Running Programs Backwards

Instruction Inversion for Effective Search in Semantic Spaces

Bartosz Wieloch Krzysztof Krawiec

Institute of Computing Science
Poznan University of Technology
Poland

8.07.2013

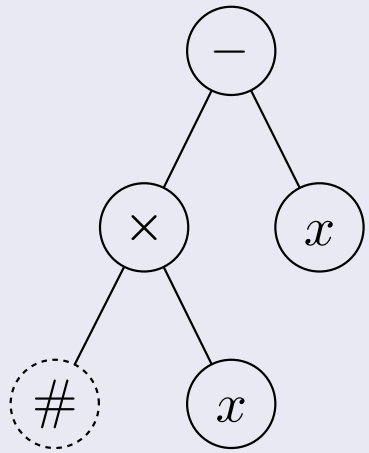
Preliminaries

- canonical tree-based genetic programming
  (can be adopted to Linear GP, Cartesian GP)
- subprograms (subtrees) can be independently executed
- subtrees can be freely replaced by other subtrees
- fitness calculation is based on a set of fitness cases

General Idea

Problem decomposition

- Let us assume, that we already have **almost correct** program for a given task, i.e.:
  - a fragment of an ideal solution (context), but
  - incorrect subprogram (subtree).
- We need to find the proper subprogram and replace the incorrect one (should be easier).



Question 1

How to get the almost correct program?

Question 2

How to find the proper subprogram for a given context?

Question 1

Question 1

How to get the almost correct program?

Answer

We do not know... ☹

Surrogate

Suppose that any random context could potentially belong to some ideal solution (sometimes true).

Question 1

Question 1

How to get the almost correct program?

Answer

We do not know... ☹

Surrogate

Suppose that any random context could potentially belong to some ideal solution (sometimes true).

Question 1

Question 1

How to get the almost correct program?

Answer

We do not know... ☹

Surrogate

Suppose that any random context could potentially belong to some ideal solution (sometimes true).

Question 2

Question 2

How to find the proper subprogram for a given context?

Several possibilities

- Use any standard metaheuristic (e.g. GP).
- Evaluate a subprogram by combining it with the context (analyze behavior of the entire program).

Our proposition

1. Calculate the desired behavior of the sought subprogram (this determines a new subtask).
2. Solve this subtask by an exhaustive search in the current population.

Question 2

Question 2

How to find the proper subprogram for a given context?

Several possibilities

- Use any standard metaheuristic (e.g. GP).
- Evaluate a subprogram by combining it with the context (analyze behavior of the entire program).

Our proposition

1. Calculate the desired behavior of the sought subprogram (this determines a new subtask).
2. Solve this subtask by an exhaustive search in the current population.

Question 2

Question 2

How to find the proper subprogram for a given context?

Several possibilities

- Use any standard metaheuristic (e.g. GP).
- Evaluate a subprogram by combining it with the context (analyze behavior of the entire program).

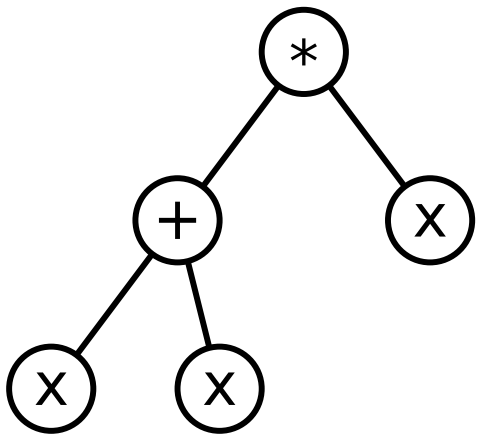
Our proposition

1. Calculate the desired behavior of the sought subprogram (this determines a new subtask).
2. Solve this subtask by an exhaustive search in the current population.

Semantics of Program

Semantics

- In general: Description of what a program does, i.e. what are the *effects* of execution of an entire program or its constituent components.
- In GP: a *list of outputs* that are actually produced by a program for all training examples (fitness cases).



x	result
-0.5	**0.5**
1.0	**2.0**
1.5	**4.5**
2.0	**8.0**

semantics=**[0.5, 2.0, 4.5, 8.0]**

Desired Behavior

Target semantics

Desired behavior of a whole program (given by task definition)

Desired semantics (of a context)

Desired behavior of a subprogram that will be composed with the context

Proper subprogram + context = ideal solution

Composition of the context and any subprogram with the appropriate desired semantics will give a program with the target semantics.

Calculating the Desired Semantics

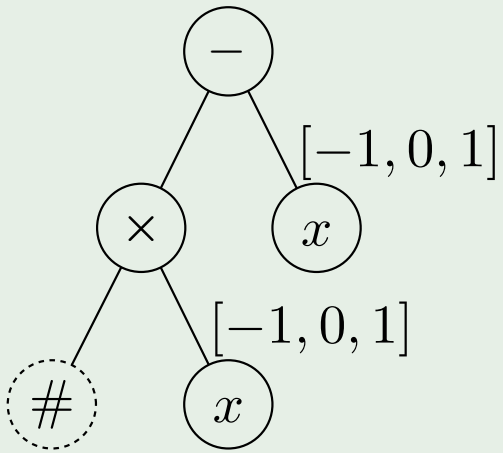
Invertible instructions, e.g.:

- $\# - x = y \quad \Longrightarrow \quad \# = y + x$
- $\# \times x = y \quad \Longrightarrow \quad \# = y / x$

Example

Fitness cases:

- inputs values: $x = [-1, 0, 1]$
- target values: $t = [2, 0, 0]$



Calculating the Desired Semantics

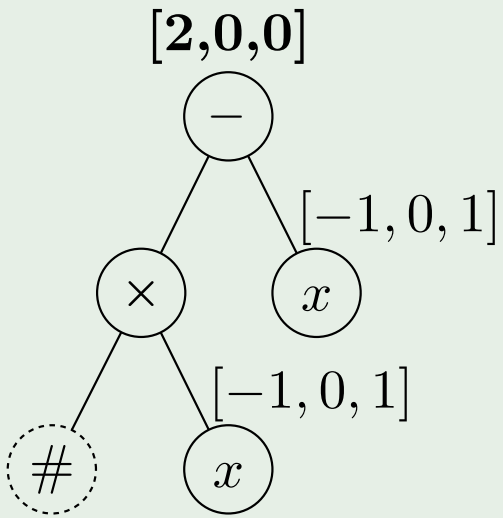
Invertible instructions, e.g.:

- $\# - x = y \quad \Longrightarrow \quad \# = y + x$
- $\# \times x = y \quad \Longrightarrow \quad \# = y / x$

Example

Fitness cases:

- inputs values: $x = [-1, 0, 1]$
- target values: $t = [2, 0, 0]$



Calculating the Desired Semantics

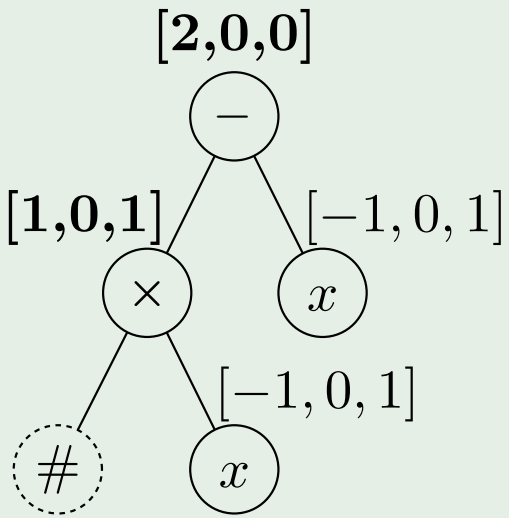
Invertible instructions, e.g.:

- $\# - x = y \quad \Longrightarrow \quad \# = y + x$
- $\# \times x = y \quad \Longrightarrow \quad \# = y / x$

Example

Fitness cases:

- inputs values: $x = [-1, 0, 1]$
- target values: $t = [2, 0, 0]$



Calculating the Desired Semantics

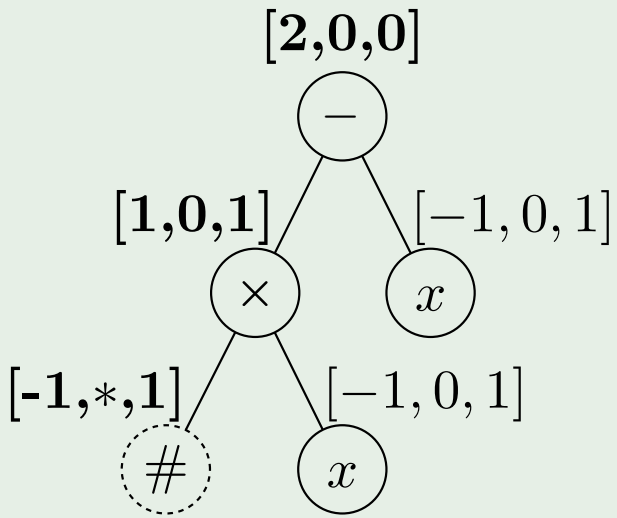
Invertible instructions, e.g.:

- $\# - x = y \quad \Longrightarrow \quad \# = y + x$
- $\# \times x = y \quad \Longrightarrow \quad \# = y / x$

Example

Fitness cases:

- inputs values: $x = [-1, 0, 1]$
- target values: $t = [2, 0, 0]$

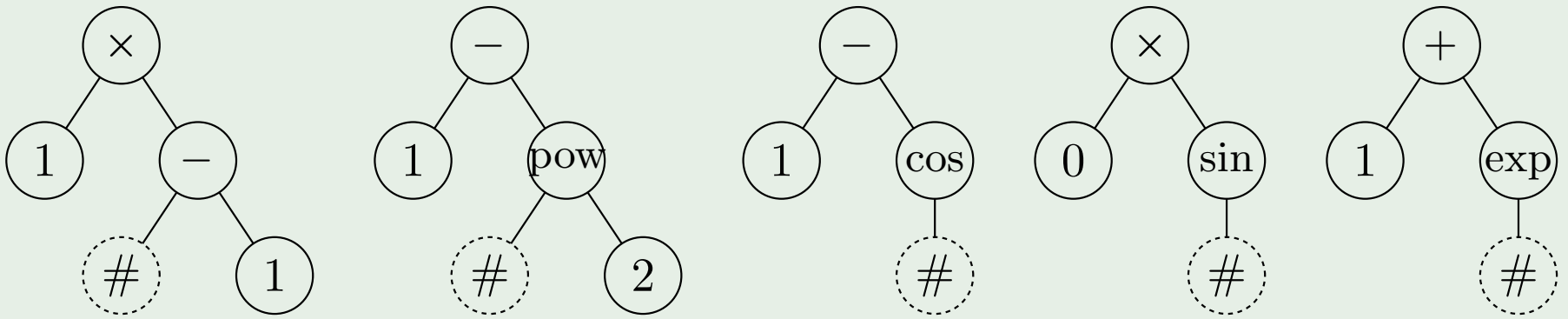


Possible Situations

For each component of desired semantics acceptable is:

1. Exactly one value.
2. Finite number of values.
3. Infinite number of values.
4. Any value ('don't care') — insignificant.
5. No value — inconsistent.

Example — target value: 0



Random Desired Operator (RDO)

1. Select random node in the parent program (determine the context).
2. Calculate desired semantics of this context.
3. Search best match in subtrees extracted from individuals in the whole population.
4. Replace old subtree with the best matching.

Benchmark Suite — Symbolic Regression Problems

	Target program (expression)	Variables	Range
F03	$x^5 + x^4 + x^3 + x^2 + x$	1	$[-1; 1]$
F04	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	1	$[-1; 1]$
F05	$\sin(x^2)\cos(x) - 1$	1	$[-1; 1]$
F06	$\sin(x) + \sin(x + x^2)$	1	$[-1; 1]$
F07	$\log(x + 1) + \log(x^2 + 1)$	1	$[0; 2]$
F08	$\sqrt{x}$	1	$[0; 4]$
F09	$\sin(x) + \sin(y^2)$	2	$[0.01; 0.99]$
F10	$2\sin(x)\cos(y)$	2	$[0.01; 0.99]$
F11	x^y	2	$[0.01; 0.99]$
F12	$x^4 - x^3 + y^2/2 - y$	2	$[0.01; 0.99]$

- instructions: $+$, $-$, $\times$, / (protected), sin, cos, exp, log (protected)
- 20 or 100 fitness cases
- success: error for each fitness cases less than $1.11 \cdot 10^{-15}$

Benchmark Suite — Boolean Problems

Problem	Instance	Bits	Fitness cases
even parity	PAR4	4	16
	PAR5	5	32
	PAR6	6	64
multiplexer	MUX6	6	64
	MUX11	11	2048
majority	MAJ5	5	32
	MAJ6	6	64
	MAJ7	7	128
comparator	CMP6	6	64
	CMP8	8	256

- instructions: AND, OR, NAND, and NOR.
- success: perfect reproduction

Experiment

Parameter	Value
Generations	100
Population size	500
Selection method	Tournament
Tournament size	3
Operators	X (crossover), M (mutation), RDO
Setups	X+M X+RDO M+RDO
Operators probability	Varying from 0 to 1 with step 0.1
Number of runs	200

Results

Setup	Rank	Setup	Rank
M+RDO 0.7	**8.63**	X+RDO 0.2	11.70
M+RDO 0.3	8.78	**RDO 1.0**	**13.40**
X+RDO 0.5	**8.90**	X+RDO 0.1	14.28
M+RDO 0.5	9.15	M+RDO 0.1	14.58
X+RDO 0.4	9.20	**X 1.0**	**20.55**
X+RDO 0.8	9.23	**X+M 0.1**	**21.30**
M+RDO 0.4	9.25	X+M 0.2	22.53
X+RDO 0.6	9.75	X+M 0.3	23.10
M+RDO 0.6	9.88	X+M 0.4	23.55
X+RDO 0.3	9.95	X+M 0.5	23.85
X+RDO 0.7	9.95	X+M 0.6	24.53
M+RDO 0.8	10.08	X+M 0.7	25.73
M+RDO 0.2	10.65	X+M 0.8	25.85
X+RDO 0.9	11.15	**M 1.0**	**27.18**
M+RDO 0.9	11.20	X+M 0.9	27.18

Setup	Rank	Setup	Rank
M+RDO 0.7	**8.83**	X+RDO 0.2	12.45
M+RDO 0.6	8.98	X+RDO 0.1	12.63
M+RDO 0.5	9.00	**RDO 1.0**	**13.18**
M+RDO 0.4	9.35	M+RDO 0.1	13.25
M+RDO 0.8	9.38	**X 1.0**	**20.70**
X+RDO 0.7	**9.75**	**X+M 0.1**	**20.70**
M+RDO 0.3	9.78	X+M 0.2	20.85
X+RDO 0.8	10.05	X+M 0.3	22.25
X+RDO 0.6	10.18	X+M 0.4	22.53
X+RDO 0.5	10.33	X+M 0.5	23.45
X+RDO 0.4	10.35	X+M 0.6	23.93
X+RDO 0.3	10.53	X+M 0.7	25.90
M+RDO 0.9	11.08	X+M 0.8	25.95
M+RDO 0.2	11.50	X+M 0.9	26.85
X+RDO 0.9	11.73	**M 1.0**	**29.63**

Conclusions

- RDO generally improves search performance.
- RDO solves Boolean problems almost perfectly.
- Choosing appropriate probability of RDO in not crucial.
- Our approach is applicable not only to evolutionary metaheuristics.

Exploiting additional known properties of problem definition (here: instruction inversion) may be very advantageous.

Thank you