Metrics for Cognitive Architecture Evaluation

Robert Wray, Soar Technology, Inc., wray@soartech.com
Christian Lebiere, Carnegie Mellon University, cl@cmu.edu

Introduction

The problem of evaluating general architectures is a difficult one (Newell, 1990). Comparative evaluations that focus on performance alone are especially problematic. It is usually feasible to develop a specialized solution for any particular problem that will outperform a general solution, such as one developed within a cognitive architecture. Thus, an evaluation of the architectural approach derives from the power of the primitives of the architecture, the generality and flexibility of these primitives in providing solutions across a range of tasks, and the resulting ability to (relatively) rapidly develop an architectural solution via a common computing framework. While this power is assumed within the cognitive architecture community (and there is significant anecdotal evidence to support it), today the community lacks a *scientific* foundation for measuring and evaluating these claims.

This paper identifies a number of metrics that could be important for assessing cognitive architectures across a range of applications domains. The metrics are organized according to a taxonomy of requirements for intelligent systems developed by Anderson & Lebiere (2003). We focus only on the functional aspects of their analysis, rather than those non-functional requirements specific to human cognition, which are also detailed in the Anderson and Lebiere evaluation approach.

These metrics together reflect our attempt to capture and measure many necessary components of general intelligent behavior, rather than solely performance metrics, which are often the primary means of evaluating intelligent systems in AI. We introduce two metrics novel to cognitive-architecture research, incrementality and adaptivity, which may prove to be useful for capturing and expressing the cumulative value of cognitive-architecture-based solutions across multiple tasks within a domain and across multiple application domains. Our approach is far from complete, in that several requirements include only notional metrics. However, this approach provides at least an empirical foundation for comparing work within and across the development cognitive architectures that can provide more objective measures of a cognitive architecture's capabilities and utility as a platform for general intelligence.

One of the primary factors that makes achieving general, intelligent behavior such a difficult problem is the complexity and variation found in the environment. For general intelligence, agents must be able to cope effectively with this complexity. However, while complex, the environment (usually) is not chaotic. It operates according to laws and general properties and can be characterized according to its complexity. Russell and Norvig's (1995) influential textbook introduces a number of contrasting dimensions that can be used for characterizing domains. For example, accessible domains provide complete access to the state of the environment while, in inaccessible domains, information relevant to a good choice at some point in time may not be available to the agent via direct perception. Different problems will have different complexity profiles based on these dimensions. Many of the metrics introduced below will also interact with these dimensions, so that quality of the overall solution and the problem complexity define a functional space for the metric for some class of related problems. A significant qualification to the work presented here is the lack of consistent measures of complexity across different application domains. Some problems and domains provide simple measures of complexity (such as the number of cities in a Traveling Sales Man tour) which can be used to evaluate the performance of a system with increasing complexity but, to-date, there are no domain-general characterizations of complexity that enable cross-domains comparisons. This will limit the utility of some of the proposed metrics to comparison within particular domains.

The remainder of the paper introduces metrics for each category of the general requirements of Anderson & Lebiere (2003). Many performance-based metrics are reused from one category to the next, suggesting specific ways in which base performance characteristics can be systematically explored to provide a more complete, multi-dimensioned characterization of performance results. While, in most cases, we propose objective, quantitative metrics, in some cases, only qualitative and/or subjective evaluation is possible today. Perhaps the workshop discussion will lead to ideas and insights for more objective approaches to evaluation in these areas as well.

Behave as an (almost) arbitrary function of the environment

Environments generally will change independently of an intelligent system ("dynamic" in the Russell and Norvig properties) and the actual state of an environment may not be known or directly perceivable (inaccessible). Thus, the intelligent system must be able to act in the situation it finds itself in (and even if it is different than the one it expected to be in). This flexibility implies a breadth of capability, meeting the complexity of the environment with appropriate responses.

Taskability is the ability of a system to adapt to new/novel problems without human (programmer) intervention. Taskability is difficult to measure because there is no "absolute" notion of taskability -- a particular quantitative measure for one domain might represent the best one could achieve, while in another, it might be a baseline.

Researchers in AI have generally evaluated taskability by adopting a set of benchmark tasks against which a system is developed, and then introduced novel tasks within the same domain and tested system performance on these new tasks (Hanks, Pollack, & Cohen, 1993). This approach provides a reasonable qualitative measure of taskability within a domain.

Incrementality is the ability to extend a cognitive-architecture-based system from one set of tasks to another set, which can be either a superset of the original set (as an example of generalization) or reflect a different set of requirements (an example of the robustness and taskability of the architecture). Incrementality could possibly be measured by the degree of overlap between the solutions to the two sets of problems. For instance, if some cognitive system provides a quite general capability, a small task-specific addition at the knowledge representation level might be sufficient to tackle a new task. On the other hand, if a system is overly specific to a particular problem, then significant reworking for the new task might be necessary and the resulting incrementality will be poor. The AAAI General Game Playing (GGP) competition is representative of this attempt to realize a more general capability than just effective play of some specific game.

Measuring incrementality will help expose excessive benchmark-driven task specialization and thus help ensure the generality of an architectural framework. To our knowledge, incrementality is a novel dimension that has not been previously evaluated, although it is consistent with the general notion of a "cumulation of results" as discussed in Newell (1990). The primary difference is that rather than a qualitative accumulation of results across different tasks, as suggested by Newell, incrementality is proposed as a quantitative measure that expresses the actual overlap at the source code level between different applications of an architecture.

Figure 1 illustrates a notional approach to measuring incrementality in the evaluation of an architecture. Here, we propose a very simple measure of incrementality. Incrementality is the ratio of the unchanged lines of source code to total lines of code needed for the solution of some problem, in comparison to the source code used to solve previous problems. One of the key points of this definition to incrementality is that it makes no distinction between the architecture source code (typically written in a standard high level language, such as Java, C, or LISP) and the knowledge representations encoded in the language(s) an architecture defines for specifying content.

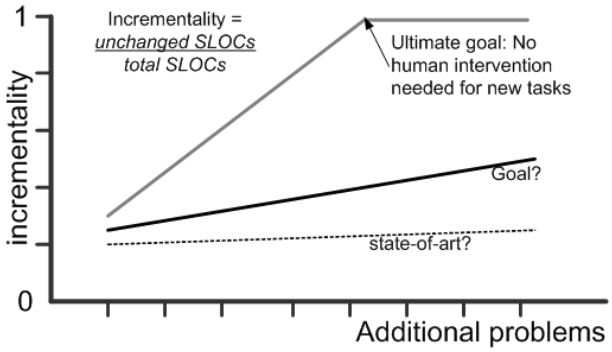

Figure 1: Measuring incrementality.

The simplicity of this definition has two advantages. First, because it does attempt to distinguish or weigh the relative contribution of different elements in the architecture-based application, it provides a direct analog to reuse metrics in software engineering generally and allows comparison to non-architecture-based approaches and their level of reuse. Second, tools to measure incrementality could be readily developed using existing source control revision comparison tools (e.g., "diff").

Today, typically, at best the source code of an architecture is reused from one application to the next, resulting in a modest but not trivial baseline. This baseline suggests some of the inherent value in cognitive architectures generally, since the incrementality of non-cognitive-architecture-based intelligent systems development is near nil (i.e., systems development begins near *de novo* for new applications).

In the ideal case, which is a long-term, not near-term goal, for any existing cognitive architecture, full incrementality is reached: no new changes to the architecture or its encoded knowledge are needed for a new task. In the meantime, as shown in the middle line, incrementality could be used as an explicit tool for understanding at a gross level how much of an architecture and its application are reusing the previous results over time. While full incrementality is likely infeasible, this approach would provide at least a coarse measure of incrementality for cognitive systems and give insight to other scientists about the level of reuse and cumulation within a particular architectural paradigm.

An obvious drawback of this definition is its coarse-grained nature. For example, the Soar source code is very roughly 50,000 source lines of code (SLOCs) of C. An obvious parallel to a SLOC in the Soar language is a production. However, most Soar systems have only a few hundred productions; the largest Soar application system, TacAir-Soar (Jones, et al, 1999), today has about 10,000 productions. Thus, significant changes in the kernel level of Soar are likely to mask reuse at the production-knowledge level of representation; similarly, the lack of reuse at the production-knowledge level will be masked by the SLOCs. We expect approaches that take incrementality seriously will provide a number of more fine-grained measures to highlight these effects.

Another possible limitation of incrementality proposed here is that it measures "latent" capability, rather than the capabilities actually used in the execution of a task. For example, using another Soar example, one could likely include the productions from a number of distinct applications into a single Soar system. In such a situation, the numerator of the incrementality metric would increase with each new application, but little reuse would be guaranteed. We propose that incrementality be used in conjunction with knowledge utilization (described below) to make explicit the actual use of the knowledge within an incrementally increasing source repository.

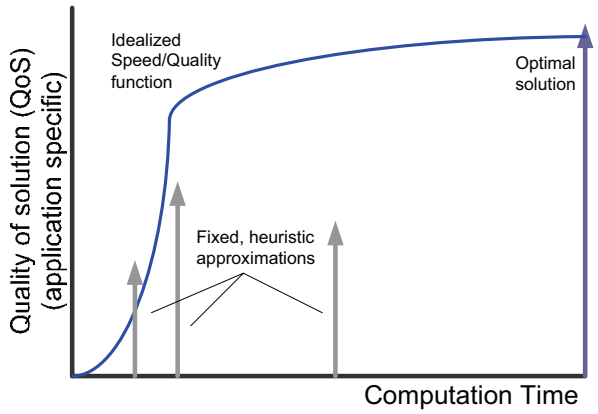

Figure 2: The relationship between computational investment and quality of solution.

Operate in real time

For an intelligent system to behave intelligently, it must able to recognize a situation that it cares about, determine an appropriate response, and then act. However, the world in which an agent operates may be continually changing. Thus, the agent must perform its internal processes quickly, relative to the speed of change in the environment, or its chances of survival/success diminish in a ling-term-existence. Many reflexes and instincts can be viewed as evolutionary solutions to the problem of fast reaction in the animal world, "hard wiring" responses to specific situations. The important requirement is that the speed of response is sufficient for the demands of the problem, rather than being as fast (in absolute terms) as possible.

There is typically also a trade-off between performance measures and the quality of solution (QoS). As an example, consider the range of potential relationships between the time taken to compute or generate some response or behavior, and the quality of the resulting solution, as illustrated in Figure 2. There may a fixed computation time that is needed before any solution or response is produced, as shown by the gray, pulse arrows. Computing an optimal solution typically provides an upper limit on compute time. While heuristics can be used to shorten the delay, except in special cases (such as admissible heuristics), more computational effort does not necessarily result in improving QoS. In the ideal case, behavior generation and reasoning has an *anytime* property, in which additional computation leads to improvement in the overall quality of solution. However, many real-world problems are difficult to formulate in terms of anytime response characteristics. The main lesson from Figure 2 is that there is typically a relationship between the time invested in generating a solution and its overall quality, meaning that absolute performance comparisons should be normalized or cast against QoS.

Metrics for real-time operation should likely include:

Response time is the time between the onset/assignment of a task (including specific subtasks) and its resolution. As noted above, response time should be accompanied by a Quality of Solution associated metric, to distinguish between satisficing/non-satisficing solutions, and to demonstrate trade offs between solution quality and response time. Also, response time should distinguish between soft and hard real time responses.

Cognitive cycles/cognitive operations per second (COPS) measures the cycles (or %age of total CPU time) devoted to cognitive-architecture operations. This metric is a poor stand-alone metric because decreases or increases cannot be evaluated in an absolute sense, similar to the way comparing operations per second in RISC vs. CISC architectures is also marginally informative. Although it has limited utility as an absolute measure, cognitive operations/time is a good *relative* metric, allowing one to assess improvements against a baseline or benchmark. Scalability (as discussed further below) can be partially evaluated according to such metrics.

Extended operation/longevity: Intelligent systems must be able to persist over long durations. What is the uptime of the cognitive architecture in a particular application? How do other performance metrics change as uptime increases? For example, does system performance degrade as uptime increases (this is often observed in some learning systems, where the addition of knowledge via learning leads to significant degradation in knowledge retrieval performance)? While it may be true that particular architectures exhibit poor uptimes due to implementation issues (a common problem is the implementation of inherently parallel processes on serial machines), the engineering-oriented reportage of these measures would give potential users insights into the maturity of current implementations and a better understanding of their actual potential value in a proposed application.

Exhibit rational, effective adaptive behavior

An intelligent system must not only respond to its environment, but it should respond in a manner appropriate for the situation. In particular, as the world changes, the agent should adapt its behavior to the situation such that it continues to make progress on its long-term goals (i.e., it cannot just be reactive). Because environments have consistent (or slowly changing) dynamics, an agent can make predictions about future states and attempt to act to effect the environment in ways that meet its goals. This capability is useful both in domains that are deterministic and non-deterministic. Different sources of knowledge available in the environment can be used to formulate goals, to act to achieve them, and to recognize when goals are met or unreachable. Adaptation to the specific environment is important because the agent's existence may span a long period of time (as above), and non-adaptive behavior may influence the survivability or viability of the agent in an application domain.

Adaptivity corresponds to both short-term adaptivity (changes in how the agent responds within an on-going episode of behavior) and long-term adaptivity (acquiring new general knowledge to improve long-term performance). Adaptivity may be a consequence of learning but is not a learning metric per se; instead, adaptivity is focused on measuring the ability of a system to respond appropriately to variation in its environment. The analog of adaptivity in control systems is the region of stability in a non-linear control problem.

Adaptivity can be quantified by comparing the performance gain of an adaptive version of an agent system, to a non-adaptive system. This approach is somewhat related to robustness (below) but does not reduce to it. Both robustness and adaptivity are necessary: robustness provides acceptable performance in unforeseen situations, adaptivity suggests a cognitive architecture simplifies a priori engineering and provides a more efficient solution-authoring process than one in which complete capability has to be specified by the designer, in addition to allowing the system to adapt on its own to the changing dynamics and task requirements of a domain.

Figure 3 illustrates a possible, albeit notional approach to a domain-specific measure of adaptivity. In this approach, adaptivity is the ratio of the size of perturbation in the environment (measured in terms of problem complexity) to the difference in the resulting quality of solution. As the difference in quality of solution increases (presumably, via a poorer quality solution), adaptivity decreases. Similarly, if quality of solution remains constant while the perturbation increases, adaptivity also increases.

In the current state-of-the-art, intelligent systems are generally designed for a given problem complexity and quality of solution, so that any perturbations, even ones in which complexity decreases, quality of solution is likely to decrease (with very sharp decreases as complexity increases, as suggested by the dotted lines in the figure). The solid lines intersecting the baseline complexity point in the figure illustrate a "minimum" standard for adaptivity. In this case, decreasing complexity preserves the baseline QoS and, as problem complexity increases, solution decreases somewhat more gracefully than the state-of-art.

The figure also suggests how robustness and learning (discussed below) can impact overall adaptivity. The stars in the figure represent a particular point on the problem complexity curve. In comparison to a non-adaptive, baseline system, processes in the agent systems that enable robustness to perturbation could result in an improved Quality of Service. In this case, we assume there are many decisions and actions an agent takes to achieve some result. Robustness processes can produce somewhat better intermediate results, leading to improvement in overall QoS at the baseline level of complexity.

Agent learning should result in an improved quality of service as well (whether resulting in improving performance or solution quality). Thus, if learning is effective, it will shift the systems resulting adaptivity up and out, providing improved quality of solution for any particular point in the problem complexity space, as suggested by the third line in the figure.

Additional metrics for rational, effective, adaptive behavior include the performance metrics introduced previously, but with different emphases and points of comparison:

Response time: What is the response time to an unexpected event? Is the system able to resume execution of an existing plan once an interrupting event to which the agent has responded? As one example, Wray and Laird (2003) developed a domain specific metric to illustrate the reaction time of a system in response to a triggering event.

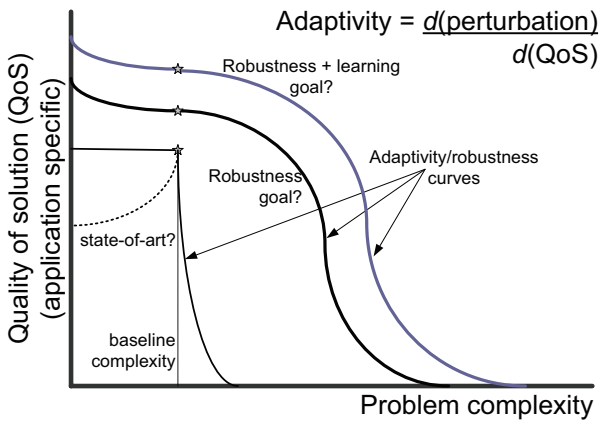

Figure 3: An adaptivity metric.

However, there is no domain general metric for evaluating this aspect of response time.

Scalability in this case is the ability to handle increasingly complex problems. Scalability in the context of this requirement is the ability to adapt to the requirements of specific problems. For example, a system might choose an analytic solution for Traveling Salesman problems up to some system-determined size *n,* then switch to a heuristic method for problems of size greater than *n.* The Traveling Salesman problem represents a problem in which the problem complexity can be systematically scaled; however, collective, subjective judgment is currently the only available approach to ranking domain problems generally along each dimension of complexity.

Use vast amounts of knowledge

There are many different objects in the environment, including other agents. Most objects obey consistent, predictable dynamics, although the agent may not have complete or correct knowledge about these laws and dynamics. These attributes of the environment make "knowledge" a fundamental requirement for intelligent systems. "Knowledge" here really means nothing more than having the means to predict future states of the environment; it does not necessarily imply deep, first-principals knowledge (e.g., the Three Laws of Thermodynamics). However, as the agent's environment becomes more complex (in terms of the objects and interactions it must manage to succeed), it will need increasingly large stores of knowledge to cope with the complexity. Potential metrics for this requirement include:

Knowledge capacity: How much "knowledge" is represented in a cognitive-architecture-based agent? Within a particular symbolic cognitive architecture, it should be relatively straightforward to characterize the size of a knowledge base. However, this metric is difficult to quantify generally. Knowledge content will be especially difficult to quantify in non-symbolic systems. Although enumerating knowledge representations is trivially simple in symbolic systems, simple enumeration can also be misleading (a Soar rule and ACT-R rule correspond to different grain-sizes of cognitive operations; how should ACT-R rules be combined with ACT-R chunks, etc.?)

Knowledge capacity may also have little meaning in architectures (and other intelligent systems) where there is no distinct line between architectural processes and knowledge content. As was observed for many of the performance metrics, knowledge capacity is better employed as a relative measure, than an absolute one.

Figure 4 illustrates one possible use of knowledge capacity. In the ideal case, across a range of tasks and domains, increasing stores of knowledge should result in overall improvements in the average quality of solution. However, the state-of-art today is often that increases in knowledge capacity actually reduce the quality of solution, chiefly due to the additional costs of storing more knowledge encodings and having to search them for retrieval.

Rather than the ideal, in the near-term, a goal of cognitive-architecture-based applications should be to be able to demonstrate that increasing knowledge capacity does not degrade quality of solution. For example, Doorenbos (1994) showed that performance (which is only one measure of quality of service) did not decrease substantially when a research application of Soar was scaled to a million productions. However, in practice, knowledge engineers within the Soar community often do attempt to limit the size of Soar knowledge bases because storing and matching any additional knowledge has some incremental cost, even if small, when implemented on serial hardware. Other architectures have similar limitations, although, as discussed above, it may be that decreasing QoS with increasing knowledge capacity represents more of an engineering issue than a theoretical one.

Knowledge utilization: Knowledge capacity reflects the total content of knowledge that *could* be applied in some situation. However, that capacity may largely be latent for any particular application (especially when increasing incrementality is an explicit goal, as described previously). Knowledge utilization reflects the knowledge that is actually used in the execution of a set of tasks. A straightforward way to measure knowledge utilization is to count the unique instances of each knowledge representation activated or applied in the course of performing a representative sampling of tasks within a domain.

A representative sampling is necessary because any single task instance may only activate part of the utilized knowledge store. TacAir-Soar offers another good example. The TacAir-Soar knowledge base spans many different missions a military pilot might fly; everything from fighter intercepts to flying air refueling missions. For any particular mission, knowledge utilization is likely to be low, but, across the span of all missions, we would assume knowledge utilization would be close to 100% within this single application.

Learning complicates knowledge utilization measures. For example, both Soar and ACT-R include mechanisms of compiling / composing production firings (Laird, et al, 1986; Taatgen & and Lee, 2003). In similar, future situations, the newly-created productions will likely supplant the original ones. Naively, a simple way to avoid this problem would be to "start counting" with the original

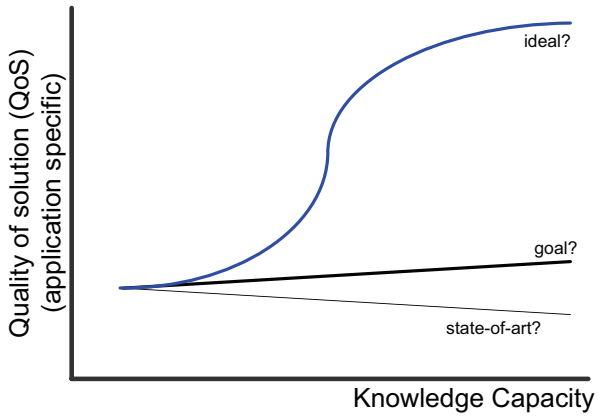

Figure 4: Knowledge capacity.

knowledge base. However, from the point of view of an agent who's knowledge base in increasing in capacity within and across domains, it may be undesirable (or infeasible) to perform some task without the learned knowledge representations.

As suggested, performance metrics also interact strongly with the notion of knowledge capacity and knowledge utilization:

Response time: How does response time change when across orders-of-magnitude differences in the knowledge capacity of a particular system? Does response time change as knowledge utilization changes?

Cognitive cycles/cognitive operations per second: How does the basic cycle time change across orders-of-magnitude differences in the encoded "knowledge" in a particular system? In Figure 4, the limitation of the state-of-art is assumed to derive from a decrease in the cycles/second with increasing knowledge.

System memory footprint: How do memory requirements scale with knowledge capacity?

Behave robustly in the face of error, the unexpected, and the unknown

An agent will always have incomplete or partially incorrect knowledge of the many objects and other agents that appear in its environment. Yet, in order to thrive, it must overcome these limitations and complexities in the environment and behave robustly. The environment itself provides some important aid in this respect. First, the environment usually has structure and abstractions that alleviate the unpredictability of the situation. A shopping agent may not have experience with the specific website just encountered in the execution of a product search, but previous experience with and knowledge of similar websites makes the situation more predictable and provides suggestions for courses of action that increase the likelihood of a successful transaction. Second, the environment provides many sources of knowledge including direct experience, observation of others, instruction, etc. These sources of knowledge can be draw on (and learned) in order to encounter the inherent uncertainty in the environment more readily.

Robustness is the ability to successfully (autonomously/dynamically/safely) withstand perturbations in expected events and tasks. There are no existing, general metrics for robustness, although domain specific metrics have been developed (Nielsen, Beard et al, 2002). One possibility would be to cast robustness as the ratio of the degree of success vs. the degree of perturbation, which is a specialization of the adaptivity metric discussed above. However, both of these measures will be domain and task dependent.

Robustness has a direct relationship with the notion of taskability introduced earlier. The primary difference is that taskability focuses on the ability of the system to handle variation in tasks, while robustness primarily focuses on success in environments where the expected dynamics are changing.

Stochastic assimilation is the ability of the system to capture and reflect in behavior the stochastic character of the environment. Where robustness reflects the ability to recover from unexpected events, in any real application an agent will also need to make rational choices in an environment where those choices are governed by probability distributions. As an example, ACT-R has been applied to a range of non-deterministic games (West, et al, 2006) and has demonstrated its ability to learn the stochastic dynamics of these games. An obvious approach to measuring stochastic assimilation within a domain is to measure the change in QoS over time. We have not yet considered a domain-general formulation of this measure.

Integrate diverse knowledge.

The objects and other agents in the environment result in many different sources of knowledge. To act appropriately, the agent must integrate its knowledge of these different objects to act appropriately in a situation. For example, in an evidence marshalling task, such as "detective's helper," the system must integrate knowledge from "scene of the crime" reports, draw on past experience, be able to reason deductively as well as by abduction and analogy, use general knowledge (language, ontology, etc.) as well as domain specific knowledge (such as the typical etiologies of particular crimes). While the task could possibly be accomplished without multiple sources of knowledge, the assumption is that the introduction of a much larger branching factor in both knowledge search and problem search is offset by the ability to reach a conclusion in just a few steps.

A key aspect of this requirement is the ability to integrate *diverse* sources of knowledge effectively. As another example, consider an agent that must conduct a TSP tour over an actual landscape, with latitude and longitude coordinates of "cities," roads, obstacles, varying constraints (e.g., the fastest tour, vs. the shortest). A common approach to a problem like this is to attempt to map these diverse constraints into an edge-cost that facilitates an algorithmic solution, such as the application of Dijkstra's algorithm. Cognitive architectures typically enable a more open-ended approach to the problem, where individual aspects of the problem can be encoded directly (and with comparatively little information loss) and then combined at run-time to provide a solution.

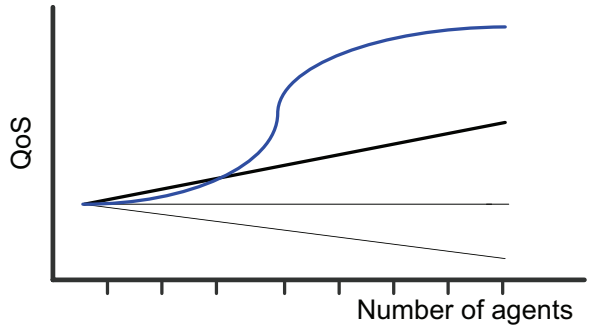

Figure 5: Benefits of social agency.

At present, we have not yet developed metrics for knowledge integration. Knowledge capacity and knowledge utilization express coarse aspects of the general capability, but are not themselves sufficient to express this requirement.

Behave autonomously in a social environment

Over a long life of continual behavior in the environment, an agent will pursue its own success and act on its own. However, the agent may live in a social environment with other agents and actors. Other agents can complicate ultimate success (competitors), but may also be the source of additional knowledge and cooperation. Other actors require that the system be knowledgeable of them (able to predict actions and evaluate intents) as well as the ability to communicate with the other actors.

Scalability: How many "cognitive agents" can interact together, given some baseline performance measure? Ideally, overall performance costs will increase at most linearly with the addition of multiple agents.

Figure 5 suggests some of the potential impacts of multiple agents. In the degenerate case, represented by the dotted line, the addition of additional agents decreases the quality of solution. In the neutral case, represented by the horizontal line, the addition of more agents does not affect the resulting quality of solution. In this case, additional agents are not providing benefit to the agent performing its task. The heavier, straight line suggests a linear benefit of the addition of more agents. In the ideal case, there is a synergistic benefit, with the addition of more agents significantly increasing to overall quality of solution. This latter effect is a common goal of many multiagent systems technologies, such as swarming (e.g., Brueckner and Parunak, 2002). Demonstrating such benefits with cognitive-architecture-based agents has not generally been undertaken.

Exhibit self-awareness and a sense of self

Because existence is long term, an agent will have many opportunities to recognize deficiencies in its knowledge of the environment, and can utilize the many different sources of knowledge in the environment to address the deficiencies. "Self-awareness" is the capability to recognize these opportunities to reflect on the state of one's self and one's behavior and to improve future action

by evaluating the efficacy of actions taken in the current situation. Self-awareness can also include notions of performance monitoring and fault localization within the overall system (i.e., extending beyond the cognitive components of the system).

Adaptivity and **Robustness** are a result of meta-cognitive capabilities, but we have not yet developed a metric that would reflect meta-cognitive capabilities generally. A subjective approach would be to enumerate and define the kinds of processes available in the system for meta-cognitive activity. For example, Soar's automatic subgoaling / impasse mechanism are assumed to provide the basis for meta-cognitive capabilities in Soar, although the basic architectural processes must be completed by encoded knowledge as well.

Learn from its environment

Can the system produce a breadth of different types of learning and improve its function? If the world is consistent and the agent's knowledge is incomplete (as will almost always be the case), then an obvious requirement for long-term success in the environment is learning. Learning, which will draw from the many sources of knowledge in the environment, re-shapes behavior. In the ideal case, learning improves outcomes of future experiences in comparison to past, similar ones.

An intelligent system must not only respond to its environment, but it should respond in a manner appropriate for the situation. In particular, as the world changes, the agent should adapt its behavior to the situation such that it continues to make progress on its long-term goals (i.e., it cannot just be reactive). Because environments have consistent (or slowly changing) dynamics, an agent can make predictions about future states and attempt to act to effect the environment in ways that meet its goals. Different sources of knowledge available in the environment can be used to formulate goals, to act to achieve them, and to recognize when goals are met or unreachable. As discussed above, learning is reflected in improving **adaptivity** within a specific environment. Non-adaptive behavior may influence the survivability or viability of the agent in an application domain

Performance measures also interact strongly with learning. The agent's existence may span a long period of time and the cognitive architecture must strike a solution to the utility problem (Holder, 1990), such that learning increases the quality of solution with experience, rather than decreasing it.

Conclusions

Evaluating cognitive architectures has proven difficult, because both their theoretical and practical value is often only emergent from a breadth of application demonstrations. Cognitive-architecture benchmarks tend to look especially poor in performance-oriented benchmarking, because they typically include an integrated collection of processes and mechanisms, some of which may not significant value in a given benchmark task. We have proposed a number of specific metrics, organized according to a general list of requirements for intelligence, which could be used to measure some of the (assumed) utilitarian advantages of cognitive architectures as general tools for building intelligent systems.

References

1. Anderson, J. R., & Lebiere, C. (2003). The Newell test for a theory of cognition. *Behavioral and Brain Science, 26*, 587-637.
2. Brueckner, S. A., and Parunak, H. V. D. (2002). Swarming Agents for Distributed Pattern Detection and Classification. *In* "AAMAS Workshop on Ubiquitous Computing", Bologna, Italy.
3. Doorenbos, R. B. (1994). Combining left and right unlinking for matching a large number of learned rules. *In* "Twelfth National Conference on Artificial Intelligence (AAAI-94)". AAAI Press, Seattle, Washington.
4. Hanks, S., Pollack, M. E., & Cohen, P. R. (1993). Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures. *AI Magazine, 14,* 17-42.
5. Holder, L. B. (1990). The General Utility Problem in Machine Learning. *In* "Machine Learning: Proceedings of the Seventh International Conference", pp. 402-410. Morgan Kaufmann Publishers, San Mateo, CA.
6. Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P. G., and Koss, F. V. (1999). Automated Intelligent Pilots for Combat Flight Simulation. *AI Magazine* **20**, 27-42.
7. Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning* **1**, 11-46.
8. Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, Massachusetts: Harvard University Press.
9. Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Prentice-Hall.
10. Taatgen, N. A., and Lee, F. J. (2003). Production Compilation: A simple mechanism to model Complex Skill Acquisition. *Human Factors* **45**, 61-76.
11. West, R. L., Lebiere, C. & Bothell, D. J. (2006). Cognitive architectures, game playing and human evolution. In Sun, R. (Ed) Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation. NY, NY: Cambridge University Press
12. Wray, R. E., & Laird, J. E. (2003). An architectural approach to consistency in hierarchical execution. *Journal of Artificial Intelligence Research*, 19, 355-398.