References

Booker, L, (1982). *Intelligent behavior as an adaptation to the task environment*, Ph.D. Dissertation (Computer and Communication Sciences). The University of Michigan.

Goldberg, D. E. (1991). Don't worry, be messy. In R. K. Belew & L. B. Booker (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 24-30). San Mateo, CA: Morgan Kaufmann.

Holland, J. H. (1990). Concerning the emergence of tag-mediated lookahead in classifier systems. *Physica D*, **41**, 188-201.

Koza, J. R. (1991). Evolution and co-evolution of computer programs to control independently-acting agents. In J.-A. Meyer & S. W. Wilson (eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior* (pp. 366-375). Cambridge, MA: MIT Press.

Riolo, R. L. (1991). Lookahead planning and latent learning in a classifier system. In J.-A. Meyer & S. W. Wilson (eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior* (pp. 316-326). Cambridge, MA: MIT Press.

Shu, L. & Schaeffer, J. (1991). HCS: Adding Hierarchies to classifier systems. In R. K. Belew & L. B. Booker (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 339-345). San Mateo, CA: Morgan Kaufmann.

Wilson, S. W. (1987). Hierarchical credit allocation in a classifier system. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 217-220). Los Altos, CA: Morgan Kaufmann.

Wilson, S. W., and Goldberg, D. E. (1989). A critical review of classifier systems. In J. D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 244-255). San Mateo, CA: Morgan Kaufmann.

mined only, as now, by the representation and the mechanics of the GA.

Many questions surround this proposal. Will it in fact alleviate the Discovery Problem, permitting us to dispense with ad hoc operators? The key here is "evaluability without vacuity". Will classifier conditions based on complete logic (including especially the `or` function) result in genetic offspring that tend, *ab initio*, to match, but at the same time contain enough specificity to fuel the GA productively? Or will the new logic simply expand the space in which matching must occur, while the `or` function becomes just a new road to over-generalization? It seems impossible to answer without some exploratory empirical research, which we hope to have done by the time of the Workshop.

In general, our approach is to try to "grow" the complexity of classifier conditions and messages in accordance with the actual demands of a problem, not an arbitrary encoding. Our specific proposal defines variables in advance (via the Kozan "terminal set") but they do not necessarily appear in classifier conditions or messages until they are needed for purposes of increased discrimination. In this the proposal has some resemblance to the messy GA (Goldberg 1991).

A different way to "grow" classifier complexity would be to create more refined detectors, as needed, without defining them or their variables in advance. For instance, a system might at one stage have only one broadly aimed thresholded light detector on its left side and a second one likewise on its right. All classifiers that used these detectors would certainly get evaluated, since the space involved would have only two bits. Later, when greater discrimination became advantageous, the detectors could split, increasing resolution, but in a way that would present only an incrementally more difficult search problem for the GA. In nature, this sort of thing is of course done over the generations, which apportions the search between phylogeny and ontogeny. It is conceivable, though, that something similar could be accomplished in artificial ontogeny, and we intend to investigate this too.

tion for classifier conditions was designed to be easy for genetic algorithm manipulation, with lots of #'s to solve the matching problem. Now, however, with the development of methods for doing genetic algorithms on arbitrary Lisp S-expressions (Koza 1991), the opportunity is available to go to fully general predicates for classifier conditions. Our hypothesis is that doing this will greatly mitigate the Discovery Problem.

The first part of our proposal is thus to represent classifier conditions as S-expressions based for instance on `or`, `and`, and `not` functions of the underlying binary variables. A particular problem may also involve some continuous variables. In that case, the function set could well include functions whose truth value depends on the arguments being within a range. This would permit conditions like "it's more or less dark now", or "his temperature is above normal" to be tested directly upon the environmental variables. Genetic operations on classifier conditions would take place along the lines developed by Koza.

The second part of our proposal has to do with internal messages. As outlined earlier, the standard system has internal messages which may be lengthy bit strings—even though the system's temporary memory requirements may be small. This length exacerbates the Discovery Problem because to produce chaining, classifiers must be generated that match these long messages and that is not easily done without, again, initializing with big percentages of #'s.

We propose to replace the standard message list with a single *message register*, and to make the consequence (message or action) part of a classifier a single S-expression that sets or clears one or more mentioned bits in the message register. The system would be initialized with very simple message S-expressions, e.g. `(setbit b2)`. Correspondingly, as previously outlined, classifier conditions could have S-expressions as simple as `(or b2)`.

As the system evolved, the message register would see various bits set, with combinations thereof sometimes being read as a unit, but the specificity of the register's "message" should never exceed what the system actually needs, and so would not have an arbitrary specificity deter-

Our approach is to design the classifier system so that, regardless of the characteristics of the problem space, most classifiers will match and be evaluable. Then the GA should be able to generate whatever is needed, without recourse to special operators.

As it stands now, the classifier system definition is stacked against evaluability. One usually has environmental messages that are relatively long and immediately place the system within a very large classifier space. Furthermore, perhaps in the interest of clearer exposition, the system's internal messages are by definition equal in length to environmental messages—even though in context many systems really should not need temporarily to remember too many different things. Of course, the standard classifier system has relatively long messages because after the system has learned, considerable message detail may be called for. And to get from its birth to that point the standard system starts with a liberal number of #'s, so that there ought to be no matching problem.

In practice this has not been an adequate solution, as shown by the later recourse to ad hoc operators, but also by the fact that systems initialized with high # percentages tend to evolve to overgenerality, having too few specific schemata for the GA to get going. What is needed is a different representation scheme that (1) permits sufficient initial generality while (2) affording a source of specific schemata that can be built upon. Let us look at the current representation.

A classifier's condition is encoded by one or more taxa, each a conjunct of some subset of the available (usually binary) variables and their complements. In effect, the condition is a predicate, but a logically incomplete one since it is a single conjunct. In particular, there is no ability to represent logical `or`. That can only be done using more than one classifier, and each must output the same message. However, `or` is an operation that permits generality without at the same time ignoring variables, as # does. For example `(or x1 x2 .. x7 .. xk)` is very general—it could mean "something going on on my left, I don't know just what"—yet at the same time it preserves mention of its arguments. A more specific condition is achieved for example by dropping mentioned variables, instead of adding them as in the standard system. The original 1,0,# nota-

Schaeffer (1991) implemented this idea in part but much further work needs to be done.

The Time Problem results from the observation that in behavior, the unit of action may range in duration from less than a second (e.g., turning your head, reaching for something) to minutes, hours, or longer (getting dressed, getting a degree, etc.). Classifier systems, on the other hand, have only one explicit unit of time, the "time-step" associated with each cycle of operation. Clearly the system must have ways of packaging reinforced action sequences into higher-order units so that bucket-brigade sequences do not become impractically long. Yet achieving this has proved difficult under the current classifier system operating principles. A more hierarchical system has been proposed (Wilson 1987), and the look-ahead work (Holland 1990, Riolo 1991) may also bear on the Time Problem.

Our subject here is the Discovery Problem. A classifier system must discover its own suitable classifiers. "Classically", this occurs through the GA. In practice, in many systems it has been difficult to get good classifiers through the GA alone. To get around this, special discovery operators like "create", "cover", and "triggered coupling", etc. have been introduced. They essentially make new classifiers based on actual inputs to the system, then use the GA for further refinement. But, why are these operators necessary? Why does the GA fail us?

I think there is a simple reason, which can be overcome. If one examines the classifier systems that have worked without use of ad hoc operators, it is always the case that the conditions of a substantial subset of all possible classifiers under that coding will be satisfied by states actually occurring in the problem. This means that genetic operators will usually produce a classifier that can be evaluated, i.e., that matches eventually and is not useless. On the other hand, in a problem where the set of evaluable classifiers is a small, often minute, subset of the classifier space, most offspring never match, so the system's search hits a brick wall. Booker's (1982) partial matching is an important possible answer to this problem, but here we will propose an approach that retains the standard notion of match.

Toward a GA Solution to the Discovery Problem

Stewart W. Wilson
The Rowland Institute for Science
100 Cambridge Parkway
Cambridge, MA 02142 USA
(wilson@smith.rowland.org)

Submitted to The International Workshop on Learning Classifier Systems

Extended Abstract

Classifier systems are an intriguing idea that, so far, has been hard to make work. The difficulties fall into three areas that I call the Discovery Problem, the Cooperation Problem, and the Time Problem. Here I will address the Discovery Problem, but the others deserve some mention.

The Cooperation Problem arises because in a classifier system, classifiers must often cooperate—and yet are also in competition under the GA. Classifiers may cooperate diachronically, forming chains along which payoff flows. Each member of a chain is dependent on the others: the early on the late and vice versa. So if the success of one member leads it to reproduce to the detriment of another member, it in effect shoots itself in the foot. Classifiers also may cooperate synchronically, sharing in the proper representation of a situation at one time. The same dilemma holds: a classifier's ultimate fitness may well depend on its contemporaries also being there—yet GA competition acts against this.

Cooperation in these senses means the fitness of classifier A depends positively on the presence of classifier B and vice versa. One solution to the Cooperation Problem is to abandon classifier systems and evolve the whole system as a unit, as in the Pitt approach. A less drastic solution was proposed by Wilson and Goldberg (1989), called the corporate classifier system. Classifiers could adaptively form clusters of one or more members that would reproduce and be deleted as a unit—so cooperators would, if they belonged to the same cluster, no longer compete. Shu and