International Journal of Artificial Intelligence,
ISSN 0974-0635; Int. J. Artif. Intell.
Spring (March) 2014, Volume 12, Issue Number 1

A Learning Procedure for Sampling Semantically Different Valid Expressions

David L. St-Pierre[1], **Francis Maes**[1], **Damien Ernst**[1] **and Quentin Louveaux**[1]

[1]Montefiore Institute, Department of Electrical Engineering and Computer Science
Liège University, B-4000 Liège, BE

ABSTRACT

A large number of problems can be formalized as finding the best symbolic expression to maximize a given numerical objective. Most approaches to approximately solve such problems rely on random exploration of the search space. This paper focuses on how this random exploration should be performed to take into account expressions redundancy and invalid expressions. We propose a learning algorithm that, given the set of available constants, variables and operators and given the target finite number of trials, computes a probability distribution to maximize the expected number of semantically different, valid, generated expressions. We illustrate the use of our approach on both medium-scale and large-scale expression spaces, and empirically show that such optimized distributions significantly outperform the uniform distribution in terms of the diversity of generated expressions. We further test the method in combination with the recently proposed nested Monte-Carlo algorithm on a set of benchmark symbolic regression problems and demonstrate its interest in terms of reduction of the number of required calls to the objective function.

Keywords: Symbolic Regression, Machine Learning, optimization.
Mathematics Subject Classification: MSC - 65C05.
Computing Classification System:G.3 PROBABILITY AND STATISTICS — Monte-Carlo.

1 Introduction

A large number of problems can be formalized as finding the best expressions, or more generally the best programs, to maximize a given numerical objective. Such optimization problems over expression spaces arise in the fields of robotics (Hu and Yang, 2004), finance (Kargupta and Buescher, 1996), molecular biology (Jones, Willett, Glen, Leach and Taylor, 1997), pattern recognition (Maulik and Bandyopadhyay, 2000), simulation and modeling (Kikuchi, Tominaga, Arita, Takahashi and Tomita, 2003) or engineering design (Deb, Pratap, Agarwal and Meyarivan, 2002) to name a few.

These optimization problems are hard to solve as they typically involve very large discrete spaces and possess few easily exploitable regularities, mainly due to the complexity of the mapping from an expression syntax to its semantic (e.g. the expressions $c \times (a+b)$ and $c/(a+b)$ differ only by one symbol but have totally different semantics). Due to the inherent difficulties related to the nature of expression spaces, these optimization problems can rarely be solved exactly and a wide range of approximate optimization techniques based on stochastic search

have been proposed. In particular, a large body of work has been devoted to evolutionary approaches known as genetic programming (Koza and Poli, 2005; Holland, 1992; O'Neill and Ryan, 2001). While genetic programming algorithms have successfully solved a wide range of real-world problems, these algorithms may be complex to implement and are often too difficult to analyze from a theoretical perspective.

(Cazenave, 2010) recently proposed to use a search technique based on Monte-Carlo sampling to solve optimization problems over expression spaces, a promising alternative approach that avoids some important pitfalls of genetic programming. One key component of this Monte-Carlo approach is the procedure that samples expressions randomly. The proposed approach was based on uniformly sampling expression symbols. This choice fails to tackle the redundancy of expressions: it is often the case that a large number of syntactically different expressions are equivalent, for example due to commutativity, distributivity or associativity. This choice also does not take into account that some expressions may have an undefined semantic, due to invalid operations such as division by zero.

This paper focuses on the improvement of the sampling procedure used in the context of Monte Carlo search over expression spaces. Given a number T of trials, we want to determine a memory-less sampling procedure maximizing the expected number of semantically different valid expressions generated $S_T \leq T$. To reach this objective, we propose a learning algorithm which takes as input the available constants, variables and operators and optimizes the set of symbol probabilities used within the sampling procedure. We show that, on medium-scale problems, the optimization of symbol probabilities significantly increases the number of non-equivalent expressions generated. For larger problems, the optimization problem cannot be solved exactly. However, we show empirically that solutions found on smaller problems can be used on larger problems while still significantly outperforming the default uniform sampling strategy.

The rest of this paper is organized as follows. Section 2 formalizes the problem and introduces notations. Our symbol probabilities learning algorithm is described in Section 3. We evaluate the quality of the improved sampling procedure in Section 4. Section 5 combines the approach with random search and nested Monte-Carlo search and applies it on symbolic regression. Finally, Section 6 concludes.

2 Problem formulation

We now introduce Reverse Polish Notation (RPN) as a way of representing expressions in Section 2.1 and describe a generative process compliant with this representation in Section 2.2. Section 2.3 carefully states the problem addressed in this paper.

2.1 Reverse Polish Notation

RPN is a representation wherein every operator follows all of its operands. For instance, the RPN representation of the expression $c \times (a + b)$ is the sequence of symbols $[c, a, b, +, \times]$. This way of representing expressions is also known as postfix notation and is parenthesis-free as

Algorithm 1 RPN evaluation

Require: $s \in \mathcal{A}^D$: a sequence of length D
Require: $x \in \mathcal{X}$: variable values
  stack $\leftarrow \emptyset$
  for $d = 1$ to D **do**
    if α_d is a variable or a constant **then**
      Push the value of α_d onto the stack.
    else
      Let n be the arity of operator α_d.
      if $|stack| < n$ **then**
        syntax error
      else
        Pop the top n values from the stack, compute α_d with these operands, and push the result onto the stack.
      end if
    end if
  end for
  if $|stack| \neq 1$ **then**
    syntax error
  else
    return top(stack)
  end if

long as operator arities are fixed, which makes it simpler to manipulate than its counterparts, prefix notation and infix notation.

Let $\mathcal{A}$ be the set of *symbols* composed of constants, variables and operators. A sequence s is a finite sequence of symbols of $\mathcal{A}$: $s = [\alpha_1, \ldots, \alpha_D] \in \mathcal{A}^*$. The evaluation of an RPN sequence relies on a stack and is depicted in Algorithm 1. This evaluation fails either if the stack does not contain enough operands when an operator is used or if the stack contains more than one single element at the end of the process. The sequence $[a, \times]$ leads to the first kind of errors: the operator $\times$ of arity 2 is applied with a single operand. The sequence $[a, a, a]$ leads to the second kind of errors: evaluation finishes with three different elements on the stack. Sequences that avoid these two errors are syntactically correct RPN expressions and are denoted $e \in \mathcal{E} \subset \mathcal{A}^*$.

Let $\mathcal{X}$ denote the set of admissible values for the variables of the problem. We denote $e(x)$ the outcome of Algorithm 1 when used with expression e and variable values $x \in \mathcal{X}$. Two expressions $e_1 \in \mathcal{E}$ and $e_2 \in \mathcal{E}$ are semantically equivalent if $\forall x \in \mathcal{X}, e_1(x) = e_2(x)$. We denote this equivalence relation $e_1 \sim e_2$. The set of semantically incorrect expressions $\mathcal{I} \subset \mathcal{E}$ is composed of all expressions e for which there exists $x \in \mathcal{X}$ such that $e(x)$ is undefined, due to an invalid operation such as division by zero or logarithm of a negative number. In the context of Monte-Carlo search, we are interested in sampling expressions that are semantically correct and semantically different. We denote $\mathcal{U} = (\mathcal{E} - \mathcal{I})/\sim$ the quotient space of semantically

Table 1: Size of $\mathcal{U}^D$, $\mathcal{E}^D$ and $\mathcal{A}^D$ for different sequence lengths D.

D	$\lvert\mathcal{U}^D\rvert$	$\lvert\mathcal{E}^D\rvert$	$\frac{\lvert\mathcal{U}^D\rvert}{\lvert\mathcal{E}^D\rvert}\%$	$\lvert\mathcal{A}^D\rvert$	$\frac{\lvert\mathcal{E}^D\rvert}{\lvert\mathcal{A}^D\rvert}\%$
1	4	4	100	11	36.4
2	20	28	71.4	121	23.1
3	107	260	41.2	1331	19.5
4	556	2 460	22.6	14 641	16.8
5	3 139	24 319	12.9	161 051	15.1
6	18 966	244 299	7.8	1 771 561	13.8
7	115 841	2 490 461	4.7	19 487 171	12.8

Table 2: Set of valid symbols depending on the current state. Symbols are classified into *C*onstants, *V*ariables, *U*nary operators and *B*inary operators

State	Valid symbols
$\lvert stack\rvert = 0$	C,V
$\lvert stack\rvert = 1$ & $d < D-1$	C,V,U
$\lvert stack\rvert = 1$ & $d = D-1$	U
$\lvert stack\rvert \in [2, D-d[$	C,V,U,B
$\lvert stack\rvert = D-d$	U,B
$\lvert stack\rvert = D-d+1$	B

correct expressions by relation $\sim$. One element $u \in \mathcal{U}$ is an equivalence class which contains semantically equivalent expressions $e \in u$.

We denote $\mathcal{A}^D$ (resp. $\mathcal{E}^D$ and $\mathcal{U}^D$) the set of sequences (resp. expressions and equivalence classes) of length D. Table 1 presents the cardinality of these sets for different lengths D with a hypothetical alphabet containing four variables, three unary operators and four binary operators: $\mathcal{A} = \{a, b, c, d, \log, \sqrt{\cdot}, inv, +, -, \times, \div\}$, where inv stands for inverse. It can be seen that both the ratio between $|\mathcal{E}^D|$ and $|\mathcal{A}^D|$ and the ratio between $|\mathcal{U}^D|$ and $|\mathcal{E}^D|$ decrease when increasing D. In other terms, when D gets larger, finding semantically correct and different expressions becomes harder and harder, which is an essential motivation of this work.

2.2 Generative process to sample expressions

Monte-Carlo search relies on a sequential generative process to sample expressions $e \in \mathcal{E}^D$. We denote by $P[\alpha_d|\alpha_1, \ldots, \alpha_{d-1}]$ the probability to sample symbol α_d after having sampled the (sub)sequence $\alpha_1, \ldots, \alpha_{d-1}$. The probability of an expression $e \in \mathcal{E}^D$ is then given by:

$$P_D[e] = \prod_{d=1}^{D} P[\alpha_d|\alpha_1, \ldots, \alpha_{d-1}]. \tag{2.1}$$

An easy way to exclude syntactically incorrect sequences is to forbid symbols that could lead to one of the two syntax errors described earlier. This leads to a set of conditions on the current state of the stack and on the current depth d that Table 2 summarizes for a problem

with variables, constants, unary and binary operators. As it can be seen, conditions can be grouped into a finite number of states, 6 in this case. In the following, we denote $\mathcal{S}$ the set of these states, we use the notation $s(\alpha_1, \ldots, \alpha_d) \in \mathcal{S}$ to refer to the current state reached after having evaluated $\alpha_1, \ldots, \alpha_d$ and we denote $\mathcal{A}_s \subset \mathcal{A}$ the set of symbols which are valid in state $s \in \mathcal{S}$.

The default choice when using Monte-Carlo search techniques consists in using a uniformly random policy. Combined with the conditions to generate only syntactically correct expressions, this corresponds to the following probability distribution:

$$P[\alpha_d|\alpha_1, \ldots, \alpha_{d-1}] = \begin{cases} \frac{1}{|\mathcal{A}_s|} & \text{if } \alpha_d \in \mathcal{A}_s \\ 0 & \text{otherwise,} \end{cases} \tag{2.2}$$

with $s = s(\alpha_1, \ldots, \alpha_{d-1})$.

Note that the sampling procedure described above generates expressions of size exactly D. If required, a simple trick can be used to generate expressions of size between 1 and D: it consists in using a unary *identity* operator that returns its operand with no modifications. An expression of size $d < D$ can then be generated by selecting $\alpha_{d+1} = \cdots = \alpha_D = identity$.

2.3 Problem statement

Using a uniformly random strategy to sample expressions does neither take into account redundancy nor semantic invalidity. We therefore propose to optimize the sampling strategy, to maximize the number of valid, semantically different, generated expressions.

Given a budget of T trials, a good sampling strategy should maximize S_T, the number of semantically different, valid generated expressions, i.e. the number of distinct elements drawn from $\mathcal{U}^D$. A simple approach therefore would be to use a rejection sampling algorithm. In order to sample an expression, such an approach would repeatedly use the uniformly random strategy until sampling a valid expression that differs from all previously sampled expressions in the sense of $\sim$. However, this would quickly be impractical: in order to sample T expressions, such an approach requires memorizing T expressions, which can quickly saturate memory. Furthermore, in regards of the results of Table 1, the number of trials required within the rejection sampling loop could quickly grow.

In order to avoid the excessive CPU and RAM requirements of rejection sampling, and consistently with the definitions given previously, we focus on a distribution $P_D[e]$ which is memoryless. In other terms, we want a procedure that generates i.i.d. expressions. In addition to the fact that it requires only limited CPU and RAM, such a memory-less sampling scheme has another crucial advantage: its implementation requires no communication, which makes it particularly adapted to (massively) parallelized algorithms.

In summary, given the alphabet $\mathcal{A}$, a target depth D and a target number of trials T, the problem addressed in this paper consists in finding the distribution $\hat{P}[\alpha_d|\alpha_1, \ldots, \alpha_{d-1}] \in \mathcal{P}$ such that:

$$\hat{P} = \underset{P \in \mathcal{P}}{\operatorname{argmax}}\, \mathsf{E}\{S_T\}, \tag{2.3}$$

where S_T is the number of semantically different valid expressions generated using the generative procedure $P_D[\cdot]$ defined by $\hat{P}$.

3 Proposed approach

We now describe our approach to learn a sampling strategy taking into account redundant and invalid expressions. Section 3.1 reformulates the problem and introduces two approximate objective functions with good numerical properties. Section 3.2 focuses on the case where the sampling strategy only depends on the current state of the stack and depth. Finally, the proposed projected gradient descent algorithm is described in Section 3.3.

3.1 Objective reformulation

Let $P_D[u]$ be the probability to draw any member of the equivalence class u:

$$P_D[u] = \sum_{e \in u} P_D[e] \tag{3.1}$$

The following lemma shows how to calculate the expectation of obtaining at least one member of a given equivalence class u after T trials.

Lemma 3.1. *Let X_u be a discrete random variable such that $X_u = 1$ if u is generated after T trials and $X_u = 0$ otherwise. The probability to generate at least once u over T trials is equal to*

$$\mathsf{E}\{X_u\} = 1 - (1 - P_D[u])^T. \tag{3.2}$$

Proof. Since at each trial the probability for $u \in \mathcal{U}$ to be generated does not depend on the previous trials, the probability over T trials that $X_u = 0$ is given by $(1 - P_D[u])^T$. Thus, the probability that $X_u = 1$ is its complementary and given by $1 - (1 - P_D[u])^T$. □

We now aggregate the different random variables.

Lemma 3.2. *The expectation of the number T_S of different equivalence classes u generated after T trials is equal to*

$$\mathsf{E}\{T_S\} = \sum_{u \in \mathcal{U}^D} 1 - (1 - P_D[u])^T. \tag{3.3}$$

Proof. This follows from $\mathsf{E}\{\sum_{u \in \mathcal{U}^D} X_u\} = \sum_{u \in \mathcal{U}^D} \mathsf{E}\{X_u\}$. □

Unfortunately, in the perspective of a using gradient descent optimization scheme, the formula given by Lemma 3.2 is numerically unstable. Typically, $P_D[u]$ is very small and the value $(1 - P_D[u])^T$ has a small number of significant digits. This causes numerical instabilities that become particularly problematic as T and $|\mathcal{U}|$ increase. Therefore, we have to look for an approximation of (3.3) that has better numerical properties.

Lemma 3.3. *For $0 < P_D[u] < \frac{1}{T}$, using the Newton Binomial Theorem to compute $1 - (1 - P_D[u])^T$, the terms are decreasing.*

Proof. Using the Newton Binomial Theorem, (3.1) reads

$$\begin{aligned} 1-(1-P_D[u])^T &= 1-\sum_{k=0}^{T}\binom{T}{k}(-P_D[u])^k \\ &= \binom{T}{1}(-P_D[u])+\binom{T}{2}(-P_D[u])^2 \\ &\quad +\cdots+(-P_D[u])^T. \end{aligned} \tag{3.4}$$

If $P_D[u]$ is sufficiently small, the first term in (3.4) is the biggest. In particular, if $P_D[u] < \frac{1}{T}$, we claim that $\binom{T}{0}P_D[u]^0 > \binom{T}{1}P_D[u]^1 > \binom{T}{2}P_D[u]^2 > \cdots > \binom{T}{n}P_D[u]^n$

As a proof, considering $n \in \{1, 2, \ldots, T\}$, we have to check that

$$\begin{aligned} &\binom{T}{n}P_D[u]^n < \binom{T}{n-1}P_D[u]^{n-1} \\ \Leftrightarrow\ &\frac{T!}{n!(T-n)!}P_D[u]^n < \frac{T!}{(n-1)!(T-n+1)!}P_D[u]^{n-1} \\ \Leftrightarrow\ &P_D[u] < \frac{n}{T-n+1} \end{aligned} \tag{3.5}$$

(3.5) holds when $0 < P_D[u] < \frac{1}{T}$, $n \geq 1$ and $T \geq 0$.

□

Observation 3.1. For $0 < P_D[u] < \frac{1}{\lambda T}$, two successive terms in the Newton Binomial Theorem decrease by a coefficient of $\frac{1}{\lambda}$.

Proof.

$$\begin{aligned} &\binom{T}{n}P_D[u]^n < \frac{1}{\lambda}\binom{T}{n-1}P_D[u]^{n-1} \\ &P_D[u] < \frac{n}{\lambda(T-n+1)} \end{aligned} \tag{3.6}$$

□

Figure 1 shows the shape of (3.2) for a fixed T of 1000. It appears that, for small values of $P_D[u]$, the expectation varies almost linearly as suggestd by Lemma 3.3. Observe that, for large values of $P_D[u]$, the expectation tends rapidly to 1.

Observation 3.2. Let $\epsilon > 0$. $P_D[u] > 1 - \epsilon^{\frac{1}{T}}$ implies that $1 - \epsilon \leq 1 - (1 - P_D[u])^T \leq 1$.

We observe from Figure 1 that the curve can be split into 3 rough pieces. The first piece, when $P_D[u] < \frac{1}{2T}$, seems to vary linearly based on Figure 1 and Observation 3.1 since the leading term of the Binomial expansion dominates all the others. The last piece can be approximated by 1 based upon Observation 3.2. The middle piece can be fitted by a logarithmic function. We therefore obtain

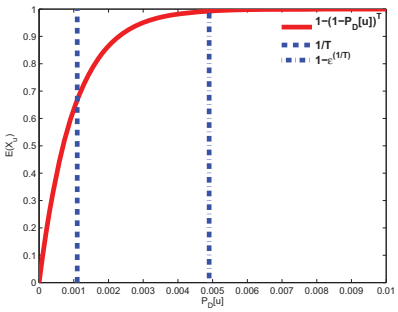

Figure 1: Expected value of X_u as a function of $P_D[u]$ with $T = 1000$.

$$\sum_{u \in \mathcal{U}} 1 - (1 - P_D[u])^T \simeq \sum_{u | P_D[u] < \frac{1}{2T}} TP_D[u] + \sum_{u | P_D[u] \geq 1 - \epsilon^{\frac{1}{T}}} 1 + \sum_{u | \frac{1}{2T} < P_D[u] < 1 - \epsilon^{\frac{1}{T}}} \left(\frac{1}{(1 - \frac{1}{T})^T} \log P_D[u] + (1 - \frac{1}{T})^T \right). \tag{3.7}$$

An even rougher approach could be to consider only two pieces and write (3.2), using Lemma 3.3 and Observation 3.2 as

$$\sum_{u \in \mathcal{U}} 1 - (1 - P_D[u])^T \simeq \sum_{u | P_D[u] < \frac{1}{T}} TP_D[u] + \sum_{u | P_D[u] \geq \frac{1}{T}} 1. \tag{3.8}$$

3.2 Instantiation and gradient computation

In this paper, we focus on a simple family of probability distributions with the assumption that the probability of a symbol only depends on the current state $s \in \mathcal{S}$ and is otherwise independent of the full history $\alpha_1, \ldots, \alpha_{d-1}$. We thus have:

$$P_D[e] = \prod_{d=1}^{D} P[\alpha_d | s(\alpha_1, \ldots, \alpha_{d-1})]. \tag{3.9}$$

We denote $\{p_{s,\alpha}\}$ the set of probabilities, such that for all $s \in \mathcal{S}$, for all $\alpha \in \mathcal{A}_s, P[\alpha|s] = p_{s,\alpha}$. Using (3.8), we can write the optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{u | P_D[u] < \frac{1}{T}} TP_D[u] + \sum_{u | P_D[u] \geq \frac{1}{T}} 1. \\ \text{subject to} \quad & \sum_{\alpha \in \mathcal{A}_s} p_{s,\alpha} = 1 \,, \forall\, s \in \mathcal{S}, && (3.10) \\ & p_{s,\alpha} > 0 \,, \forall s \in \mathcal{S}, \forall\, \alpha \in \mathcal{A}_s. && (3.11) \end{aligned}$$

where T is the total number of trials, $P_D[u]$ is the probability to generate the unique expression u, (3.10) ensures that the probabilities sum to 1 and (3.11) represents the non-negativity constraint.
Observe that the gradient is easy to compute for both objectives. We detail here the gradient when (3.8) is used as the objective function.

Lemma 3.4. *The derivative* $\frac{\partial(\cdot)}{\partial p_{s,\alpha}}$ *of Equation* (3.8) *is given by*

$$T \sum_{u|P_D[u]<\frac{1}{T}} \sum_{e \in u} n_{s,\alpha}(e) P_D[e],$$

where $n_{s,\alpha}(e)$ *is the number of times the symbol* $\alpha \in \mathcal{A}_s$ *is used from state* $s \in \mathcal{S}$ *when generating* e*.*

Proof.

$$\begin{aligned}
\frac{\partial f}{\partial p_{s,\alpha}} &= \frac{\partial}{\partial p_{s,\alpha}} \Big(\sum_{u|P_D[u]<\frac{1}{T}} T P_D[u] + \sum_{u|P_D[u]\geq\frac{1}{T}} 1 \Big) \\
&= \frac{\partial}{\partial p_{s,\alpha}} \Big(T \sum_{u|P_D[u]<\frac{1}{T}} \sum_{e \in u} P_D[e] \Big) \\
&= T \sum_{u|P_D[u]<\frac{1}{T}} \sum_{e \in u} \frac{\partial}{\partial p_{s,\alpha}} \Big(\prod_{s,\alpha|n_{s,\alpha}(e)>0} p_{s,\alpha}^{n_{s,\alpha}} \Big)
\end{aligned}$$

□

Note that this objective function is only locally convex. Figure 2 presents a problem with 3 variables projected on 2 dimensions. It shows that the problem is not convex. However, empirical experiments show that it is rather easy to optimize and that obtained solutions are robust w.r.t. to choice of the starting point of the gradient descent algorithm.

3.3 Proposed algorithm

We propose to use a classical projected gradient descent algorithm to solve the optimization problem described previously. The algorithm is equipped with a line search optimization based on the 2 Wolfe conditions. The algorithm stops when the next step in a given direction is smaller than ζ. In our case, we fixed ζ to 10^{-10}. Algorithm 2 depicts our approach. Given the symbol alphabet $\mathcal{A}$, the target depth D and the target number of trials T, the algorithm proceeds in two steps. First, it constructs an approximated set $\hat{\mathcal{U}}_D$ by discriminating the expressions on the basis of random samples of the input variables, following the procedure detailed below. It then applies projected gradient descent, starting from uniform $p_{s,\alpha}$ probabilities and iterating until the stopping condition is reached.
To evaluate approximately whether $e_1 \sim e_2$, we compare $e_1(x)$ and $e_2(x)$ using a finite amount X of samples $x \in \mathcal{X}$. If the results of both evaluations are equal on all the samples, then the expressions are considered as semantically equivalent. If the evaluation fails for any of the samples, the corresponding expression is considered as semantically incorrect and is rejected.

Algorithm 2 Symbol probabilities learning

Require: Alphabet $\mathcal{A}$,
Require: Target depth D,
Require: Target budget T,
Require: A set of input samples $x_1, \ldots, x_X$
  $\hat{\mathcal{U}} \leftarrow \emptyset$
  for $e \in \mathcal{E}^D$ **do**
    if $\forall i \in [1, X], e(x_i)$ is well-defined **then**
      Add e to $\hat{\mathcal{U}}$ using key $k = \{e(x_1), \ldots, e(x_X)\}$
    end if
  end for
  Initialize: $\forall s \in \mathcal{S}$, $\forall \alpha \in \mathcal{A}_s$, $p_{s,\alpha} \leftarrow \frac{1}{|\mathcal{A}_s|}$
  repeat
    $\forall s \in \mathcal{S}$, $\forall \alpha \in \mathcal{A}_s$, $g_{s,\alpha} \leftarrow 0$
    for each $u \in \hat{\mathcal{U}}$ with $P_D[u] < \frac{1}{T}$ **do**
      for each $e \in u$ **do**
        for each $s \in \mathcal{S}, \alpha \in \mathcal{A}_s$ with $n_{s,\alpha}(e) > 0$ **do**
          $g_{s,\alpha} \leftarrow g_{s,\alpha} + n_{s,\alpha}(e) P_D[e]$
        end for
      end for
    end for
    Apply gradient $g_{s,\alpha}$ and renormalize $p_{s,\alpha}$
  until some stopping conditions are reached
  return $\{p_{s,\alpha}\}$

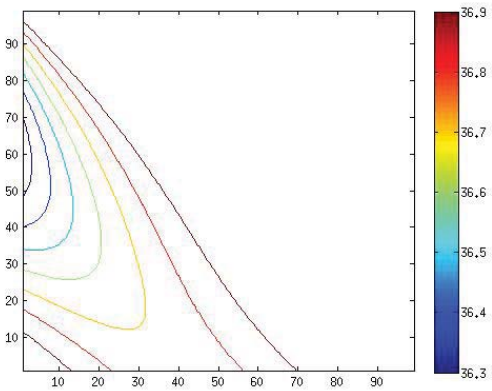

Figure 2: Optimization using (3.8) with 3 variables projected onto 2 dimensions where the y-axis and the x-axis represents different values of probability $(\%)$

Empirical tests showed that with as little as $X = 5$ samples, more than 99% of $\mathcal{U}$ was identified correctly for D ranging in $[1, 8]$. With $X = 100$ samples the procedure was almost perfect.
The complexity of Algorithm 2 is linear in the size of $\mathcal{E}^D$. In practice, only a few iterations of gradient descent are necessary to reach convergence and most of the computing time is taken by the construction of $g_{s,\alpha}$. The computation of the gradient is rather long since it requires that, for each unique expression u, we must iterate through each $e \in u$ and from each expression e, we take the partial derivative for each symbol $\alpha \in A$.
The requirement that the set $\mathcal{E}^D$ can be exhaustively enumerated is rather restrictive, since it limits the applicability of the algorithm to medium values of D. Nevertheless, we show in next section that probabilities learned for a medium value of D can be used for larger-scale problems and still significantly outperform uniform probabilities.

4 Experimental results

This section describes a set of experiments that aims at evaluating the efficiency of our approach. We distinguish between medium-scale problems where the set $\mathcal{E}^D$ is enumerable in reasonable time (Section 4.1) and large-scale problems where some form of generalization has to be used (Section 4.2). We rely on the same alphabet as previously: $\mathcal{A} = \{a, b, c, d, \log, \sqrt{\cdot}, inv, +, -, \times, \div\}$ and evaluate the various sampling strategies using empirical estimations of $\frac{\mathsf{E}\{S_T\}}{T}$ obtained by averaging S_T over 10^6 runs.
We consider two baselines in our experiments: *Syntactically Uniform* is the default strategy defined by Equation 2.2 and corresponds to the starting point of Algorithm 2. The *Semantic Uniform* baseline refers to a distribution where each expression $u \in \hat{\mathcal{U}}$ has an equal probability to be generated and corresponds to the best that can be achieved with a memory-less sampling procedure. *Objective 1* (resp. *Objective 2*) is our approach used with objective (3.7) (resp. objective (3.8)).

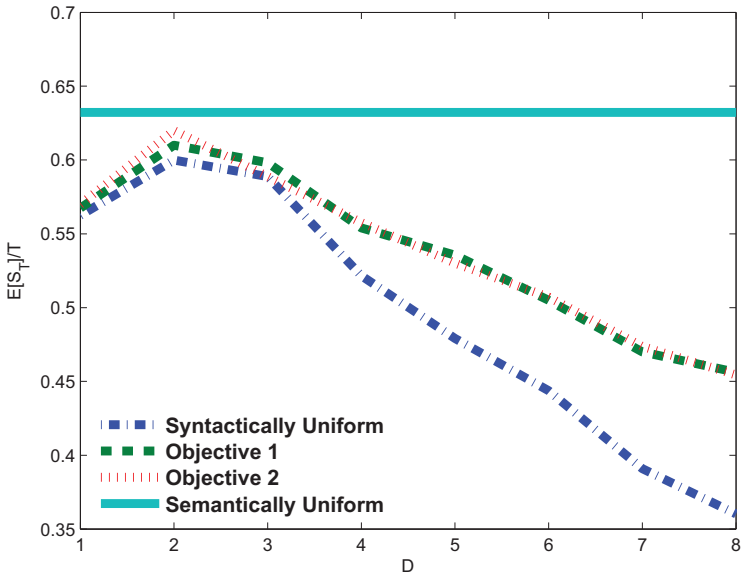

Figure 3: Ratio of $\frac{\mathsf{E}\{S_T\}}{T}$ for different lengths D with $T = |\mathcal{U}^D|$.

4.1 Medium-scale problems

We first carried out a set of experiments by evaluating the two baselines and the two learned strategies for different values of D. For each tested value of D, we launched the training procedure. Figure 3 presents the results of these experiments, in the case where the number of trials is equal to the number of semantically different valid expressions: $T = |\mathcal{U}^D|$.

It can be seen that sampling semantically different expressions is harder and harder as D gets larger, which is coherent with the results given in Table 1. We also observe that the deeper it goes, the larger the gap between the naive uniform sampling strategy and our learned strategies becomes. There is no clear advantage of using *Objective 1* over *Objective 2* for the approximation of (3.2). By default, we will thus use the simplest of the two in the following, which is *Objective 2*.

Many practical problems involve objective functions that are heavy to compute. In such cases, although the set $\mathcal{U}^D$ can be enumerated exhaustively, the optimization budget only enables to evaluate the objective function for a small fraction $T \ll |\mathcal{U}^D|$ of candidate expressions. We thus performed another set of experiments with $T = |\mathcal{U}^D|/100$, whose results are given by Figure 4. Since we have $T = 0$ for values $D < 3$, we only report results for $D \geq 3$. Note also that the small variations in *Semantically Uniform* comes from the rounding bias. The overall behavior is similar to that observed previously: the problem is harder and harder as D grows and our learned strategies still significantly outperform the *Syntactically Uniform* strategy. Note that all methods perform slightly better for $D = 8$ than for $D = 7$. One must bear in mind that beyond operators that allow commutativity, some operators have the effect to increase the probability

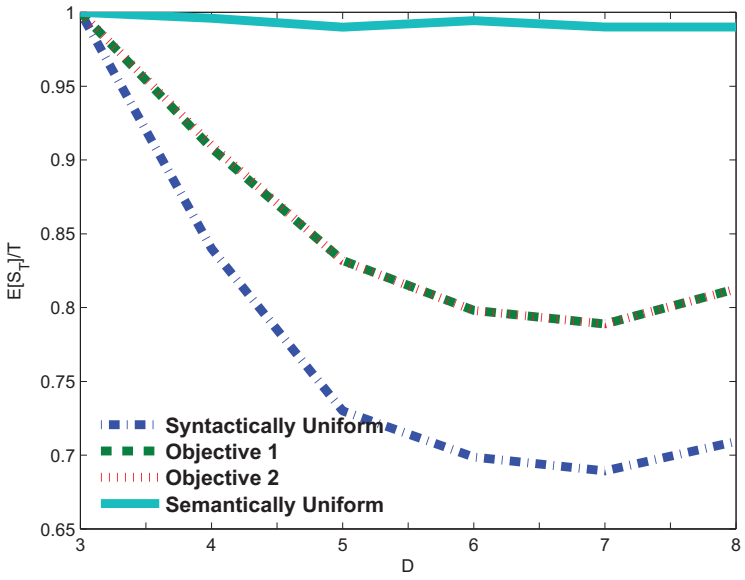

Figure 4: Ratio of $\frac{\mathsf{E}\{S_T\}}{T}$ for different lengths D with $T = |\mathcal{U}^D|/100$.

of generating semantically invalid expressions. For instance, one could think that subtractions should have a higher probability of being drawn than the additions or multiplications. However, when combined in a logarithm or a square root, the probability to generate an invalid expression increases greatly. The relation between the symbols is more convoluted then it looks at first sight, which may be part of the explanation of the behavior that we observe for $D = 8$.

4.2 Generalization towards large-scale problems

When D is large, the set of $\mathcal{E}^D$ may be hard to enumerate exhaustively and our learning algorithm becomes inapplicable. We now evaluate whether the information computed on smaller lengths can be used on larger problems. We performed a first set of experiments by targeting a length of $D_{eval} = 20$ with a number of trials $T_{eval} = 10^6$. Since our approach is not applicable with such large values of D, we performed training with a reduced length $D_{train} \ll D_{eval}$ and tried several values of T_{train} with the hope to compensate the length difference.

The results of these experiments are reported in Figure 5 and raise several observations. First, for a given D_{train}, the score $\frac{E\{S_{T_{eval}}\}}{T_{T_{eval}}}$ starts increasing with T_{train}, reaches a maximum and then drops rapidly to a score roughly equal to the one obtained by the *Syntactically Uniform* strategy. Second, the value of T_{train} for which this maximum occurs (T^*_{train}) always increases with D_{train}. Third, the best value T^*_{train} is always smaller than T_{eval}. Fourth, for any D_{train}, even for very small values of T_{train} the learned distribution already significantly outperforms the *Syntactically Uniform* strategy. Based on these observations, and given the fact that the complexity of the optimisation problem does not depend on T_{train}, we propose the following

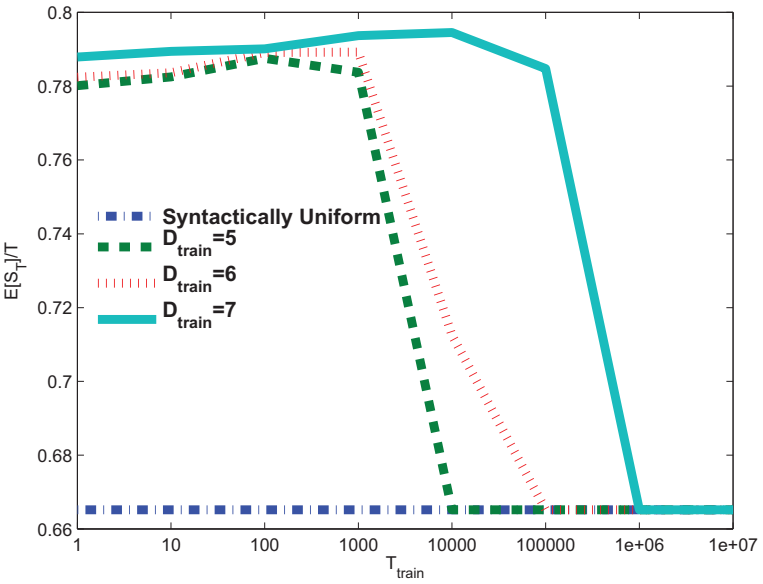

Figure 5: Ratio of $\frac{\mathsf{E}\{S_{T_{eval}}\}}{T_{eval}}$ for different values of D_{train} and T_{train} with $T_{eval} = 10^6$ and $D_{eval} = 20$.

approach to tune these two parameters: (i) choose the largest possible D_{train} value, (ii) find using a dichotomy search approach in $\{0, \ldots, T_{eval}\}$ the value of T_{train} that maximizes the target score.

Figure 6 reports for different values of D_{eval} the results obtained when assuming that D_{train} cannot be larger than 9 and when T_{train} has been optimized as mentioned above. T_{eval} is still here equal to 10^6. As we can see, even for large values of D_{eval}, the learned distribution significantly outperforms the *Syntactically Uniform* distribution, which clearly shows the interest of our approach even when dealing with very long expressions.

5 Application to symbolic regression

We have showed that our approach enables to improve the diversity of valid generated expressions when compared to a default random sampling strategy. This section aims at studying whether this improved diversity leads to better exploration of the search space in the context of optimization over expression spaces. We therefore focus on symbolic regression problems.

Symbolic regression problems. We use the same set of benchmark symbolic regression problems as in (Uy, Hoai, ONeill, McKay and Galván-López, 2011), which is described in Table 3. For each of problem, we generate a training set by taking regularly spaced input points in the domain indicated in the "Examples" column. The alphabet is $\{x, 1, +, -, *, /, \sin, \cos, \log, \exp\}$ for single variable problems ($\{f1, \ldots, f8\}$) and $\{x, y, +, -, *, /, \sin, \cos, \log, \exp\}$ for bivariable problems.

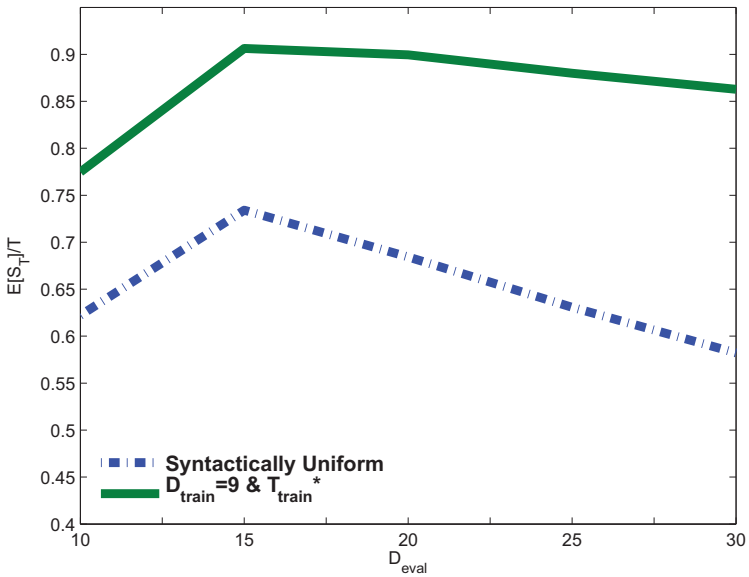

Figure 6: Ratio of $\frac{\mathsf{E}\{S_{T_{eval}}\}}{T_{eval}}$ for different evaluation length D_{eval} with $T_{eval} = 10^6$.

Search algorithms. We focus on two search algorithms: random search and the recently proposed Nested Monte-Carlo (NMC) search algorithm. The former algorithm is directly related to our sampling strategy, while the latter is relevant since it has recently been applied with success to expression discovery (Cazenave, 2010). NMC is a search algorithm constructed recursively. *Level 0* NMC is equivalent to random search. *Level N* NMC selects symbols $\alpha_d \in \mathcal{A}_s$ by running the *level N-1* algorithm for each candidate symbol and by picking the symbols that lead to the best solutions discovered so far. We refer the reader to (Cazenave, 2009) for more details on this procedure.

Protocol. We compare random search and level $\{1, 2\}$ NMC using for each algorithm two different sampling strategies: the syntactically uniform strategy (denoted U) and our learned sampling strategy (denoted L). We use two different maximal lengths D, one for which the optimization problem can be solved exactly (depth 8) and one for which it cannot (depth 20) and for which we use the procedure described in Section 4.2. Note that since we use only two different alphabets and two different depths, we only had to perform our optimization procedure four times for all the experiments[1]. The quality of a solution is measured using the mean absolute error and we focus on the number of function evaluations which is required to reach a solution whose score is lower than a given threshold $\epsilon \geq 0$. We test each algorithm on 100 different runs.

Results. Table 4 and Table 5 summarize the results by showing the median number of

[1] More generally, when one has to solve several similar optimization problems, our preprocessing has only to be performed once.

Name	Function	Examples
f1	$x^3 + x^2 + x$	20 points $\in [-1, 1]$
f2	$x^4 + x^3 + x^2 + x$	20 points $\in [-1, 1]$
f3	$x^5 + x^4 + x^3 + x^2 + x$	20 points $\in [-1, 1]$
f4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	20 points $\in [-1, 1]$
f5	$\sin(x^2)\cos(x) - 1$	20 points $\in [-1, 1]$
f6	$\sin(x) + \sin(x + x^2)$	20 points $\in [-1, 1]$
f7	$\log(x + 1) + \log(x^2 + 1)$	20 points $\in [0, 2]$
f8	$\sqrt{x}$	20 points $\in [0, 4]$
f9	$\sin(x) + \sin(y^2)$	100 points $\in [-1, 1] \times [-1, 1]$
f10	$2\sin(x)\cos(y)$	100 points $\in [-1, 1] \times [-1, 1]$

Table 3: Description of the benchmark symbolic regression problems.

evaluations it takes to find an expression whose score is better than $\epsilon = 0.5$ and $\epsilon = 0.1$, respectively. We also display the mean of these median number of evaluations averaged over all 10 problems and the ratio of this quantity with uniform sampling over this quantity with our improved sampling procedure.

We observe that in most cases, our sampling strategy enables to significantly reduce the required number of function evaluations to reach the same level of solution quality. The amount of reduction is the largest when considering expressions of depth 20, which can be explained by the observation made in Section 2.1: when the depth increases, it is harder and harder to sample semantically different valid expressions. The highest improvement is obtained with depth 20 random search: the ratio between the traditional approach and our approach is of 1.32 and 1.17 for $\epsilon = 0.5$ and $\epsilon = 0.1$, respectively. We observe that the improvements tend to be more important with random search and with level 1 NMC than with level 2 NMC. This is probably related to the fact that the higher the level of NMC is, the more effect the bias mechanism embedded in NMC has; hence reducing the effect of our sampling strategy.

6 Conclusion

In this paper, we have proposed an approach to learn a distribution for expressions written in reverse polish notation, with the aim to maximize the expected number of semantically different, valid, generated expressions. We have empirically tested our approach and have shown that the number of such generated expressions can significantly be improved when compared to the default uniform sampling strategy. It also improves the exploration strategy of random search and nested Monte-Carlo search applied to symbolic regression problems.

A possible extension of this work would be to consider richer distributions making use of the whole history through the use of a general feature function. Moreover, instead of optimizing over one set of expressions, clustering into several sets could further improve the sampling process. The generalization to several clusters is not trivial since the concept of distance in this case can either be related to syntax or to semantic. Another extension is to compare

	Depth 8 (Depth 20)					
Pr.	Random		NMC(1)		NMC(2)	
-	U	L	U	L	U	L
f1	**5**(7)	6(**5**)	6(8)	**5**(**6**)	6(**6**)	6(7)
f2	127(193)	**95**(**113**)	151(210)	**89**(**97**)	110(167)	**98**(**97**)
f3	317(385)	**183**(**179**)	331(323)	**149**(**203**)	**143**(236)	157(**203**)
f4	345(463)	**272**(**365**)	270(493)	**194**(**318**)	**205**(**247**)	262(278)
f5	21(28)	**13**(**19**)	22(24)	**15**(**19**)	18(19)	**17**(**12**)
f6	**5**(**5**)	6(6)	6(6)	6(**5**)	**6**(7)	7(**5**)
f7	7(9)	**6**(**7**)	10(10)	**8**(**7**)	7(**6**)	7(7)
f8	39(70)	**18**(**46**)	28(49)	**21**(**38**)	26(40)	**22**(**32**)
f9	4(5)	**3**(**4**)	4(6)	4(**3**)	4(4)	4(4)
f10	6(5)	**5**(5)	8(8)	**4**(**6**)	5(5)	5(5)
Mean	23.3(29.7)	**18.0**(**22.0**)	25.0(31.3)	**17.8**(**21.1**)	19.4(22.8)	**19.7**(**20.0**)
Ratio	1.3(1.4)		1.4(1.4)		1.0(1.1)	

Table 4: Median number of iterations it takes to reach a solution whose score is less than $\epsilon = 0.5$ with random search and level $\{1, 2\}$ nested Monte-Carlo search. Bold indicates cases where our sampling strategy outperforms the syntactically uniform strategy. The mean represents the geometric mean.

and evaluate a wide range of techniques for the optimization problem, from derivative free algorithms to stochastic gradient descent algorithms (e.g. (Omranpour, Ebadzadeh, Shiry and Barzegar, 2012)).

Acknowledgment

This paper presents research results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office. The scientific responsibility rests with its authors.

References

Cazenave, T. 2009. Nested monte-carlo search, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, Pasadena (CA) USA, pp. 456–461.

Cazenave, T. 2010. Nested Monte-Carlo Expression Discovery, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'2010)*, IOS Press, pp. 1057–1058.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii, *IEEE Transactions on Evolutionary Computation* **6**(2): 182–197.

Holland, J. 1992. Genetic algorithms, *Scientific American* **267**(1): 66–72.

Hu, Y. and Yang, S. 2004. A knowledge based genetic algorithm for path planning of a mobile robot, *Proceedings of the IEEE International Conference on Robotics and Automation(ICRA'04)*, Vol. 5, New Orleans (LA), USA, pp. 4350–4355.

	Depth 8 (Depth 20)					
Pr.	Random		NMC(1)		NMC(2)	
-	U	L	U	L	U	L
f1	37(53)	**28**(**(**19)	29(23)	**17**(**8**)	8(5)	**5**(**4**)
f2	18(89)	**12**(**38**)	13(25)	**6**(**18**)	7(10)	**5**(10)
f3	37(127)	**21**(**66**)	47(67)	**17**(**34**)	8(**16**)	8(17)
f4	**19**(302)	30(**140**)	**17**(127)	20(**72**)	16(**26**)	16(27)
f5	.8(2)	**.5**(**.7**)	.5(**.6**)	.5(.7)	.5(.8)	**.4**(**.6**)
f6	26(38)	**9**(**11**)	16(18)	**12**(**6**)	8(6)	**7**(6)
f7	2(6)	**.7**(**2**)	2(2)	**.7**(**1**)	**.5**(2)	.6(**1**)
f8	36(41)	**22**(**28**)	20(14)	**15**(**11**)	13(**6**)	**8**(8)
f9	.8(2)	**.4**(**.9**)	.5(1)	**.3**(**.8**)	.4(1)	**.3**(1)
f10	.6(1)	**.3**(**.6**)	.7(1)	**.4**(**.5**)	.3(.5)	**.2**(.5)
Mean	7.1(19.2)	**4.2**(**8.3**)	5.7(8.3)	**3.5**(**4.8**)	2.7(3.8)	**2.2**(**3.5**)
Ratio	1.7(2.3)		1.6(1.7)		1.2(1.1)	

Table 5: Median number of thousands of iterations it takes to reach a solution whose score is less than $\epsilon = 0.1$ with random search and level $\{1, 2\}$ nested Monte-Carlo search. The results are expressed in thousands (k) for the sake of readability.

Jones, G., Willett, P., Glen, R., Leach, A. and Taylor, R. 1997. Development and validation of a genetic algorithm for flexible docking, *Journal of Molecular Biology* **267**(3): 727–748.

Kargupta, H. and Buescher, K. 1996. The gene expression messy genetic algorithm for financial applications, *Proceedings of the IEEE International Conference on Evolutionary Computation*, Nagoya, JPN, pp. 814–819.

Kikuchi, S., Tominaga, D., Arita, M., Takahashi, K. and Tomita, M. 2003. Dynamic modeling of genetic networks using genetic algorithm and s-system, *Bioinformatics* **19**(5): 643–650.

Koza, J. and Poli, R. 2005. Genetic programming, *Search Methodologies* pp. 127–164.

Maulik, U. and Bandyopadhyay, S. 2000. Genetic algorithm-based clustering technique, *Pattern Recognition* **33**(9): 1455–1465.

Omranpour, H., Ebadzadeh, M., Shiry, S. and Barzegar, S. 2012. Dynamic particle swarm optimization for multimodal function, *International Journal of Artificial Intelligence (IJAI)* **1**(1): 1–10.

O'Neill, M. and Ryan, C. 2001. Grammatical evolution, *Evolutionary Computation, IEEE Transactions on* **5**(4): 349–358.

Uy, N., Hoai, N., ONeill, M., McKay, R. and Galván-López, E. 2011. Semantically-based crossover in genetic programming: application to real-valued symbolic regression, *Genetic Programming and Evolvable Machines* **12**(2): 91–119.