Intelligent Architectures for Knowledge Sharing: A Soar Example and General Issues

Dan Zhu
Department of Logistics, Operations and MIS
College of Business Administrations
Iowa State University, Ames IA 50011
dzhu@iastate.edu

MJ Prietula
Goizueta Business School
Emory University
Atlanta GA 30322-2710
prietula@bus.emory.edu

Abstract

In this talk we present a model of knowledge assessment based on architecture-specific metrics and present a method of knowledge sharing called Direct Knowledge eXchange (DKX). Although embryonic from a knowledge management perspective, the notion of DKX between homogeneous knowledge engines and knowledge metric assessment represents a small step to examining how distribution of knowledge can lead to remote "knowledge invocation on demand" types of distributed problem solving systems and practical mechanisms to assess their resource value.

Summary

Soar is a production system that characterizes all symbolic goal-oriented behavior as search in problem spaces and serves as an architecture for general intelligent behavior (Laird, Rosenbloom & Newell 1987). A problem space defines a set of states that can be reached within that problem space, and an associated collection of operators. Operator applications move Soar from state to state and, consequently, define search in the problem space.

Decisions are the primitive acts of the system used for search (i.e., generation and selection) of appropriate problem spaces, states, and operators, as well as the application of operators for new state configuration, in the pursuit of goals specified in a goal hierarchy.

To achieve problem-solving goals, Soar operates in terms of a two-phase *decision cycle*. Each cycle starts with an elaboration phase followed by a decision phase. Together, these phased mechanisms, coupled with an embedded set of primitive preferences, allow for problem solving to ensue through the specification of appropriate sequences of operators.

Soar is impasse-driven. In situations where the operator selection cannot unambiguously proceed (e.g., via incomplete or inconsistent preferences), then an impasse occurs, and a new subgoal is established with an associated problem space, and the process recurses.

The resolution of a subgoal in Soar is achieved by finding knowledge that resolves higher-level impasses, allowing problem solving to proceed. When this occurs, Soar learns. Specifically, "chunks" are produced that are productions that map working memory elements defining impasse situations (as antecedent conditions) into the results of subgoals (as consequent conditions). Chunking can be viewed as a form of explanation-based learning, but it is at a level articulated in specific and uniformly applied cognitive mechanisms. Subsequent encounters with similar impasse conditions can thus be resolved more directly (and with less deliberation) with the newly acquired chunks. Soar has learned.

Key to learning in general and generalized learning in particular is the ability to transfer knowledge. Soar has three basic forms of transfer: within-trial (chunks can be used as soon as they are built), between-trial (chunks are improved with repeated trials on a task), and across-task (chunks can apply to similar problems).

We are exploring another form of transfer. Imagine a set of distributed Soar knowledge engines. What would happen if they could exchange chunks? In other words, what would problem solving look like if a set of affiliated and distributed Soar problem solving engines could directly exchange their chunks – their intimate knowledge of a task?

Base Task

The task selected was the 8-puzzle (see Figure 1).[1] In this task, there is a 3 x 3 matrix of eight randomly assigned numbers (the ninth being a space) and the problem is to find the series of operators (moves) that shuffle the set into a properly ascending sequence, by moving the tiles, one-by-one, to the open space. This task has 9! initial states and, depending on how you define it, can generate an uninformed search expansion of about 3.4×10^9.

[1] Also called the N x N sliding tile problem, often appearing as a hand-held game where tiles are adjacently slid and interchanged until the right sequence/pattern is achieved.

1	2	3
4	5	6
7	8	

Figure 1

Twenty initial problems were run (with randomly assigned initial states) twice. First, each problem was run with learning inhibited, and then the problem was again run with learning enabled (generating chunks). Figure 2 shows the average decision cycles required over the 20 problems. The first (0) trial reflects effort to solve the problem with no learning. Subsequent numbers (trials 1 through 5) show between-trial improvement in problem solving behavior.

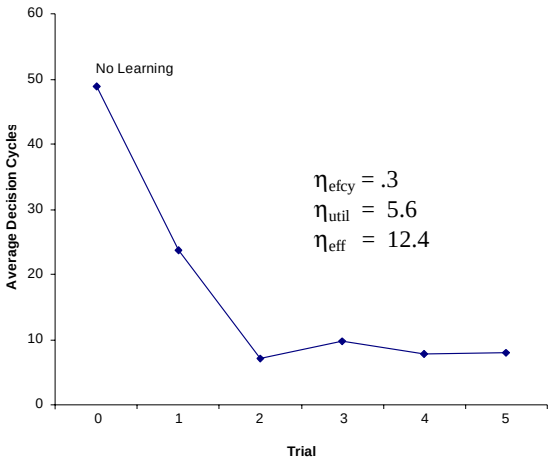

Figure 2. Learning in 8-puzzle

Knowledge Metrics

Despite many efforts at "knowledge management," *it is doubtful that knowledge can be defined independent of an architecture to interpret it.* Consequently, we used three metrics to explore how the fundamental components of knowledge in this architecture (chunks) could be viewed as a resource and measured.

The metrics we used (noted in Figure 2) were original proposed by Prietula et al. (1993) and are defined as follows.

$$[A(k)] = \begin{cases} 1 & \text{if chunk } k \text{ is applied at least once within a trial} \\ 2 & \text{if chunk } k \text{ is not applied} \end{cases}$$

Knowledge Efficiency (η_{efcy}) is defined for N chunks produced on a trial and is the proportion of the produced chunks that were eventually applied within-trial. This ranges from 0 (none of the produced chunks were applied) to 1 (all of the chunks were applied at least once within the trial).

$$\eta_{efcy} = \frac{\sum_{1 \le k \le N} [A(k)]}{N}$$

Knowledge Utility (η_{util}) reflects the number of times a chunk is applied within the trial, and can range from 0 to an arbitrary upper limit (multiple chunks defined and applied multiple times). The argument is that multiple applications may reflect an increase in utility, given the "knowledge production cost" is fixed and amortized over trials.

$$\eta_{util} = \frac{N_{applied}}{\sum_{1 \le k \le N} [A(k)]}$$

Knowledge Effectiveness (η_{eff}) attempts to relate the actual contribution of chunks to the reduction of deliberation effort (decision cycles), by dividing the effort saved (expressed in decision cycles, D_c) by the number of unique chunks produced and applied in the same trial (the denominator of η_{efcy}). This yields D_c per chunk that reflects the average within-trial contribution for the applied chunks. It is a measure of knowledge value.

$$\eta_{eff} = \frac{D_c^n - D_c^l}{\sum_{1 \le k \le N} [A(k)]}$$

The results shown in Figure 2 indicate that roughly 1/3 of the chunks that were generated were applied (η_{efcy}), the average chunk was applied 5.6 times within a trial (η_{util}), and the average contribution of a chunk was a reduction of 12.4 D_c (η_{eff}).

Also of interest is the relative *change* in these knowledge metrics from the start to the end of trials. An analysis revealed that the average η_{efcy} increased (Wilcoxon, $z = 2.93$, $p < .001$), the average η_{util} decreased (Wilcoxon, $z = 3.33$, $p < .001$), and the average η_{eff} also decreased (Wilcoxon, $z = 1.98$, $p < .05$). Therefore, it appears that most of the value of learning in these tasks is derived from the knowledge developed early in the process, with their contribution increasing (knowledge utility) with decreasing returns on generated knowledge over later trials. Much is learned early in the task.

Direct Knowledge Exchange

Of interest is how direct knowledge exchange would impact the three knowledge metrics. Five of the 20 initial conditions were selected as new problems (problems 1, 10, 15, 16, 20). The chunks generated from five other randomly selected runs were gathered (problems 2, 3, 4, 7, 12).

Figure 3 presents the results reflecting the change in Knowledge Efficiency when supplying new Soar agents with knowledge from the indicated source agent (DKX *i*). The x-axis reflects the new problems addressed by the five agents, each of which has imported the knowledge of the

agent indicated in the graphed lines. Similarly, Figure 4 presents changes in Knowledge Efficiency and Figure 5 presents changes in Knowledge Utility.

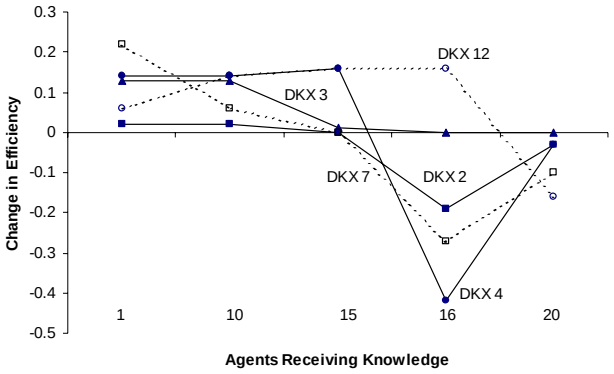

Figure 3

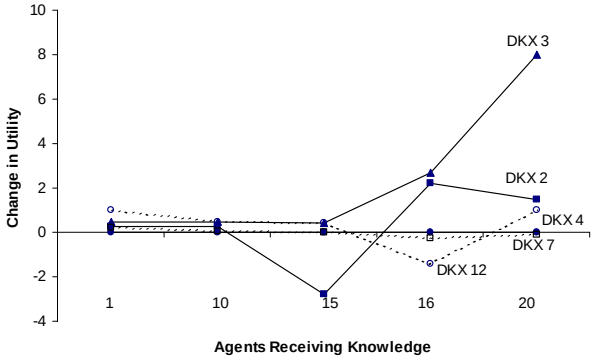

Figure 4

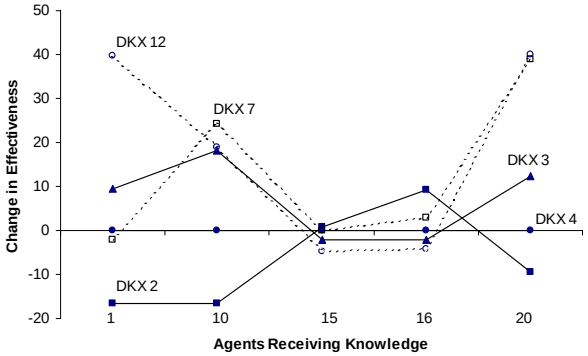

Figure 5

These results are also represented in Table 1, where the source agents (row titles) and receiving agents (column titles) are listed. The entries in this table are bit-strings where each bit represents whether there was a contribution of the supplied knowledge to an improvement in Knowledge Efficiency (1st bit), Utility (2nd bit), and Effectiveness (3rd bit). If a bit is on, then a contribution for that particular knowledge source (the row) has been made for that particular metric (the bit) for that particular agent problem (the column).

Table 1

	1	10	15	16	20	**Totals**
2	010	010	001	111	110	**242**
3	011	011	010	010	011	**053**
4	000	000	000	100	100	**200**
7	000	011	000	111	111	**233**
12	011	011	000	010	111	**143**
Totals	**032**	**043**	**011**	**332**	**443**	

The column totals are bit-sums in Table 1. For example, note that the knowledge contributed by agent 3 (row) to the problem solved by agent 20 (columns) is indicated by the bit-string entry 011, showing that it did not improve Efficiency for that problem, but did improve Utility and Effectiveness. In fact, the knowledge supplied by agent 3 improved Utility scores across all agent problems ("5" in the "053" row Totals).

The results show that DKX can improve all three metrics, but there is variance at both ends. From the source end, the contribution within and between sources to the metrics varies. Agents 2 and 7 (rows) have broader contributions across problem sets (columns), while agent 4 has minimal contributions. From the receiving end, agents 16 and 20 (columns) have broader benefits than agent 15. Therefore, this suggests that there exists a knowledge assignment problem that, in fact, could be considered a component of the DKX architecture in that agents requesting knowledge (given an impasse event) must be able to reconcile alternative knowledge sources. But at what architectural level should this occur, without doing disservice to the architecture?

References

Laird, J., Newell, A. & Rosenbloom, P. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1), 1-64.

Laird, J., Rosenbloom, P. & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(1), 11-46.

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