Genetic Reasoning
Evolving Proofs with Genetic Search

Peter Nordin and Wolfgang Banzhaf
Universität Dortmund
Fachbereich Informatik
Lehrstuhl für Systemanalyse
D–44221 Dortmund
Germany
nordin,banzhaf@ls11.informatik.uni-dortmund.de

Abstract

Most automated reasoning systems rely on human knowledge or heuristics to guide the reasoning or search for proofs. We have evaluated the use of a powerful general search algorithm to search in the space of mathematical proofs. In our approach, automated reasoning is seen as an instance of automated programming. The proof is a program executing functions that correspond to rules of inference. It transforms a statement into an axiom. Genetic programming is the technique we use for automated programming. We show how such a system can be used to evolve mathematical proofs. The domain of application is arithmetics and register machine program verification.

Keywords: Genetic Programming, Automated Reasoning, Theorem Proving

1 Introduction

We present an approach to reasoning that uses a genetic search heuristics to navigate and search in the space of true statements. An algorithm inspired by natural selection is used to search for proofs.

To use a genetic process as the architecture for mentally related activities could, at first, be considered strange. As far as we know today, genetic information processing is not directly involved in information processing in brains, though the idea of genetics as a model of mental processes is not new. William James, the father of American psychology, argued just 15 years after Darwin published *The Origin of Species*, in 1874, that mental processes could operate in a Darwinian manner (William James 1890). He suggested that ideas "compete" with one another in the brain, leaving only the best or fittest. Just as Darwinian evolution shaped a better brain in a couple of million years, a similar Darwinian process operating within the brain might shape intelligent solutions to problems on the time scale of thought and action. This allows "our thoughts to die instead of ourselves".

Genetic Programming (GP) (Koza 1992) uses the mechanisms behind natural selection for *evolution of computer programs.* This application contrasts with other evolutionary algorithms which often optimize real numbers or vectors of real numbers. GP differs from other evolutionary techniques and other "soft-computing" techniques in that it produces symbolic information (i.e. computer programs) as output. It can also process symbolic information as input very efficiently. Despite this unique strength GP has so far been applied mostly in the numerical and Boolean problem domains. In this paper we exploit GP's strength in processing purely symbolic information through search in the domain of proofs.

Genetic Programming is a method for automated programming. A formal proof of a statement could be seen as a computer program and a theorem prover could be considered as an application of automated programming. The proof program is a list of inference functions transforming a statement into an axiom or into a statement known to be false, i.e. a contradiction. Rules of inference are considered here as functions from theorems to theorems or from statements to statements. The inference rules are rules that match part of a formula and rewrite it as something equivalent or equally true. To give an example, the formula $X + 0$ could be replaced by X, as one of the axioms of Peano arithmetic tells us. This rule describes a function from one statement to another. In the same way, the reverse is true and X could be replaced by $X + 0$ through another function.)

The simplest form of a theorem prover systematically applies rules of inference in order to construct all possible valid logical deductions. Pioneering AI research tried this approach already in the 1950s, notably with the Logic Theory Machine of Alan Newell and Herbert Simon (Newell, Shaw and Simon 1957). In practice, however, such a method could only find very short proofs. Combinatorial explosion would quickly exhaust all computer resources.

Different and more efficient variants of representation and search have been introduced later, like the *resolution* method pioneered by Robinson in the early 1960s (Robinson 1965), see also (Bundy 1983). These methods were adapted more to machine reasoning than to human reasoning, hence their implementions were more efficient. Still they needed to be governed by strategies and heuristics that optimized the order in which clauses were resolved. Resolution theorem provers help against the combinatorial explosion but they cannot eliminate it altogether. As a consequence, they are only able to produce proofs of modest length.

The disappointment with some of these reasoning systems lead to the conclusion that more human knowledge needs to be put into the reasoning process, or, as Bledsoe has put it (Bledsoe 1977):

> The word "knowledge" is a key to much of this modern theorem-proving. Somehow we want to use the knowledge accumulated by humans over the last few thousand years, to help direct the search for proofs.

This knowledge is included as heuristics, weights and priorities in a theorem proving system. If the system is an *interactive theorem prover* it can have its heuristics modified by a human during execution. Regarding search algorithm, most systems rely on a hill-climbing algorithm, on back-tracking or a best-first heuristics (Winker and Wos 1978).

In our research we investigate a different approach. Instead of using heuristics that is explicitly added to guide the search process, we apply a powerful and robust general search algorithm, "Genetic Reasoning". We hypothesize that the robustness of a genetic search process could free the reasoning system from the burden of carrying specialized heuristics. The search could then be more autonomous and act more "intelligently" by producing solutions which require less a-priori knowledge.

2 Genetic Reasoning

In order to apply GP to reasoning and automated theorem proving (ATP) we need to choose an appropriate fitness function, a function set and a theorem representation. In this paper, our goal is to handle statements about arithmetics in a logic as powerful as first order logic.

The function set is made up of functions representing rules of inference. One such function could be the rule *"$X + 0$ can be replaced by X"*. All functions we use are unary functions — they take one statement as input and produce an equivalent statement as output. This means that the tree representation of individuals in GP collapses into a linear list representation[1] and that recombination will exchange linear segments of genomes as seen in Figure 1.

The actual statement to be proven "true" or "false" is represented by a tree. Universal quantification is indicated by leaving variables free. Existential quantification is represented by a Skolem function, as is common in several approaches to ATP. The natural numbers are built into our system in the form of the *zero (0)* symbol and the successor function. Figure 2 shows, how a (false) statement such as $3 = 2 + 0$ would be represented.

The inference functions in the genome are then applied in turn to this structure. After all inference functions have been applied, i.e. after the individual program has terminated, the result is another tree structure representing an equivalent statement. Let us say that we call the rule "$X + 0$ can be replaced by X" $func_1$. If this function is part of the genome it will try to match a sub-tree in the statement and, provided it finds a match, will replace it with X as shown in Figure 3.

In Figure 3 the function matches a sub-tree in the statement and the statement can be transformed. With this transformation the size of the statement structure is reduced. There exists, however, an equal number of functions in the function set that increase the size of the structure. If a function does not matches any sub-tree in the statement then it is not executed and the structure is left untouched. This procedure provides syntactic closure and also gives the opportunity of temporary storage of unused material in the genome. We call this unused genetic material *introns*. As we have discussed elsewhere, it may play an important role in the efficiency of genetic search (Nordin and Banzhaf 1995a), (Nordin, Francone and Banzhaf 1995).

2.1 The Fitness Function

The fitness function in our genetic reasoning system is very simple. It is just the number of nodes in the statement structure (cf. Figure 2). The two simplest and shortest statements are the Boolean constants **t** and **f**, each represented by only one node. These truth values are short hand for an axiom or a contradiction, respectively. So the genetic system will try to simplify any expression down to a statement of either "true" or "false", represented by the nodes **t** and **f** that carry highest fit-

[1]Note that in this application the genome structure is linear while the fitness case input is a tree structure. It is sometimes the other way around in other GP applications.

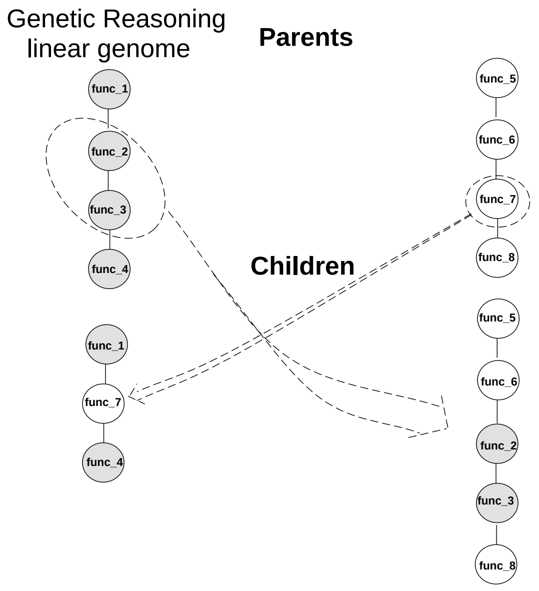

Figure 1: Crossover in Genetic Reasoning.

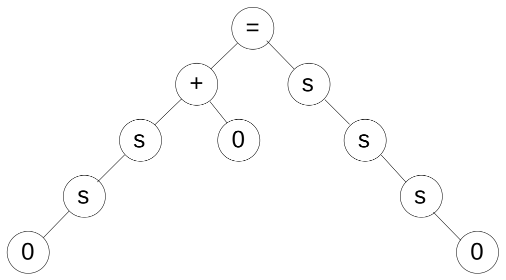

Figure 2: Representation of the statement $3 = 2 + 0$.

ness values.

Genetic search has been proven to perform well and robustly in a wide variety of highly multi-modal search domains where local optima can trap local hill-climbing approaches easily. So, the pressure towards simplifying a statement does *not* mean that the system will try to constantly shorten the structure. Instead, the population of solution candidates helps to avoid local optima. Furthermore, the selection criterion does not monotonically select the best individuals from generation to generation. Rather, it probabilistically reproduces individuals with a large variation in fitness.

The logic of the genetic reasoning system is similar to that of the automated reasoning system Nqthm (Boyer and More 1979). It is a quantifier-free, first-order logic with equality. The rules of inference are taken from propositional logic and the equality. Mathematical induction is an important part of the system.

Functions defining all Boolean and arithmetic operations ($\wedge$,$\vee$,$\neg$,$\longrightarrow$) are built-in. The Boolean constants **t** and **f** representing an axiom or a contradiction, respectively, are also built-in. There are if-then-else functions as well as equality. Natural numbers and arithmetics are defined by the Peano axioms and the symmetry relation. It is possible to add functions defining abstract data types and lemmas to support a specific application. In the register machine example below axioms describing this fictional processor are added.

2.2 The Evolutionary Algorithm and its Implementation

In principle, it is possible to use any variant of GP as the basis for Genetic Reasoning. We have used a steady-state variant with tournament selection. The size of the population has been between 100 – 1000 individuals.

The GP system is implemented on a SUN-20 in PRO-

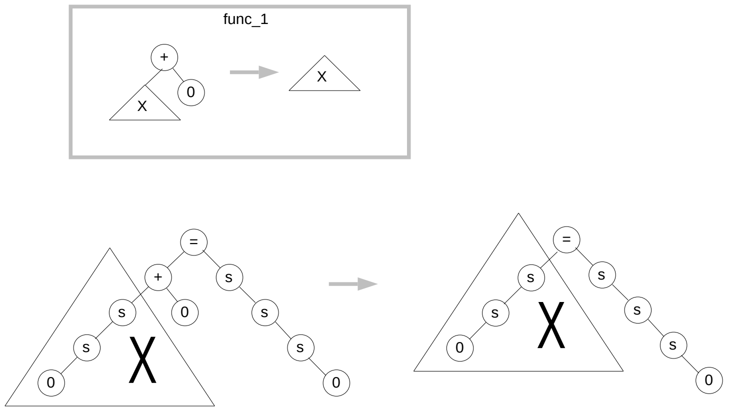

Figure 3: Application of $func_1$ to the statement $3 = 2 + 0$.

LOG. The built-in features of PROLOG, such as pattern matching and list handling, simplify the implementation significantly.

3 Results

Our reasoning system has so far been applied to two different domains: proving simple statements in arithmetics and reasoning about, for instance, halting of machine code programs. Both these applications rely to a great extent on mathematical induction as the proof method. The examples exclude that the proof can be obtained by simplification only. The system could not just hill-climb towards a solution. Instead, various steps of expansion are needed on the way, to be able to finally reach a constant false or true statement. These expansions are not defined by a lemma or heuristics but have to result from the genetic search process.

3.1 Arithmetic Problems

The arithmetic problems that we started our evaluation with were selected using two criteria. The statement should be hard to prove without induction and it should be impossible to prove by just transforming it to shorter statements. The induction principle might in itself require proofs that cannot be obtained by monotonic transformations and reductions. A typical statement used is: *"There is no natural number bigger than 3, that if added 2 to it, is equal to 4."* This statement is represented by the tree structure in Figure 4.

This kind of a statement can be proven (true) with a few hundred generation equivalents and a population size of 200 individuals. This calculation takes about 10 minutes on our SPARC-20.

3.2 Termination Proofs of a Program for a Register Machine

A register machine is a machine which operates with instructions that manipulate a limited set of registers. All CPUs in commercial computers are register machines. The axioms defining the processor and the current program of the processor are added to the system. We use the genetic reasoning system to determine the correctness of machine code programs, which often means to prove termination of the program. This approach demonstrates one of the strengths of the genetic reasoning system because termination proofs almost exclusively require induction to be part of the system.

The machine code application is slightly more complex than pure arithmetic statements and the verification of a short program of 2 to10 instructions takes about one hour on a SPARC-20 with a population size of 1000 individuals.

Correctness proofs for programs have many applications, for instance data security, high robustness in programs (e.g., satellite technology), or simplification of machine code programs (Boyer and Yu 1992), to name a few. However, GP is often accused of producing non-robust solutions. A reasoning system could judge evolved solutions to prove whether they are complete or not. The procedure resembles that of a human programmer who first might put together a program solution almost by intuition. Then he would study the solution and reason whether it will really hold for all inputs. If this is ensured he would check whether the solution could be simplified.

Termination proofs could also be used with a normal GP system in order to detect infinite loops in individual programs during evolution. Normally, a few hundred generation equivalents have been needed to reach the true and false constants in our program verification ex-

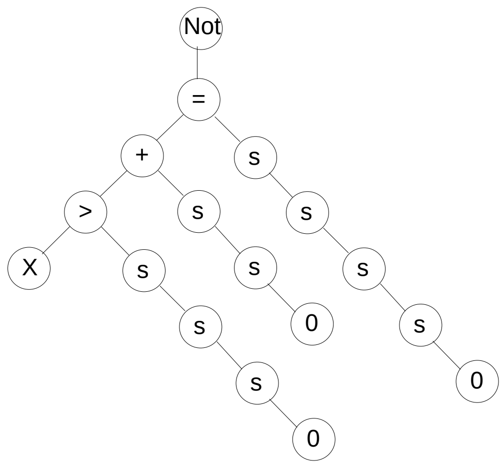

Figure 4: An example statement representation.

periments.

4 Future Work

In our experiments we have concentrated so far on the problem of proving a theorem. We are, however, convinced that the genetic reasoning method has applications in other areas of reasoning and machine learning, such as planning in robotics. We would like to continue and to extend our robot experiments on the Khepera robot platform (Nordin and Banzhaf 1995b) with the application of genetic reasoning.

We also plan to port the system to C which will result in an acceleration by a factor of 100. This would allow us to try more difficult problems using larger population sizes.

5 Summary and Conclusions

We have demonstrated that automated reasoning could be seen as an instance of automated programming. In this spirit we a have evaluated the use of a robust genetic search algorithm to search the space of proofs. The system has been able to avoid local minima in its search and has found proofs of statements from complex domains such as arithmetics and program verification. The system does not use heuristics or human knowledge to guide its search. Instead, it relies on the performance of the search algorithm. We believe that this technique will have applications in many automated reasoning and machine learning domains.

Acknowledgement

Thanks to Steffo Weber for invaluable advice in the process of writing this paper. We would also like to thank Tom Haynes for giving us pointers to his experiments with GP and theorem proving (Knight and Haynes 1994). One of us (P.N.) acknowledges support by a grant from the Deutsche Forschungsgemeinschaft (DFG) under contract Ba 1042/5-1.

References

[1] James, W. (1890) *The principles of psychology Vol.1.* Originally published: Henry Holt, New York.

[2] Koza, J. (1992) *Genetic Programming.* MIT Press, Cambridge, MA

[3] Newell, A., Shaw, J.C. and Simon, H. (1957) Empirical Explorations of the Logic Theorem Machine: A case study in Heuristic. In *Proceedings of Western Joint Computer Conference* Vol. 15.

[4] Robinson, J.A. (1965) A Machine Oriented Logic Based on the Resolution Principle. In *J. ACM*, Vol. 12, No. 1, pp. 23-41.

[5] Bundy, A. (1983) The Computer Modeling of Mathematical Reasoning, Academic Press, London

[6] Bledsoe, W. W. (1977) Non-Resolution Theorem Proving. In *Artificial Intelligence*, Vol. 9, pp 2-3.

[7] Winker, S. and Wos, L. (1978) Automated Generation of Models and Counterexamples and its application to Open Questions in Ternary Boolean Algebra. In Proceedings of 8th international symposium Multiple-Valued Logic, Rosemont, Ill., IEEE and ACM, New York, pp. 251-256.

[8] Nordin, J.P. and Banzhaf, W. (1995a) Complexity Compression and Evolution. In *Proceedings of Sixth International Conference of Genetic Algorithms, Pittsburgh, 1995.* L. Eshelman (ed.). Morgan Kaufmann, San Mateo, CA

[9] Nordin, J.P., Francone, F. and Banzhaf, W. (1995) Explicitly Defined Introns in Genetic Programming. In *Advances in Genetic Programming II.* P. Angeline, K. Kinnear (Eds.). MIT Press, Cambridge, MA

[10] Nordin, J.P. and Banzhaf, W. (1995b) Controlling an Autonomous Robot with Genetic Programming. In *1996 AAAI fall symposium on Genetic Programming.* MIT, Cambridge, MA

[11] Boyer, R.S. and Yu, Y. (1992) Automated Correctness Proofs of Machine Code Programs for a Commercial Microprocessor. In *Automated Deduction - CADE-11.* Kapur D. (Ed), pp. 416-430.

[12] Knight, L. and Haynes, T. (1994) *A GP Theorem Prover.* Technical Report CS 7213, University of Tulsa.