

# Initial Study on Handling Constrained Optimization Problems in Learnable Evolution Model

Janusz Wojtusiak

School of Computational Sciences  
George Mason University  
4400 University Drive MSN 5B2  
Fairfax, VA 22030, USA  
jwojt@mli.gmu.edu

## ABSTRACT

Learnable Evolution Model (LEM) is an evolutionary computation methodology that applies hypothesis formulation and instantiation to create new individuals. Initial study has shown that LEM significantly outperforms standard evolutionary computation methods in terms of evolution length on selected benchmark optimization problems. This paper presents initial results from handling constrained optimization problems in LEM. Constraints are classified as instantiable, which can be handled directly during instantiation process, and general, which cannot be directly instantiated. The later can be handled by applying three different methods presented in this paper.

## Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning – *Concept learning, induction*. G.1.6 [Optimization]: *Constrained optimization*.

## General Terms

Algorithms, Performance, Design, Experimentation, Theory.

## Keywords

Constraints, Evolutionary Computation, Learnable Evolution Model, Machine Learning, Non-Darwinian Evolutionary Computation

## 1. INTRODUCTION

Evolutionary computation presents an important approach to the optimization of complex functions or systems. Its popularity is stems mostly from the fact that it requires little problem knowledge to set up an optimization problem, and that individuals (candidate solutions) are created through semi-random or random operators, such as mutation and/or recombination, that are easy to implement. These methods, however, suffer from slow convergence to the optimal solutions, which makes them impractical for problems in which evaluation of the fitness

function takes a significant amount of time (and thousands of evaluations are needed).

Learnable Evolution Model (LEM) is a non-Darwinian evolutionary computation method that applies hypothesis formulation and instantiation to generate new candidate solutions [5]. It selects groups of high- and low-performing individuals from the population and uses them as positive and negative examples, respectively, for learning. A learning program generates a general hypothesis that characterizes high-performing individuals in contrast to the low-performing ones. This hypothesis is instantiated in order to produce new individuals.

The goal of this paper is to describe ongoing research on handling constrained optimization problems in Learnable Evolution Model, which constitutes part of the author's Ph.D. research. The problem of handling constraints is very important from a practical point of view, because most real world optimization problems are constrained. The presented methods are only partially developed, and are not yet sufficiently studied and tested.

Section 2 of this paper briefly describes Learnable Evolution Model and its selected experimental results (on non-constrained problems). Section 3 describes the initial study on methods of handling constraints in LEM. Future research and conclusions from the preliminary study are presented in Sections 4 and 5, respectively.

## 2. LEARNABLE EVOLUTION MODEL

This section briefly describes LEM3, the newest implementation of Learnable Evolution Model [13], [14]. Based on this description, in particular regarding learning mode, Section 3 describes methods for handling constraints.

LEM3 contains several components that are also found in traditional evolutionary algorithms, such as generation of an initial population, selection of individuals for a new population, and evaluation of individuals. These methods are well known and are not discussed further in this paper.

Components that are unique to LEM3 concern guiding evolutionary computation through machine learning, adjustment of the representation space, and multistrategy selection of actions to be executed. Figure 1 presents the top-level algorithm underlying LEM3.

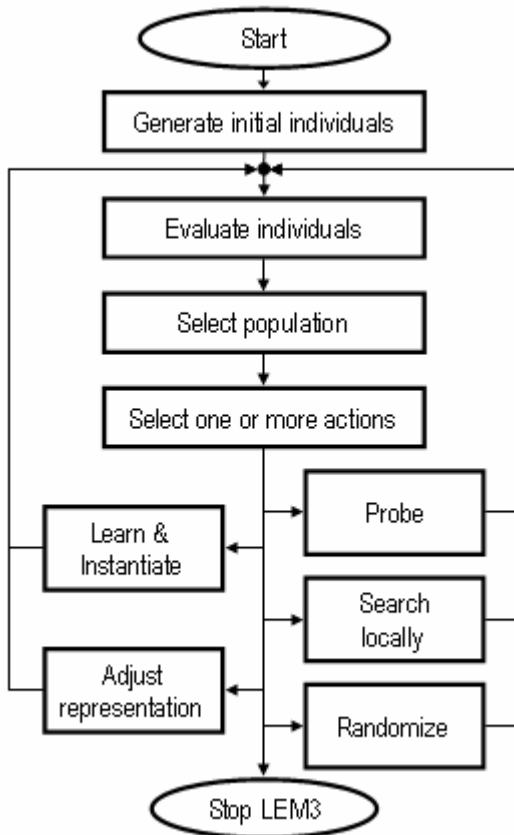
---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.  
Copyright 2006 ACM 1-59593-186-4/06/0007...\$5.00.

The major operator for creating new individuals in Learnable Evolution Model is the “Learn and Instantiate” action (a.k.a. learning mode). This action creates new individuals by performing three steps: (1) selecting the training set for the learning program (2) learning a hypothesis characterizing subspaces that likely contain the optimum, and (3) instantiating the hypothesis in various ways to create new individuals.

Step (1) selects high-performing (H-group) and low-performing (L-group) individuals from the population, according to the given fitness function. These individuals serve as positive and negative examples, respectively, for a learning program. There are two methods of creating these groups. These two selection methods, fitness-based and population-based, are described, for example, in [5] and [13]. The H- and L-groups are then passed as positive and negative examples to a learning program.



**Figure 1: Flowchart of LEM3 algorithm.**

In step (2) of LEM learning mode, the program applies hypothesis formulation to obtain a general description of the H-group against the L-group. Although any learning method can be applied in this step, our study concentrates on AQ rule learning, in particular its AQ21 implementation [12]. This program is the newest implementation of the AQ learning, an inductive learning method that produces hypotheses in the form of sets of attributional rules [6]. The simplest form of such a rule is:

$$\text{CONSEQUENT} \Leftarrow \text{PREMISE}$$

where CONSEQUENT and PREMISE are conjunctions of attributional conditions (a.k.a. selectors). The simplest form of an

attributional condition defines a relation between an attribute and attribute values that satisfy that condition. Here is an example of an attributional rule:

---


$$[\text{design} = \text{high-performing}] \Leftarrow [\text{weight} = 2..5] \ \& \ [\text{shape} = \text{rhombus} \vee \text{triangular}] \ \& \ [\text{height} < 3]$$


---

The rule states that a design is classified as high-performing if its weight is between 2 and 5 (units are presumably defined in the attribute domain), its shape is rhombus or triangular, and its height is less than 3.

A hypotheses learned by AQ21 usually consist of a number of such rules. Please note that rules learned by AQ21 have much higher expressive power than those learned by most machine learning systems.

For completeness of this discussion, Figure 2 presents pseudocode of simplified AQ21 learning algorithm. Further details on AQ learning and some of its more extended forms can be found in, for example, [7].

---

*HYPOTHESIS = null*  
 While not all H-group examples are covered  
   Select uncovered positive example  $e^+$  and use it as a seed  
   Generate star  $G(e^+, L\text{-group})$   
   Select the best rule,  $R$ , from the star according to a given criterion of optimality, and add it to HYPOTHESIS  
   Remove examples covered by  $R$  from H-group

---

**Figure 2: Basic AQ21 learning algorithm.**

The instantiation process (Step 3 of learning mode) generates new individuals that satisfy the learned hypothesis. When instantiating a rule to create an individual for the new population, the program faces the problem of instantiating values to attributes that are specified in the rule and to attributes not present in the rule. A basic instantiation algorithm implemented in LEM3 is presented in Figure 3. For a detailed description, discussion and more advanced algorithms, please refer to [14].

---

*For each rule in a ruleset (hypothesis) to be instantiated*  
 Compute the number of individuals to be created  
 For each individual to be created  
   Create the individual  
   For each attribute  
     If the attribute is specified in the rule  
       Select a random value satisfying the rule  
     Else Select a random individual from the previous population and use its value

---

**Figure 3: Basic instantiation algorithm in LEM3.**

In addition to learning mode described above, LEM3 implements several other actions. The *Probe* action applies mutation and crossover operators known in evolutionary computation. The *Search locally* action applies user defined local search operators such as gradient-based methods. The *Randomize* action either adds randomly generated individuals into a population or restarts the evolution process by regenerating randomly the entire population. These operators are, however, beyond the scope of this paper.

*Adjust representation* is applied to improve the representation space in which hypotheses are learned. This includes finding the most suitable discretization of numeric attributes, ignoring irrelevant attributes, and constructing new attributes that can better capture features of the fitness landscape.

Comparison of LEM3 with other evolutionary computation methods shows its strong advantage, which tends to grow with the number of variables. For example, LEM3 required on average 16.5 times fewer fitness evaluations than EA, a standard evolutionary computation method [2], when optimizing Rastrigin, Griewangk and Rosenbrock functions of 100 to 1000 variables. It was also about 70 times faster than results of Estimation of Distribution Algorithms on Griewangk and Rosenbrock functions of 10 and 50 variables reported in [1]. When compared to results reported on Cultural Algorithms [9], LEM3 required on average 340 times fewer fitness evaluations. The study was performed on Rastrigin, Griewangk and Rosenbrock functions of 5, 3, and 2 variables respectively. Details of the experimental study are presented in [14].

### 3. HANDLING CONSTRAINTS

The problem of handling constrained optimization problems in evolutionary computation has been studied and presented in the literature by many authors. Several methods have been proposed; some are general and applicable to a wide range of optimization techniques, and some are designed for use with specific optimization algorithms. Michalewicz [3] proposed the classification of constraint-handling methods into four main categories: penalty functions, decoders, repair algorithms, and constraint preserving algorithms. In addition to the four categories, there are a number of methods concerning constraint-handling methods such as multi-objective optimization (e.g. [11]), cultural algorithms (e.g. [10]), and the coevolutionary model (e.g. [8]).

A constrained optimization problem in Learnable Evolution Model seeks feasible solutions  $X_1, \dots, X_k$  that are optima of function  $f(x_1, \dots, x_n): E \rightarrow R$  given a set of constraints  $C$  in Disjunctive Normal Form (DNF) defining feasible solutions. For example suppose that  $E$  is the Cartesian product of the domains  $D_1 = \{\text{red, green, blue}\}$ ,  $D_2 = [0 \dots 10]$ , and  $D_3 = [0 \dots 10]$  of three attributes, one nominal ( $x_1$ ) and two ratio ( $x_2$  and  $x_3$ ) and  $f: E \rightarrow R$  is a fitness function. An example of constraints in DNF for such a problem is:

$$[x_1=\text{red}] \ \& \ [x_2 > 4] \ \vee \ [x_1=\text{blue} \vee \text{green}] \ \& \ [x_2+x_3 < 10]$$

meaning that the feasible solutions are those for which  $x_1$  is red and  $x_2$  is greater than 4, or  $x_1$  is blue or green and the sum of  $x_2$  and  $x_3$  is less than 10.

Constraints are integral part of problem definitions (in addition to the representation space and fitness function). Constraints may represent physical limitations of an optimized system or expert background knowledge.

In order to handle constraints, the LEM methodology distinguishes between *instantiable constraints* and *general constraints*. The proposed methods are applicable only to learning mode in LEM, and are described in the next two sections. Methods for handling constraints for other modes have been widely investigated in the literature.

### 3.1 Handling Instantiable Constraints

Instantiable constraints can be directly instantiated as conditions using the algorithm presented in Figure 3. For example conditions  $[x_1=\text{red}]$  and  $[x_2 > 4]$  from the above example can be directly instantiated by assigning "red" as the value of attribute  $x_1$  and a random number greater than 4 as the value of attribute  $x_2$  in an instantiated individual. Another example of the instantiable constraint condition is  $[x_4 > x_2^2 + 7]$ , where the expression on the right side can be evaluated (assuming that  $x_2$  is already instantiated). This research concerns constraints in the form:

$$[\text{attr} \ \text{rel} \ \text{expr}]$$

where *attr* is an attribute, *expr* is an expression that includes only previously instantiated attributes (and does not include *attr*), and *rel* is a relation applicable to *attr* and *expr*. Constraints in this form represent a large number of possible constraints. For constraints in many different forms (for example sets of equations), efficient instantiation methods can also be proposed.

Instantiation for this type of constraint works in two steps. First, intersect a learned hypothesis with the constraints. Because both are represented in attributional calculus, such an operation is meaningful, and results in the creation of a *target constrained hypothesis*. In the second step, the target constrained hypothesis is instantiated using, for example, the algorithm presented in Figure 3. The following example shows a step-by-step instantiation of an exemplary hypothesis and constraints.

Let a hypothesis describing high-performing individuals consist of one rule:  $\leq [\text{color}=\text{red} \vee \text{green}] \ \& \ [\text{length} > 2.7]$ , and the constraints be given by a conjunction of two conditions:  $[\text{color}=\text{red}] \ \& \ [\text{width} < \text{length} + \text{height} - 7]$ .

In this case the target constrained hypothesis is:

$$\leq [\text{color}=\text{red}] \ \& \ [\text{length} > 2.7] \ \& \ [\text{width} < \text{length} + \text{height} - 7]$$

which is instantiated in four steps: (1) assign to attribute *color* the only possible value *red*, (2) assign to attribute *length* a random value 5, (3) assign to attribute *height* a value from an existing individual, say 6, (4) evaluate expression  $\text{length} + \text{height} - 7$  and assign to attribute *width* a random value consistent with the expression, say 2. This results in creation of individual ( $\text{color}=\text{red}$ ,  $\text{length}=5$ ,  $\text{width}=2$ ,  $\text{height}=6$ ) which satisfies both the learned hypothesis and the constraints.

A problem arises when because of instantiation of one condition, other conditions cannot be instantiated. Suppose that in the example above, the attribute *length* was assigned value 8, which is consistent with the second condition. Because of that, there is no value of the attribute *width* that could possibly satisfy the third condition. A backtracking algorithm is used to solve this problem.

Please note that whenever the group of high performing individuals (H-group) are feasible, the target constrained hypotheses are always instantiable, meaning that there exist individuals that satisfy all conditions at the same time. This can be proven using the fact that the hypotheses learned by AQ21 are complete and consistent.

### 3.2 Handling General Constraints

The previous section described methods of handling a class of constraints for which there is an efficient instantiation algorithm. Please note that in general problem of handling constraints is known to be NP-Hard, thus, there is no effective way to handle all possible constraints, and all proposed methods are good only for some problems.

In this section it is assumed that constraints are given as a function:

$$c: E \rightarrow \{\text{true}, \text{false}\}$$

where  $c(i) = \text{true}$  if an individual  $i$  is feasible (satisfies all constraints) and  $c(i) = \text{false}$  otherwise. Because of this assumption, the program does not have any prior knowledge about the constraints; it can only check whether they are satisfied. It also ignores the degree to which constraints are satisfied, for example, 3 out of 5 constraints are satisfied. These issues will be addressed in future research.

Three methods of handling general types of constraints are defined. They are applicable to LEM's learning mode only. The methods are illustrated using a simple example, and initial results from testing performance of these methods are presented.

Suppose that an optimization problem is defined in a two-dimensional representation space as illustrated in Figure 4. Individuals on the plot are marked H (high-performing), L (low-performing), and X (infeasible), and the shaded area represents the feasible region.

Given sets of positive examples (H-group) and negative examples (L-group) the learning program may generate rules characterizing the H-group illustrated in Figure 5. The rules are complete and consistent with regard to training data (they cover all positive and no negative examples), but also cover a number of infeasible individuals, and large portions of the infeasible region. When instantiating the rules, the program may generate many infeasible solutions that would have to be rejected. In real world optimization problems evaluation of constraints may be a very time consuming process, sometimes as consuming as evaluation of the fitness function. Thus, the presented methods of handling constraints are designed to minimize the number of infeasible solutions generated during the optimization process and at the same time do not increase the total number of the fitness function evaluations (evolution length).

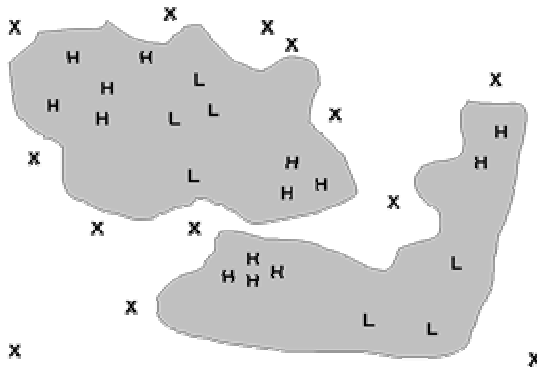


Figure 4: Feasible and infeasible individuals in the example problem.

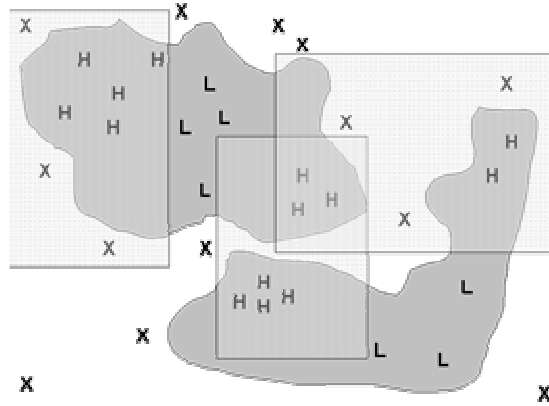


Figure 5: Example rules found using high- and low-performing examples.

#### 3.2.1 Trimming of Rules

The AQ21 learning program learns rules with controllable levels of generality. Figure 5 shows that rules cover a much larger area than is needed to cover the high-performing examples, and covers large portions of the infeasible region.

The first method of handling general constraints in LEM trims the learned rules, so they do not extend far beyond the high-performing examples. As shown in the Figure 6, the trimmed rules cover significantly less infeasible space, but large portions of the feasible region are also not covered, and solutions may be missed. This problem is solved by using *flexible rule interpretation*, that is generating 95% of the individuals to strictly match rules and 5% individuals with probabilities linearly decreasing with distance from the rule.

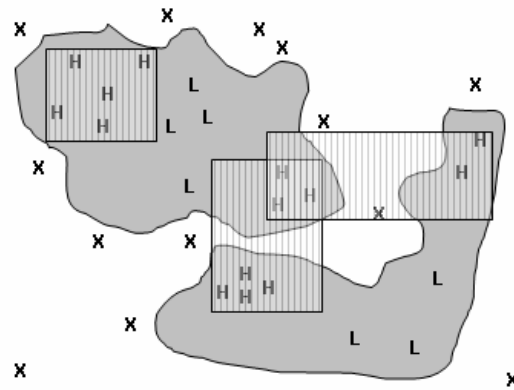


Figure 6: Trimmed rules for the example problem.

It can be also noted that the rule in the right part of the diagram covers a large portion of the infeasible area, because the program does not have any information that the area is infeasible. The problem arises when the feasible area consists of disjoint parts, or in general is not convex.

To overcome the latter problem two other methods are proposed.

### 3.2.2 Learning Approximation of Feasible Area

The general idea behind this method is to learn an approximation of the feasible area in parallel to the evolutionary optimization process. The presented method of learning approximate constraints applies the AQ21 learning program to sets of feasible and infeasible solutions. The advantage of using the same learning program for hypothesis formulation and feasible space approximation is that learned descriptions are represented in the same language (attributional calculus), and several operations can be performed on both, e.g., they can be easily intersected.

Let  $S_f$  be set of all feasible candidate solutions, and  $S_n$  be set of infeasible solutions created during the evolutionary optimization process in LEM. Using  $S_f$  as the set of positive and  $S_n$  as the set of negative examples, the method learns an approximation of the feasible subspace. Because both sets are growing during the evolution process, the approximation converges to the actual set of feasible solutions.

At each step of evolution a hypothesis describing high-performing individuals is intersected with the feasible space approximation. The intersection is then instantiated to generate new candidate solutions which are likely to be feasible and high-performing. It was aforementioned that the intersection is meaningful and can be easily computed because both, hypothesis and approximation are represented in attributional calculus and in the same representation space.

The approximation of the feasible area may miss the actual solution when the learned description is overspecialized. In such cases, some feasible candidate solutions may be missed and the actual solution to the optimization problem may also be missed, especially when it is close to the border of the set of feasible solutions. This problem can be solved by (1) learning maximally general descriptions of feasible solutions, (2) checking randomly selected candidate solutions that do not satisfy the learned description of the feasible area, or (3) using flexible rule interpretation, as described in the previous section. After instantiation, sets  $S_f$  and  $S_n$  are updated with new candidate solutions.

The method is illustrated in Figures 7 and 8.

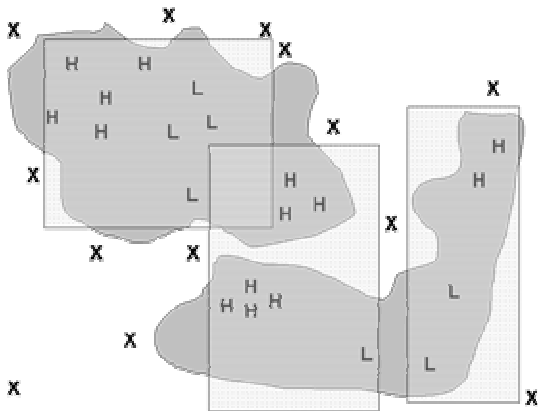


Figure 7: Feasible space approximation.

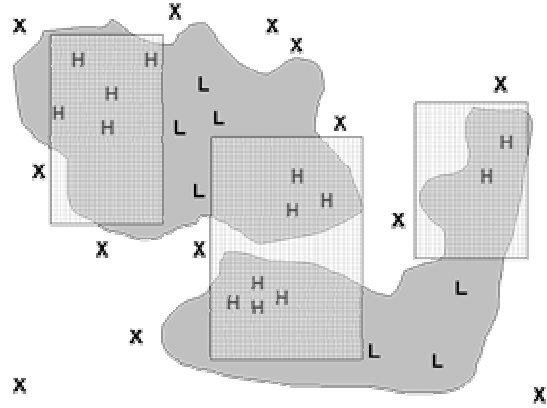


Figure 8: Intersection of learned hypothesis and feasible area approximation.

### 3.2.3 Using Infeasible Individuals as a Contrast Set for Learning

The last described method for handling general constraints keeps a list  $S_n$  of infeasible solutions and uses them as constraints for hypothesis formation in LEM, by adding  $S_n$  to the group of low-performing candidate solutions. A hypothesis learned using such a method not only describes high-performing candidate solutions, but also minimizes areas with infeasible solutions. After instantiation, the set  $S_n$  is updated with new candidate solutions.

The set  $S_n$  of infeasible individuals may be very large when many individuals are rejected. This may negatively affect performance of the learning program. To overcome this problem a subset of  $S_n$  is used as a contrast set for learning (whenever  $S_n$  is large). Selection of the subset can be random, or through choosing individuals that are the closest to known feasible ones.

Similarly to the other methods, to avoid missing solutions, when rules are overspecialized, flexible rule interpretation can be used.

Figure 9 presents an example hypothesis learned with infeasible examples used as a contrast set.

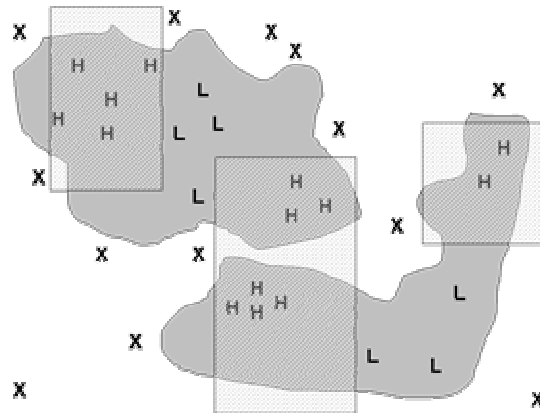


Figure 9: Hypothesis learned with set of infeasible solutions used as examples.

### 3.2.4 Initial Experimental Results

Initial application of the three methods to the G1 function [4] show that the trimming method gave the best results in terms of number of fitness function evaluations needed to find the solution. The G1 function is given by formula:

$$G1(X) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5 \sum_{i=1}^4 x_i^2 - 5 \sum_{i=5}^{13} x_i$$

with 9 constraints presented, for example, in [4].

In the presented experiment, LEM3 was executed with default parameters, most important of which are: population size 100, population-based selection method, and learning mode. Ten initial populations of feasible individuals (generated randomly with uniform distribution) were loaded by LEM3. The reported results are averages and standard deviations on the 10 runs.

**Table 1: Results of application of LEM to G1 function.**

Method	Number of fitness evaluations		Number of infeasible individuals	
	Average	Std. dev.	Average	Std. dev.
Trimming	1,383	137	1,902	600
Approximation	1,787	1086	4,045	6138
Contrast Set	2,562	831	1,731	1026

It is not surprising that the method based on the contrast set generated the smallest number of infeasible solutions, but was slow in terms of convergence to the solution. This is because it generated overspecialized rules that often missed the solution. The best in terms of number of fitness evaluations is the trimming method. It also requires only a slightly larger number of infeasible solutions.

A very large standard deviation for the approximation method is caused by one execution in which program “got stuck” in a point near optimum and switched to the probing mode to explore neighborhood of the point (it generated over 20,000 infeasible solutions). Such a point should be treated as an outlier, but clearly represents the worst case.

Results reported in [4] state that the authors applied the Genocop method, which required fewer than 1,000 generations to find the solution. It is very encouraging that LEM required in average only 14 generations to get the same result (in experiments repeated 10 times with different starting populations).

## 4. FUTURE RESEARCH

The presented methods of handling constraints in Learnable Evolution Model are in an initial stage of research. Although methods were proposed and mostly implemented, a theoretical and experimental study is needed to fully understand their behavior and applicability. In particular the research will include:

- detailed study of methods for handling general constraints (e.g. how approximations should be learned, how many examples of feasible and infeasible examples should be selected, how often the approximation should be updated),
- testing of methods of handling general constraints on selected constrained problems (e.g. other benchmark problems proposed and described in [4]),

- development of methods for handling infeasible constraints, in particular constraints in the form [att rel expr], and special functions such as *average*, *equal*, *count* [6],
- extension of the methodology to flexible constraints (which may not have to be satisfied). This includes extension of the proposed methods to reflect degrees to which constraints are satisfied.

Moreover, the methodology will be applied to a difficult real world problem.

## 5. CONCLUSION

Most real world optimization problems are constrained, thus efficient methods of handling constraints are important for the practical applicability of Learnable Evolution Model.

The presented methods are designed specifically to work with Learnable Evolution Model, in particular its LEM3 implementation, which uses the AQ21 rule learning program for hypothesis formulation. It produces hypotheses in the form of rules in attributional calculus – a highly expressive language which allows to represent both, hypotheses and constraints.

A special form of infeasible constraints has been introduced to allow efficient instantiation method. For all other constraints, three other (general) methods have been described. The very promising initial results of the presented methods encourage further investigation. Both theoretical study and large sets of experiments are needed to fully understand the methods.

The presented methods are in an initial stage of implementation and analysis. Detailed testing and comparison with existing methods is also part of ongoing research. Perhaps an even greater challenge is to find classes of problems to which these methods are applicable.

## 6. ACKNOWLEDGEMENTS

The author would like to express his gratitude to Dr. Ryszard Michalski, Dr. Kenneth Kaufman and Jarek Pietrzykowski for their comments on this paper and its earlier versions and for their help during process of development of LEM3 system.

Research presented here was conducted at the Machine Learning and Inference Laboratory of George Mason University. Research activities of the Machine Learning and Inference Laboratory are supported by the National Science Foundation Grants No. IIS 9906858 and IIS 0097476. The findings and opinions expressed here are those of the author, and do not necessarily reflect those of the above sponsoring organizations.

## 7. REFERENCES

- [1] Bengoetxea, E., Miquelz, T., Larranaga, P., and Lozano, J.A., Experimental Results in Function Optimization with EDAs in Continuous Domain. In Pedro Larranaga and Jose A. Lozano *Estimation of Distribution Algorithms*, Kluwer Academic Publishers, 2002.
- [2] Evolutionary Objects Library, downloadable from the website: <http://eodev.sourceforge.net>
- [3] Michalewicz, Z. Introduction to Constraint Handling Techniques. In T. Back, D.B. Fogel, Z. Michalewicz

*Evolutionary Computation 2*, Philadelphia Institute of Physics Publishing, 2000.

- [4] Michalewicz, Z., Schoenauer M. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation 4*. 1996.
- [5] Michalski, R.S. LEARNABLE EVOLUTION MODEL Evolutionary Processes Guided by Machine Learning. *Machine Learning*, Vol. 38, 2000, pp. 9-40.
- [6] Michalski, R.S. ATTRIBUTIONAL CALCULUS: A Logic and Representation Language for Natural Induction. *Reports of the Machine Learning and Inference Laboratory*, MLI 04-2, George Mason University, Fairfax, VA, April, 2004.
- [7] Michalski, R. S. and Kaufman, K. The AQ19 System for Machine Learning and Pattern Discovery: A General Description and User's Guide. *Reports of the Machine Learning and Inference Laboratory*, MLI 01-2, George Mason University, Fairfax, VA, 2001.
- [8] Paredis, J., Coevolutionary Constraint Satisfaction, Parallel. *Problem Solving from Nature III, Lecture Notes in Computer Science*, vol. 866, Davidor, Y., Schwefel, H-P., Manner, R. (eds.), Springer Verlag, 1994.
- [9] Reynolds, R. G. and Zhu, S. Knowledge-Based Function Optimization Using Fuzzy Cultural Algorithms with Evolutionary Programming. *IEEE Transactions on Systems, Man, and Cybernetics*, 31, 2001.
- [10] Reynolds, R. G., Michalewicz, Z., and Cavaretta, M. Using cultural algorithms for constraint handling in GENOCOP. *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, 1995.
- [11] Surry, P. D., Radcliffe, N., and Boyd I. D. A Multi-objective Approach to Constrained Optimisation of Gas Supply Networks: The COMOGA Method. *AISB-95 Workshop on Evolutionary Computing*, 1995.
- [12] Wojtusiak, J. AQ21 User's Guide. *Reports of the Machine Learning and Inference Laboratory*, George Mason University, MLI 04-3, Fairfax, VA, 2004.
- [13] Wojtusiak, J. The LEM3 Implementation of Learnable Evolution Model: User's Guide. *Reports of the Machine Learning and Inference Laboratory*, George Mason University, MLI 04-5, Fairfax, VA, 2004.
- [14] Wojtusiak, J. and Michalski, R. S. The LEM3 System for Non-Darwinian Evolutionary Computation and Its Application to Complex Function Optimization. *Reports of the Machine Learning and Inference Laboratory*, MLI 05-2, George Mason University, Fairfax, VA, October, 2005.