

# Simulated Evolution of Language and Language Processors\*

David Beaver

Dept. of Linguistics, Stanford University

## 1 Specific Aims

Recent developments in Computer Science have created a new research paradigm, that of Artificial Life (*ALife*). *ALife* concerns the study of biological or biologically inspired systems using computer simulation. The aim of the proposed project is to provide proof-of-concept results which demonstrate the applicability of *ALife* techniques to problems in linguistics and natural language processing (*NLP* — that subfield of AI and Linguistics the goal of which is to build devices that can interact with humans using natural language). More specifically, we aim to show that:

1. *ALife* techniques can be used to develop efficient *parsing* algorithms (see below) that might have application in commercial *NLP* products.
2. *ALife* techniques enable the study of the emergence of communicative conventions in coordinated activity, and the development of robust communication protocols for artificial agents (robots) engaged in such activities.

Although there has been some previous work on applying *ALife* to the study of language, the specific problems which will be tackled within the proposed project have so far never been studied. To our knowledge, furthermore, *ALife* techniques have never been utilized in the study or design of *NLP* applications.<sup>1</sup> As a result of this paucity of previous work, the basic ideas that will be presented can be understood without familiarity with any large body of previous literature. It will therefore be possible, even within the confines of this relatively short proposal, to describe the proposed program of research in non-technical terms.

## 2 Experimental Methods and Procedures

### 2.1 Genetic Programming

Of the techniques applied within this field, the proposed work will specifically make use of Genetic Programming (GP).<sup>2</sup> GP provides a way of automatically discovering solutions to problems, particularly algorithmic problems. An example application chosen from outside of the domain of language should illustrate the general principles of the GP method, as well as leading us into a discussion more directly pertinent to the proposed project.

The “Traveling Salesman Problem”, a well known combinatorial optimization problem, studied extensively in Operations Research and Complexity Theory, is the problem of finding the shortest tour passing through a set of points in some space. GP might be used either to find a particular instance of a solution to this problem, or to search for general algorithms which could be applied to arbitrary problem instances. We will concentrate on the latter case. Put another way, what we are seeking is not a particular route, but a decision procedure that a sales representative might use whenever he or she hits the road.

Normally, in order to obtain such a decision procedure, one envisages constructing the algorithm, perhaps in flowchart form, out of basic operations. But the GP approach reverses this: given some definition of what the basic operations are, there is a space of potentially appropriate decision procedures. The GP task is to search this space and ‘discover’ solutions. An analogy may be drawn with a sculptor who, rather than starting out with a view of what must be carved, attempts to ‘find’ the sculpture that lies dormant inside the stone.

Suppose that we define the problems that our putative “Traveling Salesman” algorithm must solve in terms of a list of tuples of coordinates, where these tuples define the positions of the towns to be visited. A solution to a particular instance of the salesman problem would then be a particular ordering of the list. More generally, the algorithm being sought would be one that took an arbitrary initial list, and re-ordered it appropriately.

Basic operations used in a solution might involve permuting elements on the list, establishing the distance between two elements, adding up distances, testing whether one number was greater than another, and halting. A set of such operations can be defined to form a basic vocabulary of program constructs, such that the space of all programs built from these constructs is the one that must be searched in order to find a useful decision procedure.

Although the total space of algorithms involving such operations has infinite extent, GP provides a mechanism for searching through that space. Initially a population of programs is built out of the basic program constructs, typically at random, but potentially also as mutations of some known solution program. The relative success of different members of the population is evaluated using a *fitness function*. In the case at hand, the fitness function might be based on a combination of (1) the number of steps required by the program to solve a randomly generated problem instance, and (2) the distance the salesman must travel on that solution.

Once fitness has been evaluated, a new generation of programs is created. These programs are formed by combining parts of the code from the previous population,

---

\*The proposed scheme of work grows out of a concept paper [3] drawn up by the principal researcher with colleagues at the University of Amsterdam and the University of Utrecht.

<sup>1</sup>Work of Sampson [20] comes closest.

<sup>2</sup>GP was developed by Koza [15], as a generalization of Holland’s earlier Genetic Algorithms [8].

and by introducing occasional random mutations. Importantly, a weighting is introduced such the more successful a program, the greater (statistically) the amount of its code carried over to the following generation. This ‘unnatural selection’ is what can lead to an overall increase in fitness from one generation to the next. By iterating the process of evaluating fitness and producing new populations sufficiently many times, we may hope to automatically discover algorithms which provide good routing advice in reasonable time.

## 2.2 The Traveling Sales Team Problem

The first simulation which we propose to carry out in the project concerns a problem introduced in [3]. The problem is this: Given a list of places that must each be visited by at least one member of a sales team, how should the team members coordinate their activities? This problem provides a simple illustration of how ALife methods will be applied to study the development of communicative conventions in coordinated activity.

We propose studying groups of algorithms, run in parallel, and given the capacity to share information. This communicative capacity is obtained by defining what might naturally be called *global variables*, variables which are commonly accessible to both algorithms. We may think of such variables as providing a blackboard, such that as both algorithms are run, information may be placed on the blackboard, or wiped off, by either agent.

Note that this changes the character of the algorithms. In the Traveling Salesman case the algorithms arrived at could be regarded as procedural definitions of functions from input lists to output lists. But in the presence of a commonly accessible blackboard, it becomes inappropriate to conceive of the algorithms this way. Rather, they are best thought of as rules defining the behavior of semi-autonomous agents.

In order for interesting agents to develop, the menu of basic program constructs must be altered somewhat. Clearly, there are many alternatives to be considered, but for the initial simulation we restrict ourselves to a minimal variant of the Traveling Salesman problem. In particular, we start by considering the case in which:

1. The sales team has only two members.
2. The *blackboard* consists of a list of locations to be visited, and a pair of ‘boxes’. Each box may hold a 1 or a 0 or nothing. The blackboard always holds the complete list of locations, with the boxes initially empty.
3. Each agent has an internal variable interpreted as a pointer to the current position they are addressing on the blackboard. Basic operations include moving the pointer one up or one down the list.
4. We give the agents the ability to copy the location listed at their current blackboard position onto their route list, and to read, write or erase 1s and 0s in each of the two boxes by that location.

5. The evaluation proceeds by alternating between the two agents, executing one step at a time.
6. The agents are able to test whether the other agent has halted.
7. The evaluation stops when both programs have halted, or after some arbitrary large number of program steps.

It should be noted that the population under study at each stage consists not of two agents, but of a large number of pairs of agents, with each ‘sales-pair’ being based on a different algorithm from each of the other pairs.

There are many variants to be considered. For instance, the number of agents could be increased, or each agent might be given additional memory — an extra readable and writable blackboard invisible to the other agent. Given that at present we are not able to guarantee that any effective algorithms would evolve under our initial assumptions, speculating further on these alternatives is premature.

## 2.3 The Database Coordination Task

We will consider the situation in which a number of agents each possess parts of a database of facts. The task the agents have is to swap information with each other such that all agents’ databases contain as much correct information as possible.

The databases will be realized using lists of simple one and two place predications, e.g. `professor(mary)`, `student(john)`, `supervises(mary, john)`. Partiality of an agent’s database will be represented by omitting predicates or constants in an obvious way, e.g. `professor(?)`, `?(john)`, `supervises(mary,?)`.

A generation consists of a number of identically sized groups of agents, such that in a given group all agents utilize the same algorithm. At the beginning of each generation each agent in a group is given an incomplete copy of the same randomly generated database, the incompleteness itself arising by random insertion of question marks. For each successive generation, information is initially distributed uniformly across the different groups: if in one group there is exactly one agent who has the incomplete fact `professor(?)`, then in all other groups in that generation there will also be one such agent.<sup>3</sup>

---

<sup>3</sup>It will be important to investigate the effects of allowing groups of agents to be heterogeneous, that is, to involve agents with different algorithms. In that case, the fitness function would have to be based on the amount of information an individual usefully exchanged, and not on the total number of ‘knowledge gaps’ filled in by the whole group. Note that previous authors exploring the evolution of communication have varied in this choice, whilst still attaining positive results: e.g. Koza [14] studied algorithmically homogeneous ants in colonies, whilst Briscoe [4] and Batali [1] both studied systems in which communication occurred within heterogeneous groups. By allowing heterogeneous groups, it becomes possible to restrict study to a single group per generation. Thus fitness of agents in a single generation may be evaluated at lower computational cost.

In simulating the development of a group's information, agents will be paired at random with each other for a fixed period, and whilst paired will be able to exchange information. Each pairing communicates using an initially clean *blackboard* which can hold at most one message. Agents are able to copy messages from some part of their internal database to the blackboard, or in the reverse direction, and are able to erase the contents of the blackboard. Messages will consist either of a predicate symbol, or of a constant symbol, or be drawn from a set of symbols not used in the database: call these *punctuation marks*. An additional internal book-keeping register acting as a pointer into the agents' database must also be available in order to define which part of the database is being addressed, along with operations for modifying this register.

A notion of *communicative effort* can now be introduced, corresponding intuitively to the amount of chalk used. More precisely, the effort is proportional to the total number of predicate symbols and constant symbols used, with no charge for punctuation. One group will be identified as fitter than another if there are fewer mistakes or omissions in the agents' databases relative to the original database, or, failing that, if the first group expends less communicative effort.

In this model, we can say that the basic elements of communication, namely a commonly agreed upon language for talking about agents' knowledge, are given.<sup>4</sup> What we hope to observe in this simulation is development of the following, which might be termed *communicative protocols*, or *pragmatic conventions*:

1. Turn taking, whereby one agent writes a sequence of words, and the next reads the sequence of words, before the second writes and the first reads.
2. Use of sequences of words to convey predications.
3. Questions, methods whereby an agent specifies what information is required.
4. *Elliptic* answers to questions, in which only needed information is conveyed.
5. *Discourse structure*, whereby a sequence of communicative acts share a common topic.
6. Subject to the introduction of noise (by occasional random modifications to the blackboard), development of protocols for exchanging information in noisy environments ("Pardon?", "Did you hear me", repetition of signals).

---

<sup>4</sup>According to time available, we will also specify and implement a variant simulation, in which the hard-wired link between message symbols and database symbols is broken. That is to say, agents will not be given operations for copying bits of the database onto the blackboard. Rather, they will be given additional set of symbols to be used for communication, and a set of operations for 'learning' symbols by guessing their meaning. The basic principles of such a system are clear from previous work, such as [17, 16, 22, 19, 2, 4, 12, 11, 13, 9]. However, a word of caution: the complexity of the additional specification and programming task is large, and the amount of CPU time that would be needed for agents to develop interesting languages from scratch may also be large. The negative character of the results in [5] may be indicative of this. So it seems likely that such an extension would be better left to a follow-up project.

We postulate that all of these developments would enhance fitness, by reducing the communicative cost of synchronizing agents' information.

## 2.4 An NLP Application: Parsing Algorithms

A parser uses a grammar to syntactically analyze strings of some language. Normally, one thinks of a Computational Linguist inventing a parser by constructing an algorithm. But the approach that will be adopted here, as may by now be expected, will involve viewing the process in reverse: there is a space of potential parsing algorithms, and to obtain a parser we must search that space.

Within the large space of *potential* parsing algorithms, Computational Linguists have invented/discovered a few classes of *actual* parsing algorithm. There can be no doubt that there are many more parsing algorithms within that space which are of comparable or better efficiency than those so far identified. What is meant by *efficiency* here? Loosely, efficiency is measured in terms of how many program steps a parser will take to analyze a given length string. But which string? Formal analyses of parser efficiency normally concentrate on what is called *worst-case time complexity*, the relationship between string length and the longest time taken to parse any string of that length, relative to some class of grammars.

Parsers, and also complexity results, are divorced from linguistics, in so far as they commonly apply to classes of grammar which do not match very closely (as far as is known) the class of human languages. Complexity results typically reveal nothing about how parser performance may vary from one human language to another. Similarly, only a very little is known about how parsers might be tailored to particular languages. Such tailoring could prevent a parser from first attempting analyses which are unlikely to be appropriate in some language, and thus lead to parsers more efficient than those currently available for NLP applications.

In order to automatically discover parsers using GP, we must define the search space of algorithms in which parsers are to be sought, and we must also define a fitness function, the metric according to which different algorithms may be compared. It is proposed that the space of parsers under study be restricted to the class known as *chart* parsers.<sup>5</sup> These are parsers which operate on similar principles to those familiar within the AI community under the heading *blackboard reasoning systems*. In such a system we may say that while one component does the real work, another component — the blackboard — organizes the schedule of work to be done. In the case of a chart parser, the schedule consists of a list of partial analyses of the input string. The scheduler can be thought of as a black box which, at various times during the parse process, permutes the chart list in some way, or sends out requests for fuller analysis of whatever is at the top of the list. The question we

---

<sup>5</sup>See Kay's [10], Thomson's [21] and e.g. the textbook [7].

will ask is, given that other components of the parser are held constant, how do we find a suitable black box?

Before briefly considering how the scheduler might be defined, two alternative (but potentially combinable) approaches to the fitness function will be outlined.

The first proposal for a fitness function requires a separately written *generation* system to produce test data. The test data consists of a set of grammatical sentences and a set of non-grammatical sentences. It is a relatively trivial task to implement a generator which will pseudo-randomly produce well-formed sentences relative to some grammar. The problem of producing ill-formed sentences is, in a sense, more interesting: it is not a task which is often called for. We suggest permuting well-formed strings, and checking that the results are ill-formed with respect to the grammar by attempting to parse them with some standard parser. A simple absolute fitness metric would be based on (1) the percentage of correct classifications of strings as grammatical or ungrammatical, and (2) the number of program operations needed in order to classify all the strings in the problem set.

A second notion of fitness function would require a previously syntactically analyzed corpus, and a grammar capable of providing the given analyses. Fitness would be determined according to how closely the tree returned by the parser matched that in the corpus. In other words, we would be scoring the parser not merely according to whether it provided a syntactic analysis, but according to whether it provided an analysis similar to that produced by a human. It is important to realize that the implementation of such a notion of fitness will require significantly more work than the previous approach described. The main reasons for this are that currently available hand annotated corpora (of which the largest and best known is the *Penn. Tree Bank*) are not based on complete and fully documented grammars, and furthermore may in some cases contain only partial analyses.<sup>6</sup>

Note that, especially as regards the first definition of the fitness function, we might choose between attempting to develop an all-purpose chart parser, and attempting to optimize a chart parser for a specific grammar. To develop an all-purpose chart-parser, we would have to test for fitness with respect to a range of different grammars, possibly generating these grammars automatically, and without respecting any particular constraints of natural language grammars. On the other hand, by far the easier task would be to use just one grammar in a given simulation. In that case, we might expect to produce just the sort of *tailored* parser discussed above.

Regarding the basic program constructs out of which

---

<sup>6</sup>Given that the first notion of fitness function is likely to be significantly simpler to implement than the second, it may be questioned why it is felt necessary to consider the second at all. The main reason concerns a potential weakness of the first fitness notion, namely the fact that it enables selection of parsers only relative to some artificial method of generation. The danger is that the simulation may only result in parsers optimized to a particular method of generation, and not optimal in the face of naturally produced linguistic input.

the scheduler should be built, we have not so far produced a specification, so that this specification task must be carried out within the lifetime of the project. However, some general comments are in order. It has to be said that not much is known about chart parser scheduling. Current systems tend to make use of seemingly *ad hoc* heuristics, there being little (if anything) published on the general theory of such heuristics, and little sign of progress being made on the problem. So the general approach would be to start by using an extensive assortment of basic program constructs, and then later in the development process perhaps try to narrow down on sets of constructs which produced comprehensible, as well as effective, heuristics.<sup>7</sup>

### 3 Anticipated Results and Significance

Positive results for the *Sales Team* and *Database Coordination* simulations would consist in the production of algorithms which solved the given problems significantly better than the best algorithms not using special communicative conventions.<sup>8</sup> Such results would (a) provide support for the assertion that in the future ALife techniques might be used to study the evolution of more complex and human-like systems of communicative conventions, and (b) open the door for use of the same techniques in practical applications involving autonomous agents.

An example should clarify this second point. At the Daimler-Benz Intelligent Systems Laboratory there is a project called "Adaptive Distributive Guidance and Control." The project involves simulated studies of traffic flow with (to simplify somewhat) robot chauffeurs driving the cars. The company is planning to investigate the possibility of introducing inter-chauffeur communication, to see to what extent that might help deal with or avoid awkward traffic situations. But there is currently no theoretical model to indicate what should be communicated, or what conventions should be used. Here we make one simple observation: the type of task being considered at Daimler Benz has a structure which combines elements of the *Sales Team* and *Database Coordination* simulations. The task involves agents that (1) must choose spatial paths (2) have partial information (e.g. about each other's destination and manoeuvring intentions). The existence of such applications indicates

---

<sup>7</sup>In [3] a number of alternative approaches to delimiting the space of parsers are considered. One is to add weightings to grammar rules, and make the scheduler sensitive to these weightings. The simulation would then be used to evolve both schedulers and grammar-rule-weightings in tandem.

<sup>8</sup>In the case of the *Sales Team*, we might find such an algorithm by running the simulation without allowing agents to write on the blackboard. The *Database Coordination* task is more subtle, in that it cannot be solved without two-way communication. But we might, e.g., compare the best algorithm discovered in the normal situation with that discovered in a version of the simulation without the extra punctuation symbols, in order to establish the potency of conventions developed as concerning the punctuation symbols.

that there is potential for industrial interest in the coordinated communication simulations being discussed in this proposal.<sup>9</sup>

As far as the *Parsing Algorithms* simulation is concerned, it even a minor improvement in fitness over currently available parsers would represent a significant result. Parsers with architectures similar to those described above are used extensively in NLP applications, and the parser is often responsible for a processing bottleneck. If faster parsers were available, they could see immediate application due to their potential to clear such bottlenecks.

Furthermore, it should be noted that the proposal is in an important sense *conservative*, in that it potentially provides away of improving parsers without changing overall system architecture. Suppose that we were presented with a working NLP system containing a parser. We might apply optimization techniques very like those described above to create a new parser which had identical input-output behavior to the original parser, but operated slightly faster. The new parsing module could thus be 'slotted in' in place of the old one, without introducing any concerns about unwanted changes to overall system performance. This would not be the case if one replaced the parser with a parser based on a completely different technology.<sup>10</sup>

## References

- [1] John Batali, unpublished m.s. *Computational Simulations of the Emergence of Grammar*, UCSD.
- [2] John Batali, 1994. "Innate Biases and Critical Periods: Combining Evolution and Learning in the Acquisition of Syntax", in R. Brooks & P. Maes, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press, Cambridge, Mass.
- [3] David Beaver, Willem Groeneveld and Breannán Ó Nualláin, Unpublished m.s. *Synthetic Evolution of Communication*, University of Amsterdam.
- [4] E.J. Briscoe, Unpublished m.s. *Language acquisition: the bioprogram hypothesis and the Baldwin Effect* (Under consideration for *Language*.)
- [5] Joseph J. Crumpton. 1994. Evolution of Two Symbol Signals by Simulated Organisms, MS thesis.
- [6] Mark Johnson and Martin Kay, 1990. "Semantic Abstraction and Anaphora", in *Proceedings of COLING 90*, Helsinki, Finland, pp. 17–27.
- [7] Gerald Gazdar and Chris Mellish, 1989. *Natural Language Processing in LISP*, Addison-Wesley.
- [8] John Holland, 1975. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.
- [9] James Hurford, to appear. "Artificially growing a numeral system", in Jadranka Gvozdanovic (ed.), *The history of numeral systems*.
- [10] Martin Kay, 1986. "Algorithm Schemata and Data Structures in Syntactic Processing", in B. Grosz, K. Sparck Kones & B. Webber (eds.), *Readings in Natural Language Processing*, Morgan Kaufmann, Los Altos, pp. 35–70.
- [11] Simon Kirby, to appear. "Fitness and the selective adaptation of language", in J. Hurford, M. Studdart-Kennedy & Chris Knight (eds.) *The Evolution of Human Language*, CUP.
- [12] Simon Kirby, 1996. "Competing motivations and emergence: explaining implicational hierarchies", *Language Typology*, 1.
- [13] Simon Kirby and James Hurford, to appear. "Learning, Culture and Evolution in the Origin of Linguistic Constraints", Proceedings of the 4th European Conference on Artificial Life (1997).
- [14] John Koza, 1992. *Genetic Evolution and Co-Evolution of Computer Programs*, in C. Langton, C. Taylor, J. Doyne Farmer & S. Rasmussen (eds.), *Artificial Life II: The Second Workshop on the Synthesis and Simulation of Living Systems*, Santa Fe Institute Studies in the Sciences of Complexity, proceedings Vol. X, Redwood City, CA: Addison-Wesley, pp. 603–629.
- [15] John Koza, 1992. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass.
- [16] Bruce J. MacLennan and Gordon M. Burghardt, 1993. "Synthetic Ethology and the Evolution of Cooperative Communication", *Adaptive Behavior*, 2:161-187.
- [17] Bruce MacLennan, 1992. "Synthetic Ethology: An Approach to the Study of Communication", in C. Langton, C. Taylor, J. Doyne Farmer & S. Rasmussen (eds.), *Artificial Life II: The Second Workshop on the Synthesis and Simulation of Living Systems*, Santa Fe Institute Studies in the Sciences of Complexity, proceedings Vol. X, Redwood City, CA: Addison-Wesley, pp. 631–658.
- [18] Noble, J., Cliff, D., 1996. *On Simulating the Evolution of Communication*, Cognitive Science Research Paper 420, School of Cognitive and Computing Sciences, University of Sussex.
- [19] M. Oliphant and J. Batali, Unpublished m.s. *Learning and the Emergence of Coordinated Communication*, UCSD. Submitted to *Cognitive Science*.
- [20] Geoffrey Sampson, 1996. *Evolutionary Language Understanding*, Cassell, NY/London.
- [21] Henry Thomson, 1983. "MCHART - A flexible, modular chart parsing framework.", in *Proceedings of the Third Annual Meeting of the American Association for Artificial Intelligence*, AAAI, Stanford, CA.

<sup>9</sup>The head of the relevant Daimler-Benz group is Pat Langley, a consulting professor at Stanford: he has already suggested the possibility of collaborative work on this problem area.

<sup>10</sup>Examples of such technologies would be *statistical analysis* or *stochastic optimization*.

- [22] Gregory Werner and Michael Dyer, 1992. “Evolution of Communication in Artificial Organisms”, in C. Langton, C. Taylor, J. Doyne Farmer & S. Rasmussen (eds.), *Artificial Life II: The Second Workshop on the Synthesis and Simulation of Living Systems*, Santa Fe Institute Studies in the Sciences of Complexity, proceedings Vol. X, Redwood City, CA: Addison-Wesley, pp. 659–689.