the intended task. The main idea they were testing was the Zeigarnik effect (Roskenfeld, 1964).

It is important to note that participants were told which script was the prospective script only after they memorized both scripts to criterion. This prevented them from privileged access (e.g., through additional information in the prospective script or the neutral script during the initial study phase). After they were told which script was the prospective script, they were immediately given the LDTs.

In Figure 1a, we report Marsh et al.’s data from Experiment 1. As can be seen, participants were faster in accessing the memory items related to the prospective scripts compared to the access to the items related to the neutral script. Nevertheless, this difference did not exist for the prospective script that people were told to observe.

Since Marsh et al. found no significant differences between the prospective and the neutral scripts in the observe condition in their first two experiments, they decided to focus on the perform condition in Experiments 3 and 4. In Experiment 3, the participants were told to perform the prospective script in both blocks of the experiment. In one of the blocks of the experiment, the LDTs were collected before they performed the script, and in the other, the LDTs were collected after they performed the script. As can be seen, they replicated the basic results from their previous two experiments in a within-subjects design. Namely, people were quicker to access the memory items of the prospective script compared to those of the neutral script before they performed the intended task, but after they completed the intended task, they were slower to access the memory items of the prospective script compared to those of the neutral script.

In Experiment 4, Marsh et al. Basically followed the procedure outlined in the previous three experiments, but in one of the blocks of this experiment, they interrupted the participants while they were performing the intended task and gave the LDTs, and in the other block, the LDTs were given after they completed the intended task. The main idea they were testing was the Zeigarnik effect (Roskenfeld, 1964).

Table 1: Production rules for Lexical Decision Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Production Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>IF the goal is to perform lexical access on spelling and there is a chunk mapping spelling to word THEN note in the goal that the desired word is word.</td>
</tr>
<tr>
<td>None</td>
<td>IF the goal is to perform lexical access on spelling THEN note that no word can be found.</td>
</tr>
<tr>
<td>Output</td>
<td>IF the goal is to perform lexical access and the word is word THEN output word and focus on a new goal.</td>
</tr>
</tbody>
</table>

Procedural knowledge consists of three production rules. The most important production, map, implements lexical access. Given a word’s spelling that was entered from the environment and is present in the current goal, map retrieves from declarative memory the lexicon chunks associating that spelling to a word and adds the word to the goal. If the retrieval fails, then the production, none, notes in the goal that no word can be found associated to that spelling. After either of these two productions fails, the production output, outputs the word then focuses on a new goal. The English form of these three production rules is given in Table 1. As is the case for the declarative knowledge, these productions are quite simple and are potentially learnable, an important constraint on any model.

Subsymbolic Level

While at the symbolic level the model is appealing simple and straightforward, it wouldn’t generate the prospective memory effects described previously. The symbolic level of productions and chunks merely provides the structure of the model on which the statistical learning mechanisms of the ACT-R architecture operate to tune the performance to the structure of the environment by determining the optimal subsymbolic parameters that control the availability of symbolic structures. The probability and time to retrieve a chunk from declarative memory is a function of its activation, which is given by the activation equation:

\[ A(t) = B + \sum_{i} W_i \cdot S_i \]
**T** = \( F \cdot e^{-\lambda} \)  
**Lawrence Equation**

\( T \) is the latency to retrieve a chunk, and \( F \) is a time scaling factor. Thus, the higher the activation of a chunk, the faster its retrieval latency, and vice versa.

**Assumptions and Results**

The basic assumption of this model lies in the composition of the original context, which determines the intensity of the reactivation of activation. As previously described, in addition to the essential components of the lexical decision task, namely the scrutiny and the word to be accessed, the goal also includes a slot that encodes the correct context or task to be accomplished. Because any context is known to the task to be strongly predictive of the words that need to be accessed, as it is the case here, each word that is associated with a limited number of words, including the target chunk as a source of activation is a reasonable assumption in trying to measure the activation of the chunks to be reactivated.

This provides a useful inhibition mechanism. The chunks that are retrieved for a given task will be more active when the task is over, because the base-level learning mechanism has boosted their activation to reflect their recent use. To prevent these chunks from interfering on the following task because of this temporal bias in activation, changing the context to a new task not only boosts the activation of the words most likely to be encountered in that task, but also lowers the activation of the words that are not relevant to that task, including the words that are temporarily more active due to retrieval in the previous task.

By boosting performance through the strategic use of base architectural learning mechanisms, this assumption is therefore compatible with the empiric of the functional analysis underpinning the architecture (Anderson, 1990). Including the goal in an additional component that generally reflects that different contexts are similar to the key assumption of an ACT-R model of sequence learning (LeBiere & Walklei, 1998, 2001). In this model, the additional goal component is the previous visit in the sequence, but it serves the same purpose of providing a discriminating source of activation for different memory items, i.e., the current stimulus. This assumption is also compatible with the view of the current goal as ACT-R's working memory (Lovett, Forrer & LeBiere, 1999) as it allows a natural place to keep the task(s) to be completed active as a reminder of their impending necessity.

The one thing left to specify is which context is active in the goal given the various experimental conditions. The general rule is that if a task is expected to be performed in the near future (and no other pressing one is currently being performed) then the context is set to that task to facilitate the retrieval of related information. When a task has been performed, the context is changed to some other task, even if it is not expected to be performed soon, to prevent the information associated with the task that was just performed from generating excessive interference. The model worked as follows. Both perspective and neutral scripts were first studied. This means setting the goal context in the title of the script and performing lexical access (through the same procedure used in the lexical decision task) on all the words present in the script (LD for each script). For each word, this typically means firing the map and output productions. The retrieval of the lexicon chunk for the word in the map production led to the measuring of the strength of association between the current context and that lexicon chunk. The rest of the operations would vary with each experimental condition. None of the initial parameter were optimized to fit the data. The important aspect of the model is how simply it can capture the effects in the data, but not the maximization of a quantitative measure of fit.

In Experiment 1, participants engaged in the Lexical Decision Task (LDT) before observing or performing the scripts, but after having been instructed which script they would have to observe (Observe condition) or perform (Perform condition). In the Perform condition, the context was set to the script to perform, because subjects would have to actively generate the script. In the less demanding Observe condition, the context was randomly set to either scripts with equal probability, on the assumption that subjects did not have to set the right context because they would merely have to observe the experimenters perform the script, which provided enough spreading activation without needing to make the script itself a source of spreading activation. This reflects the fact that maintaining a concept in the goal exacts some cost, including the additional conditions presented to the subject. In Experiment 2, the context could be left empty or set to another task other than those of the experiments, which would yield comparable results. Figure 2a presents the model results in terms of the average latency to perform the task in each condition. Comparing it to Figure 1a, all the significant effects in the subject data are reproduced. In the Observe condition, neutral and perspective scripts produce similar latencies because the context is equally likely to be set to one or the other, and the experiment condition is set to neutral. In the Perform condition, the perspective script is recalled faster than the neutral condition because the context is set to that script, leading to the strengths of association to an activation boost to lexical items present in the scripts. Results show that it is not only that the additional rehearsed scripts in the perspective scripts have the highest strength between those words from the perspective scripts. Since half the time the context in the LDT is the neutral script, the advantage of the rehearsal is learned and access to lexical items associated with the perspective script is only slightly faster. In the Perform condition, the neutral script has much lower latency because the context has always been set to that script, whereas the perspective script, despite its additional rehearsal, has higher latency because of constraints inhibition from the neutral script.

In Experiment 2, the only the Perform condition was used, with the LDT administered either before or after the performance of the perspective script. The setting of the context was the same as in the previous experiment, namely the context was set to the perspective script before (and during) performances and to the neutral script afterwards. Figure 2c displays the results, which are consistent with those of the previous experiments, and can be compared to Figure 1c. When the LDT is given before performance, latency is lower for the perspective scripts because the context is set in its favor. When the LDT is given after performance, latency is lower for the neutral scripts, because the context is then set in its favor and higher for the perspective scripts. The overall latency is however lower than before the additional rehearsal.

In Experiment 4, the procedure was similar except that the performance phase was interrupted halfway through and the LDT task administered both during the interruption and after the performance phase had been completed. Again, the context was set according to the usual rule, meaning to the perspective scripts during the performance phase (including the interruption) and to the neutral script afterwards. Figure 2d displays the model results, which again reproduce the effects...
of the subject data given in Figure 1d. During the interrupt domain, words associated with the interrupt script benefited both from the (potential) rehearsal and from the scenes being in the context. After the performance phase, the search of the context to the neural script then favors words associated with that script.

**Discussion**

Geoffray and Kuhl (1993) and Mamer, Hicks, and Bink (1998) associated these findings in terms of the ACT* cognitive theory (Anderson, 1983), a predecessor of the ACT-R theory, and found that results consistently conformed with the theory. They suggested that intervals were represented as goal nodes, which confirmed them additional activation, which was quickly dissipated when the goal was popped off the stack and replaced by another. Mamer, Hicks, and Bink (1998) moreover suggested that after completing a task people naturally direct their attention toward making a decision of which task to complete next, providing a rationale for our use of switching the context in the neural task after completing the respective task. They also noted the inhibitory nature of such a switch, which we also observed. While many results have changed between ACT* and ACT-R, crucial changes in the current case is the change in activation as a result of goal shifting or in switching activation order (rather than a decay in goal activation) the basic account of the data in terms of the ACT theory remains valid.

Another advantage of the model is its compatibility with existing models of language (Anderson, Busch, & Reder, in press; Lehman, 1993). As such, this model of the lexical decision task provides a bridge between the written form of words, i.e., their spelling, and their internal representations, in terms of symbols. An important model for the model could quickly be written for the mapping between auditory input and of various component, i.e., phonemes, in the word themselves. In either case, after the mapping between two representations to internal representations are performed words can be manipulated appropriately to generate meaningful words, providing a very desirable abstraction in the form of symbolic characters. Such abstraction might be one of the fundamental purposes of language.

**Conclusion**

The main contribution of this paper is to present a simple yet precise model of the language processing effect. The model brings on the fundamental assumption that a task such as or will be accomplished in the near future is kept in the goal as a source of activation, leading to faster access to related lexical items and inhibition of items related to other, competing contexts. However, once the task is completed, it is removed from the goal and attention is shifted to a different context, thereby providing inhibition of the past completed task. Additional empirical and modeling work is clearly needed to determine the limits and circumstances of context maintenance. But the fact that such a simple model can provide a unique account of this data, as well as the surprising phonetic change is indicative of the power of cognitive modeling to illuminate empirical data.

**Acknowledgments**

The research reported in this study was supported by the Office of Naval Research, Cognitive Science Program, under Contract Number N00014-95-10231. All correspondences should be addressed to Christine Lehman.

**References**


**Abstract**

Unified modeling approaches have come to play an important role in both theoretical and applied modeling of cognitive processes, most notably natural language. Attempts to model such processes using neural networks have met with some success, but have faced serious hurdles caused by the limitations of standard connectionist modeling. As a contribution to this field, this paper presents recent work in infinite RAAM (RAAM), a new connectionist-unification model. Based on a focus of recurrent neural networks with functional geometry, RAAM allows us to understand the behavior of these networks as dynamical systems. Using a logical programming language as our modeling domain, we show how this abstract formalism supports unified models of the problems faced by typical connectionist models, supporting unification over arbitrarily large sets of recursive expressions. We conclude that RAAM can provide a principled computational substrate for unification in a variety of computational modeling domains.

**Language and Connectionism: Three Approaches**

Language, in a cognitive sense, can be held to include natural language and the "language of thought" (Fodor 1973), as well as symbolic programming languages developed in situ such as, like PROLOG and Planner. Approaches to build connectionist models of such systems have generally followed one of three approaches. The first of these, exemplified by (Rumelhart and McClelland 1986), dispenses entirely with traditional representations (data structure and rules algorithms on these structures), in favor of letting the network "learn" the patterns in the data being modeled, via the well-known back-propagation algorithm (Rumelhart, Hinton, and Williams 1986) or a similar training method. This approach becomes the subject of a heated criticism from members of the tradition prior to symbolic and rule algorithms. The second computational approach goes beyond the rules-and-representations view and deals directly with the heart of the problem by using a declarative model, thus allowing for a recurrent neural network to perform all the operations of a Turing machine, or more (Siegelmann 1993). Such systems may hold a good deal of abstracted mental behavior, but they do not address the degree to which a particular computational paradigm (connectionism) is suited to a particular real-world task (language). They are therefore not of much use in arguing for or against the merits of connectionism as a model of any particular domain of interest (Mitchell 2001). Any number of techniques have since taken up the Turing and provide a picture of the cognitive and the neural network called an infinite RAAM corresponds directly to one such process, unification, thereby supporting a systematic, computational model of linguistic structure.

**Unification**

Unification, an algorithm popularized by Robinson (1965) as a basis for automated theorem-paving, has come to play a central role in both computer science and cognitive science. In computer science, unification is at the core of logical and computational languages like Prolog (Clocksin and Mellish 1994); in cognitive science, it is the foundation of a number of "category-based" approaches to the analysis of natural language ('Shieber 1984). The basic unification algorithm can be found in many introductory AI textbooks (e.g., Rich and Knight 1991 p. 152), and can be summarized recursively as follows: (1) A variable can be unified with a literal. (2) Two literals can be unified if their literal predicate symbols are the same and their arguments can be unified.

If, for example, we have a Prolog database containing the assertions "na2(eve\(\theta\))", meaning "a1(eve\(\theta\))", and we perform the query query(male(eve\(\theta\))). Asking "Who is male?" the unification algorithm will first attempt to unify trying to unify m\(\lambda\)ale(eve\(\theta\)) with male(eve\(\theta\)), and will succeed in matching on the predicate symbol male\(\lambda\)e by rule (2). The algorithm will then recur, attempting to unify the variable with the same literal (e.g., L\(\lambda\)ight\(\theta\)) and will fail, but will success in matching on the predicate symbol m\(\lambda\)ale\(\theta\)e by rule (2). The algorithm will then recur, attempting to unify the variable with the same literal (e.g., L\(\lambda\)ight\(\theta\)) and will fail, but will success in matching on the predicate symbol m\(\lambda\)ale\(\theta\)e by rule (2).