Genetic Operators and Inductive Logic Programming: Fisher's Theorem of Natural Selection

Paper accepted as poster at GECCO 2003

Lutz Hamel

Department of Computer Science and Statistics
University of Rhode Island, Kingston, Rhode Island 02881, USA
hamel@cs.uri.edu

Abstract. We are interested in inducing equational theories from facts. Our current, limited prototype that implements inductive equational logic programming using evolutionary techniques shows good convergence behavior for small problems but relatively poor convergence for larger problems. Therefore we are concerned with the design of our genetic operators. We chose to study the evolutionary population dynamics of our system through a set of experiments and compare it to the theoretical behavior predicted by Fisher's Fundamental Theorem of Natural Selection in order to determine the quality of our genetic operators. The evolutionary population dynamics of our system behaves as predicted by Fisher's Theorem and therefore we conclude that the design of our genetic operators is appropriate and that the poor convergence behavior in larger problems is due to limitations of the prototype.

1 Introduction

We are interested in inducing equational theories from examples or facts. This can be considered a special case of the general notion of concept learning where the aim is to induce a description of a concept from a set of examples. Typically the set of examples are ground sentences in a particular representation language. In our case the representation language is equational logic. Concept learning can be seen as a search over all possible sentences in the representation language for sentences that correctly explain the examples and also generalize to other sentences that are part of that concept [9, 14]. We refer to the induction of equational theories from facts as inductive equational logic programming [6].

In recent years specialized search heuristics in both the inductive first-order and equational logic setting have been proposed. Consider Muggleton's Progol system whose underlying search paradigm is based on inverting logical entailment [16]. In the equational setting, inverse narrowing has been proposed as the main search strategy in the FLIP system [7]. Since concept learning and inductive logic programming imply complex searches, it is natural to ask whether evolutionary

algorithms are applicable in this area. To date evolutionary algorithms, particularly genetic programming systems, have successfully been applied to concept learning and inductive logic programming tasks in a variety of formalisms. For example, they have been successfully applied in the propositional case [11], in the first-order logic setting [19, 8, 2], as well as in the higher-order functional logic programming setting [9].

In this paper we continue our study of an evolutionary approach to concept learning based on equational logic first described in [6]. In this paper we study of an evolutionary approach to concept learning based on equational logic. Equational logic is the logic of substituting equals with equals. The examples or facts are ground equations and the induced concept descriptions are first-order equational theories. We have implemented a prototype by incorporating a specialized genetic programming engine into the equational logic programming system and algebraic specification language OBJ3 [5]. Informally, the system operates by maintaining a population of candidate theories that are evaluated against the facts using OBJ3's deductive machinery. The fittest theories are allowed to reproduce in accordance to standard genetic programming practices.

One of our key concerns is the quality of our genetic operators: cross-over, mutation, and fitness. Typically, the design of these operators spans the entire spectrum of possibilities. The design ranges from operators that utilize highly specialized heuristics to perform cross-over and mutation [9] to operators that apply cross-over and mutation in a straight forward random fashion under the assumption that the term structure is closed [12]. We took the latter approach to our own operator design with the only exception being that our term structure is typed and the genetic operators have to respect this typing [15]. What makes the genetic operator design even more crucial is the fact that the fitness landscape for inductive logic problems tends to be rugged with steep steps in it. This is due to the fact that logic statements are either true or false, there is no gradual error approximation as in other genetic programming tasks. Given our straight forward design of our operators we are concerned with the convergence behavior of our system. Indeed, as mentioned in [6] we observe that our system exhibits good convergence Indeed, we observe that our system exhibits good convergence for smaller problems but poor convergence behavior for larger problems. Given that our prototype is limited to a maximum population of 200 individuals, we want to understand if the convergence in smaller problems is a consequence of evolutionary search rather than incidental and conversely we want to understand if the poor convergence behavior in larger problems is due to the limited size of the population and therefore the lack of appropriate genetic variety rather than due to a fundamental limitation of our genetic operators.

Fisher's fundamental theorem of natural selection states that the increase of the average fitness of a population is proportional to the variance in the genetic fitness [3]. This characterization of the evolutionary population dynamics in natural systems gives us a way to check our artificial evolutionary system. We postulate that if our system exhibits an evolutionary population dynamics as predicted by Fisher's Theorem then our genetic operators are appropriate.

In this paper we describe a set of experiments that show that our system does indeed exhibit an evolutionary population dynamics as characterized by Fisher's Theorem. It follows that our genetic operators are appropriate and we conclude that the convergence behavior in small problems is not incidental but a consequence of the genetic operators. We also conclude that the poor convergence behavior in larger problems is due to the fact that our prototype only supports limited population sizes that simply do not provide enough genetic variety for the larger search spaces at hand.

The rest of this paper is organized as follows. Section 2 provides a brief introduction to many-sorted equational logic. In Section 3 we develop an algebraic semantics for inductive equational logic programming and sketch the actual implementation of the system. Section 5 and Section 6 describe the experiments. We end with the conclusions in Section 7.

2 Equational Logic

Equational logic is the logic of substituting equals for equals with algebras as models and term rewriting as the operational semantics [13, 18]. The following formalizes these notions.

An equational signature defines a set of sort symbols and a set of operator or function symbols.

Definition 1. *An* **equational signature** *is a pair (S, Σ), where S is a set of sorts and Σ is an $(S^* \times S)$-sorted set of operation names. The operator $\sigma \in \Sigma_{w,s}$ is said to have arity $w \in S^*$ and sort $s \in S$.* [1] *Usually we abbreviate (S, Σ) to Σ.*

We define Σ-algebras as models for these signatures as follows:

Definition 2. *Given a many sorted signature Σ, a* **Σ-algebra** *A consists of the following:*

- *an S-sorted set, usually denoted A, called the* **carrier** *of the algebra,*
- *a* **constant** *$A_\sigma \in A_s$ for each $s \in S$ and $\sigma \in \Sigma_{[],s}$,*
- *an* **operation** *$A_\sigma \colon A_w \to A_s$, for each non-empty list $w = s1 \ldots sn \in S^*$, and each $s \in S$ and $\sigma \in \Sigma_{w,s}$, where $A_w = A_{s1} \times \ldots \times A_{sn}$.*

Mappings between signatures map sorts to sorts and operator symbols to operator symbols.

Definition 3. *An* **equational signature morphism** *is a pair of mappings $\phi = (f, g) \colon (S, \Sigma) \to (S', \Sigma')$, we write $\phi \colon \Sigma \to \Sigma'$.*

[1] Notation: Let S be a set, then S^* denotes the set of all finite lists of elements from S, including the empty list denoted by []. Given an operation f from S into a set B, $f \colon S \to B$, the operation f^* denotes the extension of f from a single input value to a list of input values, $f^* \colon S^* \to B$, and is defined as follows: $f^*(sw) = f(s)f^*(w)$ and $f^*([]) = []$, where $s \in S$ and $w \in S^*$.

A theory is an equational signature with a collection of equations.

Definition 4. *A Σ-**theory** is a pair (Σ, E) where Σ is an equational signature and E is a set of Σ-**equations**. Each equation in E has the form*

$$(\forall X) l = r,$$

where X is a set of variables distinct from the equational signature Σ and $l, r \in T_\Sigma(X)$ are terms over the set Σ and X. If $X = \emptyset$, that is, l and r contain no variables, then we say the equation is **ground**. *When there is no confusion Σ-theories are referred to as theories and are denoted by their collection of equations, in this case E.*

The above can easily be extended to conditional equations[2]. However, without loss of generality we continue the discussion here based on unconditional equations only. Also, our current prototype solely considers the evolution of theories with unconditional equations.

The models of a theory are the Σ-algebras that satisfy the equations. Intuitively, an algebra satisfies an equation if and only if the left and right sides of the equation are equal under all assignments of the variables. More formally:

Definition 5. *A Σ-algebra A* **satisfies** *a Σ-equation $(\forall X) l = r$ iff $\overline{\theta}(l) = \overline{\theta}(r)$ for all assignments $\overline{\theta} \colon T_\Sigma(X) \to A$. We write $A \models e$ to indicate that A satisfies the equation e.*

We define satisfaction for theories as follows:

Definition 6. *Given a theory $T = (\Sigma, E)$, a Σ-algebra A is a T-model if A satisfies each equation $e \in E$. We write $A \models T$ or $A \models E$.*

In general there are many algebras that satisfy a particular theory. We also say that the class of algebras that satisfy a particular equational theory represent the denotational semantics of that theory.

Semantic entailment of an equation from a theory is defined as follows.

Definition 7. *An equation e is* **semantically entailed** *by a theory (Σ, E), write $E \models e$, iff $A \models E$ implies $A \models e$ for all Σ-algebras A.*

Mappings between theories are defined as theory morphisms.

Definition 8. *Given two theories $T = (\Sigma, E)$ and $T' = (\Sigma', E')$, then a* **theory morphism** *$\phi \colon T \to T'$ is a signature morphism $\phi \colon \Sigma \to \Sigma'$ such that $E' \models \phi(e)$, for all $e \in E$.*

In other words, the signature morphism ϕ is a theory morphism if the translated equations of the source theory T are semantically entailed by the target theory T'.

[2] Consider the conditional equation, $(\forall X) l = r$ if c, which is interpreted as meaning the equality holds if the condition c is true.

Our approach to equational logic so far has been purely model theoretic. A proof theory for many-sorted equational logic can be defined by defining rules of deduction for the following [13]: reflexivity, symmetry, transitivity, substitutivity, abstraction, and concretion. Given a theory (Σ, E), we say that an equation $(\forall X)t = t'$ is *deducible* from E if there is a deduction from E using the rules of deduction whose last equation is $(\forall X)t = t'$ [18]. We write: $E \vdash (\forall X)t = t'$.

The model theoretic and the proof theoretic approaches to equational logic are related by the notion of soundness and completeness [13].

Theorem 1. (Soundness and Completeness of Equational Logic) *Given an equational theory (Σ, E), an arbitrary equation $(\forall X)t = t'$ is semantically entailed iff $(\forall X)t = t'$ is deducible from E. Formally,*

$$E \models (\forall X)t = t' \text{ iff } E \vdash (\forall X)t = t',$$

where $t, t' \in T_\Sigma(X)$.

This theorem is very convenient, since it lets us use equational deduction to check the theory morphism conditions above which plays an important part in our system implementation.

Term rewriting [10, 13] can be considered an efficient implementation of *unidirectional equational deduction* by viewing equations as rewrite rules from left to right. Term rewriting forms the basis of the operational semantics of the OBJ specification language [5].

3 Semantics and Implementation

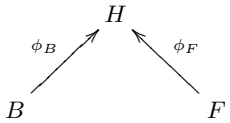
Inductive logic programming concerns itself with the induction of first-order theories from facts and background knowledge [17]. Although it is possible to induce theories from positive facts only, that is from facts that are to be entailed by the concept, having negative facts, that is facts that are not to be entailed by the concept, helps to limit the domain. Therefore, both positive as well as negative facts are typically given. Before we develop our semantics we have to define what we mean by background knowledge and facts.

Definition 9. *A theory (Σ, E) is called a* **Σ-facts** *theory if each $e \in E$ is a ground equation. A theory (Σ, B) is called a* **background theory** *if it defines auxiliary concepts that are appropriate for the domain to be learned. The equations in B do not necessarily have to be ground equations.*

In the inductive logic programming literature induced theories are usually referred to as hypotheses [17]. We adopt this terminology here. We define our algebraic notion of hypothesis as follows,

Definition 10. *Given a background theory $B = (\Sigma_B, E_B)$, positive facts $P = (\Sigma_P, E_P)$ (facts to be entailed), and negative facts $N = (\Sigma_N, E_N)$ (facts not*

to be entailed), then an **hypothesis** $H = (\Sigma_H, E_H)$*, is a theory with a pair of mappings* ϕ_B *and* ϕ_F



where

- $\phi_B \colon B \to H$ *is a theory morphism,*
- $\phi_F \colon F \to H$ *is a theory morphism,*
- *and* $F = (\Sigma_P, E_P) \cup \neg(\Sigma_N, E_N)$ *is a positive* Σ*-facts theory.*

Here, $\neg(\Sigma_N, E_N)$ denotes the representation of the negative facts as positive facts by coding them as inequality relations that have to hold in the hypothesis. More precisely, $\neg(\Sigma_N, E_N) = (\Sigma_N, \neg E_N)$ and $\neg E_N$ is a set of equations such that each $l = r \in E_N$ corresponds to an equation $(l \neq r) = \texttt{true} \in \neg E_N$. The above union operator is the a component-wise, sort-indexed operation.

Taking a closer look at ϕ_B, from the definition we have $\phi_B \colon B \to H$ is a theory morphism if $H \models \phi_B(e)$, for each $e \in E_B$. This is equivalent of saying that in order for this mapping to be valid the hypothesis must semantically entail the given background knowledge. Of course this holds trivially if ϕ_B is the inclusion morphism. Our prototype implementation treats ϕ_B as an inclusion morphism.

Similarly, ϕ_F maps the facts into the hypothesis. Again from the definition, $\phi_F \colon F \to H$ is a theory morphism if $H \models \phi_F(e)$, for each $e \in E_F$. Please note, by replacing the semantic entailment with proof theoretic deduction which follows from the soundness and completeness of equational logic we obtain a computable relation. This is precisely what we use in our current system implementation.

Note that this semantics does not say anything about the quality of a particular hypothesis. In fact, it is interesting to note that this semantics admits a number of trivial solutions; for instance, let $H = P$. Also consider the case where $B \models p$ for every $p \in P$. Typically, the weighing of one hypothesis over another is left to the operational or search semantics of a system. In our case we are using an evolutionary approach to this search.

We have implemented this semantics in our prototype system within the OBJ3 algebraic specification system [5]. The current prototype incorporates a genetic programming engine based on Koza's canonical Lisp implementation [12]. The main modifications in the engine are the addition of a type structure to the terms as well as the fact that the engine is cognizant of the syntactic structure of equational theories, that is, the genetic programming engine does not have to rediscover the concept of an equational theory with every run. The cross-over and mutation operators are implemented in the same straight forward manner as in Koza's system with the only exception that they respect the type structure on the terms. The fitness function used by the system to evaluate each candidate theory is

$$\text{fitness}(T) = (\text{facts}(T))^2 + \frac{1}{\text{length}(T)}\,,$$

where T denotes a candidate theory, facts(T) is the number of facts or fitness cases entailed by the candidate theory, and length(T) is the number of equa-

tions in the candidate theory. The fitness function is designed to primarily exert evolutionary pressure toward finding candidate theories that match all the facts. In addition, the function also exerts pressure toward finding the shortest theory that supports all the facts. Our implementation is similar to the implementation given in [6].

4 Fisher's Theorem

R. A. Fisher stated his fundamental theorem of natural selection in 1930 [3]:

Theorem 2. *The rate of increase in fitness of any organism at any time is equal to its genetic variance in fitness at that time.*

We can state a corollary of the theorem more appropriate for artificial evolutionary systems [1, 4]:

Corollary 1. *The rate of change in the average fitness of a population is proportional to its genetic variance in fitness, more formally:*

$$\Delta\overline{F} \propto Var(F),$$

where $\Delta\overline{F}$ is the change in the average fitness of a population from one generation to another and $Var(F)$ is the genetic variance in the fitness of a population.

Fisher states his theorem in terms of increasing fitness. However, it is not uncommon in genetic programming systems to express the fitness of an individual as well as the average fitness of a population in normalized terms where the normalized fitness is defined as follows:

Definition 11. *Let $\nu(F)$ denote the* **normalized fitness** *for a fitness value F then,*

$$\nu(F) = \beta_F - F,$$

where β_F is the best possible fitness value attainable by an individual in a population.

A consequence of normalizing the fitness is that the fitness of an individual as well as the average fitness of a population decreases as the individuals become better adapted. In this setting, an individual in a genetic programming system that has found a solution has a normalized fitness value $\nu(F) = 0$.

We are now in a position to articulate the core idea of this paper.

Proposition 1. *The genetic operators of our system are appropriate if the evolutionary population dynamics in our systems behaves according to Corollary 1.*

In other words, our genetic operators are appropriate, if the evolutionary population dynamics in our system mimics natural evolutionary population dynamics. In the following sections we look at a set of experiments that show that our genetic operators seem indeed to be appropriate, since the system exhibits an evolutionary population dynamics in accordance to Corollary 1.

5 Experiment I

In this first experiment we are interested in inducing an equational theory for a recursive definition of the predicate `even`. This predicate returns true if its argument is even and false if it is not. The following is the canonical equational definition of this predicate given in OBJ3 notation.

```
obj EVEN is
  sort Int .
  op 0 : -> Int .
  op s : Int -> Int .
  op even : Int -> Bool .
  var X : Int .
  eq even(s(s(X))) = even(X) .
  eq even(0) = true .
endo
```

The theory begins with sort, operator, and variable declarations. The actual definition of the predicate are the two equations at the bottom of the theory. Please note that here we give the naturals in Peano notation where $s(0) \mapsto 1$, $s(s(0)) \mapsto 2$, *etc.* Our aim is to have our genetic programming system evolve this theory from a set of facts. The facts theory for this problem looks like this.

```
obj EVEN-FACT is
  sort Int .
  op 0 : -> Int .
  op s : Int -> Int .
  op even : Int -> Bool .
  eq even(0) = true .
  eq even(s(s(0))) = true .
  eq even(s(s(s(s(0))))) = true .
  eq (even(s(0)) =/= true) = true .
  eq (even(s(s(s(0)))) =/= true) = true .
  eq (even(s(s(s(s(s(0)))))) =/= true) = true .
endo
```

Again, the theory begins with a set of sort and operator declarations followed by a set of ground equations representing the facts. Note the negative facts coded as inequality relations that need to hold in the hypothesis.

We chose this problem as our first problem since the strong typing (the domain of the predicate are the naturals, it's co-domain are the boolean values) limits the search space the evolutionary algorithm has to traverse. However, it is still big enough to not allow for enumeration.

First we chose a population of 150 individuals which is big enough to ensure convergence. We obtain a convergence rate of about 90% over 50 runs. We limited each run to a maximum of 20 generations. Figure 1 highlights the convergence behavior for this set of runs. It is worth mentioning that out of the 50 runs the large majority converged on a solution within 2-6 generations. Only 4 runs did not converge on a solution within the bounds of 20 generations.

Figure 2 shows the mean standardized fitness and the fitness variance for all the runs that converged in 9 generations. The curves are characteristic for all the runs that converged.

Next we limit the size of the population to 25 individuals. Here we obtain a convergence rate of about 10% with 50 runs over 100 generations. Of the 50 runs, 6 converged on a solution. It is interesting to note that the rest of the runs in fact converged on a local minimum, which is simply the enumeration

of the positive facts as the hypothesis. Figure 3 shows the averaged means and variances for the 44 runs that did not converge on a solution.

It seems that in both, the convergence case and the non-convergence case, the mean fitness and the fitness variance behave as predicted by Corollary 1: with an increase in fitness variance the slope of the mean fitness curve increases (even if it is only slightly in this example), with a decrease in fitness variance the mean fitness curve levels off.

6 Experiment II

In the next experiment we want to induce a recursive definition of addition. In our case we represent the addition of naturals with the function symbol `sum`. Here is the canonical equational definition of addition.

```
obj SUM is sort Int .
  op 0 : -> Int .
  op s : Int -> Int .
  op sum : Int Int -> Int .
  vars X0 X1 : Int .
  eq sum(X1,0) = X1 .
  eq sum(X0,s(X1)) = s(sum(X0,X1)) .
endo
```

Again, naturals are given in Peano notation. For this experiment we have 16 positive and negative facts identifying examples and counter examples of what addition is.

```
obj SUM-FACT is sort Int .
  op 0 : -> Int .
  op s : Int -> Int .
  op sum : Int Int -> Int .
  eq sum(0,0) = 0 .
  eq sum(s(0),s(0)) = s(s(0)) .
  eq sum(0,s(0)) = s(0) .
  eq sum(s(s(0)),0) = s(s(0)) .
  eq sum(s(0),0) = s(0) .
  eq sum(s(0),s(s(0))) = s(s(s(0))) .
  eq sum(s(s(0)),s(s(0))) = s(s(s(s(0)))) .
  eq sum(s(0),0) = s(0) .
  eq sum(s(s(s(0))),s(0)) = s(s(s(s(0)))) .
  eq sum(s(s(s(0))),s(s(0))) = s(s(s(s(s(0))))) .
  eq (sum(s(0),0) =/= 0) = true.
  eq (sum(0,0) =/= s(0)) = true .
  eq (sum(s(0),s(0)) =/= s(0)) = true .
  eq (sum(s(0),0) =/= s(s(0))) = true .
  eq (sum(0,s(0)) =/= s(s(0))) = true .
  eq (sum(0,s(0)) =/= 0) = true .
endo
```

In the first instance we chose the maximum population our system supports for this problem which is a population of 100 individuals. We obtained a convergence rate of about 20% with 10 runs limited to 100 generations each. Out of the 10 runs, 2 converged; one after 21 generations and one after 88 generations. Figure 4 shows the mean and variance in fitness for the run that converged after 21 generations. Again, we observe the typical increase in fitness variance just before a solution is found. As an interesting aside, this run induced a theory that was different from the canonical solution but just as fit:

```
  eq sum(X1,0) = X1 .
  eq sum(X0,s(X1)) = sum(s(X0),X1) .
```

This illustrates very nicely the "resourcefulness" of genetic algorithms that sets them apart from other search heuristics.

Next we restrict the size of the population to 25 individuals. Here we obtain a convergence rate of around 5% over 50 runs limited to 100 generations. Figure 5 shows the averaged variances and means for the 48 of the 50 runs that did not converge. After an initial ramp-up in diversity, the variance levels off. Also notice the extremely shallow slope of the mean fitness curve. It seems that after the initial ramp-up the maximum diversity with 25 individuals has been achieved and only very little progress is being made toward finding a solution given this limited diversity. Again, it seems that the mean fitness and the fitness variance behave according to Corollary 1. We strongly suspect that with a larger population we would see a better convergence behavior, since our genetic operators behave appropriately.

7 Conclusions

We are interested in inducing equational theories from facts. Our current, limited prototype that implements inductive equational logic programming using evolutionary techniques shows good convergence behavior for small problems but relatively poor convergence for larger problems. The fundamental question we asked: is the convergence behavior in small problems due to our genetic operators or is it incidental and conversely is the poor convergence behavior in larger problems due to the limitations of the prototype implementation which can only support a maximum populations with 200 individuals (100 individuals for larger problems) or is it due to limitations in the design of our genetic operators. In order to answer this question we chose to study the evolutionary population dynamics of our system and compare it to the theoretical behavior predicted by Fisher's Fundamental Theorem of Natural Selection. It seems that the evolutionary population dynamics behaves as predicted by Fisher's Theorem and therefore we conclude that our genetic operators are appropriate. We conclude that the good convergence behavior in smaller problems is due to our genetic operators and that the poor convergence behavior in larger problems is due to the limitation on population size in the current prototype.

We find it remarkable that our straightforward design of the genetic operators is appropriate for a complicated domain such as inductive logic programming.

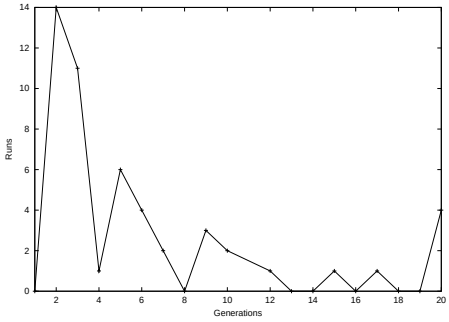


Fig. 1. Convergence behavior for the Even problem, pop. 150, gens. 20.

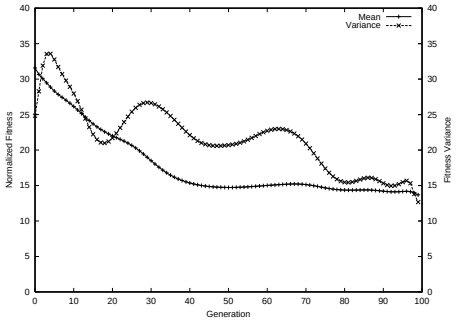


Fig. 3. The Even problem with pop. 25 for runs that did not converge.

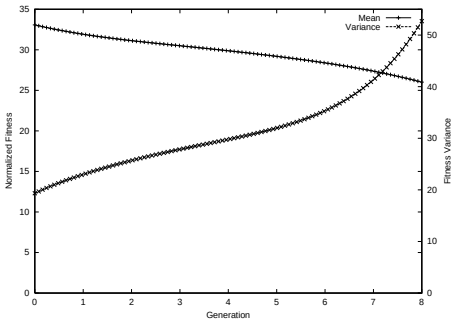


Fig. 2. The Even problem with pop. 150 and gens. 9.

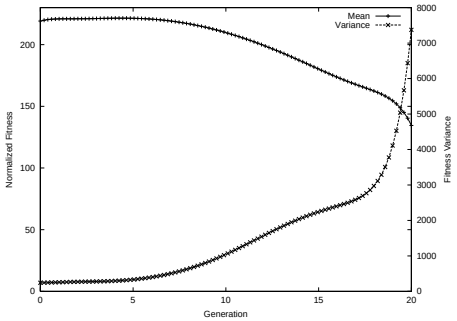


Fig. 4. The Sum problem, pop. 100, gens. 21.

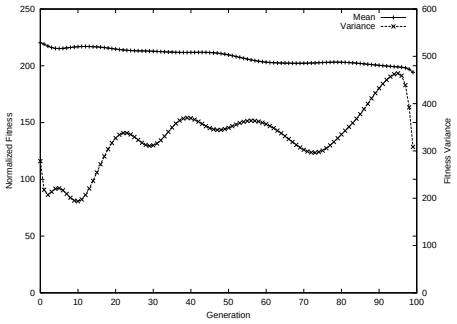


Fig. 5. The Sum problem, pop. 25, gens. 100.

References

1. L. Altenberg. The Schema Theorem and Price's Theorem. In *Foundations of Genetic Algorithms 3*, 1995.
2. F. Divina and E. Marchiori. Evolutionary concept learning. In W. B. Langdon *et al.*, editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers, 2002.
3. R. A. Fisher. *The Genetical Theory of Selection*. Claredon, Oxford, 1930.
4. S. A. Frank. The Price Equation, Fisher's fundamental theorem, kin selection, and causal analysis. In *Evolution*, volume 51, pages 1712–1729. 1997.
5. J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J. Jouannaud. *Software Engineering with OBJ: algebraic specification in action*, chapter Introducing OBJ. Kluwer, 2000.
6. L. Hamel. Breeding algebraic structures—an evolutionary approach to inductive equational logic programming. In W. B. Langdon *et al.*, editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers, 2002.
7. J. Hernández-Orallo and M. J. Ramírez-Quintana. A strong complete schema for inductive functional logic programming. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634. Springer-Verlag, 1999.
8. R. Ichise. Inductive logic programming and genetic programming. In H. Prade, editor, *European Conference on Artificial Intelligence*, 1998.
9. C. J. Kennedy and C. Giraud-Carrier. An evolutionary approach to concept learning with structured data. In *Proceedings of the fourth International Conference on Artificial Neural Networks and Genetic Algorithms*. Springer Verlag, 1999.
10. J. W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.
11. J. R. Koza. Concept formation and decision tree induction using the genetic programming paradigm. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496. Springer-Verlag, 1-3 1991.
12. J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, 1992.
13. J. Meseguer and J. Goguen. Initiality, induction and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*. Cambridge, 1985.
14. T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2), 1982.
15. D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2), 1995.
16. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4), 1995.
17. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20, 1994.
18. W. Wechler. *Universal Algebra for Computer Scientists*. Springer-Verlag, 1992. EATCS Monographs on Theoretical Computer Science, Volume 25.
19. M. Wong and K. Leung. Genetic logic programming and applications. *IEEE Expert*, October 1995.