A Survey of Artificial Life and Evolutionary Robotics

John F. Walker and James H. Oliver *

April 1, 1997

Abstract

The authors present a summary of classic literature addressing the topic of Artificial Life (A–Life). From roots in mathematics, computer science, and biology, this new discipline has proven itself as a valuable tool in a wide variety of engineering applications.

The most commonly used A–Life technique in engineering applications is genetic algorithms (GAs). However, there are many other variations and methods based upon the evolutionary or biological analogy. This paper presents a survey of the entire family of A–Life techniques available to the engineering research community. The paper concludes with a detailed investigation of one particularly promising application, a new hybrid field encompassing engineering, robotics, and A-Life: evolutionary robotics (ER).

*Mechanical Engineering Department and Iowa Center for Emerging Manufacturing Technology, Iowa State University, Ames, IA, 50011, email: (volker|oliver)@icemt.iastate.edu

1 Introduction: Why Artificial Life?

Darwin[17] was the first to describe evolution as the force which propels life into complexity. Evolution is the most powerful force our planet has ever witnessed and it has transformed the Earth from a collection of thermally cycling chemicals to an indescribably complex interaction of behavior and diversity.

The forces which initiated and drive this process are not understood. Like all observable physical events there must, however, be an underlying set of laws and principles which govern life and evolution. Humanity does not understand these 'rules of life' but may be able to imitate them. If this force can be harnessed the potential benefits of the resulting revolution of technology would be difficult to imagine.

2 A Brief History of Artificial Life

For nearly fifty years, scientists from a variety of fields including computer science, biology, and mathematics have been fascinated by the potential to abstract the mechanics of evolution. Until the mid- 1980's pockets of seemingly disparate research in areas such as cellular automata, self replicating machines, and genetic algorithms were pursued more or less independently of one another.

The term "Artificial Life" (A-Life) was first used to describe this broad spectrum of interests in 1987 when a loosely organized gathering of scientists and enthusiasts took place at Los Alamos National Labs[42]. In 1989 the first conference[38] devoted specifically to A–Life took place, marking the beginning of this new field as a unique hybrid of biology, computer science, simulation, and engineering. Since then A–Life has grown quickly, gaining popularity in a variety of applications, including engineering design.

2.1 Early Ideas and Pioneers

The origins of A–Life predate modern digital computing technology. The earliest explorations were conducted by creative theoreticians primarily as thought experiments. The field matured concurrently with digital computer technology, and as the theoretical foundations were established, it became feasible to implement A–Life methods in software. We summarize the contributions of three particularly influential A–Life pioneers who inspired many others to pursue research in this field.

2.1.1 Von Neumann

If any one person could be called the father of A-Life, it would be John von Neumann. He identified the minimum requirements of any self–replicating organism, invented cellular automata, and is credited with inventing game theory (with Oskar Morgenstern)[1]. Von Neumann also laid the foundations for a bold endeavor: attempting to understand how to simulate the processes of life and evolution.

Von Neumann was fascinated with the self–reproduction of life. Through thought experiments, he attempted to define the minimum required system which could perform this elusive task. The result was a four component system defined by:

A. A duplicator component which reads instructions and copies them.

B. A control apparatus to direct the activity of the creature.

C. A factory component which follows instructions to build.

D. The instructions themselves.

The system is assumed to exist in an environment that contains all of the material necessary to construct a replica. The duplicator (A) reads a piece of data in the instructions (D) and copies it. The controller (B) receives the instruction from the duplicator (A) and directs the factory (C) to make the relevant part. This process is repeated until a complete copy of the apparatus is created by the factory.

When the 'child' is complete, it is detached into the environment and is given the copy which was made of the instructions (D) as they were being read by the duplicator (A) and executed. The great importance of the duplicator is now revealed: without its own copy of the instructions, the child would never be an independent copy of the apparatus. With the instructions, it is a complete functional duplication *including the ability to produce its own offspring*[63]. Note that von Neumann obtained this result through thought experiment and *preceded* the discovery of the DNA molecule by several years[42].

Von Neumann is credited also with the introduction of the cellular automata (CA)[13] in 1953[42]. CA are machines that perform fast computation using massively parallel simple processing. CA are discussed in greater detail later in this paper.

2.1.2 Dyson

Freeman Dyson is not often cited as an A–Life pioneer. His contribution lies in the exploration of ideas and honing of the mind–set surrounding the A–Life movement.

Dyson was fascinated with the work of von Neumann and the potential offered by the union of biology and technology. Rather than concentrating on the implementational details of today, he envisioned the future impact of such systems and explored the long term consequences (both good and bad) which the implementation of self replicating machines.

The power of self–replicators is in their exponential growth potential for payoff. Extending von Neumann's principles, Dyson described how one self–replicator could, given enough time, expand to fully utilize any resource capable of supporting it. An example of this is the 'fresh water' system also known as 'Artificial Living Plants'[48]. The concept uses self replicating ships afloat on the oceans of the world which reproduce and create an extra by–product: fresh water[19](p.197–198).

Dyson provided a great deal of vision and spoke of the promise of these systems. He was also aware of the potential hazards of self–replicators and always included words of caution in any discussion about them. The systems seem to yield unbounded payoff for finite investment, by seemingly violating the second law of thermodynamics. (Of course there is no violation, it only looks like it is being 'cheated' from our perspective. The total amount of entropy in the universe still increases.)

2.1.3 Holland

John Holland invented Genetic Algorithms (GA) in 1973[32]. In broad terms, GA's imitate the mechanics of evolution by applying genetic operators to populations of encoded solutions that are improved over many generations via 'survival-of-the-fittest'. When studying adaption and preparing a book covering the subject [33], the idea arose to apply this methodology to computer algorithms. Holland is a theoretical computer scientist at heart and preferred to confine his experiments to the chalk board and pad of paper.

One of Holland's graduate students (David Goldberg) became interested in GAs and implemented the theoretical algorithm and began to apply it to real problems in design and engineering. The technique amazed Goldberg with its robustness and ability to deal with very complex solution environments. Goldberg continued to study GAs; eventually writing an entire book dedicated to the subject[25]. This is still considered the definitive

(if slightly dated) work on the subject by many in the A–Life community.

In addition to GA's Holland is also credited with the concept of classifier sytems. Both of these techniques are described in detail later in this paper.

3 The Technologies of A–Life

The major A–Life methodologies are briefly summarized below. The methods are presented in the order in which they were developed. The section concludes with a focus on one of the most popular A–Life techniques, the genetic algorithm, and is impact in engineering.

3.1 Cellular Automata

Cellular automata (CA) techniques comprise a very rich subject area and an entire literature review could be dedicated to this topic alone. The first paper on CA is generally credited to von Neumann. This introductory work presented CA as a mathematical basis for biology, together with an explicit demonstration of self-reproduction in a complicated two-dimensional cellular automation. [67] Von Neumann proved that a CA of 26–states is sufficient to demonstrate the self–replicating machine[13],

A CA can be described as an n–dimensional array of cells which follow a deterministic set of rules to determine their future state. Over time very complex behaviors can emerge from two– and even one–dimensional CA with only two possible states.[8]

3.1.1 Linear CA

Stephen Wolfram experimented with one dimensional CA extensively. Although there are an infinite number of possible rule sets, broad similarities of behavior are observed in the record of the CA's behaviors. The existence of definable categories of behavior was formulated and cataloged.

Wolfram stratified their behavior into four groups. Type I CA converge to a state of all on or all off. Type II converge a simple repeating pattern. Type III degenerate to a state of random noise. Type IV, however, demonstrate complex, emergent behavior which can be utilized and investigated[68]. Type IV CA live in the transition between

type II and type III. Swimming in the sea of chaotic behavior, this is where complexity, useful computation, and the *potential* of life reside.

3.1.2 Conway's Game of Life

Conway conducted a number of experiments with two–dimensional CA. It is amazing that many of the initial discoveries were made with a physical implementation, i.e., sea–shells on checkerboards and other bits of manual paraphernalia. Most of these earliest experiments were conducted in the common room of the Mathematics Department at the University of Cambridge[42].

Conway sought to reduce the complex 26–state von Neumann CA to a *binary* CA; one where each cell is either off or on. If Conway could prove that Turing (universal) computation was possible, then all CA could be shown to reduce to this simple representation. Although the goal was never achieved with manual methodology, the interest of people with access to actual computers allowed more complex structures to be obtained.[8]

The simple behaviors observed in the common room such as 'gliders', 'blinkers', and 'puffers' of were replaced with clouds of complexity. For example, huge shimmering structures emit gliders and formations of intricate repeaters combining together into explosions of chaotic (but not random) interaction. These were the tools needed to demonstrate [69] that general computation was possible with a binary CA.

3.1.3 Self Reproduction and Langton

Christopher Langton noted that cellular automata were considered to be ideal structures for modeling self-reproducing "machines" by von Neumann[13] and yet no one had demonstrated such a CA. (It had only been proven that it exists, but not shown.) Thus, Langton began his quest to achieve this goal.

Codd[16] proved that as few as 8 states could be used implement a self replicating CA. Langton was able to build a functional 8–state CA which could self–reproduce. The details of this are presented in his 1983 paper[37].

Langton also explored the idea of a single parameter λ to describe the 'sweet spot' where the Wolfram type IV CA lived[39]. While it can be argued that a single parameter is too much of a reduction to use as a description[41], the idea of life as a dance on the shores of randomness had returned.

3.2 Simulated Annealing

Simulated Annealing (SA) is classified as a direct parameter optimization technique developed by Kirkpatrick[34]. The name is derived from the imitation of the physical process of annealing used in the material processing of metals and ceramic compounds. The basic idea of SA is described below.

3.2.1 Steps in SA

First, a point is chosen somewhere within the allowable solution space of a problem to be optimized. Then, randomly vary the input parameter(s) in proportion to the temperature parameter T (this can also be thought of as *mutation*). Check this new solution: if the new solution is better, automatically 'accept' the new solution and start the next iteration. If the new solution is worse, calculate a probability of acceptance of the new solution based upon the current 'temperature'; where a higher temperature indicates a greater probability of accepting a bad solution.

Thus, T and its rate of decay determine the course of the optimization. Generally T decreases as the simulation proceeds which has one or both of the following effects: a) the perturbations of the current 'solution' for the parameters become increasingly small; and b) the probability of accepting a candidate solution which is less optimal than the current solution decreases. The process terminates when either the temperature has dropped below a certain level, or the desired level of optimization is achieved.

3.2.2 Not Life, but Still Nature

Why is this considered A-Life? Granted, the SA algorithm is very simple, but the stochastic nature of its optimization makes it very similar to the other classical A-Life techniques. SA has been used and tested extensively in the optimization literature, but it can be slow to converge and is almost as vulnerable to local optima traps as many of the other direct and conjugate gradient techniques [49]. SA is very robust when faced with discontinuous fitness landscapes, so it continues to see use in the scientific world.

3.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are an extension of SA. In an EA, the current 'solution' (set of parameters) is replaced by a *population* of solutions. Following the methodology of the SA, each of the solutions is now mutated independently. These new solutions are tested and compared to their 'parent' point. The new solution is accepted if superior to its parent or rejected if worse. It may also have a decreasing probabilistic acceptance if worse, as in the SA.

An EA is much less likely to become trapped in a local optima as long as the initial population is sufficiently spread over the allowable solution space at start–up. Adjustments can also be made to push the solutions apart so the genes are not all trapped in the same optima. For example, give each member of the population a 'charge' and model the repulsion of the solution points in the fitness space.[3]

3.3.1 One Step Beyond SA

There is not much difference between running an EA and repeating a large number of SAs and statistically looking at the results or brute-force searching all the end solutions. The addition of solution repulsion makes the analogy less valid, but these 'spreading' techniques increase the high computational costs of EAs.

3.3.2 Mutation is the Name of the Game

For both the SA and EA techniques, mutation (random variation of input parameters) is the only operator. (SA can also be thought of as an EA with a population size of one.) In classical optimization terms, the algorithm is simply performing a stochastic hill climb in the local solution space. Considering the large number of candidate solution tests required, the gain in robustness of the method is seldom worth the relative slowdown compared to methods like Hooke & Jeeves[49], Nelder-Mead Simplex[20], or Powell[70]. The SA/EA techniques still win out in robustness, so their continued use in science is not in doubt.

3.4 Genetic Algorithms

Genetic Algorithms (GAs) are another step in the direction of real biology from the EA. John Holland began wondering why so many biologists virtually ignored genetic recombination as a vehicle for change in evolution. A practical reason to ignore breeding was because it was so difficult to incorporate into simulation and was poorly understood. This was justified by biologists with arguments such as "This has little effect since it operates on a uniform population" or "The effects are negligible compared to those of mutation"[25].

3.4.1 John Holland

Holland formulated the early ideas of the GA while preparing his book, *Adaptation in Natural and Artificial Systems*[33]. Initially the GA idea was basically ignored; how could a randomly directed process yield good solutions to hard problems? The answer lies in the implicit parallelism of the GA which is discussed later in this paper and by Goldberg[25].

The basic algorithm of the GA is simple and is shown in figure 1. Start with a random population of genes. Evaluate the *fitness* of each of the members of the population (each gene). Based upon the fitness, choose a percentage of the genes to be *parents* (with either a probabilistic or deterministic choice). Breed pairs of these fit creatures to create offspring genes (e.g., cross–combine two genes into a new gene). Replace a portion of the population (stating at the low end of fitness) with the child genes. Finally, go to the fitness evaluation step and repeat.

By repeated application of this routine, the *fit* genes will be more likely to remain in the population and be selected for the breeding step. This means that as the GA proceeds, poor solutions will be removed from the population while good, intermediate solutions will remain to be mutated and bred with other good solutions. Good sub–ideas have the potential to be combined with other good sub–ideas into a more fit gene which incorporates both. Putting this in the terminology of optimization: the mutation can be thought of as a local exploration of the solution space and the breeding as an 'informed random jump' in the solution space.

Create starting population
Repeat
  Fitness evaluation
  Selection for breeding
  Breed offspring (with possible mutation)
  Replace unfit with offspring
Until Done*

* a *deep* subject

Figure 1: Summary of the genetic algorithm.

3.4.2 Coming of Age

The GA has become the most commonly used A–Life technique, both in engineering and within the A–Life community itself. It has taken its place as one of the most heavily used parameter optimization techniques for solving a vast variety of problems. The GA is among the easiest to use yet most generally applicable of the A–Life methodologies and it has enjoyed wide success and increasing use in the technical world.

GAs are also useful as a sub–component of other systems. Classifier Systems (detailed below) use the GA for the creation of new rules, GAs are used to program Artificial Neural Networks (ANNs) for a plethora of purposes, and the GA is used as a benchmark for comparison of new methodologies. Since its acceptance, the method has grown in use so quickly that a comprehensive list of its uses would be nearly impossible to prepare and would be out–of–date before publication.

3.5 Rule Based Systems

Classifier Systems and L–Systems are similar in structure, but vary in their application and purpose. Classifier Systems attempt to understand learning and/or interactive development while within an environment. L-Systems attempt to understand the development of a *phenotype* (the manifestation) from a given *genotype* (the encoding). At first glance, the grouping of these two methods may seem inappropriate. However, both of them share the purpose of trying to find artificial representations of rules or

directives. These formulations are then used to guide development of a phenotype or direct the process of decision in an AI.

3.5.1 Classifier Systems

In GAs, the candidate solution (aka. creature, agent, or gene) is frozen during the fitness evaluation. While this is appropriate for testing a design or the physical structure of a creature, there is no vehicle to model learning of the creature *during* the fitness evaluation.

Classifier Systems were introduced by John Holland in an attempt to create and model a methodology of ongoing learning. The algorithm creates and tests sets of *rules* which recommend actions and make bids to indicate confidence levels. Successful actions are, however, usually the result of a number of actions carried out in sequence which lead to the final, high pay–off action. If the intermediate actions are never rewarded, then the system will never learn. By what mechanism can the algorithm identify the beneficial intermediate or preparatory actions? (This is the "credit assignment" problem.)

Imagine that someone is trying to teach a program to play checkers. The task is relatively simple, yet the successful player needs to implement tactics of setting–up the opponent which may span several turns. For example, consider the very successful move of a double jump. Waiting for the opportunity to just appear is unlikely; it is usually the result of several preparatory moves which expose the opponent.

Holland implemented the algorithm to reward not only the pay–off action, but to also reward actions which lead to the result. Each preceding action is given a partial pay–off based upon its distance from the paying action (how long ago was it used). As the algorithm executes, the actions which lead to success will be repeatedly rewarded with small pay–offs and remain part of the creature's knowledge base. (This is the "bucket brigade" method.)

The system evolves by starting with a random set of rules and actions. The rules are used for an arbitrarily chosen time and then their relative success and usefulness is evaluated. The most fit rules are bred (i.e., parameters are cross–combined) with a GA and new rules are created. These replace the least fit rules in the creature and the process repeats.

3.5.2 L–Systems

L-Systems were created by Aristid Lindenmayer[43] in the attempt to imitate the recursive developmental cycles of biological life forms. In biological life, every cell contains all the information necessary to construct the entire creature. Yet each cell seems to 'know' its purpose in the larger organism and develops into the specialized cell needed to this location. Lindenmayer theorized that there must be a control mechanism in the code or in the mechanism which decodes it which can direct the application of internal components of the gene and ignore others. He believed this mechanism to be recursion.

Ben Hesper and Pauline Hogeweg used the recursive application of relatively simple rules to recreate this phenomena in simulation[42]. Although this result simulates simple plants and their development, it does not prove that L-Systems accurately model biological processes. Beginning with very simple microscopic plants, the work was later extended to the development of the branching structures of larger flora. This work continues and is being extended by Przemyslaw Prusinkiewicz[50].

3.6 Artificial Micro–Biology

In 1989 Thomas Ray, a rain forest biologist, set out to produce a truly open–ended system of evolution in a computer. The basic premise was to build a nearly minimal program which was able to reproduce itself in the core memory of a computer.

3.6.1 Thomas Ray and Red Code

What is the result of randomly combining or mutating binary executable code? The answer is "a program that will just core–dump" or the equivalent statement of almost certain disfunction. The reason for this is that computer code is *fragile*. Changing only one bit will cause a wrong memory address to be changed, will send the processor's attention to some nonexistent space, or will cause some similarly fatal task to be requested. In a protected OS (like Unix), this will only cause the program to crash; in an unprotected OS (DOS, Win95, Mac OS) it is likely to freeze the entire machine. Either result makes it difficult to run any sort of evolutionary process or simulation since the whole task will die with the bad command.

Thinking of this in mathematical terms, consider the set of all binary strings below a certain length. A subset of these strings will be valid computer programs relative to a certain computer language (where valid in this arena simply means "it will not

crash the machine"). A subset of these will function properly to do some desired tasks or at least some sub–part of one of the desired tasks. Unfortunately, the ratio of this final set is so small compared to the starting set that it will be nearly impossible to find[40]. The same is true for even the second set; randomly perturbing *even one bit* of a valid, functional program is very likely to completely destroy its functionality. How then would one ever be able to 'evolve' computer code?

To solve this problem, Ray imitated the function of a game played by hackers known as *Core Wars*. The game is played within a software simulated processor running inside a real processor. This layer isolates the real computer, its operating system, and its memory from destructive commands of the simulation. It also prevents code derived for the simulated processor from affecting the real computer since the two languages are functionally incompatible. This other language is named *Red Code* and Ray used a similar system for his research. (Ray was 'helped' with this by some friends: his computer training was taking place *while* he was writing the code.)

By running the evolution in a 'simulation of a simulation' the evolution can be observed and tested safely. Thus 'illegal' instructions can be penalized and become another fitness pressure optimizing the code. This now provides the mechanism for a gradually increasing fitness measure which can allow reasonable exploration of the solution space. Without this layer, the test becomes too stratified: most changes would simply make the creature 'non–functioning' with a fitness of zero. This would isolate the first functional gene in the solution space and cause evolution to all but stop since there is no 'slope' or direction of improvement for the algorithm to 'climb' in its quest for higher fitness. Thus, by making the use of invalid commands a penalty, the evolving ecology can still explore the solution space. An innovative strategy with a few minor errors can survive long enough to optimize the errors away.

This creation was dubbed *Tiera* and became well known for its truly revolutionary origin and adherence to minimal human influence. *Tiera* survives to this day, having been extended and expanded to add new experimental possibilities and the ability to run dispersed around the Internet on any machine whose owner is willing to donate CPU time or who wants to participate in the continuation of the experiment[51].

3.6.2 The Promise of Evolving Binaries

Imagine now that we use the isolation and robustness of testing on a simulator but start to evolve code in a language which would actually function on the machine. Actual functioning binaries could be evolved and tested. Once the creator was satisfied, the

simulation layer could be removed and the resultant code could be run on the machine to perform the desired task it was created for.

The potential is present for entirely new methodologies and algorithms to be 'bred' to solve or optimize various computational tasks. The increasing trend toward intricate architectures of modern computers only strengthens this idea since true understanding and optimization of these very complex designs is a quickly fading reality. One could even imagine coupled design and evolution of not only the code to run a given configuration of hardware, but to design the hardware as well.

3.7 Genetic Programs

The above methods are very effective at optimizing systems which have externally created structures into which we can put parameters to describe them. This also requires that a possible form of the solution can be created by the human designing the GA, EA, Classifier, or whatever technology they choose. Unfortunately, there are many problems where it is difficult to know what is needed before the start. Other problems may be sensitive to parameter ordering or other subtle details which can greatly hinder the evolutionary processes. What is needed is a general method of algorithmic representation which can organize its own structure and adapt to the complexity requirements of the problem.

John Koza was a student of Holland's. Koza was inspired by the proceeds of one of the early International Conferences on Genetic Algorithms (ICGA). After reading about the advances and acceptance of GAs, he said,"What we really need is a way to evolve programs!" [42].

The problem is (as discussed in the section on *Tiera*) that normal computer code, both in binary and language form (C++, LISP, or FORTRAN) is very fragile. The misplacement of a single character or bit may render an otherwise optimal program disfunctional. The programs also do not provide a mechanism for gradual improvement since most 'good' intermediate solutions are centered in a large space of entirely non–functioning or 'bad' programs[36]. Koza needed a way to formulate programs which would be be both universally valid (if not correct) and be capable of incremental improvement.

3.7.1 GAs goto GPs

The framework used for these evolvable programs is parse trees. Borrowing inspiration from the LISP programming language, Koza set about creating a system to be universally valid, capable of incremental improvement, and efficiently explored by the use of cross–combination and minor perturbation (i.e., a GA).

The tree is made up of a number of different types of *nodes* separated into two broad categories: *branches* and *leaves*. Leaves have no nodes descending from them and are self contained input values. In the initial work, they were constants. A leaf could also be the input from a sensor, the output of a pseudo–random number generator, or any other value not requiring any internal computation by the GP. The branches are nodes which have a variable number of inputs from either leaves or the output of other branch nodes. The branches perform some computation using, modification, or test of the input branches and pass this on as their output (which are the inputs to other nodes higher on the tree). An example of a parse tree is shown in figure 2.

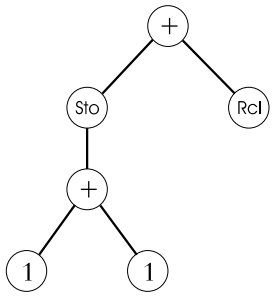

Figure 2: An example parse tree [4]

The parse tree is executed by starting at the top or final node and examining the inputs to it. Each input which requires computation is traversed until every input is calculated and the final result can be obtained. Navigating the parse tree is equivalent to executing a program, so the evolution of a GP is equivalent to evolving a program.

The structure of the parse tree is allowed to grow and shrink within specified size constraints during the mutation and breeding, and the algorithm actively 'trims' or reduces portions of the tree which are unused or growing too large. Mutation is simple: Pick a random node and change it to a new type. If it has a larger number of inputs than are already present, randomly generate new nodes to fill the empty 'link'. If it has fewer inputs or is a leaf, delete the extra, unneeded branches. Crossover is done by randomly selecting a link on two GPs; taking everything below the link on one GP, and exchanging it with everything below the link on the other.

3.7.2 A Few Problems

This idea would seem to provide a road to evolve programs, compilers, and languages. For any given computer the GP can fully utilize the power of evolution to optimize the performance of any given architecture. GPs are proven on simple problems, are fast to execute once evolved, and can be optimized further once the evolution is complete. At least, that is the theory...

Unfortunately, there are problems, particularly with scaling. As the size of the problem (and GP) increases, the application of either the mutation or crossover operations becomes less and less likely to produce any improvement[1]. Thus, the mechanism which the GA needs to evolve is stopped; along with the GA's progress.

Another problem with GPs is cross–combination of nearly identical parents. Consider the case of two identical GPs being bred. If a random location is selected on each one for crossover, the probability is very high that the same location will not be selected on both parents. Thus, the child creature produced is likely to have a very different character from the parents. This is not a problem from a 'valid program' standpoint, but this greatly slows convergence of the population and makes it unlikely that the breeding algorithm will preserver useful sub–structure within the GP.

GPs are still used and offer great potential as the components of higher level systems. Consider the GP a 'building block' which can be used in simple sub–components or systems with a higher level controller (like a finite state machine[1]) selecting among these 'behaviors'.

Despite its implementational difficulties, interest in GPs is increasing and there is now an entire conference dedicated to the application of Koza's invention. There are also a number of methodologies which can be used to improve the efficiency of the evolution by making intelligent choices about cross–over and mutation of the parse tree. One example of this is PORS[4] where the probability of selecting new child nodes and cross sites is evolved along with the parse tree.

3.8 Summary: A–Life in a Bottle

Many different methodologies have been presented. They all have their strengths and weaknesses; some methods are better suited to certain tasks than others. As a group the faults of one method are often the boon of others; and hybridization of techniques is also effective in dealing with persistent difficulties. A–Life techniques have repeatedly

demonstrated their power and robustness and will undoubtedly continue to spread in the scientific and engineering communities.

3.8.1 Many Technologies...

A–Life techniques form a loose family of approximately 7–9 (depending upon who is counting) different methods and algorithms. There are also fringe techniques relating other areas (such as flocking behaviors[52] and machine & animal vision[30, 62]) which are of interest to A-Life researchers but most do not consider to be "classical A-Life".

3.8.2 ...One Paradigm

The criteria for classifying methods as 'inside' or 'outside' the A-Life research community are not formalized, however the following list of common characteristics are observed:

- Inspired by natural (biological) phonomea: particularly evolution, learning, and behavior. (It may also be based upon failed theories of the functioning of the above, e.g., Lamarckian evolution.)
- Stochastic selection is heavily used during the process.
- Human direction of intermediate results is avoided. (This does not preclude 'steering'; only that the random element is included and allowed to explore the solution space.)

4 Genetic Algorithms in Engineering

When GAs were developed by John Holland, they were not perceived as pratically implementable, probably due to the abstraction necessary for encoding real-world problems. They were first used to solve engineering problems by students of Holland's at the University of Michigan. Holland knew that GAs had enormous potential, and the subtle power of GAs for solving practical problems did not remain undiscovered very long.

The key to the GAs' strength is illustrated by Holland's *schemata theorem*[33]. The proof demonstrates that the GA can process n^3 schemata in linear (*order* n) time with

respect to the number of operations (i.e., fitness evaluations) performed on a language of $\{0,1\}^*$. This effect is referred to as *implicit parallelism* [25].

4.1 GA's Versus Classical Optimization

GAs have been shown to be an extremely powerful optimization tool (see [25], chapter 4). The keys to the GA's success are three fold: First, they have the ability to explore local optima without becoming trapped. Second, GAs exhibit robust behavior in conditions where large numbers of constraints and/or huge amounts of training data are required. Third, GAs do not require a particular data structure for the gene being optimized and are thus less sensitive to arbitrary implementational issues than other methods.[49]

4.1.1 Parameter Optimization

Parameter optimization is a mature subject and much effort has been devoted to creating fast and robust solution methods. In some sense, a GA is simply another parameter optimization method to specialists in the area. Comparing it to some other classical parameter optimization techniques it shares many common features:

- There are a finite numbers of parameters to be optimized. (As opposed to the 'infinite degrees of freedom' of optimal control)
- It is dependent upon humans to formulate a set of parameters which can contain the solution. (e.g., a data structure which can describe the application of the parameters to the problem.)
- The optimization algorithm has low computational overhead. (This is independent of the fitness or gradient calculation.)

Many optimization techniques require derivative information in order to function[49]. These are called *gradient optimization methods*. Simple gradient–descent methods are easy to implement, *very* fast, and provably quadratically convergent (to the optimum of a quadratic function) in a finite number of steps[49]. Unfortunately, they are fragile and vulnerable to large numbers of divergent conditions. They also require second derivative information which can be hard to calculate or estimate. Gradient descent methods include Newton's Method[66, 65] and BFSG[49]. Steepest descent methods(Caoch,

1847)[49] do not require high accuracy of the gradient (particularly when coupled with Goldstein or Armijo's step size rules[49]), but are slow and easily fooled. Conjugate direction methods are quadratically convergent and avoid a number of the problems of direct gradient usage, but also suffer vulnerability to discontinuous spaces and traps of local optima. Some CG methods are Conjugate Gradient (Hestenes&Stiefel, 1952), Gradient PARTAN (PARallel TANgents), and Davidon, Fletcher, Powell (DFP)[49].

Another class of optimization methods are the *direct optimaztion methods*. The basic concept is to avoid gradient information and operate on the parameters using only a direct measure of the resulting fitness. Most of these methods use a local sampling of the surrounding fitness landscape to determine a new solution point to be used as the starting point for a new local search. Some of the most commonly used direct methods are Hooke & Jeeves[49], Nelder–Mead Simplex[20], and Powell's Method[70]. Thus, GA's can be considered as another direct optimization method with which it shares the following characteristics:

- No derivative information is required.
- The methodology is 'simple' and relatively straight forward to implement.
- The optimization algorithm has low computational overhead. (This is independent of the fitness calculation.)
- The performance is dependent upon efficiency and overhead (CPU & RAM) of quality measurement (fitness) evaluations.

GAs are particularly sensitive to the last characteristic since the fitness calculation routine is called much more often than with the other methods. This is because of the stochastic nature of the GAs search and large 'populations' of candidate solutions which are maintained. As computer power increases, this will be less of a concern; but the problem will persist as scientists want to incorporate increasingly accurate (and time consuming) tests into the fitness evaluation.

Perhaps the strongest advantage of the GA is in dealing with constraints. Most classical optimization methods increase in complexity and lose robustness as the number of constraints is increased. GAs do not suffer from this problem since constraints are easily encoded into the fitness function; often as a increasing penalty–for–violation over time (such as in Penalty Function Methods[21]). This property is again resident in the ability to use very large data sets for training. The GA is *improved* with the addition of more data to use for training. (This is not the case with all classical optimization methods)[49]

GAs are not perfect; along with advantages come the inevitable pitfalls. GAs are computationally expensive relative to other parameter optimization techniques. Consider a comparison to the Hooke & Jeeves method. Hooke & Jeeves starts with a single solution point (like SA) and then checks for improvement by varying one parameter by a finite amount in both the positive and negative directions. Of these three solution points, Hooke & Jeeves picks the best one and uses this as the starting point for investigating the next parameter. Once every parameter (which can be thought of as a dimension in the solution space) has been checked, the process begins again: usually the parameters are checked in a different order.

In the above Hooke & Jeeves implementation, each iteration takes $2n$ fitness evaluations; Where n is the number of parameters being optimized. When using a GA, a population size of $2n$ is probably too small to maintain diversity in the population. A larger population translates into more time for testing since a greater number of fitness evaluations must be performed for each generation (i.e., each iteration).

4.2 Earliest Uses in Engineering

David Goldberg was a Civil Engineering student who was interested in optimizing the configuration of natural gas lines. When he proposed using the GA to solve the problem, he was cautioned about the difficulty of the task, but allowed to continue. Surprising even their creator, the GA was able to solve the problem to a level never expected by his (Goldberg's) committee[25]. The success of the method in this and a few other early experiments was enough to interest others in this biological optimization methodology.

The robust nature of the GA and its early successes attracted researchers to this new technique. By the end of 1986, the GA had been applied to a significant number of difficult engineering tasks. Goldberg[24] and Kuo[26] used the GA for designing pump–pipeline systems; Brady used the method to attack the infamous 'Traveling Salesman Problem'[9]; Minga[47] and Goldberg & Samtani[27] attacked structural optimization problems in aircraft and trusses; and Fourman[23], Davis, and Smith[18] used GAs to compact and layout VLSI circuits.

These were very difficult tasks, especially when considering the modest computing power available in 1986 (by today's standards). With these successes, knowledge of Holland's invention spread and grew until entire conferences and companies were built around the GA.

4.3 The GA Decade

Since the mid-1980's the profound utility of the GA as a general solution technique is demonstrated by the explosion of it's appearance in technical literature. As evidence of this trend, consider figure 3 which shows the number of papers referencing GA's each year between 1985 and 1995 in one of the most comprehensive data bases for engineering literature (Ei COMPENDIX).

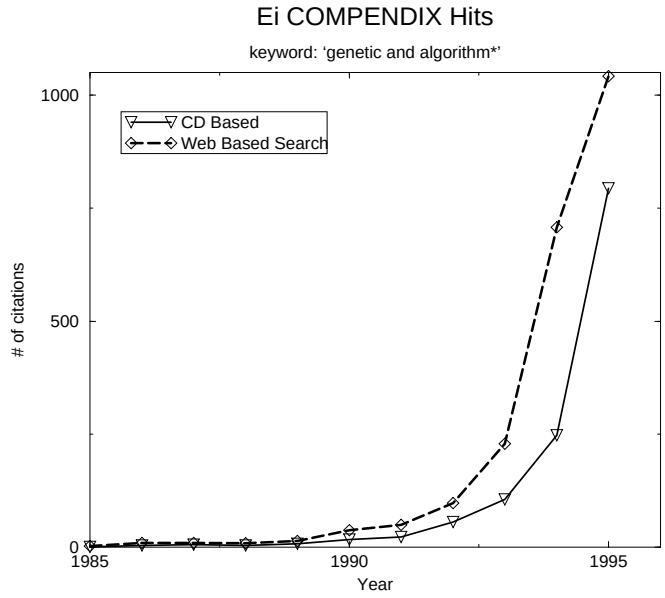

Figure 3: Number of Papers Referencing GAs by year

Obviously GAs are able to solve a great number of problems well. The method would not continue to see such expanded use if this were not true. As an optimization method, GAs are unparalled for robustness and functionality in complex constraint type problems. GAs also require fewer preconceptions and parametric assumptions than many methods. This may increase the danger of accepting incorrect solutions due to errors in the model or fitness function, but results can (and should) be verified by other methods and detailed traditional simulation and prototyping. GAs also efficiently utilize a basic axiom of optimization: it is easier to check if a solution is true (or of high quality) than to search for a solution.

GAs also show great promise as a pre–processor, or software and simulation "bug–hunter". Classical optimization is much more efficient if started near the optimum or in a 'smooth' region of the fitness landscape. There is great advantage, therefore, in using

the GA methodology as a pre-processor to find interesting and small (e.g., searchable) regions of the solution space for traditional investigation.

5 Evolutionary Robotics

The control of an autonomous robot, i.e., its *intelligence* is an inherently difficult problem. In 1960, artificial intelligence (AI) to the level of imitating a human being (the Turing test[35]) was predicted to be possible in 10–20 years. Obviously that did not occur. However, Human intelligence is certainly more than is required for basic activities of mobility and searching, so early attempts at autonomous robots were built upon AI technology of the times (now referred to as knowledge-based AI). Unfortunately, they were basically considered a failure[42].

The greatest barrier to truly autonomous robotics is not mechanical limitations of hardware. The breadth of application of fixed 'moving arm' type robotics shows that everything from handling heavy equipment to precision assembly and testing of micro–electronics is commercially usable. For mobility, one need only investigate the intracies of current radio–controlled (R/C) cars and airplanes or look at a high–end full suspension mountain bike to see that this is also not the bottleneck.

As the progress of AI systems repeatedly stalled and slowed, researchers began searching for a way to attack these goals. The problem must be solvable, because Earth's ecosystem was able to produce the complexity and intelligence exhibited by insects, fish, animals, and even humans. A logical course of action is to try and discover how nature solves these problems. Evolutionary Robotics (ER) is the manifestation of the idea of imitating nature to produce functional robots which react on a human timescale to events and changes in their environment.

The different approaches to ER all share a few basic beliefs or assumptions. First, the emergence of intelligence and complex, robust function comes from solid low level behaviors rather than from high–level representations of tasks or environments. Second, the solutions discovered through millions of years of evolution are worthy of imitation; both in the final function and in the method of discovery. Before the implementations of ER are described, a contrast of the traditional AI paradigm and its proposed replacement is given.

5.1 Knowledge Based versus Behavior Based AI

The origins of ER are tied to the origins of Behavior Based Artificial Intelligence (BBAI). BBAI is a different structure of AI which was created by scientists attempting to understand how nature implements intelligence. A structure was desired which could build upon itself; again imitating how intelligence has been built in nature: starting with the simple, single–cellular behaviors, progressing to the selection of various reflexive behavior in insects, and finally progressing to animals like opossum, cats, monkeys, and (eventually) man.

Traditional or "Knowledge Based AI' (KBAI) began in the early 1960's and continues to be an active area of research. KBAI was very successful in dealing with specific expert based tasks and reasoning about particular problem domains (e.g., a chess playing computer or medical diagnosis assistant) [45]. KBAI is designed for high level decisions, but not for quick, reflexive responses to changing or unstable situations. The weaknesses of KBAI are identified as follows[45]:

- Resultant systems are often 'slow'.
- Systems are inflexible to changing or unexpected environmental conditions.
- Systems are brittle and prone to sudden, non-gentle failure.
- Systems are difficult to link internal representation to physical stimuli.
- Systems are require accurate human–designed model of the environment.
- The systems' contingency models are vulnerable to combinatorial explosions. [14]

KBAI is generally unable to deliver the performance and stability which would be needed for independent robots. To help identify the problem, Pattie Mays summarized the characteristics of KBAI[45]. These system are:

- Model isolated
- Closed
- Singular
- Built upon declarative knowledge structures

- Non–adaptive or developmental
- Dependent upon symbolic internal representations

Unfortunately, control of autonomous robots produces many situations which violate some or all of the above conditions. The environment is always changing and there are many unexpected circumstances or combinations of events which render the predictions, conditions, and assumptions of the designer false. Since predicting every possible condition is impossible, KBAI stumbles as its inability to adapt, and the internal structure of its knowledge causes the internal expectations to be violated. Being fragile, the system is often 'stuck', chooses inappropriate actions, or spends a large amount of time trying to incorporate the conditions into its world model.

BBAI is the response: a new method imitating nature based on behavior and specific outcomes rather than on explicit knowledge. Rodney Brooks described the intelligence of nature as not a top–down approach but rather a *bottom–up* structure where very simple activities form a foundation of behavior[11]. Once the foundation behaviors are satisfied, the next higher level activities (themselves just simple selections between several lower level activities) come into play. This 'layered architecture' builds upon itself until the highest level of control is empowered.

BBAI has the following characteristics:[45]

- Multiple integrated competencies
- Situated in its environment (open)
- Emphasis on autonomy
- Emphasis on resultant behavior rather than internal 'knowledge'.
- Developmental approach to construction and continued adaptation.

BBAI is the common thread shared among the entire ER community. KBAI has its place at the highest levels of control, but to date the systems constructed are far below the point where they need or can effectively utilize KBAI[44]. The preferred method to implement and train BBAI systems is still in flux; this is the realm of the ER researchers.

5.2 Overall Paradigms and Dogma of ER

There are three basic paradigms to ER. Subsumption architecture [11], The Sussix Approach[31], and Behavior–Oriented Design[57]. Each has its strengths and weaknesses which are summarized at the end of the section.

Subsumption architecture is based upon the idea of a layered hierarchy of function. Rather than trying to create a system which has one control structure which can 'do everything', the higher level goals and tasks are built upon a foundation of more basic sub–tasks. These sub–tasks rest on a number of even lower level tasks. The process of sub–division ends at very simple activities which can be effectively modeled, understood, and implemented in hardware[11].

The Sussix Approach is based upon a very pure imitation of the methods of biology. The ANN evolved through test and simulation is the basic fundamental unit of control. The tasks and their nets are much more complex than the 'lowest level' of the subsumption architecture and a much greater burden of design is placed upon evolution rather than the hand–designed method of subsumption[31].

Behavior–Oriented Design (BOD) is in some ways a hybrid of the above methods, and yet differs significantly in implementation and hardware architecture. It is fundamentally different in that a dynamic systems approach is used to implement the hardware. Physics and the nature of the physical hardware of the robots is exploited as much as possible to reduce computation and complexity of the control architecture. It also uses a continuous operation of the system (e.g., akin to analog versus digital electronics)

These broad groupings are being presented for comparison only. Many other research sites combine, change, and otherwise tweak these ends of the spectrum. (This is even done within the 'home labs' of these methodologies.)

5.3 Subsumption Architecture

The ideas of BBAI presented above are an abstract presentation of the methods and output of the earliest work at the AI Lab of MIT. Rodney Brooks was the leader in the work here; attempting to solve the problems of building 'real robots'. Brooks also strongly expressed an opinion that simulating robots provided only a very limited insight into the problems of robotics. The only way to study robots was to build them; to directly tackle the problems and challenges in physical hardware operating in the real world.[11, 10, 22]

5.3.1 Bottom–Up Design: the Subsumption Architecture

Traditional AI attempted to design robots in the manner of a pyramid. Top level tasks were desired. The algorithm to decide between tasks was then written. Then each task had its sub–task determined and written. The usual result was something disfunctional or an algorithm so complex that a super–computer was required to execute it. Another problem of classical AI was the notion of an internal representation of the environment. The construction and maintanence of such a model is slow to compute, very hard to do, and prone to error.

Brooks imagined a *subsumption architecture* where all the tasks of controlling the robot would run in parallel, asynchronously. This was meant to provide a framework in which incremental improvements and additional complexity could be added to a functioning robot design. Each addition would not change the previous work; it would only add a layer of new functionality. The asynchronous aspect is important for two reasons: 1) The speed at which the other units function is not affected by the additional layer, and 2) the robot is always acting on the best available information and never waits for a new or existing complex system to update before reacting to its environment.

The different layers can communicate and actively suppress each others inputs and outputs. Each layer also has access to some or all of the sensory inputs from the robot and possibly the outputs of other layers. This internal network of connections is not fully connected and in Brooks' robots is generally sparse[11, 12].

5.3.2 Layered Incremental Design

The best illustration of building a robot with this method is a concrete example. Imagine that someone wants to design a robot to do searching behavior. Since insects are very good at finding food in large dispersed spaces, its structure will be imitated: six legs coming out of a central body with a 'head' containing the majority of sensory apparatus.

Let the process begin by designing the 'zeroth' level of control. This will have as inputs the positions of the legs and whether or not two touch sensitive 'antenna' indicate that it is touching a wall or other obstacle. The purpose of the zeroth layer is as follows: stand and balance without falling. If on the ground, stand and balance. Its output will be desired leg positions and whether or not it is currently successful at standing (i.e., the body is not touching the ground).

Next, a first layer will be created. This layer will have access to a battery of sonar sensors which will give a crude 360 degree measurement of the range to the nearest

object in the ray extending straight out form the sensor. It will also have access to the output of the zeroth layer and the antenna. The purpose of this layer is to perform movement and collision avoidance. If the zeroth layer is not currently standing, this layers action is suppressed by the unsuccessful output of above. Collision avoidance is done by moving away from objects the antenna are touching and not coming too close to objects sensed by the sonars during movement. If the sonar sensors indicate ranges becoming smaller when the robot is not moving in that direction or faster than the robot is moving, it will move away from that direction (e.g., run away). Note that this layer contains several sub–tasks such as comparing range data to old values, and logic to chose what directions will correct antenna collisions. The algorithm will attempt to center the robot in the given available space if it is run as the top level controller.

The second layer will provide a motivation to move. It will contain a search algorithm and will chose directions of movement by identifying corridors and rooms from camera data. It will have sensory input from two video cameras and the sonar sensors. Its outputs will be able to suppress the 'centering' goal of the first layer, but will be unable to over ride the 'run away' or 'stand and balance' directives.[11] Additional layers can be added, following the continued building approach given in the example.

This architecture is much more robust in changing conditions. If the ground is uneven, for example, the robot will give balance its first priority. This will probably cause movement and the other tasks to become slower, but they will still be successful if the roughness of the terrain is within a range the robot is capable of physically dealing with the new, harsher conditions. (e.g., It is unlikely that it would be able to stand and balance on water.) [12]

5.3.3 Behavior Replacing Internal Models

Rather than attempting to model the environment, the robot is given desired behaviors. The different behaviors come into play depending upon the sensory state which is simply a combination of the environmental state and noise conditions. Since performing behaviors changes the robot's environmental perception, the behaviors which will be used are being changed by the behaviors which were used in the past. Thus, an internal world model is not necessary because the different behavioral modules of the robot are communicating through the world[22].

Another challenge to overcome was the issue of sensor fusion. The goal of sensor fusion is is to incorporate the input from several different types of sensors and incorporate it into one model of the environment. This task is very difficult and prone to produce

erroneous results in degenerate combinations. With the emphasis on behavior and the existence of independent tasks provided by the subsumption architecture, this task became irrelevant and sensor *fission* was implemented to separate the sensory tasks between the competing agents of the architecture. Thus, the different sensors are tied to the initiation of different behaviors and the need for resolving sensor conflict and noise is removed[22].

5.3.4 Pros and Cons

The subsumption architecture was able to produce results superior to everything which came before it. In this ultimate test, it could only be considered a great success. The implementation of robots using this method also showed how little true complexity was needed to achieve complex behavior if one was willing to accept the task of designing everything from scratch; from processor boards, to sensors, to compilers and languages[22].

Although the imitation of nature was a guiding philosophy, subsumption and its building blocks were still being meticulously designed by humans. The 'evolution' of designs didn't mean a GA operating on the design or a classifier system training the behavior selection routine; it meant the next version of the robot or a re–write of some aspect of code. More of natures methods were waiting to find their way into robotics.

5.4 Neural Nets, Incremental Complexity, and COGS

The School of Cognitive and Computing Sciences ('COGS') at the University of Sussex at Brighton is another group playing a key role in the early development of ER. COGS has and is attempting to bring even more imitation of the animal world into the field of autonomous robotics. The basic assumption is that if humans could build robots that worked as well as animals or even insects, we would be very far ahead of the state-of-the art in robotics. This imitation was not to be confined to physical structure; but would also include the imitation of a biological brain: in the form of an artificial neural network (ANN)[29].

The COGS group has a decidedly different approach to bottom–up design than the subsumption architecture. This methodology is based upon the gradual increase in complexity of neural nets. They begin the method in the same way; create the lowest level of control for very simple tasks. Once this is fully functional and debugged, proceed to the next level. Now, the divergence: rather than having each layer of the control

system be designed as a separate add–on, COGS simply adds complexity to the original network and allows the new net to re–converge at the higher complexity level[31].

Inman Harvey (one of the leaders of COGS) was able to design algorithms for dealing with the variable length genes needed for this paradigm. The technique, Species Adaptive Genetic Algorithm (SAGA) [28] was designed during his Ph.D. work and has been continually extended and applied since. The basic idea is to allow the size of the gene to change during evolution without losing the gains achieved by earning the *status quo*.

The COGS group has another distinction from MIT; simulation is heavily used in the initial design and improvement of the design. Regular testing in hardware and correlation of the simulations to reality is performed to ensure validity of results. The increased use of simulation allows a wider variety of experimentation to be performed and different issues of ER to be explored with minimum hardware modification[29].

5.4.1 Vision and Cognition

The COGS group is also involved in the investigation and simulation of animal vision and behavior selection. Although this is partially outside ER, effective, efficient vision has long been a stumbling block to autonomous agent design[15]. The evolution and experimentation with vision systems began in simulation, but quickly moved to a hybrid of simulation and testing in physical reality: the *gantry robot*[30]. This device allows the sensors and environment to be in reality while giving positional, and heading control to an external controller. During the evolution of the control system, the gantry can prevent illegal (damaging to the robot or its environment) maneuvers, accurately reposition the probe to the starting point of the next fitness evaluation, etc.

The relative levels of simulation and physical control are variable. At one extreme the sensors may be directly piped into a purely numerical simulation, while at the other end, the robotic probe may be completely self contained with the only connection being the output of the robot's wheels being converted to motion of the gantry[29].

5.4.2 Evolving Net Architectures and Sensor Placement

Karthik Balakrishnan and Vasant Honavar are exploring the simultaneous evolution of ANN architecture and programming. The concept is to not only evolve the parameters of the net, but also evolve the structure (topology) as well[5]. Any criteria may be used for selection, but generally the goal is to find as small a net as possible since this is

easier to optimize, cheaper to build, and less prone to failure. Most of the work in the Iowa State University AI group has been done with discrete systems[7, 6] (in a model similar to Teller's[60]), but extending some of this work to a 'continuous' environment has been completed by Walker[64].

There is a danger of creating a 'meta–algorithm' where each candidate net structure must be independently optimized before its potential can be evaluated. In other words, Each fitness test would have to consist of a complete evolution of the parameters for the candidate design. There are two primary problems with this. 1) Each candidate architecture requires a full optimization of the connection and neuron parameters it uses to fairly evaluate it. This causes orders of magnitude increase in the required number of fitness evaluations needed. 2) There is increased danger of good designs being missed. This is because of low probability stochastic combinations derailing the evolutionary test of an otherwise good net architecture. This problem is solvable by performing repeat tests but this only aggravates the problem of too much fitness testing.

There is great potential in pursuing minimization; the work of Ashlock, Walker, and Oliver[2] confirms that very simple control networks are capable of very complex behavior. There have also been good results in combining the minimal approach with the evolved sensor placement action: the GA often ignored (by making its connections to the control net near zero) redundant sensors which were not needed to model its environment[64].

5.4.3 Pros and Cons

The Sussix Approach most closely follows the A–Life paradigm of all the ER techniques. The COGS group is also building a strong arsenal of general methods and techniques of construction which can be universally applied and reused in other projects. The research here is focused toward very practical solutions to not only the end design of autonomous robots, but controlling cost and difficulty during the design and evolutionary process.

Another advantage is the automatic mixture of sub–tasks within the ANN provides. The interface between sub–components does not require special design consideration; this is handled by the evolution of the ANN. This facilitates automatic integration of additional hardware and allows the GA to make minor adjustments to other systems to optimize for the synthesis of the new hardware with the old.

The greatest pitfall to the Sussix Approach is the lack of intuitive understandability in the resultant ANNs and control structures. The groups' early work is adequately analyzed and the ability to scale the methodology has been shown. The COGS group

also admits that developing the techniques to evolve the ANNs has required significantly more effort than hand design would have. In the words of one member of the group, their focus on developing general methodologies "is a bet on the tortoise"[31].

5.5 Behavior–Oriented Design

BOD is the creation of Luc Steels from the AI Lab at the Vrije Universiteit Brussel. The two guiding principles of this approach are specified as follows: 1) All behavioral responses are run in parallel with weight given to each activity or reaction altered dependent upon the given state. 2) Initially, easy but non-optimal actions seed the process of emergence and are gradually replaced by more complex and efficient behaviors[57].

To illustrate 1), consider the following comparison to subsumption: In the attempt to design a robot which patrols a given space looking for trash dropped on the floor, subsumption might arrive at the following architecture:

- collision avoidance
- backing away from corners or traps
- seeking recharge station (if low on power)
- pick up nearby trash
- search for trash

Since each lower level of the architecture can over–ride the ones above it, the search for trash will only occur *if* there is no nearby trash, there is enough power, it is not stuck in a corner, and there are no imminent collisions.

BOD is fundamentally different in that all the routines are run together and summed. The summation is weighted by the given internal state of the robot. The internal state is determined by the environmental factors. Using the above example, if the robot is low on power, the weight of the 'seek the power station' task will become a progressively higher priority to the robot as reserve power decreases. Due to the summed nature of the control, the gradual drain on power might cause the robot to drift its search toward the area *near* the power station when power is getting low rather than making a discrete switch at some arbitrary power level. Thus, the task would continue as long as possible and would allow the robot to consume more of its energy completing its task before

being force to recharge (since the power station will be conveniently close when that time arrives)[57].

Steels maintains that using a dynamical systems approach is a better way to utilize the complexity of the environment than imposing an external clock or discrete nature to the control system (which both subsumption[11] and Sussix[31] employ). Many creatures have been shown to do this[56, 59] and the removal or distortion of this effect was not only counterproductive, but also an undesired human restriction on what the system could extract from the complexity of the environment.

Steels summarizes the characteristics of the PDL robot architecture[55] using four comparisons[58].

- Cooperation vs. subsumption
- Sub-symbolic vs. symbolic
- On–line vs. off–line
- Open vs. closed functionality

Cooperation describes the summation of all inputs in the control system. This was discussed above. *Sub-symbolic* indicates the use of the BBAI (rather than KBAI) paradigm; the lack of internal symbolic representations of the environment. *On–line* is a description of the training environment of the evolution: everything is done 'live' in hardware.

Experiments with BOD have involved control systems designed with human control and through an evolutionary process. Unlike COGS, the evolutionary training is done completely in hardware with physical robots operating in their environment. This also enables the utilization of subtle environmental effects which would often be missed in preparing a simulation.

Open functionality is the most difficult facet to describe. Most current GA experiments assume fixed fitness landscapes and an external measure for quality evaluation. Inspired by Ray's *Tierra*[51] Steels strives to create environments where natural selection, not artificial selection, determines the course of evolution and the resultant behaviors of the robots.

5.5.1 Pros and Cons

The evolution of control exclusively in physical hardware guarantees functionality of the final design. There is no chance of loss between simulation and reality since this layer is removed[58].

The simultaneous operation of all control forces a harmonious existence between different components of the control system. This facilitates a smooth, efficient transition between different behaviors and a gentle but constant force toward low priority but slightly more optimal behaviors.

Environmental complexity and characteristics are fully utilized by the control system. This avoids harmful external constraints imposed by environmentally isolated systems while allowing minimization of internal complexity of the robot[56].

Evolution in hardware guarantees functionality. Unfortunately, it is very time consuming and prone to problems. Disturbance of the process, hardware failures, or the lack of available staff to "baby–sit" the robots' environment stop the process until the difficulty is corrected. The systems evolved to date are designed for simple tasks in limited environments. Since any evolutionary process slows as the number of variables is increased, the complexity needed for control will soon outgrow a practical amount of training time.

There can also be problems with the blending of multiple goals within a single control architecture. Walker demonstrated the possibility of the interference from multiple task rendering each individual control system ineffective[64]. Thus, radically conflicting tasks may require a discrete priority (subsumption) or an internal method to remove the conflicting interference such as a 'switching neuron' in an ANN implementation. This interference can also impede evolutionary progress of the combined system if it is being evolved holisticly[64].

Finally, the majority of research has focused on investigating environments with Steels' *open functionality*. This is very interesting from a research point of view, but eventually a useful solution is desired for specific tasks. Without practical use, BOD will remain a experiment in the lab rather than a method for building robots for valuable real–world tasks.

5.6 Virtual Agents for Synthetic Environments

One way to answer the complaints of Brooks is to remove the problem. If the only good robots are the ones which actually operate in the "real world" (their final operating environment) then why not make the world inside the computer the final destination? This will obviously not work if we want physical robots, but if the final application is synthetic, then this is the only environment in which it needs to function.

This is fundamentally different from building a simulation to prototype creations for the real world. The sensory apparatus of the agent may now utilize the advantages of noise–free sensors, oracles of information about the environment, or access to sensory or modeling information not available to the human users of the environment.

Another possibility is to explore idealized simulations of the environment to study issues like evolution of vision, memory, communication, or other perception strategies. For purely SE applications, the situation may be radically altered or reversed; perhaps trying to find a minimal, non–visual representation of the world or aspects of it which allow natural behavior without the overhead of simulating natural sensory methodologies.

For example, consider a virtual agent which we do not want to give oracle information. To imitate reality, we could render two views from its eye positions, and then train an ANN to do the image processing of the rendered bitmap. Training the ANN would be a very difficult task, and all the rendering and calculation would place a large computational burden upon the machine supporting the synthetic environment.

Consider instead an agent which uses something akin to *Symbot* vision[2] where objects are simply modeled as energy sources and the 'vision system' of the agent is simply a sensor sensitive to this type (e.g., frequency, color, etc) of signal. The simple representation of the environment created with this method would be invisible to the user, but would be computationally inexpensive and easy to train agents to utilize[64]. Thus, the agents would not use oracles and would be operating, like the user with only their perceptions of the environment. The perceptions would simply be different.

This work has been pursued actively by Terzopoulos[62] with artificial fish and active vision systems[61]. Karl Simms has done extensive work creating simulated animals which learn locomotion in different environments[54] and direct competition for resources[53]. These two authors represent only a small portion of the work being done.

5.7 Summary of Evolutionary Robotics

All three methods have their strengths and weaknesses. Subsumption has a proven track record, but it requires manual implementation. Programming a robot for subsumption is a difficult, highly skilled activity. Brooks has formed a company (IS Robotics) to apply his methodology to commercial autonomous robots, but the success or failure of this venture is yet to be determined.

The Sussix Approach is more established in simulation than in hardware. The Klephera[46] group is, however, attempting to commercially market robots; but these are primarily for research and experimentation rather than practical or industrial usage.

Subsumption and BOD are robust and understandable, but suffer from complexity problems when scaled to large problems. They are also both dependent upon human intuition and understanding in the design process.

The Sussix Approach more closely follows nature, but the resultant evolved systems are often not understandable to humans. There is also a great dependency on the accuracy of simulation used for much of the training; more so than with subsumption or BOD since the designers cannot intuitively check the validity of controls.

Brooks has been said to have relaxed much of his earlier criticism of simulation in ER. Great advances have been made in physically based modeling and simulation during the late 80's when his demands of 'hardware or nothing' were made. Obviously the requirement to implement in hardware before really trusting the results is obvious and maintained. (e.g., Would anyone fly in an aircraft which had only been tested in simulation?)

Figure 4 visually illustrates the shared nature of three differences of implementation between the major approaches of ER. Subsumption and COGS favor incremental increases in complexity during the process of design and programming of the robot; BOD robots contain full complexity as they are thrust into the environment to find their own natural selection of fitness. BOD and COGS use evolution to program their control systems; subsumption is manually designed by humans. Subsumption and BOD claim the only hardware results and training are valid; COGS makes extensive use of simulation and partial hardware testing.

All three of these paradigms of ER share characteristics of the others. None of the three can be completely separated from the others with unique methodology or principles. This is less surprising if the youth of the science is remembered. These methods were created at different locations, but the leaders of the different groups interacted (and

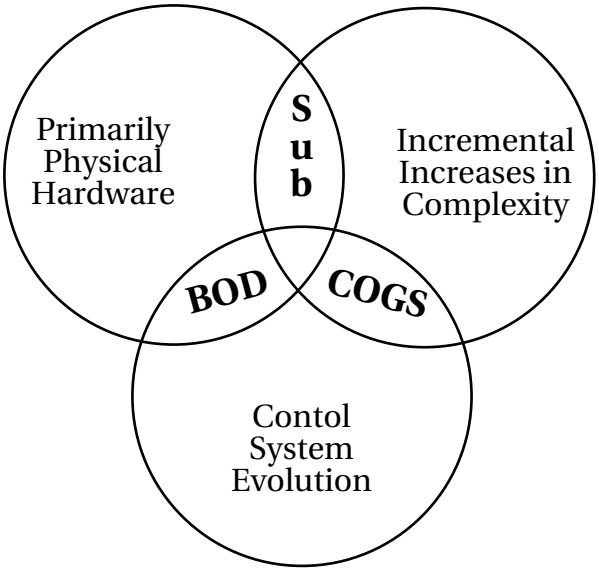

Figure 4: ER: sharing and contrast

continue to meet) at conferences, through the Internet, and with the exchange of papers. All three methods are still evolving (along with their robots) so the drift, fusion, and and convolution is certain to continue.

6 Looking to the Future

It can be said that any significant advance in technology is akin to releasing a long sleeping genie from its magical prison. It may be difficult to release, but once the cork is pulled the genie may be impossible to put away. Technology is like the genie, resistant to discovery but stubbornly persistent to remain in use once developed.

Some of the greatest thinkers in A–Life, when considering the lessons of the atomic bomb, the warnings of Dyson [19], or the predictions of Levy[42], speak of caution. Computer viruses are annoying, but they have no power to affect the physical world outside their computer (once it is switched off or isolated). One can only imagine the detrimental effects of self–producing robots gone awry or of self replicating factories with their own agenda.

Dyson suggested that A-Life may be the beginnings of the next phase of our evolution; that the shells of flesh we wear are as doomed as the dinosaurs compared to technological

life which and survive easily in vacuum or radiation and can adapt its genetic code on the fly. He also suggests that these "children of humanity" may be our guides and helpers in exploring the cosmos. Only the future knows which statement is the truth, but the genie of A–Life is already out of the bottle...

Acknowledgments

The authors gratefully acknowledges the financial support provided by the National Science Foundation Young Investigator Award, (Grant No. DDM-9258114) and the Iowa Center for Emerging Manufacturing Technology (ICEMT).

Thanks are given to Jim Lathrop for numerous discussions on the subject material, reference location, and preliminary text reviews. We would also like to thank Dan Ashlock for content input and Jennifer Donnelly for editorial reviews and providing a biologist's perspective.

References

[1] D. Ashlock. Optimization and Modeling with Artificial Life. Text Book in Progress, 1995.

[2] D. Ashlock, J. Walker, and J. Oliver. The evolution of ultrasimple virtual robots. In N. G. Bourbakis, editor, *Proceedings of 2nd IEEE Intl. Joint Symp. on Intelligence and Systems*, Proceedings of 2nd IEEE IAR, pages 170–177. IEEE, 1996.

[3] D. A. Ashlock. Personal communication, 1994.

[4] D. A. Ashlock and J. I. Lathrop. An algorithmic test suite for genetic programming. In *Proceedings of the Fourth International Symposium on Mathematics and Artificial Intelligence*, 1996.

[5] K. Balakrishnan and V. Honavar. Properties of Genetic Representations of Neural Architectures. In *Proceedings of the World Congress on Neural Networks*, 1995.

[6] K. Balakrishnan and V. Honavar. Analysis of Neurocontrollers Designed by Simulated Evolution. In *Proceedings of the IEEE International Conference on Neural Networks*, 1996.

[7] K. Balakrishnan and V. Honavar. On Sensor Evolution in Robotics. In *Proc. Genetic Programming 1996*, 1996.

[8] E. Berelecamp, J. Conway, and R. Guy. *Winning Ways for your Mathematical Plays*. Academic Press, New York, NY, 1982.

[9] R. M. Brady. Optimization strategies gleaned from biological evolution. *Nature*, 317:804–806, 1985.

[10] R. Brooks. Achieving artificial intelligence through building robots. AI Memo 899, 1986.

[11] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), 1986.

[12] R. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. AI Memo 1091, 1989.

[13] A. Burks, editor. *Theory of self-reproducing automata.* Univ. of Illinois Press, Urbana, Ill., 1966.

[14] D. Chapman. Planning for conjunctive goals. *Journal of Artificial Intelligence*, 32(3), 1987.

[15] D. Cliff, P. Husbandsand, and I. Harvey. Evolving visually guided robots. In J.-A. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats 2, Proc. of 2nd Intl. Conf. on Simulation of Adaptive Behavior*, SAB93, pages 374–383, Boston MA, 1993. MIT Press.

[16] E. Codd. *Cellular Automata.* Academic Press, New York, NY, 1968.

[17] C. Darwin. *The Origin of Species.* John Murray, 1859.

[18] L. Davis and D. Smith. Adaptive design for layout synthesis. Technical report, Texas Instruments, 1985.

[19] F. Dyson. *Disturbing the Universe.* Harper and Row, New York, NY, 1979.

[20] F. A. L. (ed.). *Numerical Nethods for Non–Linear Optimization.* Academic Press, 1972.

[21] A. Fiacco and G. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* Wiley, 1968.

[22] A. Flynn and R. Brooks. Battling reality. AI Memo 1148, 1989.

[23] M. Fourman. Compaction of symbolic layout using genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 141–153, 1985.

[24] D. E. Goldberg. *Computer–aided gas pipeline operation using genetic algorithms and rule learning.* PhD thesis, University of Michigan, Ann Arbor, MI, 1983.

[25] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[26] D. E. Goldberg and C. H. Kuo. Genetic algorithms in pipeline optimization. *Journal of Computers in Civil Engineering*, 1(2):128–141, 1987.

[27] D. E. Goldberg and M. P. Samtani. Engineering optimization via genetic algorithm. In *Proceedings of the Ninth Conference on Electronic Computation*, pages 471–482, 1986.

[28] I. Harvey. *The Artificial Evolution of Adaptive Behavior.* PhD thesis, University of Sussix, Sussix, UK, 1995.

[29] I. Harvey, P. Husbands, and D. Cliff. Issues in evolutionary robotics. In J.-A. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats 2, Proc. of 2nd Intl. Conf. on Simulation of Adaptive Behavior*, SAB93, pages 364–373, Boston MA, 1993. MIT Press.

[30] I. Harvey, P. Husbands, and D. Cliff. : Seeing the light: Artificial evolution, real vision. In J.-A. M. D. Cliff, P. Husbands and S. Wilson, editors, *From Animals to Animats 3, Proc. of 3rd Intl. Conf. on Simulation of Adaptive Behavior*, SAB94, pages 392–401, Boston MA, 1994. MIT Press.

[31] I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi. Evolutionary robotics: the sussix approach. *Robotics and Autonimous Systems*, 1996.

[32] J. H. Holland. Genetic algorithms and the optimal allocations of trials. *SIAM Journal of Computing*, 2:88–105, 1973.

[33] J. H. Holland. *Adaption in Natural and Artificial Systems.* The MIT Press, Cambridge, MA, 1975.

[34] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[35] M. Koppel. Structure. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 435–452. Oxford University Press, Oxford, 1988.

[36] J. R. Koza. Evolution and co–evolution of computer programs to control independently–acting agents. In J. Meyer and S. Wilson, editors, *Animals to Animats: Proceedings of the First European Connference on the Simulation of Adaptive Behavior*, Cambridge, MA, 1991. MIT Press.

[37] C. G. Langton. Self reproduction in cellular automata. In F. D., T. T., and W. S., editors, *Cellular Automata: Proceedings of an Interdiciplinary Workshop*, pages 135–144. North Holland Physics Publishing, 1983.

[38] C. G. Langton. *Artificial Life*, volume 5 of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Reading, MA, 1990.

[39] C. G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42:12–37, 1990.

[40] J. I. Lathrop. Personal communication, 1996.

[41] J. I. Lathrop. Compression depth and the behavior of cellular automata. *Complex Systems*, 1997. To appear.

[42] S. Levy. *Artificial Life: the Quest for a New Creation*. Pantheon Books, New York, NY, 1992.

[43] A. Lindenmayer. Mathematical models for cellular interaction in development. *Journal of Theoretical Biology*, 18:280–315, 1968.

[44] P. Maes. A bottom–up mechanism for behavior selection in an artificial creature. In J. Meyer and S. Wilson, editors, *Animals to Animats: Proceedings of the First European Connference on the Simulation of Adaptive Behavior*, Cambridge, MA, 1991. MIT Press.

[45] P. Maes. Behavior–based artificial intelligence. In D. Cliff, P. Husbands, J.-A. Meyer, and S. M. Wilson, editors, *Animals to Animats III: Proceedings of the Third European Connference on the Simulation of Adaptive Behavior*, Cambridge, MA, 1993. MIT Press.

[46] O. Miglino, H. H. Lung, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434, 1995.

[47] A. Minga. Genetic algorithms in aerospace design. In *Proceedings of the AAAI Southeastern Regional Student Conference*, 1986.

[48] E. F. Moore. Artificial living plants. *Scientific American*, 195(4):118–126, 1956.

[49] B. Pierson. Aeroe–635: Parameter optimization. unpublished lecture notes, Iowa State University; Ames, IA, 1995.

[50] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Mech. Visual models of plant development. In G. Rosenberg and A. Salomaa, editors, *Handbook of Formal Languages*, Berlin, 1996. Springer–Verlag.

[51] T. S. Ray. An approach to the synthesis of life. In C. G. Langton, editor, *Artificial Life II*, volume 10 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 313–324, Reading, 1991. Addison-Wesley.

[52] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 4(21):25–34, 1987.

[53] K. Simms. Evolving 3D morphology and behavior by competition. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 28–39, Cambridge, MA, 1994. MIT Press.

[54] K. Simms. Evolving virtual creatures. In *Computer Graphics*, Proceedings, Annual Conference Series, pages 15–22. ACM SIGGRAPH, 1994.

[55] L. Steels. Building agents with autonimous behavior systems. In L. Steels and R. Brooks, editors, *The 'artificial life' route to 'artificial intelligence'*, New Haven, 1993. Lawrence Earbaum Associates.

[56] L. Steels. The artificial life roots of artificial intelligence. *Artificial Life*, 1(1), 1994.

[57] L. Steels. A case study in the behavior–oriented design of autonomous agents. In J.-A. M. D. Cliff, P. Husbands and S. Wilson, editors, *From Animals to Animats 3, Proc. of 3rd Intl. Conf. on Simulation of Adaptive Behavior*, SAB94, Boston MA, 1994. MIT Press.

[58] L. Steels. Emergent functionality in robotic agents through on–line evolution. In C. G. Langton, editor, *Artificial Life IV*, volume 11 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 8–14, Reading, 1995. Addison-Wesley.

[59] L. Steels. Intelligence – dynamics and representations. In L. Steels, editor, *The Biology and Technology of Intelligent Autonomous Agents.*, Berlin, 1995. Springer–Verlag.

[60] A. Teller. The evolution of mental models. In K. Kinnear, editor, *Advances in Genetic Programming*, pages 199–220, Cambridge, MA, 1994. MIT Press.

[61] D. Terzopoulos and T. Rabie. Animat vision: Active vision in artificial animals. In *Proceedings of the Fifth Intl. Conf. on Computer Vision (ICCV '95)*, pages 801–808, Cambridge, MA, 1995.

[62] D. Terzopoulos, X. Tu, and R. Grzeszczuk. Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Artificial Life*, 1(4):327–351, 1994.

[63] J. von Neumann. The General and Logical Theory of Automata. *Collected Works*, 5:288–328, 1961.

[64] J. F. Walker. Evolution of simple virtual robots using genetic algorithms. Master's thesis, Iowa State University, Ames, IA, 1995.

[65] D. J. Wilde. *Optimum Seeking Methods*. Prentice-Hall, Englewood Cliffs, NJ, 1964.

[66] D. J. Wilde and C. S. Beightler. *Foundations of Optimization*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[67] S. Wolfram. Preface. In F. D., T. T., and W. S., editors, *Cellular Automata: Proceedings of an Interdiciplinary Workshop*, pages vii–xii. North Holland Physics Publishing, 1983.

[68] S. Wolfram. Universality and complexity in cellular automata. In F. D., T. T., and W. S., editors, *Cellular Automata: Proceedings of an Interdiciplinary Workshop*, pages 1–35. North Holland Physics Publishing, 1983.

[69] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[70] W. Zangwill. *Computer Journal*, 10:293–296, 1968.