Tuning Experiments with an Adaptive Computational Framework

Joseph Lewis
San Diego State University
5500 Campanile Dr.
San Diego, CA 92182
01-619-594-2014
lewis@cs.sdsu.edu

Chris Ansoff
San Diego State University
5500 Campanile Dr.
San Diego, CA 92182
01-619-594-2014
cansoff@san.rr.com

Categories and Subject Descriptors
I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *coherence and coordination, intelligent agents, multi-agent systems.*

General Terms
Design, Experimentation, Measurement.

Keywords
Adaptive computation, self-organization, complex systems, parameter tuning, Copycat, Starcat.

Complex systems can be defined as those in which collective behavior arises from among the interactions of a large number of locally acting agents with each other and the environment. They have recently become the subject of much investigation. Their behavior can exhibit a fluid-like sensitivity, both to pressures internal to the system and to external context from the environment. A number of computational architectures have been produced which adapt their behavior to a changing environment (e.g., classifier systems). The work presented here builds upon and refines the ideas of the Copycat architecture of Hofstadter and Mitchell.

The Copycat architecture has three components: the slipnet, the workspace, and the coderack. The slipnet is a set of nodes and links among those nodes that together correspond roughly to the concepts of the problem domain and the relationships between those concepts. The workspace is a perceptual arena in which structures are built corresponding to the concepts in the slipnet, as driven by the interactions with the environment. This structure-building is achieved by a flurry of small pieces of executable code known as codelets. The codelets originate from the nodes in the slipnet and are inserted into the coderack, which is essentially a stochastic priority queue. When they leave the coderack, codelets arrive at the workspace to engage in incremental structure-building. In the slipnet, each node gets activation from emitted codelets that are successful in building a corresponding structure. As activation accumulates, a node has an increasing chance of emitting codelets. Such codelets are called top-down, as they are driven by the activation of concepts currently relevant to the program. The whole process gets underway via bottom-up codelets that look for instances of particular concepts in the environment to instantiate in the workspace. Activation spreads among nodes through links and also decays over time. Some of the links between nodes change length, making possible 'slippages' from one node to another. This provides a mechanism for context-dependent behavior. In the workspace each individual structure comes about from the activity of (possibly several) codelets. No individual structure by itself represents a whole perception. Rather, the emergent relationships among the structures are what carry meaning. When a codelet arrives at the workspace it selects, in a biased random manner, some item and proceeds to determine if it can build some new structure related to that item. The bias comes from the activation of related nodes in the slipnet, providing another mechanism for context-dependence and self-organization.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.
ACM 1-59593-186-4/06/0007.

Copycat's original application domain was the completion of analogies between letter-strings, such as "abc -> abd, ijk -> ?". This microdomain allowed for the precise exploration of relevant slippages between concepts under various pressures. Essentially the same architecture has been applied to increasingly complex domains with successful results. These projects include efforts from Hofstadter's research group, such as TableTop, LetterSpirit and Metacat. Other work expanded these ideas to domains with different features and constraints, such as Lewis' Madcat and the object recognition and visual processing by Bolland and Wiles, as well as work by Mitchell and her current graduate students. With this number of applications constructed based on the Copycat architecture, it is natural to ask whether a general framework could be built on these premises, from which particular applications could be instantiated. Further, what new possibilities arise for the use and development of this architecture in the context of such a framework? Lewis and Lawson have described just such an effort in the Starcat project.

Starcat's components—the slipnet, workspace, and coderack—are asynchronous, each running in its own separate thread. The system is entirely event-driven, and the events that are exchanged among components are codelets. Each component is essentially a data structure driven by a metabolism whose activity (the thread for behavior) can be easily parameterized. In Starcat, each component has its own update thread. Lewis and Lawson presented a simple application—an ant colony simulation—that tests their design in a degenerate configuration (i.e. no significant slipnet is used). They have also reported in that this generalized framework highlights the important idea that behavior drives representation and not the other way around. Moreover, the environment triggers changes in the system's functioning but does not specify the nature of those changes. The system's own internal dynamics makes that choice.

Before attempting to build serious applications using this framework, we feel that it is necessary to have a better understanding of the system's functioning under various settings of its many parameters. In order to experiment with the behavior of the Starcat architecture under a variety of parameter settings, we wanted a simple application with which to instantiate the framework that would have elementary behaviors that we could show being sequenced, repeated, and selected. From this we could argue the universality of behavior for this architecture, as well as acquire empirical data about the various parameter settings that give rise to certain kinds of behavior.

The simple application we developed, called Lettercat, has a workspace populated with five individual letters, initially all A's. The task of the program is to cycle each of these letters through the sequence A-B-C-D-E-F and then back around to A, repeating. A run is considered successful when this cycle is repeated indefinitely. A run is considered to fail when the program does not continue this replacement behavior. The slipnet for this application has six nodes, one for each of the letter categories possible in the workspace (A through F). Each node emits codelets which select at random one of the five letters in the workspace. If the selected letter is the same as the source node for the codelet, then that codelet will replace that letter with its successor and add activation to the node corresponding to the succeeding letter, thereby setting it up for its replacement in the cycle. If the codelet does not acquire a letter that matches its source node, it fails to make the substitution and adds activation back to its own source node, thereby keeping that possible replacement active.

In our experiments with Lettercat we have identified 18 independently tunable parameters which have impact on the behavior of the program. We selected five parameters to test based on design analysis and some early experimentation. In our experiments we have attempted to explore the parameter space where the program's behavior is either successful or at its 'edge'. We are essentially trying to identify the edges of the attractor in parameter space for the program's successful behavior. We consider a parameter configuration to produce successful behavior when, out of a set of runs with that configuration, all execute successfully. A configuration is not successful when all of these runs fail that criterion. We call it edge behavior when there is a mixture of these two.

There are five basic parameters that our experiments vary. The first parameter varied was the interval for the update of the slipnet; we used the values of 1, 5, and 10 milliseconds. For each of these settings we varied a second parameter: the connectivity of the slipnet. We used three different slipnets. One simply had nodes corresponding to each of the letters for which the program attempts to make substitutions; there were no connections between them for spreading activation. The second slipnet had a ring topology—that is, each node had a link to its successive neighbor in the alphabet. Finally, there was a slipnet with full connectivity among the nodes. For each of these configurations we then varied the number of codelets produced by each slipnet node during the slipnet update, with values of 1, 5, and 10. Then for each of these we varied the amount of activation added to slipnet nodes upon success of their emitted codelets, taking on values 20, 25, and 50. Finally, for each of these values we varied the amount of activation added to slipnet nodes upon failure of their emitted codelets, taking on the values 0, 5, and 10. The remaining parameters were kept constant. With these five parameters and their chosen values, we have a total of 243 possible experiments. With the no-link slipnet, successful execution was found when the number of codelets was between 5 and 10, the success activation was 50 and the failure activation was 5 or 10. The edge of successful behavior was seen when the success activation was at 25 with 10 codelets and 5, 10 failure activation. With the medium-links slipnet, the edge of successful behavior was much lower than with the no-link slipnet. With the fully-linked slipnet, there was successful behavior at all parameter settings even when the neighbor links in a configuration were set at a relatively high length of 85 and the remaining links (across the ring of neighbor connections) were set to a length of 95. We believe this illustrates the importance of the slipnet and its links in maintaining program execution through high activation levels.

We maintain the following order of importance for the parameters varied in our experiments: 1) the slipnet, 2) the ratio of success to failure activation from the codelets, and 3) the actual number of codelets emitted by active slipnet nodes. Incidentally, we do see a breakdown of program execution when this number of emitted codelets is too high as well.

There is a peculiar phenomenon seen in some of the experiments that we refer to as 'lockstep behavior'. As the system was executing, all the letters in the workspace would change to the same letter before updating to a subsequent letter. For some test cases, the system would eventually stop (or hang), whereas in others, the system would eventually break out of this behavior. Further experimentation is required to determine what is responsible for this phenomenon. In test cases near the edge, the initial configuration of the slipnet activation was an important factor on the system having the ability to produce successful behavior indefinitely. This framework targets problems where the ongoing production of behavior is paramount, therefore initializing codelets will matter for maintaining node activity and the robustness of system behavior.

We feel that the basic values for the rates of activity on threads and slipnet nodes established in these experiments is sufficient groundwork for moving forward with similar experiments on other, more realistic application domains. The simple test application was shown to be too limited to discriminate between different rates of update for the slipnet, indicating that such tuning experiments would be an important part of the pre-deployment phase of any application instantiated with the framework. Also, initial activation mechanisms were shown to be important. We believe that these experiments show that the choices in the current design of Starcat are viable, in that similar behaviors to past applications are supported. Moreover these choices give more flexibility and robustness to the architecture, amplifying the biological inspiration that was paramount to the original work.