Limerick Generation by Evolutionary Computation with Neural Network as Fitness Evaluator: A Simulation of Human Creativity

Robert Levy
Candidate for B.A. Degree
in Cognitive Science and Linguistics

Oswego State University
College Honors Program

December, 2000

Abstract

A computer program can arrange text to fit the meter and rhyme scheme of a limerick, while behaving randomly in other respects. Collections of computer-generated limericks constitute the "first generation" in the abstract model of natural selection applied here. A neural network, trained on human ratings of assorted limericks, generalizes, determining the "fitness" of novel limericks, which may be mutated, recombined, or directly copied. The objective is that the quality of poems should improve after a number of generations. Creativity is defined here as the ability to produce things that are not merely novel, but also unique or valuable in some sense. POEVOLVE, the evolutionary model of limerick composition implemented here, shows signs of real creativity: over a period of time left running fitness values shift upward significantly.

Contents

1 Advice to Future Honors Students

"Carelessly planned projects take three times longer to complete than expected. Carefully planned projects take four times longer to complete than expected, mostly because the planners expect their planning to reduce the time it takes." —

Depending on who you are, you may or may not be looking forward to working on a large scale project that will deeply interest you and steal away most of your waking hours for the next few semesters. If you are the kind of person who is looking forward to getting involved in something like this, I can understand how excited you are. I can both empathize with and pity your lack of self-control when it comes to such consuming passions. My advice to you is to get carried away but to be sure never to neglect your other obligations. However, you may discover at some inopportune moment, at a stage far too advanced, that most "other" aspects of your life have subtly become subsumed by your thesis work and reframed inside its context. If this happens, there is no use fighting to loosen the grip your work now has on you. The only advice I feel licensed to offer is that you remember the bigger picture, and remember that the "picture" can always be "bigger" than what you might be required to imagine right now. On a side note, please remember never to let your work get in the way of your health. The one major exception I can think of is that you may sometimes choose not to sleep. Although this may not always seem to be a good idea at the time, in my experience it has provided boosts in productivity at times when such boosts were needed. As the old cliche goes, the problem with undergrads working on their honors theses is that they have to sleep every few days.

2 Acknowledgements

I want to thank all of the cognitive science faculty and organizers of the cognitive science degree program here at SUNY Oswego. Their efforts have made it possible for me to do my undergraduate thesis project in the exciting area of cognitive science, and to graduate with a degree in cognitive science. I thank my family, especially my parents, for everything. I thank Craig Graci and Vera Kempe who have advised me and frequently worked with me on this project. From the very being they made the project possible and pointed me in the direction of successful completion. The courses they have taught have been for me the most interesting, useful, and memorable aspects of my education because these courses have given me a rare and valuable undergraduate introduction to cognitive science with a much appreciated emphasis on computational modeling approaches. Thanks to David Vampola for encouragement and inspiration, and for influencing my decision to study at SUNY Oswego in the first place. Thanks to David Sargent for distributing limerick questionaires to his class. Lastly, I must thank the many people who provided questions, criticism, interesting and original ideas, and who donated their time filling out questionaires.

3 Introduction

Computer poetry, the practice of writing programs capable of writing their own poetry, is a sparsely populated but extremely interesting area where AI modeling techniques interact with poetic and literary ideas. Though it is largely unexplored the concept of computer poetry is only as new as AI itself. The prospect of a computer sytem autonomously generating a believable sonnet was suggested by Alan Turing in his landmark 1950 paper to illustrate his now famous test of machine intelligence (Turing, 1950). Some recent work with programs that produce poetry can be found in Charles O. Hartman's book *The Virtual Muse* (Hartman, 1996) where he discusses his programs in detail, and in the futurist writer and technologist Ray Kurzweil's *The Age of Spiritual Machines* (Kurzweil, 1999). Kurzweil's program, which started as a hobby and turned into a recent commercial software release, can compose poetry in other poets' styles and in combinations of the styles of different poets, using a number of pattern recognition techniques and heuristics for writing poetry. Similarly, Hartman's programs employ numerous heuristics designed to produce literary effects for the reader to pick up on. In addition to these two events, among others there is a subculture of computer literate poets who compete with one another to see who can produce the best Haiku-writing programs. Outside of poetry there is a considerable amount of work being done exploring musical and and visual expressive potential of autonomous software.

In the development of my own program, POEVOLVE, which this paper exposes and explores in much detail, an aim has been to include a minimum of explicit instruction on how to write poetry, elaborating instead on tailoring a system to pick up these skills spontaneously in its ongoing process of experimentation. This approach may yield results less immediately than others, but it is one more likely to produce informative surprises if given time. It is closer, or is at least an attempt at getting closer, to the mostly uncertain and self-governed experience of composition by a genuinely creative human poet. Introspective descriptions of creative activity are typically Dionysian, and if these are at all relevant then neatly rational processes might be inappropriate as models. It seems that the deciding, acting individual finds it necessary to be immersed in an overwhelming swarm of representations (imagined thoughts, memories, ideas) in order to be creative, for reasons of this making it possible for patterns to form that are concrete versions of high-level descriptions of what is valued or desired. In other words, out of all the possible patterns to become aware of in a frenzy of activity, the most interesting ones get discovered most readily; the more possibilities there are, the better the chances are of seeing something unique and interesting. This is the basic argument for a scruffy, massively parallel model of human creativity. The overview of the literature on "creativity" detailed in this paper shows that researchers of mechanisms and the process of creativity commonly regard this assumption as useful to have. Metanalysis of the available literature on creativity points firmly in the direction of an evolution-like process as the basis of the creativity of biological cognitive systems. It is not clear yet how specific the analogy is, which has been the case before in the history of science. Metaphors that map crudely but somewhat accurately onto the data have a history of transforming into mature scientific theories. The model of natural selection, with regard to the value-driven experimentation that people and other animals engage in, may be analagous to how the planetary model once related to a cruder conceptualization of atoms. A massively parallel and abstraction-driven cognitive system with similarities to biological evolution is, it seems, the most viable model of creativity that exists today. Parallelism, adaptativity, and the interaction of stochastic experimentation and selective interests is what POEVOLVE aims to model.

The POEVOLVE model's architecture is fairly unique, and in fact there seem to have only been four other models that share its basic structure. The structure referred to here is that of a Darwinian adaptationist model utilizing a neural network specifically to evaluate the fitness of individuals in the evolving system. Of the four models referred to, one is an industrial application (Dagli, 1993), another is an analytic and synthetic tool for biologists (Schneider, 1995), and two are musicological. One, the GenJam model (Biles, 1996) has had limited success at its goal of modelling jazzy improvisation guided by a skilled mentor. The other known musicological use of the basic modelling idea (Burton, 1997) is an incomplete work in progress, including a proposal for the invention of an interactive drum machine that would act with some degree of freedom in creating rhythms while also learning from ongoing external stimulation provided by a drummer. Burton and Vladimirova's model is in theory a genetic algorithm in which the neural net (acting as judge of the fitness of rhythmic patterns) belongs to a class of unsupervised neural networks called Adaptive Resonance Theory (ART) networks. Like the network in the GenJam model, this network would continue to learn throughout the course of the system's evolution. POEVOLVE differs from these in that the neural network is trained prior to its introduction into the Darwinian program, so though it is actively involved in determing fitness of poems, no further training occurs. It is important to note that though the POEVOLVE model was planned and developed prior to any knowledge of the existence of similar models, the differences between POEVOLVE and similarly designed models are empasized here for the sake of explanation. Having the evolutionary algorithm as the single source of changes in the system makes the model easier to understand and interpret than it would be if an interactive approach were taken. The introduction of an additional "outside" influence into the system, a dynamically evolving fitness function. More importantly, the "train once, use many" dictum gives POEVOLVE the important advantage of autonomy. POEVOLVE is autonomous and does not require continuing stimulus from human readers of poetry, but because the neural network extrapolates the data from the human ratings it was trained upon, it also may be considered an interactive genetic algorithm (IGA) like the others. However, this term is less applicable to POEVOLVE than to GenJam or the proposed interactive drum machine, though all three are more autonomous than previous IGA models, including as an example the intricate work of Karl Sims and his programs (Boden, 1996). The baroque visual imagers are products of evolutionary computation in which the evaluation of fitness is the full-time job of a human end user. This is reasonable for visual art in which a collection of thumbnail images are beheld instantly and selected from soon afterward, but in the case of poetry, music and other temporally structured activities, the demands are less reasonable. In any case, developing on themes of autonomy is an appropriate idea if the long-term aim is anything like "artificial intelligence" or a computational demonstration of genuine creativity. A neural network is in many ways an obvious candidate for "missing link" or surrogate thinker, because neural nets are famous for their ability to recognize and complete patterns among other relevant properties. It is interesting and exciting to know that two of the only four published implementations of this design are applications in the humanities. Perhaps the future will see this class of models frequently adopted and developed by scientists and artists developing and testing models of creativity and learning.

On the more pragmatic side of this work, the system seems to have demonstrated some optimization of limericks: the average ratings of limericks increase over the course of generating 1000 populations of 100 limericks. Pearson's test shows that there is a correlation, and a T-test comparing the initial ratings to ratings at the end of the simulation shows there is reasonable certainty (probability of error ¡ .05) that some progress occurred. The results are ambiguous at this point however because a longer simulation demonstrated no grounds to reject the hypothesis that nothing happenned at all. In addition to this ambiguity an important factor in determining the viability of the system will be to administer a "Turing test" in which humans beings will be asked to once again rate limericks. This time however, "population zero" limericks (the unevolved ones) will be mixed in with limericks produced by POEVOLVE that the neural network predicts should be given higher ratings. Such a test would be premature at this point in time, but it will be necessary sometime in the future. I suspect that after executing this system and allowing it to run for a number of generations on a large population of limericks, that the system should produce limericks that are qualitatively different from limericks that were generated randomly. These limericks should be "better" in the sense that people will rate them higher on average.

4 Poetry

The aim of this project as a whole is to probe processes of artistic generativity by using a program that aims to model these processes. To do this a domain is required. The domain could be visual art in its many forms, music, literature, or even scientific discovery. The domain that has been chosen is poetry. Poetry has not been chosen for special reason, other than that some domain is needed. Drawing, composing stories, or jazz music would all work equally as well, each with their unique and shared challenges to to the computational modeler. What is important for the present purpose is that there is both a constraint that can be described algorithmically, and an element of freedom that can be exploited by the artist creatively. Any poetic, musical, or artistic form that fits these requirements could theoretically be applied in an implementation of the computational model described here.

4.1 Form in Poetry

Having decided on working with poetry, a number of options arise. There a number of diverse forms of poetry, ranging in their degree of structuredness from being very constrained to having no explicit form stated. Free verse, an illustration of the latter, is interesting because styles and genres can become apparent without a clear description of them, but the approach taken here will focus on forms that have clear constraints that house the poet's creative expression.

The form of a poem sets a number of parameters that define the form's uniqueness. These parameters include the number of lines, the rhyme scheme (if any), the number of syllables and their prosody. There are also qualities unique to forms, such as specifying that the last and first line be the same, for example. Along with these objective qualities, there can be additional special characteristics of forms at level of theme and content. For instance, in the limerick form, it is common to compress into the five lines a humorous anecdote with a suprising final line.

Incidentally the limerick form has been chosen for use here. It has been selected not because of its semantic side, but because having only 5 lines it is a small form, yet in its short duration still affords many options. The shortness of the limerick form reduces the program's complexity so that it may run faster, and the many options gives it the potential for high variability.

4.2 Characteristics of Limericks

The limerick form is of five lines in length. Its rhyme scheme is $A\ A\ B\ B\ A$. This means that there are two patterns of rhyme, one for the first two lines and the last line, and the other for the third and fourth lines. Lines of limericks have what is called an "anapestic" prosody. The rhythm of a single anapest is of the form *weak weak strong*, where strength refers to the stressing of syllables. Although a limerick can be entirely anapestic, most limericks actually start of every line with an iamb, which is essentially a truncated anapest. Iambs are of the form *weak strong*. To put all this together, a limerick's meter consists of short lines of the form *iamb anapest*, and long lines of the form *iamb anapest anapest*. The meter of the limerick form maps directly onto the rhyme scheme. The longer first, second, and fifth lines are the A rhyming lines in the $A\ A\ B\ B\ A$ scheme, and the shorter third and fourth lines are the B rhyming lines.

In addition to these rules there is also another rule allowing for trailing week syllables after the last strong syllable in a line. The corollary to this rule is that the trailing weak syllables must match up as equal in number and must be part of rhymes. This rule is actually self-evident and is a part of the rules for rhyming in general, and not just in limericks. The rules of rhyme are that the rhyme must begin on the accented syllable, and need not include the initial consonants, if any, of that syllable. The rhyme must include all syllables of the word after the accented syllable, including consonants. In short, the rhyme is the latter part of a word divided just before the vowel of the accented syllable. If the accented syllable is word final, then the rhyme is called "masculine", and if the accented syllable is otherwise located the rhyme is "feminine".

The limerick form has a long history of telling a short anecdote with a surprise ending. It is usually funny and normally has either a sexual theme or some subject that is considered taboo and shocking. The fact that the limericks people write so often tell funny and surprising stories might actually be a side effect of the very form of the limerick. The first line normally sets the stage and introduces a character. The second line can then be used for character exposition or to introduce a second character, and/or begin to reveal a plot. The next two lines, shortened in length, produce a suspenseful effect as the story unfolds. The final element, the element of surprise is unleashed at the final line of the poem.

4.3 Qualifications of the Limerick Form

The limerick form has a core set of objective rules, but also has semantic expectations tied to it. It is the semantic aspect of limericks that will clearly delineate the difference between anything the computer can possibly do and what people do. Not only does the "wit" of limericks depend on an element of humor, the element of humor depends on the existence of a developed set of interconnections between references to lived reality. It is not the goal of this study to endow a computer program with an ability to have the experiences that make humor possible, or even to get the program to impersonate humor. At best POEVOLVE will pick up on objective features available in the text itself that contribute to the limerick's quality. These features may be syntactical properties, or even semantic properties in the form of word collocations. The hypothesis is that in this way the program can learn some of the processes that give the human poems a better score than random ones.

Here is an example of a limerick to illustrate the above description:

There was a young lady of Kent
Whose nose was most awefully bent
One day I suppose
she followed her nose
for no one knew which way she went
– *Edward Lear*

The limerick form fits the criterion of not being too small, and not being too large. This should allow large quantities of limericks to be generated in reasonable amounts of time. Within the five lines, there are a number of parameters that can be varied to create different sorts of effects. The rhyming words can include the same word twice or rhyme different words, the latter being a normal requirement of rhyme, the former an oddity which breaks a rule of rhyming. Words can be arranged in any number of ways in the thirty or so places of a typical limerick, but stress also constrains what words go where. Lines can trail weak beats or stick to the central anapestic rhythm. Lines can also either be purely anapestic or begin with iambs. These dimensions seem to provide the possibility for sufficient variation producing a wide space of possible limericks.

The obvious problem with limericks as opposed to other candidate forms is that there are some high level semantic expectations associated with them. Limericks are expected to be funny and preferably profane, not to mention embodying the dramatic effect of build-up and release. The fact that this is difficult to accomplish with a poetry generation program makes this obstacle an important reminder that the goal of the project is not to pass a Turing test, but to build a system that in some future form could conceivably pass a Turing Test. This modest hope for the implementation developed here is that it will produce some poems that are slightly better than what could be achieved using the basic random limerick generator. To use an "easier" form could be seemingly more rewarding in some respects, and the same problems would still exist but would be less obvious. In other words, easier forms, the less semantically weighty ones, are not significantly easier. The limerick form is a good choice because it honestly both displays failures and sucesses in the computational poet's ability to do what human poets do.

4.4 The "Limerable" Lexicon

The lexicon is the source of the program's ablity to select words. It is structured such that the phonological and prosodic dimensions of words can be directly accessed. The words in the lexicon come from a collected pool of limericks written by people. Some of the poems were written by famous poets, some have been picked arbitrarily out of world wide web pages and books, and a few were written by myself and my advisors. The final set of limericks from which the words were extracted consists of 50 limericks. After putting this list in order and fixing it so that no more than one of each word is included, the lexicon's final size is 1107 entries. The decision to construct the lexicon in this way is due to the logic that if these words have all been used in limericks before, they stand a better chance of being part of a limerick again in the future. The word "limerable" is coined to describe this property the lexicon. The limerability of the lexicon helps to ensure that there is a sufficient quantity of words in the lexicon that rhyme, since rhyming occurred in the source limericks. Another point in favor of the limerable words is that they are all the correct number of syllables to fit into limericks. More subtle semantic effects of using a limerable lexicon may or may not exist as well.

4.5 Examples of Limericks

Below are some examples of limericks. Some are written by people and others by computer. The computer generated limericks are random products of the limerick generation toolkit.

There once was a man in a hat
and he sat and he sat he sat
and he ate and he ate
now can you guess his fate
the man in the hat became fat
Craig Graci

attempted especially all
dozed businessman attitude fall
couch Easter eggs ate
calligrapher fate
calligrapher cautiously ball
Computer-Generated

Dolorous Dolorous Jane other

obscene whale blast curious brother

Dolorous sloan dress

obscene stole well guess

fat canny felt latter another

Computer-Generated

died entered cat jolly cat night

supplying swarm sleepy curve white

upon climbed dine blast

Mercator flat passed

professor returned fast do right

Computer-Generated

5 Thoughts on Creativity

5.1 Distilled Wisdom (Quotes) on Creativity

"Genius is 1 percent inspiration and 99 percent perspiration" — *Thomas Edison*

"Anything created must necessarily be inferior to the essence of the creator." — *Claude Shouse*

"Einstein's mother must have been one heck of a physicist." — *Joseph C. Wang*

"Originally a chaos of ideas. Those which were consistent with each other survived, the remainder perished – and are perishing." — *Friedrich Nietzsche*

"Researchers in the area of creativity are fond of identifying a small number of mental processes that they believe are crucial to creative genius. Thus, Albert Rothenberg pinpointed such mechanisms as homospacial and Janusian imagery. Silviano Arieti has demonstrated the importance of amorphous and primitive cognition. Arthur Koestler underlined the impact of bisociation. And so forth. Who's correct? What's the most basic process behind the creative act? Answer: everybody is wrong. But, like the famous proverb of the blind men and the elephant, all have captured a portion of the truth." — *Dean Keith Simonton*

"Both the bionic and heuristic programming approaches to artificial intelligence are primarily descriptive; that is, there is an attempt to model nature as it is observed to exist. In contradistinction, the evolutionary approach is primarily normative, in that it is an attempt to model evolutionary processes as they *might* occur in nature; to describe what *ought to be* rather than what *is*." — *Lawrence Fogel, Alvin Owens, and Michael Walsh*

"Full-scale creativity consists in having a keen sense for what is interesting, following it recursively, applying it at the meta-level, and modifying it accordingly." — *Douglas Hofstadter*

"I tell you– one must still have chaos in one to give birth to a dancing star!" — *Friedrich Nietzsche*

"Every artist knows how far from any feeling of letting himself go his 'most natural' state is–the free ordering, placing, disposing, giving form in the moment of 'inspiration'– and how strictly and subtly he obeys a thousandfold laws precisely then, laws that precisely on account of their hardness and determination defy all formulation through concepts (even the firmest concept is, compared with them, not free of fluctuation, multiplicity, and ambiguity" — *Friedrich Nietzsche*

"Artificial intelligence is realized only if an inanimate machine can solve problems that have, thus far, resisted solution by man; not because of speed and accuracy, but because it can discover for itself new techniques of solving the problem at hand." — *Larry Fogel, Alvin Owens, and Michael Walsh*

"The process of evolution can be described as four essential processes: self-reproduction, mutation, competition, and selection (Mayr, 1988; Hoffman, 1989; and many others). ... The implication of these very simple rules is that evolution is a procedure that can be simulated and used to generate creativity and imagination mechanically." — *David Fogel*

"Having creativity is an automatic consequence of having the proper coding of concepts in a mind. It is not something you add on afterward. It is built into the way concepts are. To spell this out more concretely: if you have succeeded in making an accurate model of *concepts*, you have thereby also succeeded in making an accurate model of the creative process, and even of consciousness." — *Douglas Hofstadter*

5.2 Introduction: A General Picture of Creativity

"Creative" work refers to work that is both novel and valued. The creative process or processes are those processes that function to produce such work and creative people are those who do the work and exercise the these processes. Creativity has always taken place within the context of a culture. Creative individuals have done work that has been both valued in their cultural context and which has been unexpected in ways that have many times catalyzed change. Creativity is ubiquitous. From the everyday lives anyone might achieve, to the exceptionally creative lives of "geniuses", it is both necessary and valuable. It is as important in the arts as it is in scientific research. What makes creativity possible for us? The history of interest in creativity is long, and there has always been a need to conceptualize it in one way or another. Creativity seems to be an essential part of what it means to be human, so any adequate model of human nature must account for creativity. For this reason, creativity is of interest in modern cognitive science, the multidisciplinary study of cognitive processes. Since cognitive science aims to bring together the intelligence of the various overlapping disciplines that study aspects of cognition, mysteries as deep as creativity and consciousness are currently the objects of scientific study. The creative process has always been mysterious, and its very definition implies that it will always involve the element of the unknown and mysterious. This does not at all conflict with acquiring a detailed understanding of how it works. If anything, such an understanding will enrich our ability to be creative and make intelligent decisions. Artificial intelligence, if successfully developed applying principles of creativity, should undoubtedly surpass models that are mere caricatures of certain areas of human expertise.

5.3 Defining Creativity

There is not complete consensus on how to define "creativity", but the similarity of the definitions used is reassuring evidence that researchers are not are not merely using the same word to talk about different phenomena. In ancient times, creativity was understood within the ideology of the time and place, but was not clearly defined in a way that would allow for scientific study. As the world changed and history took its course, philosophers such as Friedrich Nietzche, Gregory Bateson, William James, B. F. Skinner and George A. Miller sought a more detailed conceptualization of the creative process that would be consistent with overarching perspectives on the nature of the reality and life. These trends in philosophy eventually brought us to the twentieth century's scientific approaches to understanding creativity. D. T. Campbell (Campbell, 1960) revived the ages old concept of an individual's creativity as a mirror of life's evolution and reframed it within the context of modern biology. Arthur Koestler, who wrote in the later half of this century, defined creativity as a process of "bisociation", meaning the fusion of two previously unrelated forms into a suprising new form. He illustrated this with his exploration of "wit", a kind of creativity that is judged both on its novelty and humor (Koestler, 1964). Teresa Amabile has applied the working definition "novel and appropriate" in her studies on the social psychology of creativity (Amabile, 1993). Theorists who apply the neodarwinian model sometimes say that creativity is that which is "novel and adaptive". These two definition have obvious similarities. They differ in that one works better when discussing creativity at the level of an individual's work in relation to a social collective, whereas the alternate definition, stressing the idea of some things being retained as others are discarded, is more of an attempt to describe the creative process at some level within the individual. The term "appropriate" in some contexts is insufficient, because the concept of appropriateness does not encompass the fact that creative work most often is not merely adequate but is exceptional and even unusual. Mihalyi

Czikzentmihalyi, who (like Amabile) is interested in the role the social environment plays in fostering or inhibiting creativity, defines the concept differently by wording it instead as "novel and valued" (Czikzentmihalyi, 1999). The idea that creativity is a process of doing what is novel and valued in one's work is a formulation compatible with the majority of examples of creativity, both in diverse contexts and at the different angles of individual, society, process, and artifact. The generality of Czikzentmihalyi's definition is due to the generality of "value". It does not specify that a thing is valued for its humor, for its quality of being appropriate, for its fitness within the system, for its aesthetic properties, etc.. It can be useful then to assume from here on that creativity requires a source of value and a source of novelty. This definition may or may not serve as a fact that can apply universally to all creativity, but has been assigned as an operational definition of creativity that applies to the computational model presented here.

5.4 Evolution as a Paradigm of Creativity

In 1960, Donald T. Campbell published "Blind Variation and Selective Retention in Creative Thought as in Other Knowledge Processes" (Campbell, 1960). Campbell's paper introduced a workable model of creativity inspired by biology's Darwinism. In the paper Campbell states that "all inductive achievements, all genuine increases in knowledge, all increases in fit of system to environment" are fundamentally due to processes of "blind variation and selective retention". In adition to this central idea, which we will explore and elaborate on in detail, Campbell proposed two additional corollaries. The first is that there are processes that "shortcut" the full processs of blind variation and selective retention. These shortcuts are themselves inductive achievements developed by means of blind variation and selective retention. The other corollary is that even shortcut processes can utilize a blind variation and selective retention process in their operation.

The concept of blind variation and selective retention is a powerful idea because it provides a very specific framework in which to understand how creative work is possible. The idea is of course not a theory, but is at least a powerful analogy that may lead to fruitful discoveries. The basic process requires that three conditions be met. There must be a mechanism in place that functions as a source of variation, there must be a selection process (ie. a way of applying value judgments to objects in question), and there must be a way of preserving and making copies of the selected variations. Having made this clear, the term "blind" stands out as opposed to "random". Campbell explains that randomness does not have the same meaning that the metaphor of blindness has. Randomness implies equal probability of all outcomes, as well as statistical independence, and neither of these conditions are related to blind variation. Blindness, encompasses two important facts about the kinds of variations that are necessary. The variations that occur must be independent of context, and they must also be "uncorrelated with the solution", which is to say that an important implication of blindness is that any and all selection takes place subsequent to variation. The requirement of blindness is obviously not always the case. It does not apply to the aforementioned shortcut processes because these need not be blind even though blindness was involved in the original construction of this shortcut process. When a shortcut modularly calls upon blind variation and selective retention as a subprocess the rules of blindness do apply, but only within the subprocess of the shortcut.

Campbell stresses the fact of "substitution" in creative thought. Substitution refers to the general capacity to go through an exploration of conceptual space before any call to action is answered with a specific choice of what to do. Campbell's view is that when an individual is in a situation requiring a creative solution, a process of blind variation and selective retention is initiated. Because a representational system stands in for the necessity to feel one's way around the real world trying out variations, a solution occurs more quickly and safely than would be possible without substitution. William Calvin, whose book *How Brains Think* is very much in the tradition of Campbell's ideas, echoes his position on substitution, and has elaborated on it further (Calvin, 1996). Calvin's book explores how the older methods of overt foraging and immediate experimentation in one's direct sensory experience have become increasingly substituted by creative processes in which most of the effort is done before any of potential motions are really enacted. For Calvin, and from the standpoint of this view in general, questions of how much freedom a biological organism has in its volition can be answered in terms of the type and degree of creativity that organism is capable of. Developing a better understanding of the complexities of volition is an exciting possibility that future research on creativity is likely to bring about.

Judging by the large amount of work being done, it seems that a majority of the computationally oriented cognitive scientists share some version of the idea that a system, artificial or natural, that can begin to qualify as "intelligent" must necessarily behave adaptively. The major avenues of research into adaptation in computational systems are self organizing neural networks, agent-based architechtures, and evolutionary computation. The classic AI modeling techniques are being reframed within these newer approaches. David Fogel, seeking a definition of artificial intelligence, has argued for an approach that rests heavily on the idea that intelligence is implicit in natural evolution and systems that reflect it in the right ways (Fogel, 1995). Fogel's definition of artificial intelligence excludes software that simply solve a problem– the software must go beyond this to "solve the problem of how to solve problems." Darwin's basic algorithm *is* a sufficient solution to the problem of how to solve problems, whether in the context of ecosystems over the course of geological time, in an ant colony, in a higher mammal's neural system, or in a digital computer.

The process of Darwinian evolution appears sufficient as a means to achieve creativity. However, even if a model of the process were to demonstrate intelligent behavior using this basic algorithm, it would not necessarily indicate anything significant about the intelligence of biological organisms. The idea that human creativity employs a process with strong similarities to Darwinian evolution needs to be tested and refined. It could be the case that there are important differences between the kinds of evolution that allow different kinds of systems to demonstrate their unique forms of intelligence. It is likely that computer models of creativity will make a more detailed understanding of creative evolutionary processes possible. One thing we may find more evidence for both in human creativity and in biological evolution is that self-organization may be an essential property of our adaptively creative systems (Goertzel, 1995). Self-organization is defined as system's capacity to influence its own control parameters.

It is necessary to remember that Darwinism is a metaphor when using it to study creativity. We have very little reason to claim anything more than that it is a useful metaphor. With this in mind, a reasonable question to ask is the question of what specific differences there are (if any) between the biological and cognitive creativity. David Perkins (Perkins, 1994) has made some helpful suggestions in a paper entitled "Creativity: Beyond the Darwinian Paradigm". Perkins divides the problem up into not two but three kinds of creative systems: the evolutionary creative system, the meme creative system, and the inventor creative system. The evolutionary creative system is made up of genetic variation, survival long enough to breed, and transfer of traits to offspring. In the meme creative system, (where "meme" refers to a supposed elemental unit in cultural evolution analogous to the gene) variation occurs due to mistakes in copying and by individuals' changes to the memes, selection refers to the meme being retained by a person long enough for it to be copied, and preservation means the meme is successfully distributed to more people. Perkins's formula for the inventor creative system is identical to Campbell's system (described above), but in addition to blind variation a spectrum of classes of variation are layed out ranging from "safe bet" to "sheer chance". Perkins illustrates the similarities and differences among the three creative systems with the thought experiment of "Klondike spaces". Searching for gold in the Klondike, one encounters the condition of rarity in which payoff is sparsely distributed, the isolation problem of payoff being isolated, the "oasis" where regions of payoff are difficult to leave, and the "plateau" where the direction to better fortune is unclear. Each of the creative systems has its own unique strengths and weaknesses for meeting challenges in Klondike spaces. Biological evolution meets the challenges of rarity and plateaus because of the advantages of time and massively parallel operation, and meets the oasis problem by radiating in all directions regardless of how viable the direction is. Evolution has difficulty with the isolation problem because "leaps" are out of the question. Memetic creativity in culture is very similar to evolutionary creativity except it has an advantage in solving the isolation problem due to the fact that individuals can make creative judgments that effectively allow for leaps into more profitable regions. The inventor creative system is weak at solving the rarity problem due to its brief time constraints, but compensates by utilizing heuristics that shortcut the slow but sure algorithmic approach. Inventors have at their disposal a unique advantage in the rarity problem by being able to consider forms that are not viable– this advantage is not present in biological evolution. Inventors, uniquely in this case as well, are able to avoid getting stuck on an oasis by intentionally avoiding making the mistake. To solve the plateau problem, inventors alone are able to systematically employ an element of chance in making their decisions, and are also able to control the grain of their searches, choosing between higher or lower levels of resolution. What Perkins's analysis shows is that while different classes of creativity bear important similarities, there are fundamental differences between them as well.

To summarize this overview of creativity, these are insights into creativity can be derived. While "creativity" is difficult to define, the definition "novel and valued" has the advantage of being both general and practical. Creativity applies to individuals, their creations, their environments, and the processes they go through in creating. Generalized as a property of systems, creativity also applies to biological evolution, which in turn can be used as an an analogy to understand the individual artist's and scientist's creative process. Creativity can involve altering one's own constraints or "transforming conceptual space". Creative thought is nonlinear and divergent rather than linear and convergent. An evolutionary model of creativity is both a useful analogy with which to explore human creativity with, and a smoothly adaptable candidate for computational approaches to intelligence. Darwin's simple algorithm describing evolution qualifies as a solution to "the problem of how to solve problems". Although the model is useful, there are fundamental differences between the kind of solutions that adaptive process achieve in shapes an ecology, and the kind person is capable of arriving at.

6 The POEVOLVE Model

6.1 The Architecture of POEVOLVE

POEVOLVE, a computational model of poetry composition, loosely models a description of biological evolution, and in doing so models the evolution-based conceptualization of creativity. POEVOLVE can be divided up into three sections. One part of the system is called the **limerick generation toolkit (LGTK)**. The LGTK includes all the computational materials used in making limerick poetry. It includes the lexicon, the phonemicon, the template for the limerick poetic form, and all the functionality that generates limericks or assists in the generation of limericks. Another module, referred to simply as the **evaluator network**, determines the fitness of forms generated within POEVOLVE. This part of the program is a neural network trained on empirical data from human evaluation of limericks. The main part of program is simply referred to as **POEVOLVE** because it is the top level of execution. POEVOLVE calls on the LGTK when generating, mutating, or meshing together limericks, and uses the evaluator to determine which individuals to select out of limerick populations based on their adaptive value.

The working model POEVOLVE has been coded in the language Common Lisp, making use of Common Lisp's object oriented sublanguage, the Common Lisp Object System (CLOS). The evaluator utilizes a separate program, the Tlearn neural network simulator, which is triggered by but runs independently of the Lisp interpreter. Specific details on how POEVOLVE is coded in Lisp are not included because the detailed functional description that follows should suffice to guide readers toward an understanding of the actual source code (see Appendix B). Three kinds of computational constructs are referred to for purposes of clarity. **Object classes** are descriptions or definitions of *kinds* of objects, while **objects** are particular, specific manipulable things, and **methods** are actions performed, often on or with objects.

6.2 The Limerick Generation Toolkit

The LGTK consists of a collection of functional components used in constructing arrangements of text that fit the limerick form. It is referred to as a toolkit because the components are tools utilized in the variety of processes for generating poems. LGTK introduces a number of interesting object classes describing some things that need to be created in order to produce limerick poetry. There is the lexicon class, defining an object that will store knowledge about words. The entry class is the elementary unit of the lexicon, encoding the knowledge of a single word. The phonemicon class narrows the focus of linguistic knowledge to pinpoint the individual phonetic sound and how it is articulated. There are a variety of methods, such as "genlimerator" for instance, the random limerick generator.

6.2.1 Lexicon

The lexicon in this computational system is a model of the kind of knowledge a person has on words. It models a person's knowledge of a subset of all the words available in a language. Like a linguistically competent human being, the system must be able to rapidly access information on a number of different aspects of any given word. To make this information available to the system, the limerick generation toolkit uses a lexicon made up of a store of lexical entries, providing four basic pieces of information on each of the 1107 included words. These pieces of information are the spelling of the word, the spelled word divided into syllables, a phonemic coding grouped into syllables, and a number indicating what syllable to put emphasis or stress on.

The lexicon class has two properties: length and entries. Entries holds a list of entry objects and length contains a number. Length is the length of the list of entries. When a lexicon is created these two abstract labels are filled with the relevant information which may then be accessed from the object. Make-lexicon loads the list of entries from the lexicon file and creates a new lexicon object with these entries.

To clarify exactly what information becomes available when a lexicon is instantiated, the genlex method is described here. The genlex method helps the user to maintain the lexicon. The process of correcting errors and making other alterations to the lexicon is made much easier by this method. The lexicon is saved in a file called "lex.l", but the lexicon file is maintained indirectly by editing a file called "word-list.txt". The reason for this is that during run time it is convenient to explicitly store (in the entry objects) information that is the result of computations based on minimal information (from word-list.txt). If it were not stored explicitly, it would be necessary to perform much larger number of computations in the program runs, making them astronomically slower. It is also convenient to only have to edit four pieces of information rather than the nine explicitly stored in entry objects. The function of the genlex method is to generate the lexicon from the data in word-list. It is only necessary to run genlex after making changes to the word list file. Genlex is a kind of lexicon "compiler" that reads information in the word list. The four pieces of information on each word are separated unobtrusively by hyphens, and the sonic coding is divided up into syllables using commas. Genlex separates the word-list file into lines and transforms the data from each individual line into a separate entry in the lexicon being constructed.

Example of a line from the word list:

"behave" - "be" "have" - "b" "ee" , "h" "ay" "v" - 2

How Genlex Processes the Word List

```
For each entry, genlex

1. Separates the raw text of the line into the four items

  "word" (spelling information),
  "syllables"

  "sounds" (with syllable dividers)

  "stress"

2. Assigns this info to the appropriate slots on the entry object:

  "word", "syllables", and "stress" are extracted.

  "sounds" from "sounds" information (ignoring syllable dividers)

  "syllasounds" from "sounds" information (respecting syllable dividers)

  "index" from an incrementing counter

  "rhyme" from the "format-rhyme" method, a rhyme-extracting

          algorithm using the "sounds" and "stress" info

  "rhythm" from the format-rhythm method, using "stress" and

           "syllasounds" info to construct a coding of rhythm

  "nn-input" from the format-nn-input, which generates the neural

             net coding based on "index", "syllasounds", and "stress".
```

The “format-rhyme” method uses sounds and stress info. The algorithm uses this information to discover the rhyme and stored it on the entry explicitly. If the stress is on the last syllable a ”masculine” (monosyllabic) rhyme is chosen, which means, beginning with the vowel sound, all of the sounds of the last syllable are included. If the stress is otherwise located in the word, a ”feminine” (multisyllabic) rhyme is used instead, and this jargon translates to starting by including the vowel sound of the stressed syllable, also including the part the word after this sound. Try this with any word to see how a rhyme can be picked up automatically given the prosody and sounds of the word.

The “format-rhythm” method uses the information from the stress and syllasounds (short for "syllable demarcated sound list") properties of an entry. The method generates a list of zeroes and ones, the zeroes and ones corresponding to syllables landing on "weak" or "strong" (stressed) beats. This coding of prosody is used by the make-poem-line method which matches up fitting entries in a rhythmic template.

The method "format-nn-input" uses the index, syllasounds, and stress information associated with entries. It functions to construct an encoding of the entry that can be fed to the input cells of a neural network. This is necessary because the neural network has to be able to process limericks after they are generated. In order to convert the information on entries into their neural network encodings, the first step is to fit each syllable into a "c c v c c" template. The syllables are then converted into binary format including the indexical information specifying what word the syllable belongs to, and the stress marker which is turned on or off depending on whether the syllable is stressed. The coding of each phoneme in binary is acquired by accessing the phonemicon, described in greater detail later.

The method "save-lexicon" saves a generated lexicon into a lisp structure encoding the entire lexicon with all of the necessary information stored explicitly. Saving and loading the lexicon in this way has the advantage of bypassing the immense amount of computational work necessary to generate the lexicon from scratch each session. This has the noticeable effect of the program loading much faster.

6.2.2 Entry

Entries of the lexicon contain more than just the words themselves. They contain the phonetic and prosodic knowledge of the words that people must also have in some form.

```
The entry object has 9 properties:
word, a string containing the word as it is spelled
syllables, the word as it is spelled, broken up
           into separate strings for each syllable
sounds, a list containing a symbol for
       each of the word's phonemes
rhyme, a list of only the phonemes used for rhyming
rhythm, a list specifying which beats are
        strong and which are weak
stress, a number specifying which beat is strongest
index, a number unique to only this word
syllasounds, a list of syllables (a list of lists of phonemes)
nn-input, a list containing the entry's coding for the nn.
```

6.2.3 Phonemicon

Constructing the Phonemicon

The make-phonemicon method creates the phonemicon with all of the data on phonemes "hard coded" (as opposed to dynamically constructed or loaded from a separate data file.) Most of the information included on phonemes is for easy reference. The pragmatic aspect of the phonemicon is its role in cosntructing the neural net coding. When the lexicon is being generated the phonemicon is accessed and a network coding is built for each syllable. When the data file for a population is being put together, the binary encoded syllables (also representing stress and word index) are accessed from the lexical entries. The structure of a syllable is CCVCC, and the structure of a phoneme is as follows: The first bit encodes the parameter of consonant/vowel The second bit encodes the parameter of voiced/voiceless. The third through fifth bits encode the seven possible places of articulation. The sixth through eighth bits encode the seven possible manners of articulation. The ninth, tenth, and eleventh encode the five possible heights. The twelfth and thirteenth bits encode the three possible depths.

6.2.4 Limerick Generation

The Population Zero Limerick Generator

The "genlimerator" method (with optional file-out for writing it to disk) is what generates the initial arbitrary population, making random choices while abiding by the rules of the limerick form. To generate a random limerick, the population zero limerick generator first picks up three rhyming words to end the first second and fifth lines with, picks out two rhyming words for the center (third and fourth) lines. Next a miniature lexicon of a prespecified size is extracted randomly from the big lexicon (both to minimize computational effort of searching through large ammounts of words, and to focus the range of words to choose from on a more special set than merely "all known words". After choosing the rhyme- words and the lexicon, the lines of the limerick are composed and stored in the limerick-poem object. They are composed through the random selections of the make-poem-line method which fills a rhythm template with entries from a lexicon. The templates given to make-poem-line are (0 1 0 0 1 0 0 1) for the three long lines and (0 1 0 0 1) for the two shorter lines. The templates are preequipped with the allready chosen rhyming entries at the start, so make-poem-line is typically given something like (0 1 0 0 ¡ENTRY¿).

The limerick-poem object class properties are "lines" and "rating". Rating is assigned after the limerick is evaluated by the neural net. Lines is the list lines that make up the limerick (A line is a list of entries). The “make-poem-line” method is an important part of the population zero limerick constructor. Given a set of entries and a "rhythm” field (a list of zeroes and ones where the ones are strong beats) the blanks become filled. Replacing zeroes and ones with entries, the list gradually fills with entries that fit the rhytm template, and the list is returned when there are only entries in the list. The “rhythm-filter” method searches through the lexicon for words with the specified rhythmic template. The “rhyme-filter” method searches through the lexicon for words that rhyme with the specified sound.

Here is an example of a limerick being randomly generated:

```
PCL>(setf limerick (genlimerator))
#<LIMERICK-POEM 1047066074>

PCL>(display limerick)
wench graphing deep knitting hit fight
amazes been packets cut night
deep year gown again
queer graphing cant then
Attila Jane hornpipes clothes might
```

6.3 The Evolution Program

6.3.1 Introduction: Evolutionary Computation Fundamentals

Evolutionary computation is a class of programming methods mainly used for learning, optimization, search and modeling. This class of programming methods is characterized by its used of a metaphor reflecting kind of evolution that occurs in natural, biological ecosystems. Sometimes the adherence to naturalistic paradigms is strict, and other times it is be only similar to evolution in a very abstract sense. Melanie Mitchell describes the evolution as an abstract process of "searching in parallel among an enormous number of possibilities for 'solutions' to the problem of survival in an environment, where solutions are particular designs for organisms." This definition zeroes in on an important element of evolution, the fact that it is the search for solutions to "survival" that brings out novel forms. This property of evolutionary computation makes it useful for applications where there set of parameters to be optimized exists but the desired "solution" is not easily circumscribed. In the evolutionary approach to such problems, the valued parameters become the measure of fitness, which determines "survival". Given a sufficient amount time and a large enough population size, surprising forms emerge relecting the qualities specified by the fitness operator.

Evolutionary computation has traditionally been divided up into three independent paths of development. Larry Fogel invented "evolution programs", John Holland came up with "genetic algorithms", and Rechenberg and Schwefel introduced the "evolution strategy". All of these approaches began as strictly specified algorithms that changed over time as computer scientists advanced new ideas. Genetic algorithms began as an attempt to model naturally adaptive systems on a computer, but have more recently been used in applications similar to the applications of evolution programs and evolution strategies. Evolution strategies were originally applied for purposes of parameter optimization (for example in the development of more aerodynamic airplane wings). Evolution programs were originally created by artificial intelligence researchers who wished to create programs that could effectively deal with unexpected situations and to learn from experience. The three approaches have so much in common that the categories might in fact be merely historical distinctions. David Fogel (Fogel, 1995) has suggested that genetic algorithms differ from evolution programs in that the genetic algorithms tend to imply a "selfish gene" philosophy emphasizing genetic mechanisms, whereas evolution programs place less emphasis on the gene. This relates to an assumption that the evolutionary process can be at work at levels than other selfish genes. It may also be a useful model for systems unrelated to genetics, such as the real-time optimization of behavior and the shuffling of concepts. This of course is the meaning of "evolution" that pertains to the POEVOLVE model.

6.3.2 The Basic Process

The POEVOLVE model begins with an object called "population". the population object class models a set of individuals, in other words, a collection of limerick poems. The population object class defines two properties. "individuals" is the set of individual limerick poems objects, and size is the size of the set of individuals in the population.

The "establish-parameters" method

The establish parameters method is executed before the execution of POEVOLVE. This method's arguments are saved as the global values "*population-size*" and "*nr-generations*", one specifying the size of populations, and the other specifying the number of generations for which the program will run. The population size is constant over the generations in this model. When the population size is determined, three other constants are established implicitly, based on a preconfigured ratio specifying the percentage of individuals to go through a "crossover" method, the percentage to be altered by the "mutation" method, and the percentage of individual items to be copied directly, a process labelled "direct copy". When the population size is determined, the exact quantities of individuals to undergo these processes are registered in the variables "*direct-copy-group-size*", "*crossover-group-size*", and "*mutation-group-size*".

This is how the parameters are established:

```
PCL>(establish-parameters)
The program's parameters have been set to
90% crossover, (4500 individuals),
9% direct copy, (450 individuals),
1% mutation, (50 individuals),
to evolve a population of 5000 individuals for 60 generations.
(5000 60)

PCL>
```

6.3.3 The Initial Pool of Individuals

The "generate-limericks" method

After establishing parameters, the working model can be started, and generate-limericks is the name of the method which initiates and conducts this process. First, generate-limericks creates the initial population of the specified size. Since it is what is known as "population zero" this population is generated randomly (using the "genlimerator" method described elsewhere). Next, a loop is set into motion that lasts for the specified number of generations. Evolving each generation means running the "evolve-one-generation" method again and again.

When the evolution program is executed, it evolves for a set number of generations of a fixed size. Evolving a population begins with a tournament selection process that weeds out all but a few of the "fitter" individuals. A certain percentage of the new population is created by mutating individuals from the exclusive group, another portion is generated by crossing over its members, and another percentage is directly copied into the next generation. The ratios are fixed at 90 percent crossover, 9 percent direct copy, and 1 percent mutation. After a selected individual is chosen to be mutated or crossed over with another individual, another decision determines what particular version of the selected operator to use. To ensure variety, four mutation operators and five crossover operators exist that are used interchangeably under the generic headings of mutation and crossover. The"evolve-one-generation" method is what brings about the new generation from the old one. In order to make this happen, the pivotal fitness function, here dubbed the "evaluate" method, must assign a rating to every individual in the population. The details of this are described in greater depth below, but the basic idea is that a neural net, trained at an earlier date by human ratings of limericks, autonomously scans the population, applying a numeric judgement to each limerick. After this process is completed, what is referred to as a "tournament selection process" occurs: a small handfull of individuals are put in the "*fit-list*", labelled as such because it contains the most fit individuals from the previous generation. The size of this very small selection is specified by the global constant "*select-few*" at default size of seven individuals. The third and last step of producing a new generation out of the old one is the step where the "crossover", "mutation", and "direct-copy" methods are applied. For mutation, a group of the size specified at initialization is randomly created by repetitively selecting individuals from the "*fit-list*", and then mutation operators are run on each of these, producing a list of the same size, but with new limerick poems bearing similarities to their predecessors. Similarly crossover, in which two parents are required to create a child, is executed on two groups of the specified size. Pairs meet up from these two groups and a crossover operation produces a child on each pair with similarities to both parents. The details on what mutation and crossover actually do might be influential, perhaps having a non-negligable effect on the system's functioning and its ability to produce viable forms. For this reason, mutation and crossover are next expanded upon in detail.

6.3.4 Mutation

The central mutation method performs the task of activating one of a collection of mutation methods. The choice is made randomly, but the operation is specific to the method chosen. The mutation operators are named and called according to their letter. There are a few things that all mutation methods implemented for POEVOLVE have in common. One is that all of them will produce different results each time they are used, even if they are executed on the same individual twice. This is because there is a great deal of freedom involved both in the selection of words to use and in the choices surrounding how words are to be arranged within the poem. The following example demonstrates both the variability and the similarity of two limericks produced by mutation of the same unaltered indivual:

```
PCL>(setf limerick (genlimerator))


#<LIMERICK-POEM 47573644>


PCL>(display limerick)

jigs sitting aunt potion ill pegs

tv voice laredo asked eggs

voice keen pat weeds make
```

```
off grey overtake

Lloyd ricochet course build grey legs


#<LIMERICK-POEM 47573644>


PCL>(display (mutate-a limerick))

grey drawl overtake checks off pegs

Lloyd dialing interred off ill eggs

make frantic spoon make

ill mice overtake

goes rhyme overtake voice year legs


#<LIMERICK-POEM 51522434>


PCL>(display (mutate-a limerick))

obscene asked hired ricochet pegs

resisted voice magnum keen eggs

obscene aunt ate make

charm get overtake

asked psyches Egyptian id legs


#<LIMERICK-POEM 52646614>


PCL>
```

Mutation method A

Mutation A does not affect the rhyming words at the end of the lines, but transforms the lines of the orignal poem without. In order to generate the new lines it reuses the old words and gathers fifty extra words. This design choice is intended to create some variety without spoiling the new limerick's connection to the originating limerick from which it is produced.

The following example illustrates what mutation A does:

```
PCL>(display limerick)
wench graphing deep knitting hit fight
amazes been packets cut night
deep year gown again
queer graphing cant then
Attila Jane hornpipes clothes might

#<LIMERICK-POEM 1047066074>

PCL>(setf limerick2 (mutate-a limerick))
#<LIMERICK-POEM 1062134304>

PCL>(display limerick2)
rhyme circus apparel him fight
Jake awefully graphing go night
retired smiled again
```

```
Europas cant then

amazes obsessed been queer might


#<LIMERICK-POEM 1062134304>


PCL>
```

Mutation method B

Mutation B is a low fidelity mutation operator, discarding almost all the features of the orginal limerick mutation is performed on. The only thing preserved are the words from the original limerick, and fifty extra are added for the same reasons they are added to limerick A. The effect B has on a limerick is to lose everything but the words used in the original, and to generate the rest from scratch. Here is a demo to illustrate it:

```
PCL>(display limerick2)

rhyme circus apparel him fight

Jake awefully graphing go night

retired smiled again

Europas cant then

amazes obsessed been queer might


#<LIMERICK-POEM 1062134304>


PCL>(setf limerick3 (mutate-b limerick2))
```

```
#<LIMERICK-POEM 1063457114>


PCL>(display limerick3)

composing again him once plenty

asked dinosaur boarding once twenty

Europas debate

retired gad Jim ate

queer circus retired klutz once plenty


#<LIMERICK-POEM
```

Mutation method C

In this mutation operation C, the first line stays constant and does not change, but all the other lines are randomly generated. Here is what it looks like:

```
PCL>(display limerick3)

composing again him once plenty

asked dinosaur boarding once twenty

Europas debate

retired gad Jim ate

queer circus retired klutz once plenty


#<LIMERICK-POEM 1063457114>
```

```
PCL>(setf limerick4 (mutate-c limerick3))
#<LIMERICK-POEM 1063322214>


PCL>(display limerick4)
composing again him once plenty
repelled grew Egyptian found twenty
gown apples agree
imprudent gown glee
droll withered attempted click plenty


#<LIMERRICK-POEM 1063322214>
```

Mutation method D

Mutation D, the fourth and final mutation operator regenerates the fourth and fifth lines of the limerick, leaving the other lines as they are arranged in the originating limerick.

```
PCL>(display limerick4)
composing again him once plenty
repelled grew Egyptian found twenty
gown apples agree
imprudent gown glee
droll withered attempted click plenty


#<LIMERICK-POEM 1063322214>
```

```
PCL>(setf limerick-5 (mutate-d limerick4))

#<LIMERICK-POEM 1066620710>


PCL>(display limerick-5)

composing again him once plenty

repelled grew Egyptian found twenty

gown apples agree

pawtucket agree

large Viceroy forge twentyfoot plenty


#<LIMERICK-POEM 1066620710>


PCL>
```

6.3.5 Crossover

The crossover operations are like those of mutation but different in the sense that it crossover is based on the concept of synthesizing a new artifact out of two artifacts, whereas mutation is the alteration of only one. Crossover puts stock in the idea that combining two good things together may produce a third good thing. Crossover can combine limericks in five different ways. In other words, the main crossover method is in charge of randomly picking a method of crossing over the two parent limericks to produce a child limerick. So in this respect it is very similar to the main body of the mutation method. Also similar to mutation, the many degrees of freedom make it unlikely that the same limericks will appear more than once if even if a the exact same crossover method is executed twice on the exact same limerick.

Crossover method A

Crossover method A combines limericks out of the 3rd and 4th lines of one limerick and the 1st, 2nd, and 5th lines of the other one. An example of it actually happenning has been included below:

```
PCL>(display (crossover-a (display a) (display b)))
right withered filet rod known out
queer mapmakers spain that without
blast program although
bed shaking spain throw
long hotter Europa's about

nose maiden charm Niger die fall
cape legal horn four Jack son mall
debate four cape kitty
Attila sank city
reside Jack hoped snowman glob drawl

right withered filet rod known out
queer mapmakers spain that without
debate four cape kitty
Attila sank city
long hotter Europa's about
```

```
#<LIMERICK-POEM 1067302024>

PCL>
```

Crossover method B

Crossover operator B combines the rhythm of limerick I with vocabulary of limerick II. Each word in limerick I is replaced by a different word from limerick II with the same rhythm. The rhyming words remain unaltered. Here is how it works out in practice:

```
PCL>(display (crossover-b (display a) (display b)))
Alfredo Alfredo Iraq
revealed been Alfredo cape sack
inventor lets swell
agree hurt shoe fell
cape Easter aghast throw warts Jack

imprudent hitched smarty grey word
owls smarty dressed ever cooked nerd
appall sank would class
imprudent locked ass
appall til entire wished interred

imprudent imprudent Iraq
```

```
entire dressed imprudent sank sack

imprudent cooked swell

appall til nerd fell

sank smarty appall locked wished Jack


#<LIMERICK-POEM 1071304004>
```

Crossover method C

Crossover method C is, in a certain sense, a combination of crossover A and B. The flip-flopped rhyme scheme is used. Additionaly, and consistent with this flip-flopping, the rhythm-structure and vocabulary tranformation is performed just as it is done in B. This is an example of crossover B.

```
PCL>(display (crossover-c (display a) (display b)))

insisted insisted lear him

papyrus suppose seeds ass swim

efficient grass loud

went walker clone crowd

suppose say efficient click Jim


behave underdone sat formed hat

park hillbilly Ida class flat

inquired old throat mean

concluding Kathleen
```

```
unpleasant professor men mat


concluding concluding sat him

professor Kathleen formed old swim

suppose loud crowd mean

insisted Kathleen

inquired sat concluding mean Jim


#<LIMERICK-POEM 1061402640>
```

Crossover method D

The crossover method D takes lines 1, 3, 5 from limerick I and takes lines 2 and 4 from limerick II. Lines 2 and 4 from the second limerick are used only for their rhyming words. Those two lines are recreated, borrowing the words of their vocabulary from limerick II. Here is the sort of effect it will produce:

```
PCL>(display (crossover-d (display a) (display b)))

high graphing bologna shade green

goat maiming tv more fifteen

Flo Germany out

Flo only without

rode roar new tv high grist seen


efficient alas Sid replied
```

```
say lasted efficient off Hyde

efficient awoke

about queer sit spoke

lob notion aquatic reside


high graphing bologna shade green

efficient replied Hyde fifteen

Flo Germany out

tv Sid without

rode roar new tv high grist seen


#<LIMERICK-POEM 1070764510>


PCL>
```

Crossover method E

Crossover E is like crossover method D in the way it combines and reuses the words lines. It is also like Crossover A and Crossover C in the way it flip-flops the lines. Here is an illustration:

```
PCL>(display (crossover-e (display a) (display b)))

weeds over without wench chased two

lob sweaty fermented til knew

poupon prime nose click
```

```
imprudent glob sick

amazes instead pears froze few


potato along clothes sound three

commends bank technician grass be

potato grass cannister

iambs get stiles bannister

potato commends clothes break me


weeds over without wench chased two

stiles bannister sweaty clothes knew

potato grass cannister

be over sick bannister

amazes instead pears froze few


#<LIMERICK-POEM 1065526204>
```

6.3.6 Fitness

In order for the program to make selections out of the large number of possible individuals, there must exist a measure of fitness. It must have a way to make the machine equivalent of judgments on the value of what it produces. In the limerick evolution system this vital function is performed by a neural network. The neural network, trained on the data from a questionaire filled out by about 160 human research participants asked to rate limericks. The behavior of the neural network is intended to reflect the essence of what the group of human raters have based their judgments on. Neural networks and this particular instance of a neural network are discussed in the following section.

6.3.7 Integrating the Tlearn Network into the CLOS Program

Before launching into a discussion of the neural network, first we will explore the interface that allows the system to run smoothly as a whole. Outside the basic random limerick generator and its associated accessories and paraphenalia (including functions that display limericks on the screen or call on an external speech synthesizer to read them aloud), the system requires that limericks also are prepared to be speedily entered in the neural network. When the "evaluate" method is executed, first the tlearn files must be constructed, next the tlearn script executes, and then ratings are assigned from the results of the tlearn script. Making tlearn files means simply that a "data file" is generated that encodes all of the limericks into the neural network's coding scheme (described earlier), and a reset file is also created. The reset file is necessary because the context nodes of this recurrent network must be reset regulary so that information is built up for each individual word over time but also so that processing of one individual word does not influence the processing of a proximal word simply due to being in that place in order.

After making the new files, the tlearn script uses the weights of trained network and the configuration file (already in the "evaluator" directory). It is estimated that there are about 40 syllables in each limerick, so the size of the population is multiplied by 40 to arrive at an accurate number of times to sweep the data patterns into the trained network. After executing the neural network, the rating for each limerick is placed onto the "rating" property of that limerick-poem object. After doing this, the system procedes as normal, selecting the top seven limericks and going on to apply crossover, mutation, and direct copy to get the next generation. The top seven limericks are displayed along with a table of how ratings were distributed in that generation overall.

6.4 The Evaluator Network

6.4.1 Introduction: Neural Network Fundamentals

The neural network is an approach to computation that applies current theories on how networks of interacting neurons in real brains function within thinking, acting organisms. Classical models are based on the idea of hierarchical design where components have clearly defined functions, and are built up logically into more functions. In the distributed system, a number of computations occurring at the local level contribute in different ways to the macroscopic behaviors of the network. In distributed neural nets the functions of individual processing units are uninterpretable except in terms of how they contribute to larger scale patterns of activation in the network.

Neural networks are comprised of processing units and interconnections among them. A single connection between two units is unidirectional and has a weight. The weight of the connection helps to determine if a processing unit will fire or not. The weight is multiplied by the activation value of the firing node for each connections to a given node, and this list of products is added together. The receiving node then fires depending on the result of passing the combined signal through its activation function. The behavior of a neural network is determined by the weights of connections between units, which are set in a process called training. A number of methods exist for training neural networks, including backpropogation, Hebbian learning, and evolutionary algorithms.

Neural nets can be used to make associations, to recognize patterns of stimulation, to classify data, to make predictions, to generalize to never before seen information, and are used both in commercial applications and in scientific research on cognition. Neural networks come in a number of different architectures. Some networks have recurrent connections allowing for temporal feedback, and other networks only feed forward in their propogation of signals. Most neural networks have hidden units but some do not. Recurrent neural networks, incorporating an element of feedback, are capable of being influenced by temporal relations in the input stream. These networks can become receptive to patterns that include time as an important variable.

6.4.2 The Questionaire

To gather data to train a neural network on, a questionaire was distributed to a total of about 160 human research participants in psychology, computer science, and cognitive science classes. The questionaire's instructions requested that the students rate limericks on a scale from one to six describing how "creative" each limerick is. The questionaire that was distributed lists 50 limericks. Four different versions of the questionaire were handed out which contained the same limericks in different arrangements to expose any possible carryover effects caused by the order of the limericks. Half of the included limericks were generated randomly using the limerick generation toolkit, and the other half were found in collections of limericks or were written by myself and my advisors.

6.4.3 The Data from the Questionaire

After gathering the data, it was analyzed, yielding the information that the data was evenly distributed and that there was a certain degree of consensus on the creativity of the different limericks. This made it possible to use the collections of ratings, after entering them all into the computer, to generate a modal rating for each limerick. The number encodes the most commonly chosen value for the corresponding limerick. Looking at the data, one thing that is clear is that limericks written by people are granted higher creativity scores than the mechanically generated limericks.

6.4.4 Network Architecture

The recurrent network was selected as the architecture to utilize for the task of associating and generalizing the connections between limerick poetry and its assigned value. There are 77 input cells encoding the template of a syllable, 40 context cells, 40 hidden cells, and 6 outputs.

This network accepts syllables of a limerick one at a time. Once a syllable arrives, a rating is established at the output cells. Because of the recurrent character of the network, the rating ascribed to the previous syllable influences the rating ascribed to the next syllable. In this architecture, the network builds up a rating over time– precisely the time it takes to read in one limerick. After the last syllable of the limerick is processed, the activations of context cells are reset. The rating of a limerick is the rating built up after examining the sequence of syllables that make up the limerick.

6.4.5 Encoding of Limericks in the Network

There are two important facts concerning the coding of limericks in the neural network.

1. There is an distributed coding of entries using an 11-bit vector. The 11 nodes encode a number from 1 to 1107 corresponding to the index of the lexical entry particular syllable belongs to. Using an orthogonal localist encoding of 1107 nodes would be a better approach as it would ensure that the network would not establish spurious connections, but this idea did not survive because it produced a network too unweildly in terms of computational complexity. The 11 nodes encode 1107 configurations because there are that many entries in the lexicon. To make clearer the purpose of these nodes, they are there so that the network may associate various words and ordering of words as significant in the processes of determining what properties of the limericks are associated with what fitness values. The coding makes some semantic inductions possible.

2. Phonology is encoded by presenting each syllable as a CCVSCC cluster, where V=vowel, C=consonant, S=strong/weak beat. Each phoneme is encoded as 13 bits that encode the features of consonant/vowel, voicing, manner1, manner2, place1, place2. This coding system is similar to the one Plunkett and Marchman employ in their demonstration of English past tense learning (Plunkett, 1991). The phonetic coding must be applied to all the elements in the CCV[S]CC cluster along with the bit encoding stress. The five 13-bit phonemic codings along with a single bit to indicate where the stress is, make up 66 of the input units encoding the syllable. The previously mentioned 11 units encoding indexical information are added to this, giving the syllable's coding a size of 77 input units.

6.4.6 Training the Neural Network

The neural network was built in Windows Tlearn, where it was trained with limericks from the questionaire as data and their corresponding modal ratings as the teach set. To allow for later testing of the neural network's ability to generalize, ten of the limericks were omitted from the training sessions. The excluded ten limericks were half human limericks and half computer limericks, in order to preserve the original ratio. The result of this training is that the neural network now demonstrates the behavior of assigning high scores to the meaningful limericks written by people and low scores to the nonsense poems. More significantly, the neural net demonstrates the ability to generalize to the limericks that were excluded from the training set, assigning high scores to the meaningful ones and low scores to the randomly generated limericks. This indicates that the network has picked up on some of the features that make the poems written by people better than the poems written by the machine so far. The neural net, with its current set of weights does not differentiate between the six levels of creativity that the people have provided it with, but it does make a clear distinction between highly rated and poorly rated limericks. The network has demonstrated successful differentiation between two types of limericks: good and bad. This is important because though the grain of its assessments are course, it has provided a viable measure of fitness for use in POEVOLVE.

6.5 POEVOLVE in Context

The underlying considerations involved in the design of POEVOLVE relate to the earlier discussion on "blind variation". The blindness of variation refers to the fact that it is not influenced in any way by the subsequently gauged value of things it produces. The kinds of variation that the mutation and crossover operators perform are algorithmic and are selected in a way that has no relationship to the success or failure. This aspect of the evolution program is without a doubt, essential to the conceptual evolution model, and is accomplished by the current system. For this reason, the specific algorithms used are not all too important as long as they make it possible for something recognizable good to be produced out of something else recognizably good. This is possible despite the arbitrariness of the kinds of transformations the mutation and crossover methods perform.

One theoretical problem with this this model is that it lacks the aspect of being able to develop ways of shortcut the process of blind variation and selective retention, and thereby increasing the amount of successful production. The program is not capable of transforming its own conceptual space. In this model, the "conceptual space" is made up of both the "sense of value" provided by the neural network and the constraints on variation inside of the absolute constraint of the limerick form. The process of making variations on individuals is defined by the instructions for blind crossover and mutation. The system includes no way of altering the variation process or developing preferences for certain kinds of variation over others. In addition, the neural network as a measure of value is a fixed function that can only ever be explored and can never be transformed. A system based on the current model but which is capable of reshaping its conceptual space will be much closer to genuine creativity, but a discussion of how this might be accomplished is beyond the scope of the present study. Nevertheless, POEVOLVE has all the essential properties of something that can be called creative, including evidence that it actually did produce creative work.

6.6 Is POEVOLVE Creative?

The system is theoretically capable of developing limericks that are more favorable than limericks from previous program runs. The reason for this is that the fitness evaluator, a component that has demonstrated an ability to exhibit some of the same criteria used by people in judging the poems, it pick up on those limericks that happen to meet these criteria by chance. The mutation, crossover, and direct copying operators will in turn focus on these "fit" limericks by bringing them into the next generation, and by bringing variations of them into the next generation. Since the next generation will contain copies of the adaptive limericks and possibly adaptive mutations and combinations of them, it is theoretically sound to hypothesize that the system is capable of keeping this process up for a while as well. However, looking at the data produced by POEVOLVE, there is not clear evidence that it has yet demonstrated the ability to sustain the flow of success. This seems to be what has happenned, but it is not yet clearly the case that the system has learned. If it can be shown that the system improved according to the network's standard it is arguable that POEVOLVE has already demonstrated creativity by being novel and by upholding its own quality constraints on novelty.

The graphed data found in Appendix A are the average ratings given to populations of 100 limericks by the evaluator over a period of 1000 generations. Two different tests showed that a significant amount of progress was acheived. Pearson's test of correlation revealed a trend toward better ratings over time. A T-test using the initial ratings and final ratings after 1000 generations indicated that change occurred in the system. If these results are significant, then POEVOLVE has changed for the better over time, in the sense that it optimizes what it deems valuable (a sense acquired from exposure to the central tendency of human ratings). If the program has in fact improved in this sense then the goal of making a simple model of creative processes can be considered successful, and the program can be considered creative, at least according to the view supported here that a creative process is a process that both produces novel material and is driven by a sense of quality.

Confusingly, however, results counter to these have also been discovered. A program run of 10,000 generations, still utilizing a population size of 100 individuals, returned results indicating that no learning occurred. This indicates that further testing will be necessary.

6.7 A Proposed Turing Test

The POEVOLVE software may have demonstrated learning and the ability to be novel while optimizing quality on its own terms, but a more convinving test of the creativity of the program is a tailored version of the classic Turing Test which was based on the "imitation game". The classic Turing Test was designed for what are now referred to as "chat bots". A chat bot is a computer program who attempts to carry on a conversation with a human who interacts with the program as he or she would with another person (limitted only to textual communication). To pass the test, the software must be able to trick the human user into believing it was actually a person. The argument in favor of this test's validity as a measure of a machine's genuine intellectual potential is that a machine would have to be quite intelligent in order to be convincing enough to appear intelligent.

For purposes of judging the intelligence of a program that writes poetry, the Turing Test may be stripped down to its essentials: a questionaire aimed to answer the question of whether or not the program's autonomously improvised "better than random" poetry seems better to humans. The questionaire will contain limericks from a set self-rated as good, mixed up with a set self-rated as bad. If those limericks the program evolved and considered better are significantly matched with the higher human ratings on average, there will be enough evidence to say that the program actually improved the quality of its own work in a way consistent with human concepts of what is good in the texts of the poems. Recall that the neural net has already shown the ability to generalize accurately (although admittedly at a coarse grain) to novel limericks left out of the network's training session. This in itself shows that the program's probable improvement on its own self-graded terms may more than just an artificial self-assessment of its own success– it might be correct about its own work. The best way to tell will be to run a Turing Test. The test can be performed at this time, but at the time of this writing it has not yet been done.

7 Bibliography

Section 3: Introduction

Biles, J. A., Anderson, P. G., and Loggi, L. W. Neural Network Fitness Functions for a Musical IGA. Technical Report, Rochester Institute of Technology, 1996. online. http://www.it.rit.edu/ jab/SOCO96/SOCO.html

Boden, M. A. (1996) Artificial Genius. *Discover*, October.

Burton, A. R. and Vladimirova, T. Genetic Algorithm Utilising Neural Network Fitness Evaluation for Musical Composition. Technical Report, University of Surrey, 1997. online. http://citeseer.nj.nec.com/burton97genetic.html

Dagli, C. H. & Sittisathanchai, S. (1993) Genetic neuro-scheduler for job shop scheduling. *Computers and Industrial Engineering*, 25(1-4):267-270, 1993.

Ferrero diRoccoferrera, G.M. (1969) Poetry by Computer. *Computers and Automation*, 18, 34-35.

Hartman, C.0. (1996) *Virtual Muse: Experiments in Computer Poetry.* Wesleyan University Press

Schneider, G., Schuchhardt, J., and Wrede, P. (1995) Amino acid sequence analysis and design by artificial neural network and simulated molecular evolution– an evaluation. *Endocytobiosis and Cell Research.* , 11(1):1-18, 1995.

Turing, A. (1950) Computing Machinery and Intelligence. *A Historical Introduction to the Philosophy of Mind.* ed. Morton, P.A. Ontario: Broadview Press.

Section 4: Poetry

Turco, L. (1986) *The New Book of Forms: A Handbook of Poetics.* Hanover: University Press of New England.

Section 5: Thoughts on Creativity

Boden, M.A. (1994) What is Creativity? *Dimensions of Creativity.* ed. Boden, M.A. Cambridge: MIT Press.

Calvin, W.H. (1996) *How Brains Think: Evolving Intelligence, Then and Now* New York: Basic Books

Campbell, D.T. (1960) Blind variation and selective retention in creative thought as in other knowledge processes. *Psychological Review*, 67 380-400.

Csikszentmihalyi, M. (1999) Creativity. *MIT Encyclopedia of Cognitive Science.* Cambridge: MIT Press

Finke, R.A., Ward, T.B., & Smith, S.M. (1992). *Creative Cognition: Theory, research, and applications.* Cambridge: MIT Press.

Gabora, L. (1997) The Origin and Evolution of Culture and Creativity *Journal of Memetics-Evolutionary Models of Information Transfer*, vol. 1.

Goertzel, B. (1995) Evolutionary Dynamics in Minds and Immune Systems. *Chaos Theory in Psychology.* Westport: Greenwood Press.

Gruber, H.E. & Davis, S.N. (1988) Inching our way up Mount Olympus: the evolving-systems approach to creative thinking. *The Nature of Creativity.* ed. Sternberg, R. Cambridge: Cambridge University Press.

Harth, E. (1999) The Emergence of Art and Language in the Human Brain. *Journal of Consciousness Studies*, 6, 97-115

Hofstadter, D. & Fluid Analogies Research Group (1995) *Fluid Concepts and Creative Analogies: Computer models of the fundamental mechanisms of thought.* New York: Basic Books.

Johnson-Laird, P.N. (1988) *The Computer and the Mind.* Cambridge: Harvard University Press.

Perkins, D.N. (1994) Creativity: Beyond the Darwinian Paradigm. *Dimensions of Creativity.* ed. Boden, M.A. Cambridge: MIT Press.

Simonton, D.K. (1993) Genius and Chance: A Darwinian Perspective. *Creativity* ed. Brockman, J. New York: Simon and Schuster.

Skinner, B.F. (1953) *Science and Human Behavior.* New York: Macmillan.

Ward, T.B., Smith, S.M., & Vaid, J. (1997) Conceptual Structures and Processes in Creative Thought. *Creative Thought: An investigation of conceptual structures and Processes.* Washington, DC: American Psychological Association.

Section 6: POEVOLVE

Fogel, D.B. (1995) *Evolutionary Computation: Toward a new philosophy of machine intelligence.* New York: IEEE Press.

Fogel, L.J., Owens, A.J., & Walsh, M.J. (1966) *Artificial Intelligence Through Simulated*

Evolution. New York: John Wiley.

Grefenstette, J.J. (1991) Strategy Acquisition with Genetic Algorithms. *Handbook of Genetic Algorithms* ed. Davis, L. New York: Van Nostrand Rheinhold.

Holland, J.H. (1998) *Emergence: From chaos to order.* Reading, MA: Addison-Wesley.

Michalewicz, Z. (1992) *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin: Springer-Verlag.

Mitchell, M. (1996) *An Introduction to Genetic Algorithms.* Cambridge: MIT Press

Mitchell, M. (1999) Evolutionary Computation. *MIT Encyclopedia of Cognitive Science.* Cambridge: MIT Press

Parks, R.W., Levine, D.S., & Long, D.L. (1998) *Fundamentals of Neural Network Modeling: Neuropsychology and cognitive neuroscience.* Cambridge: MIT Press.

Plunkett, K. & Elman, J.L. (1997) *Exercises in Rethinking Innateness: A handbook for connectionist simulations.* Cambridge: MIT Press.

Plunkett, K. & Marchman, V. (1991) U-shaped learning and frequency effects in a multi-layered perceptron: Implications for Child Language Acquisition. *Cognition*, 38, 43-102.

Pruett, P.S., Levine, D.S., Leven, S.J., Tryon, W.W., & Abraham, F.D. (1995) Introduction to Artificial Neural Networks. *Chaos Theory in Psychology.* Westport: Greenwood Press.