1999. *Kognitionswissenschaft* [Journal of the German Cognitive Science Society] Special issue on cognitive modelling and cognitive architectures, D. Wallach & H. A. Simon (eds.). 20-29.

Knowledge Level Learning and the Power Law: A Soar Model of Skill Acquisition in Scheduling

Josef Nerb, Frank E. Ritter, Josef F. Krems
Department of Psychology, University of Freiburg, Niemenstr. 10, D- 79085 Freiburg, Germany
School of Psychology, University of Nottingham, Nottingham NG7 2RD UK
Department of Psychology, University of Chemnitz, D-09107 Chemnitz, Germany
(nerb@psychologie.uni-freiburg.de, frank.ritter@nottingham.ac.uk, josef.krems@phil.tu-chemnitz.de)

Summary. The model presented here gradually learns how to perform a job-shop scheduling task. It uses Soar's chunking mechanism to acquire episodic memories about the order of jobs to schedule. The model was based on many qualitative (e.g., transfer effects) and quantitative (e.g., solution time) regularities found in previously collected data. The model was tested with new data where scheduling tasks were given to the model and to 14 subjects. The model generally fit these data with the restrictions that the model performs the task (in simulated time) faster than subjects, and its performance improves somewhat more quickly than subjects. The model provides an explanation of the noise typically found in problem solving times—it is the result of learning actual pieces of knowledge that transfer more or less to new situations but rarely an average amount. Only when the data are averaged (i.e., over subjects) does the smooth power law appear. This mechanism demonstrates how symbolic models can exhibit a gradual change in behavior and how the apparent acquisition of general procedures can be performed without resort to explicit declarative rule generation. We suggest that this may represent a type of implicit learning.

1. Introduction.

Soar is a candidate unified theory of cognition (Newell, 1990). There are several aspects of the architecture that particularly influence and support learning. These can be illustrated with a model, that performs a job-shop scheduling task (i.e., a task of optimally scheduling work on different machines). The model matches data showing that symbolic learning (in particular, production learning using the chunking mechanism in Soar) can account for power-law effects, and that learning on the knowledge level within a symbolic architecture that matches human performance does not have to be represented explicitly. Instead, the power law of learning can arise from individual learning events when performance is aggregated across trials or subjects. Whereas similar claims are implicit in Newell (1990), they are not examined in as much detail.

VanLehn (1991, p. 38) differentiates between learning events (where changes occur in procedural knowledge) and rule acquisition events (where changes occur on the knowledge level). Explicit learning mechanisms are easier to understand and model, but understanding how acquisition of rules on the knowledge level can occur slowly is likely to be important as well. We will explore here how acquiring partial episodic rules can slowly approximate rule-based procedural behavior, that is, learning events that lead gradually to rule acquisition. In this paper, chunking is the basis for arguing that aspects of human learning often characterized as subsymbolic or numeric can be represented symbolically and thereby provide detailed process accounts for phenomena like the decision to seek advice. We will cover in turn, Soar, the task and initial regularities, the model, and testing the model with new data.

1.1 Overview of Soar. Soar is an architecture that provides a small set of mechanisms for simulating and explaining a broad range of phenomena as different as perception, reasoning, decision making, memories, and learning. It is this restriction on one hand side and the breath of intended—but only partially yet realized—applications that renders Soar as an candidate unified theory of cognition (Newell, 1990). Unified does not mean that all behavior must be expressed by a single mechanism (although Soar uses relatively few), but that the mechanisms must work together to cover the range of behavior.

There are extensive explanations and introductions to the architecture (Lehman, Laird, & Rosenbloom, 1996; Norman, 1991), with some available online (Baxter & Ritter, 1998; Ritter & Young, 1998), so we will only briefly review Soar, as shown in Figure 1.

Soar is best described at two levels, the symbol level and the problem space level. Behavior on the symbol level consists of applying (matching) long-term knowledge represented as production rules to the current state in working memory (including the context stack). This level implements the problem space level.

The problem space level represents behavior as search through and in problem spaces of states using operators.

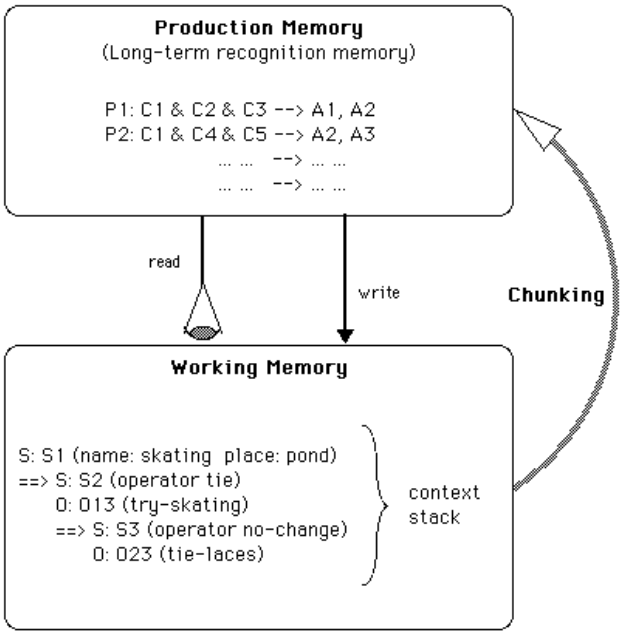

Fig. 1: The main processes in Soar.

(In previous versions of Soar, problem spaces were explicitly represented, in version 7 they can be implicitly represented.) Operators are created and implemented using the symbol level.

In routine behavior, processing proceeds by a repeated cycle of operator applications. When there is a lack of knowledge about how to proceed, an impasse is declared. An impasse can be caused, for example, by the absence of operators to apply, by an operator that does not know to make changes to the state (no-change), or by tied operators (i.e., if there are more than just one operator that is applicable at a time). In an impasse, typically further knowledge can be applied about how to resolve the problem. States S2 and S3 in Figure 1 are impasse states. As problem solving on one impasse may lead to further impasses, a stack of impasses often results. An impasse defines a context for problem solving, that is, what knowledge is required for progress based on the architecture of choosing operators and applying them to states. For S2, the impasse is that two operators cannot be differentiated, and knowledge is required to differentiate them or to accept them as equivalent. For S3, is it that the operator could not be directly implemented; knowledge is required to implement it. The stack changes as impasses get resolved through problem solving or when new impasses are added.

When knowledge about how to resolve an impasse becomes available from problem solving within the context of the impasse, a chunk (a newly acquired production rule) is created. This acquired rule will contain as its condition part the knowledge of the higher context that has been used for resolving the impasse and as its action part the changes that happened during resolving the impasse. In addition, the conditions and actions will be generalized by replacing constants with variables. Thus, the chunk will recognize the same and similar situation in future problem solving and—by applying the chunk's actions—problem solving will proceed without an impasse.

Figure 2 provides a simple example of how learned rules (chunks) are acquired. In this example, the operator

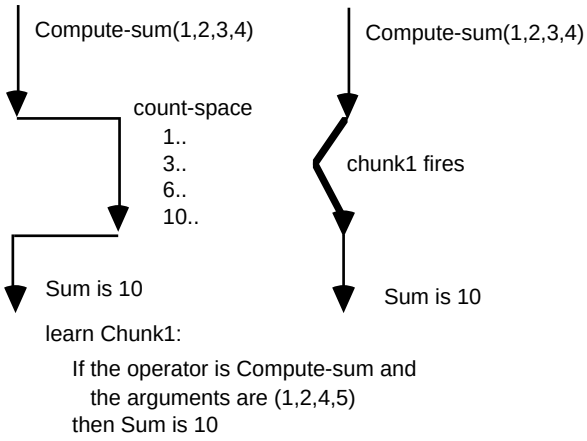

Fig. 2: How chunking encodes a new rule. Adapted from Howes and Young (1997).

Compute-sum has been selected, but immediate knowledge is not available to implement it by providing the answer. An operator no-change impasse is declared by the architecture, because Compute-sum made no change to the state. Based on this impasse and its type, knowledge about counting in a counting problem space is proposed, and counting is performed. A value (the number 10) can be returned as the result of Compute-sum. When the result is returned, a new rule, Chunk1, is created. It will have as its conditions the information from the higher level context used to solve the impasse. In this case, it will include the name of the operator and its arguments. The action of Chunk1 will be the results passed back to the higher context, in this case, that the answer was 10. In the future, when Compute-sum is selected with the same arguments, Chunk1 will match, providing the result and an impasse will not occur.

1.2 Forms of learning in Soar. Soar's chunking mechanism is well-constrained by psychological data and theory and has proven predictive in a number of cognitive applications. These qualities make it an important and general tool for analyzing learning processes. There have been many higher-level forms of learning implemented using it. Work with this chunking mechanism began with the Xaps production system (Rosenbloom & Newell, 1987) and continued in Soar with simulations of the Seibel task and the R1 Vax configuration task (Newell, 1990). The type and meaning of the impasses differentiate the forms of learning. If the impasses are on operators to retrieve an association from memory, the results look like declarative learning. If the impasses are on a set of tied operators, the results look like search control knowledge. If the impasse is structured like in Figure 2, the result is speedup due to caching.

Many of these learning approaches have been part of AI or cognitive models that have not been tested closely against human data, with the exception that the models exhibit behavior that humans generally exhibit as well, such as learning through reflection. These include models that learn through instructions (Huffman & Laird, 1995), reflection (Bass, Baxter, & Ritter, 1995; Nielsen & Kirsner, 1994), analogy (Rieman, Lewis, Young, & Polson, 1994), and abduction (Johnson, et al., 1991; John-

son, Krems, & Amra, 1994). Further examples are available (Baxter & Ritter, 1998; Rosenbloom, Laird, & Newell, 1992).

There are, however, several learning models in Soar that have had their predictions compared with human learning data, starting with some of the earliest work with Soar (Rosenbloom & Newell, 1987). Many of these models are in the field of human computer interaction (HCI). Altmann and John (in press) have looked at learning and interaction; John, Vera, and Newell (1994) have looked at reflecting after interacting. Numerous models of computer users that learn have been developed. One model illustrates how mappings between tasks and actions are acquired (Howes & Young, 1996). There are also models that learn through external scanning and internal comprehension (Rieman, Young, & Howes, 1996), and through exploration to recognize states and information near the target (Howes, 1994). Diag (Ritter & Bibby, 1997) learns which interface objects to examine and how to implement its internal problem solving more efficiently.

There are also models outside of HCI that have been compared with data. Able-Soar, Jr. (Ritter, Jones, & Baxter, 1999) simulates the novice to expert transition in solving physics problems using Larkin's (1981) transition mechanism. Driver-Soar (Aasman & Michon, 1992) drives a simulated car. It learns to compile plans for activities and how to control the car more accurately. SCA (Miller & Laird, 1996) learns concepts by starting with very general classification rules and learns more specific rules to classify novel stimuli. Chong and Laird (1997) have created a series of models that become better at solving a dual task using a model of interaction called Epic (Meyer & Kieras, 1997). There are also models of abduction (Johnson et al., 1994) and concept acquisition in development (Simon & Klahr, 1994).

There exist computational models of learning besides Soar (e.g., case-based or comprehension-based systems) where models learn gradually using a strengthening approach and that have been compared with data (e.g., Anderson & Lebiere, 1998; Kitajima, Soto, & Polson, 1998; Rumelhart et al., 1986; Siegler & Shrager, 1984; Weber, 1996). Many of these models are successful quantitatively but can be opaque qualitatively and do not always provide a detailed analysis of the associative knowledge and controlled processing involved in learning. For a comprehensive overview and comparison of different learning approaches including an example model that was refined and extended incrementally by successive empirical tests see Schmalhofer (1998).

Many of these models compared with data simulate learning that could be categorized as explicit or as complete learning, that is, learning that is based on a correct and complete domain theory or which work on a subsymbolic (i.e., numeric) level to match behavior, and where the learner knows that it is learning, knows why competence and performance have improved, where performance can become correct, or where several of these features are true. Most such models also have either not predicted reaction times or had them tested, although for numerous counterexamples, see Anderson and Lebiere (1998).

The model we propose deals with procedural learning in a task in which human subjects improve incompletely and implicitly, that is, they improve with a less explicit learning mechanism without knowing why and how; they use an incomplete domain theory; and they do not learn to do the task perfectly. The model should perform the task and improve in performance both qualitatively and quantitatively, and in a way similar to human subjects using a partial domain theory. We will confirm this by comparing it with human behavior acquired explicitly to test it.

First, however, we will describe the task and prior empirical regularities. These regularities will serve as initial empirical constraints on the model. The validity of the model will be scrutinized in a further empirical study using, among other regularities, reaction times. Utilizing model predictions, we will examine subjects' performance, including learning in this domain in a principled, detailed way.

2. Skill acquisition in scheduling.

From a psychological point of view, planning can be considered a problem-solving activity where an ordered sequence of executable actions has to be constructed in a prospective manner. In a more formal sense, this means specifying a sequence of operators with well-defined conditions and consequences that transform a given state into a goal state. For interesting problems, the entire problem space cannot be searched, and heuristics must be used to guide the search.

Scheduling problems are a specific, important subset of planning. Here the task of the problem solver is to find an optimal schedule based on the given constraints (e.g., minimal processing time). Factory scheduling (so-called job-shop scheduling) is a further subset of scheduling tasks, namely, to find the optimal ordering of activities on machines in a factory.

Job-shop scheduling has direct, practical importance. Over the last two decades algorithms have been derived that produce optimal (or near optimal) solutions for scheduling tasks using operations research techniques (e.g., Graves, 1981) as well as non-learning AI techniques (e.g., Fox, Sadeh, & Baykan, 1989). Other AI based approaches have used the general learning mechanisms in PRODIGY (Minton et al., 1989) and Soar (Akyurek, 1992; Prietula, Hsu, Steier, & Newell, 1993). These systems rely on the assumption that general methods for efficient problem solving can be discovered by applying large amounts of domain knowledge with a learning mechanism.

In psychology, on the other hand, little is known about how scheduling is performed as a problem solving activity and about the acquisition of scheduling skills. For a counterexample and earlier call to arms, see Sanderson (1989).

2.1 The job-shop scheduling task. The task—for the subjects as well as for the computational model—is to schedule five actions (jobs) optimally, as a scheduler or dispatcher of a small factory. Figure 3 illustrates our task. Thus, one out of 5! possible sequences of actions has to be suggested. Jobs had to be scheduled to run in order through two machines (A and B). Each job had to be run in a fixed order, first A and then B, requiring different processing time on each machine for each job.

Sets of five jobs with randomly created processing times (ranging from 10 to 99) were given to the subjects on a computer display. Subjects tried to find the order of

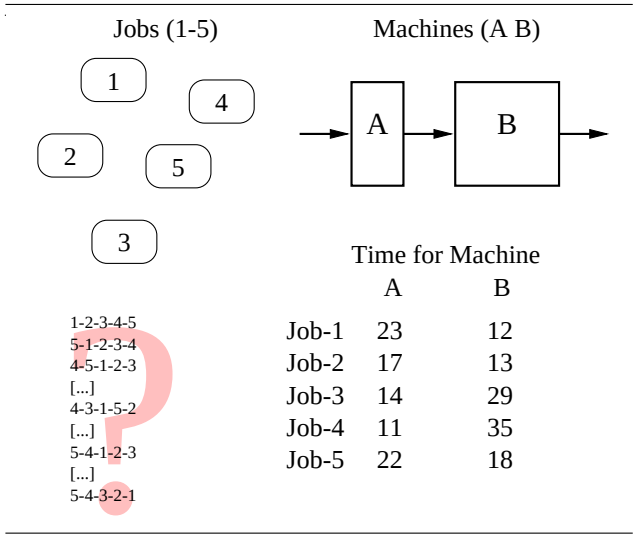

Fig. 3: The task solved by subjects and the model: Finding the sequence of jobs to minimize processing time?

jobs that produced the minimal total processing time, determining which out of the five jobs should be run first, which second, and so on. They were not otherwise instructed in how to perform this task.

For this kind of scheduling task an algorithm to find the optimal solution is available (Johnson, 1954). The general principle requires comparing the processing times of the remaining jobs and finding the job with the shortest time on one of the machines. If this job is on machine A than it has to be run in the next available time in the schedule, if it is on B, then in the last remaining time in the schedule. This principle is applied until all of the jobs are scheduled. Suboptimal sequences result if only parts of the general principle are used, for example, if only the demands on Machine A are used for ordering the jobs.

For the sets of jobs in Figure 3, this algorithm would first choose Job 4, with the shortest job on Machine A, to put in the first slot in the schedule. Job 1 has the next shortest time (12) on a machine. This is on Machine B, so it scheduled in the last available slot in the schedule, at this point the last slot. Job 2 has the next shortest time (13). It is on Machine B, so it goes in the latest available slot, or fourth. The remaining two jobs are scheduled in the same way.

This task of modest complexity is a good task with which to study learning because (a) it is simple enough to assess the value of each trial's solution by comparing it to the actual optimal solution, but (b) the task is hard enough to be a genuine problem for subjects, who have to solve it deliberately, and (c) to solve the task without errors appears to require discovering and applying general principles.

2.2 What is learned in this task? In the Soar architecture, learning to solve scheduling tasks, like learning in general, requires the acquisition as well as the storage of rules in memory. In this task, acquisition is discovering the general rule or inferring at least useful scheduling heuristic rules while performing the task. This is what Prodigy models do when they solve job-shop tasks.

If no rule on how to schedule jobs is available and the problem solver progresses through blind search and no learning occurs, on average no improvement will occur across trials. Only if the subject generates internal hypotheses about the schedule ordering rules, and if feedback about the correctness of these assumptions is available, will the subject be able to discover efficient scheduling rules. And then, only if a discovered rule is stored in memory will the improvement be applied in later trials. As in impasse-driven learning theories (VanLehn, 1988), it is assumed that rule acquisition particularly takes place when subjects face a situation in which their background knowledge is not sufficient to solve the problem immediately.

Of course, as in other domains, learning in scheduling tasks depends on the amount of practice and it is highly situated. An essential situational factor, which facilitates or inhibits the acquisition of rules and thus the progress in learning, is the interaction of the problem solver with the environment. In this task, interaction can provide feedback about the quality of the solution and therefore about the efficiency of their heuristics.

In previous experiments investigating this task, Krems and Nerb (1992) had 38 subjects each generate 100 schedules. Although the main focus of their work was to investigate the effect of different kinds of feedback on learning, they also reported some general regularities. An initial set of empirical results that can be used to constrain the design of process models of scheduling skill acquisition is shown in Table 1.

3. Sched-Soar.

Sched-Soar is a computational model of skill acquisition in scheduling tasks. The architectural component—Soar—is strictly separated from the task specific knowledge—in this case, knowledge about maths, scheduling, and memorizing strategies. The model predicts human reaction times, learning, and transfer on this task. The rules it learns while solving the task are used to examine and explain transfer between problems. These learned rules, which are created about once every simulated second (10 model cycles) and a learning mechanism based on partial knowledge, start to explain how rule-based behavior can arise from apparently more chaotic behavior. The model is available at http://www.psychologie.uni-freiburg.de/signatures/nerb/sched-soar/

In addition to the empirical constraints, we include the following general constraints that are consistent with or based on the Soar architecture. (a) The task is described and represented in terms of problem spaces, goals and operators, as a Problem Space Computational Model (Newell, 1990). All knowledge is implemented as productions. Soar's bottom-up chunking mechanism is used for learning, which means that chunks were built only over terminal subgoals. This has been proposed as basic characteristic of human learning (Newell, 1990, p. 317).

(b) An initial knowledge set about scheduling tasks is provided as part of long-term knowledge (e.g., to optimize a sequence of actions it is first necessary to analyze the resource demands of every action). Also, basic algebraic knowledge is included, such as ordinal relations between numbers. Together this amounts to 401 production rules implementing eight problem spaces.

Table 1. Important regularities of behavior on this task taken from Krems and Nerb (1992).

(a) ***Total processing time.*** The task takes 22.3 s, on average, for a novice (minimum: 16 s, maximum: 26 s).

(b) ***General speed-up effect.*** On average, the processing time for scheduling a set of jobs decreased 22% from the first ten trials to the last ten.

(c) ***Improvement of solutions.*** The difference between the subject's answers and the optimum answers decreased more than 50% over the 100 trials.

(d) ***Suboptimal behavior.*** Even after 100 trials the solutions were not perfect.

(e) ***Algorithm not learned.*** None of the subjects detected the optimal scheduling rules (i.e., nobody could give a verbal description of the underlying principle when asked after the trials), but most subjects came up with some simple ideas and heuristics (i.e., they had some partial knowledge of the optimal algorithm).

(c) The model is feedback-driven. If the available internal knowledge is not sufficient to choose the next action to schedule, the supervisor is asked for advice. These are situations in which a human problem solver would have to do the same or to guess. So when the type of impasse is the inability to select between tied operators proposing jobs to schedule, there is knowledge that suggests asking for outside advice to resolve the impasse.

3.1 Processing steps. Sched-Soar begins with an initial state containing the five jobs to be scheduled and a goal state to have them well scheduled, but without knowledge of the actual minimum total processing time. Its initial knowledge leads to these main processing steps, which are applied to scheduling each of the jobs as shown in Table 2.

Learning thus mainly consists of episodic rules about which job to schedule out of a set of jobs. There are also minor speedups that occur through learning how to do the bookkeeping and implement the calculations. These contribute to the general speedup, and will also show some small transfer effects. If given additional knowledge and dependent on the exact knowledge and its organization, the correct algorithm can be implemented with the same basic mechanism, giving rise to a power law of learning, but probably faster and better than subjects.

3.2 Sched-Soar's behavior. The model's behavior can be examined like a subject's behavior, as individual runs on sets of problems. Figure 4 shows Sched-Soar's solution time on four series of 16 randomly created tasks. These times include requests for advice, but this takes only 1 model cycle. Neither the power function (R^2 = .55) nor a simple linear function (R^2 = .53) proves a good fit to these data. However, when averaged these series fit a simple power function well ($T = 274*N^{-.3}$, with R^2 = .95).

An assumption in Soar is that the learning rate should be constant. In Sched-Soar, the learning rate (as chunks per trial) indeed proved constant over all 4 x 16 trials (Chunks (trial) = 15.48 * trial + 417.8; with R^2 = .98).

Table 2. The scheduling steps in Sched-Soar.

(1) Sched-Soar analyses the situation trying to find a job to schedule. The analysis includes recalling the amount of time on Machine-A that each remaining job takes, ordering them based on this, and noting how many jobs have been scheduled so far. An operator is proposed for each remaining job to schedule, which will often lead to an operator tie impasse. Previous knowledge, in the form of a learned rule (chunk), may indicate which job should be scheduled next, allowing the model to proceed directly to Step 3.

(2a) If no decision can be made, despite examination of all available internal knowledge, Sched-Soar requests advice from the environment. The advice specifies the job that is the optimal choice to schedule next in the current set.

(2b) Sched-Soar reflects on why the advice applies to the current situation. Sched-Soar uses its scheduling and basic arithmetic knowledge to figure out what makes the proposed job different from the others, using features like the relations between jobs, the resources required, and the position of the job in the sequence.

(2c) Based upon this analysis, Sched-Soar memorizes explicitly the aspects of the situation that seem to be responsible for the advice. Sched-Soar only focuses at one qualitative aspect of the suggested job, namely the ranking in processing time of the job on the first machine. Restricting what is memorized to just one feature of the situation clearly represents a heuristic that tries to take into account the limited knowledge and memorizing capability of humans. This kind of memorizing is goal-driven and would not arise from the ordinary chunking procedure without this deliberation.

We call such a learned rule an episodic chunk. These episodic chunks implement search-control knowledge, specifying what has to be done and when. An example is: If two jobs are already scheduled, and there are three operators suggesting different jobs to be scheduled next, and Job 1 has the shortest processing time compared to the other jobs on Machine A, then prefer the operator to schedule Job 1.

If in subsequent trials a similar situation is encountered, then Sched-Soar will bring its memorized knowledge to bear. Of course, because the memorized information is heuristic, positive as well as negative transfer can result. Consequently, because only explicit, specific rules are created, general declarative rule-based behavior will arise slowly and erratically to an outside observer.

(3) The job is assigned to the machines and bookkeeping of the task and experiment is performed.

Note, however, as performance only improves as a negatively accelerated power function, the newly learned chunks are less less effective than the older chunks.

A closer look at Sched-Soar's problem solving process shows that the variance in the individual trials comes from two main sources: the negative transfer of chunked knowl-

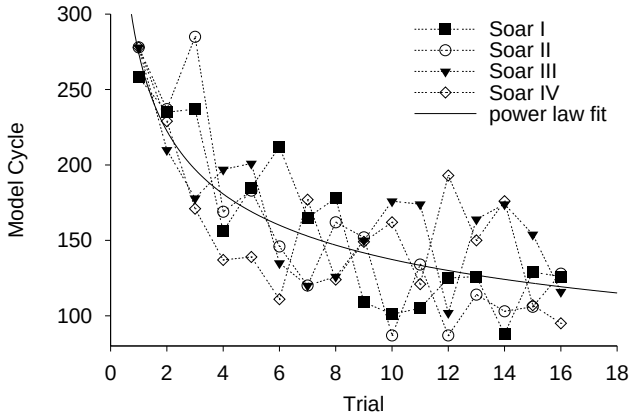

Fig. 4: Individual processing times of the Soar model for four sets (Soar I-IV) of simulated data and a power law fit.

edge and the number of requests for advice. Negative transfer results when an episodic chunk, built while solving a previous task, suggests an action in a new situation that appears similar, but it in fact requires a different action. If this is the case, Sched-Soar gets feedback that the proposed action is not optimal. This requires the situation to be evaluated again to find the proper job to schedule.

Finally, if there is no suitable knowledge, the model still has to ask for advice. This explains why we found in the model's performance that additional requests for advice are often preceded by one or more instances of negative transfer. Both negative transfer and asking for advice directly lead to more processing time. Note, however, that the rules that caused the negative transfer will not be deleted from memory nor will they be corrected. There is no distinct correction mechanism implemented within Sched-Soar. On average, however, the newly created acquired rules are quite useful and produce more positive than negative transfer. Otherwise the model would not improve over time.

While this reflective process is used to do credit assignment, the process does not have a complete view of the procedure being learned, or of the way that the outcomes of the learning process accumulates to provide a more complete scheduling skill. This learning process suggests that learning on the symbol level may give rise to behavior that follows higher level rules on the knowledge level without the higher level rules being explicitly represented. The results of this model demonstrate that knowing the algorithm (for some definition of "knowing", such as often behaving in the same way as an agent following the correct algorithm) can be reduced to having learned (sub)rules that each only partially implement the algorithm, and the learning algorithm may only have local knowledge.

This approach can be contrasted with learning mechanisms that are both less explicit and those that are more explicit. In the Altmann and John (in press) episodic indexing model, learning is integrated with attention in a way that is independent of what is attended and indeed of the task itself. Learning in Sched-Soar, which involves a deliberate cue-selection process, is qualitatively more explicit, yet not completely explicit.

This approach can also be compared with learning mechanisms that reason explicitly about strategy shifts with complete knowledge, such as those in solving the Tower of Hanoi (Anzai & Simon, 1979), and instruction taking (Huffman & Laird, 1995). Other models use complete knowledge to change their behavior. The original models of the Seibel task (Rosenbloom & Newell, 1987) shifted strategies from dealing with single lights to dealing with sets of lights with a single operator. There was no exploration, but processing changed because of learning. In these models, the learning mechanism led directly to correct, higher-level behavior.

3.3 Comparison with regularities. The empirical constraints in Table 1 are met in general by the model. (a) Solving the task requires 151 model cycles (decision cycles), averaged over all trials and runs. The Soar architecture initially specified the rate of model cycles to be between 30 ms to 300 ms (Newell, 1990). The model performs slightly faster than the log mean expected rate of 100 ms per cycle, but at 147 ms/cycle it is well within the theoretical bounds.

(b) The speed-up of the model in 16 trials is greater than the subjects' speedup. The model improves 56% (from 270 cycles in the first trial to 118 cycles in trial 16), compared with 22% by the subjects from trial one to ten. The more comparable speed-up of the model from trial one to ten was 51%. However, the model's greater learning might be an effect of different modes of receiving feedback (in the empirical study feedback was given after a complete solution, whereas Sched-Soar is advised immediately after every single scheduling level decision).

(c) The improvement in correctness cannot be explored yet, because Sched-Soar was initially programmed to use advice to produce always correct solutions. (d) Sched-Soar's behavior is always suboptimal after 16 trials (and negative transfer might still occur in later trials). (e) The model did not discover or implement the general algorithm. In order to validate and further improve the model, more data are necessary that match more exactly the model's situation, including how the model can learn through interaction and the tasks it solves.

4. Comparison of Sched-Soar with additional data.

To make the model's and subjects' data more comparable, we conducted an additional study where we assessed participants' behavior in solving the task under conditions that were more equivalent to the simulation studies. For this study, the model results can be considered more direct theoretical predictions about subjects' behavior because the length of trials and form of feedback are more similar.

4.1 The experiment. The experiment was carried out with 14 subjects, all undergraduate students at the University of Regensburg receiving course credit for participation. In this study, participants were not told the scheduling algorithm, but could get advice immediately after every scheduling decision. This provides a more similar setting to the model's situation.

Subjects were instructed to separate their decisions based on knowledge from those based on guesses: they were requested to ask for advice when they did not know what to do. When they asked for advice, they were told

where to schedule the next job. Each subject solved a total of 18 different scheduling problems. At the end of the experiment, subjects were debriefed and asked whether they had detected regularities within the task.

4.2 The comparison. The subjects' processing time for each task is shown in Figure 5. The time for asking advice and getting the feedback was not included (as this is not fully modelled), but time thinking about the advice was. The average processing times for trials 1 to 18 vary between 99.4 and 36.3 s. The times are longer than the times in Table 1 because subjects here did not have as much practice and spent time receiving advice. We found that a power function ($T = 110 * N^{-.38}$) accounts best for the averaged data ($R^2 = .82$, compared with .71 and .73 for linear or exponential functions). It appears that subjects learn faster with more detailed feedback (the power law has a steeper slope).

It has been noted before that cognitive models often predict less time than subjects take (Kieras, Wood, & Meyer, 1997). Like many cognitive models (e.g., Peck & John, 1992), Sched-Soar performs the task more efficiently than subjects do. Sched-Soar predicted times on these tasks to be between 270 and 116 model cycles. To match the subjects' time, one has to assume between 313 or 369 ms/model cycle, which is slightly above the region defined by Newell (1990). The learning rate (power law coefficient) of the subjects is only somewhat higher than the learning rate of the model (-.38 vs. -.30), suggesting that the task was equally complex for subjects and model (see Newell & Rosenbloom, 1981 for a discussion of this coefficient).

If these time constraints are taken seriously, they indicate that Sched-Soar is not doing as much work as subjects are. This is almost certainly the case. Future extensions to Sched-Soar should include more of the tasks that subjects must perform, such as reading the jobs from the screen and typing in the schedule. Subjects may also be taking more time to comprehend and understand the advice. This will take time to perform, but it will offer further opportunities for learning, leading to both an increase in average processing time and to a somewhat higher learning rate because these are easy tasks to learn.

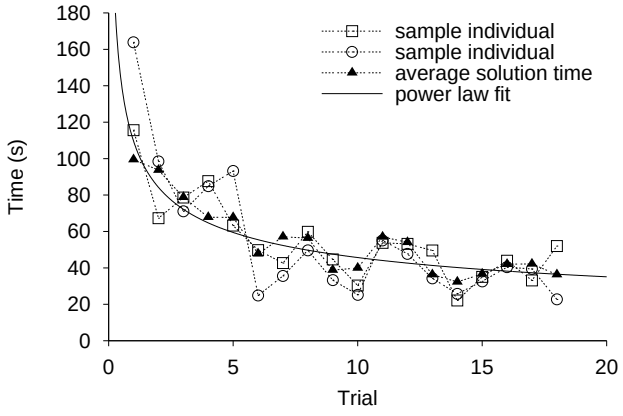

Fig. 5: Processing time from trial 1 to 18 for two sample individuals, the average solution time of all subjects, and a power law fit (log-log) to the average.

We also found a correlation of .46 between processing time by the subjects and the number of their requests for advice due to lack of knowledge or wrong decisions. The same behavior is exhibited by the model where negative transfer of chunk knowledge leads to advice seeking. The model's mechanism at least provides a partially sufficient explanation of what prompted subjects to ask for advice.

Interestingly, none of the subjects discovered the optimal algorithm for solving the task. This is indicated by their sub-optimal performance and their comments in the debriefing session (i.e., they could not express the correct scheduling algorithm or how they were improving, but could recognize important features). Because subjects did not detect regularities with the task but obviously improved over time, a less than perfect and not completely conscious learning mechanism is suggested. The quality of the comparison suggests, however, that subjects used a set of heuristics similar to those in Sched-Soar.

4.3 Relative strengths and weaknesses. Sched-Soar has several relative strengths. Sched-Soar's weaknesses in many ways mirror its strengths. As part of an architecture, it should be able to be combined with other Soar models to form a more unified theory. However, models in cognitive architectures are not often combined. This is true for Sched-Soar as well, although it did resuse another model of maths and of memorization itself. It provides principled predictions of reaction times over a series of trials while learning that generally match human behavior on this task. Not all of Sched-Soar's predictions have been tested, such as individual series of reaction times. This work has to be done by hand, and is tedious and time consuming.

5. Conclusions.

Sched-Soar provides a detailed account of how the power law can arise out of apparently noisy behavior, how transfer rules can be learned, and how the noise in reaction time data can be explained by varying amounts of transfer of knowledge. The model also explains how learning while problem solving can lead to a slowly improving performance, and how rule-like behavior can arise out of apparently noisy behavior. Such a demonstration, in a new domain and compared to data from a new experimental paradigm, is a contribution if only because many instances are necessary for us as cognitive scientists to infer and confirm a complex concept such as human learning. This complements the SCA model (Miller & Laird, 1996), which slowly learns declarative (categorization) knowledge.

5.1 One way the power law arises. A major claim of our analysis of Sched-Soar is that the power law of practice will appear when performing the kind of scheduling task used in these studies, but that the smooth power law curve will only appear when the data are averaged over subjects or trials. We saw the model and subjects improve in a non-linear and non-continuous way. Sched-Soar, like other models of learning and transfer (e.g., Gobet, 1997; Ritter & Bibby, 1997), suggests fairly clearly that some of the variance in solution times is not due to noise in the measurement process or variance in the processing rate of the underlying cognitive architecture (which might have been proposed for simpler, more perceptual tasks). Sched-Soar strongly suggests that the variance in solution time

in this case is due to how much learned knowledge transfers to new situations. Sometimes a learned episodic rule transfers very well, and sometimes it does not transfer, and sometimes it transfers but is incorrect. Similar claims are implicit in Newell (1990) and in Singley and Anderson (1989), but are not examined in as much detail. Sched-Soar provides a more complete explanation of how rules that transfer can be learned.

These results are consistent with studies showing that the power law of practice applies across strategies (Delaney, Reder, Staszewski, & Ritter, 1998). Sched-Soar goes further in that it suggests that the rate of improvement in the task, the power law, arises out of individual learning events. The improvements in problem solving in this task at this time scale are not due to some form of tuning, but due to knowledge compilation and transfer.

5.2 How rule-like behavior arises. Sched-Soar shows how the gradual acquisition of rule-like procedural behavior can be achieved in the Soar architecture. It learns through the creation and storage of specific, context dependent rules based on simple heuristics; it does not infer general rules. In this way it is similar to EPAM (Feigenbaum & Simon, 1984). It is slightly different from Anderson's (1987) theory of skill acquisition of passing through declarative, procedural, and tuning stages. Sched-Soar does problem solving to create procedural knowledge from declarative information, and implements tuning as additional procedural knowledge, to the effect that the model reduces the final stage of tuning to the same mechanisms underlying the previous stages.

This model does not know enough to become perfect. Its representation and problem solving knowledge is too weak. It does, however, know enough to improve with practice. People exhibit this type of slow improvement in many circumstances. This model suggests that in such situations, people's behavior might be optimal—it is just their knowledge that is incomplete rather than their processing mechanisms or planning algorithms. For problem solvers like this, further improvements in task performance will have to come with additional knowledge.

One bigger question is how to view this mechanism in the context of larger tasks, for example, as a grammar learning mechanism. This model shows how rule-like behavior, such as language, could arise gradually even in a symbolic architecture from a simple learning mechanism with incomplete knowledge and non-explicit representations. The rule-like behavior is not defined explicitly as a rule or definition, but implemented as an expanding set of simpler rules implementing and arising out of a simple algorithm. In line with findings from the field of implicit learning (e.g., Haider & Frensch, 1996), the improvements in this task come from recognizing regularities based on simple heuristic knowledge and attention to a partial set of necessary features while trying to minimize the use of resources. In this way, it is consistent with previous work to learn grammars through chunking (Gobet & Pine, 1997; Servan-Schreiber & Anderson, 1990).

Acknowledgments

We would like to thank two anonymous referees, Fernand Gobet, and John Josephson for useful comments.

References

Aasman, J., & Michon, J. A. (1992). Multitasking in driving. In J. A. Michon & A. Akyürek (Eds.), *Soar: A cognitive architecture in perspective.* Dordrecht, The Netherlands: Kluwer.

Akyurek, A. (1992). Means-ends planning: An example Soar system. In J. A. Michon & A. Akyurek (Eds.), *Soar: A cognitive architecture in perspective.* 109-167. Dodrecht, NL: Kluwer.

Altmann, E. M., & John, B. E. (in press). Episodic indexing: A model of memory for attention events. *Cognitive Science.*

Anderson, J. R. (1987). Skill acquisition: Compilation of weak method problem solutions. *Psychological Review, 94*, 192-210.

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

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review, 86*, 124-140.

Bass, E. J., Baxter, G. D., & Ritter, F. E. (1995). Using cognitive models to control simulations of complex systems. *AISB Quarterly, 93*, 18-25.

Baxter, G. D., & Ritter, F. E. (1998). The Soar FAQ, http://www.nottingham.ac.uk/pub/soar/nottingham/soar-faq.html.

Chong, R. S., & Laird, J. E. (1997). Identifying dual-task executive process knowledge using EPIC-Soar. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society.* 107-112. Mahwah, NJ: Lawrence Earlbaum Associates.

Delaney, P. F., Reder, L. M., Staszewski, J. J., & Ritter, F. E. (1998). The strategy specific nature of improvement: The power law applies by strategy within task. *Psychological Science, 9*, 1-8.

Feigenbaum, E. A., & Simon, H. A. (1984). EPAM-like models of recognition and learning. *Cognitive Science, 8*, 305-336.

Fox, M., Sadeh, N., & Baykan, C. (1989). Constrained heuristic search. In *Proceedings of IJCAI'89.* 20-25.

Graves, S. C. (1981). A review of production scheduling. *Operations Research, 29*, 646-675.

Gobet, F., & Pine, J. (1997). Modelling the acquisition of syntactic categories. In *Proceedings of the 19th Annual Conference of the Cognitive Science Society*. 265-270. Mahwah, NJ: Lawrence Erlbaum.

Gobet, F. R. (1997). Roles of pattern recognition and search in expert problem solving. *Thinking and Reasoning, 3*, 291-313.

Haider, H., & Frensch, P. A. (1996). The role of information reduction in skill acquisition. *Cognitive Psychology, 30*, 304-337.

Howes, A. (1994). A model of the acquisition of menu knowledge by exploration. In *Proceedings of CHI'94 Conference on Human Factors in Computing Systems*. 445-451. New York, NY: ACM Press.

Howes, A., & Young, R. M. (1996). Learning consistent, interactive, and meaningful task-action mappings: A computational model. *Cognitive Science, 20*, 301-356.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *Journal of AI Research, 3*, 271-324.

John, B. E., Vera, A. H., & Newell, A. (1994). Towards real-time GOMS: A model of expert behavior in a highly interactive task. *Behavior and Information Technology, 13,* 255-267.

Johnson, K. A., Johnson, T. R., Smith, J. W. J., DeJongh, M., Fischer, O., Amra, N. K., & Bayazitoglu, A. (1991). RedSoar: A system for red blood cell antibody identification. In *Fifteenth Annual Symposium on Computer Applications in Medical Care.* 664-668. Washington: McGraw Hill.

Johnson, S. M. (1954). Optimal two and three-stage production schedules with set up times included. Naval *Research Logistics Quarterly, 1,* 61-68.

Johnson, T. R., Krems, J., & Amra, N. K. (1994). A computational model of human abductive skill and its acquisition. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. 463-468. Lawrence Erlbaum.

Kieras, D. E., Wood, S. D., & Meyer, D. E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *Transactions on Computer-Human Interaction, 4,* 230-275.

Kitajima, M., Soto, R., & Polson, P. G. (1998). LICAI+: A comprehension-based model of the recall of action sequences. In F. E. Ritter & R. M. Young (Eds.), *Proceedings of the 2nd European Conference on Cognitive Modelling.* 82-89. Thrumpton, Nottingham: Nottingham University Press.

Krems, J., & Nerb, J. (1992). Kompetenzerwerb beim Lösen von Planungsproblemen: experimentelle Befunde und ein SOAR-Modell (Skill acquisition in solving scheduling problems: Experimental results and a Soar model) FORWISS-Report FR-1992-001). FORWISS, München.

Larkin, J. H. (1981). Enriching formal knowledge: A model for learning to solve textbook physics problems. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition.* 311-334. Hillsdale, NJ: LEA.

Lehman, J. F., Laird, J. E., & Rosenbloom, P. S. (1996). A gentle introduction to Soar, an architecture for human cognition. In S. Sternberg & D. Scarborough (Eds.), *Invitation to cognitive science,* vol. 4. Cambridge, MA: MIT Press.

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

Miller, C. S., & Laird, J. E. (1996). Accounting for graded performance within a discrete search framework. *Cognitive Science, 20,* 499-537.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence, 40,* 63-118.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Newell, A., & Rosenbloom, P. (1981). Mechanism of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition,* 1-56. Hillsdale, NJ: Erlbaum.

Nielsen, T. E., & Kirsner, K. (1994). A challenge for Soar: Modeling proactive expertise in a complex dynamic environment. In *Singapore International Conference on Intelligent Systems (SPICIS-94)*. B79-B84.

Norman, D. A. (1991). Approaches to the study of intelligence. *Artificial Intelligence* 47, 327-346.

Peck, V. A., & John, B. E. (1992). Browser-Soar: A computational model of a highly interactive task. In *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems.* 165-172. New York, NY: ACM.

Prietula, M. J., Hsu, W. L., Steier, D. M., & Newell, A. (1993). Applying an architecture for general intelligence to reduce scheduling effort. *ORSA Journal of Computing, 5*(3), 304-320.

Rieman, J., Lewis, C., Young, R. M., & Polson, P. G. (1994). "Why is a raven like a writing desk" Lessons in interface consistency and analogical reasoning from two cognitive architectures. In *Proceedings of the CHI 94 Conference on Human Factors in Computing Systems.* 438-444. New York, NY: ACM.

Rieman, J., Young, R. M., & Howes, A. (1996). A dual-space model of iteratively deepening exploratory learning. *International Journal of Human-Computer Studies,* 743-775.

Ritter, F. E., & Bibby, P. A. (1997). Modelling learning as it happens in a diagrammatic reasoning task (Tech. Report No. 45). ESRC CREDIT, Dept. of Psychology, U. of Nottingham.

Ritter, F. E., Jones, R. M., & Baxter, G. D. (1999). Reusable models and graphical interfaces: Realising the potential of a unified theory of cognition. In U. Schmid, J. Krems, & F. Wysotzki (Eds.), *Mind modeling - A cognitive science approach to reasoning, learning and discovery.* 83-109. Lengerich: Pabst Scientific Publishing.

Ritter, F. E., & Young, R. M. (1998). The Psychological Soar Tutorial, http://www.psychology.nottingham.ac.uk/staff/ritter/pst-ftp.html (vers. 12.).

Rosenbloom, P. S., Laird, J. E., & Newell, A. (1992). *The Soar papers: Research on integrated intelligence.* Cambridge, MA: MIT Press.

Rosenbloom, P. S., & Newell, A. (1987). Learning by chunking, a production system model of practice. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development.* 221-286. Cambridge, MA: MIT Press.

Rumelhart, D. E., McClelland, J. L., & The PDP Research Group (1986). *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations.* Cambridge, MA: The MIT Press.

Sanderson, P. M. (1989). The human planning and scheduling role in advanced manufacturing systems: An emerging human factors domain. *Human Factors, 31,* 635-666.

Schmalhofer, F. (1998). *Constructive knowledge acquisition: A computational model and experimental evaluation*. Mahwah, NJ: Erlbaum.

Servan-Schreiber, E., & Anderson, J. R. (1990). Learning artificial grammars with competitive chunking. *Journal of Experimental Psychology: Learning,*

Memory, and Cognition, 16, 592-608.
Siegler, R. S., & Shrager, J. (1984). Strategy choices in addition and subtraction: How do children know what to do? In C. Sophian (Ed.), *Origins of cognitive skills.* 229-293. Hillsdale, NJ: Lawrence Earlbaum Associates.
Simon, T. J., & Klahr, D. (1995). A computational theory of children's learning about number conservation. In T. J. Simon & G. S. Halford (Eds.), *Developing cognitive competence: New approaches to process modeling.* 315-353. Hillsdale, NJ: Lawrence Erlbaum.
Singley, M. K., & Anderson, J. R. (1989). *The transfer of cognitive skill*. Cambridge, MA: Harvard University Press.
VanLehn, K. (1988). Toward a theory of impasse-driven learning. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems*. 19-41. New York, NY: Springer.
VanLehn, K. (1991). Rule acquisition events in the discovery of problem-solving strategies. *Cognitive Science*, *15*, 1-47.
Weber, G. (1996). Episodic learner modeling. *Cognitive Science, 20*(2), 195-236.