Geometric Semantic Genetic Programming (GSGP): theory-laden design of variation operators

Andrea Mambrini

University of Birmingham, UK
NICaiA Exchange Programme

LaMDA group, Nanjing University, China
7th March 2014

Genetic Programming

Genetic programming is an evolutionary algorithm-based methodology to evolve functions (or computer programs).

- Individuals are functions represented for example as parse trees
- Mutation operators traditionally perform operations on the syntax of the trees
- The fitness function measures the input-output behaviour (semantic) of the function, usually comparing it with a target behaviour that is aimed to be reached.

Research question

How can we get **guarantees** on the convergence time of genetic programming (GP)?

Black-box boolean function learning

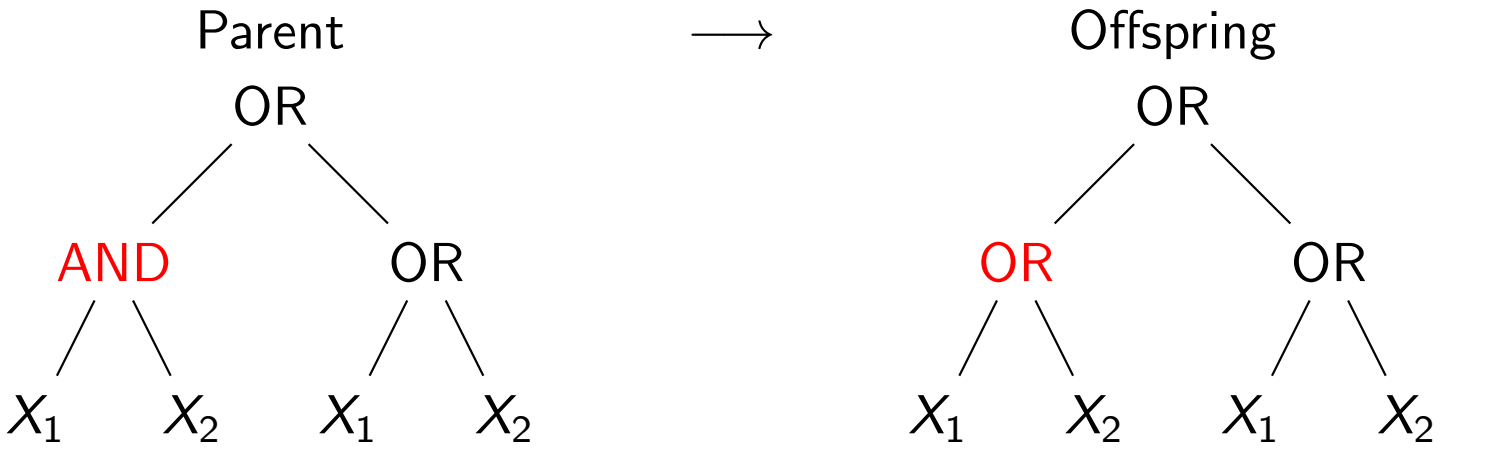
Which boolean expression produces the required output behaviour?

X_1	X_2	Output
1	1	1
1	0	0
0	1	1
0	0	0

Black-box setting:

- We don't have access to the truth table
- We can give our candidate function $f(X_1, X_2)$ to an oracle that will answer us the number of matched output rows.

Traditional Genetic Programming

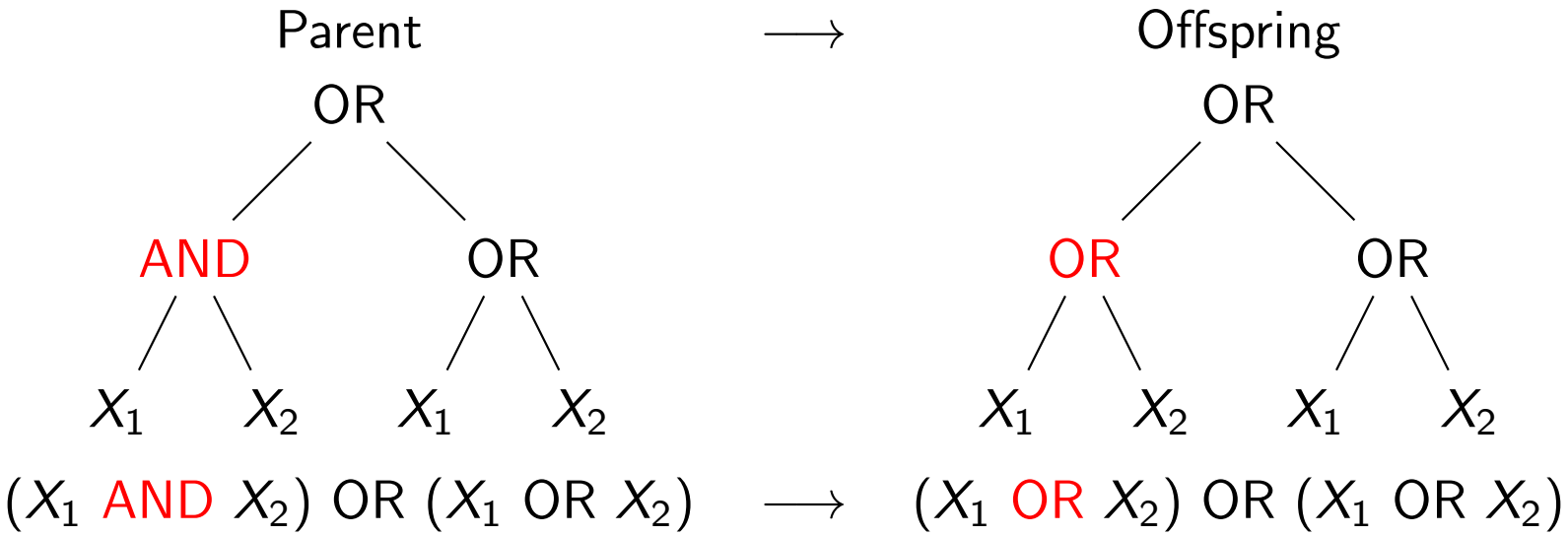


$(X_1 \text{ AND } X_2) \text{ OR } (X_1 \text{ OR } X_2) \longrightarrow (X_1 \text{ OR } X_2) \text{ OR } (X_1 \text{ OR } X_2)$

- Expressions are represented by trees.
- Variation operators produce offspring by syntactic manipulation of parent trees

X_1	X_2	Target Output	Parent's Output
1	1	**1**	1
1	0	**0**	1
0	1	**1**	1
0	0	**0**	0
Fitness			3

Traditional Genetic Programming

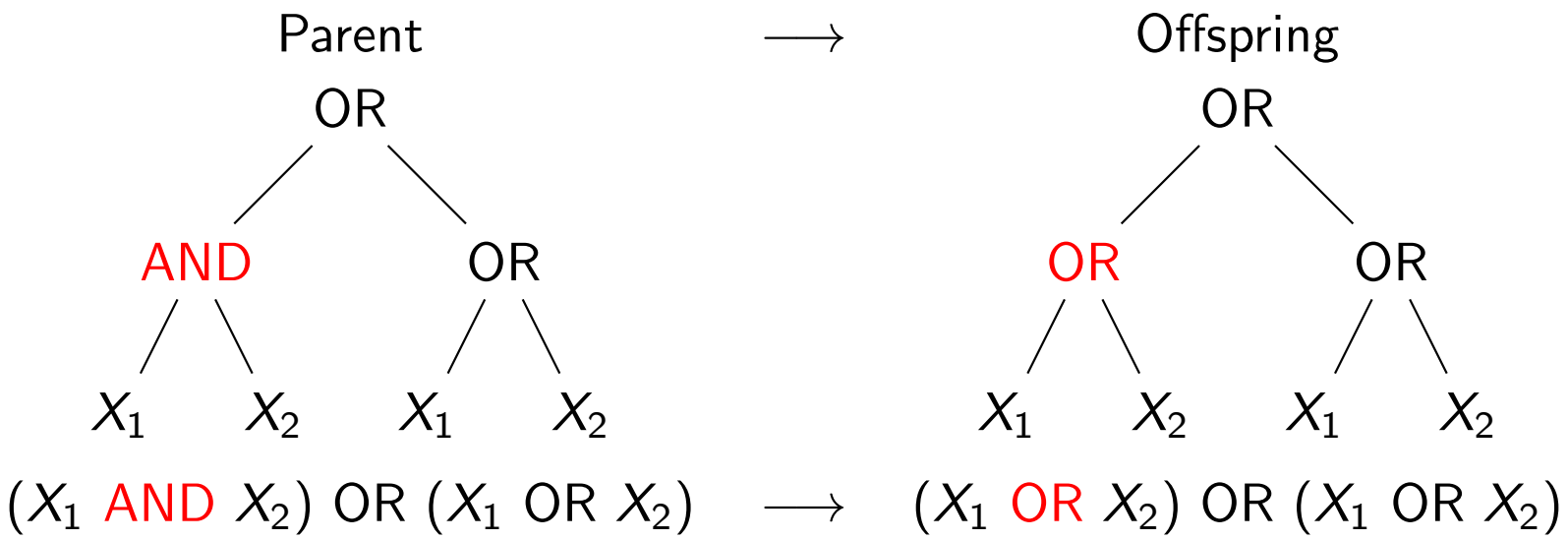


- Expressions are represented by trees.
- Variation operators produce offspring by syntactic manipulation of parent trees

Why GP is hard to analyse (and thus to efficiently design)?

- The fitness depends on the behaviour (semantic) of the function to evolve, while variation operators act on the syntax.
- How a syntactic variation operator affects the semantic of the expression?

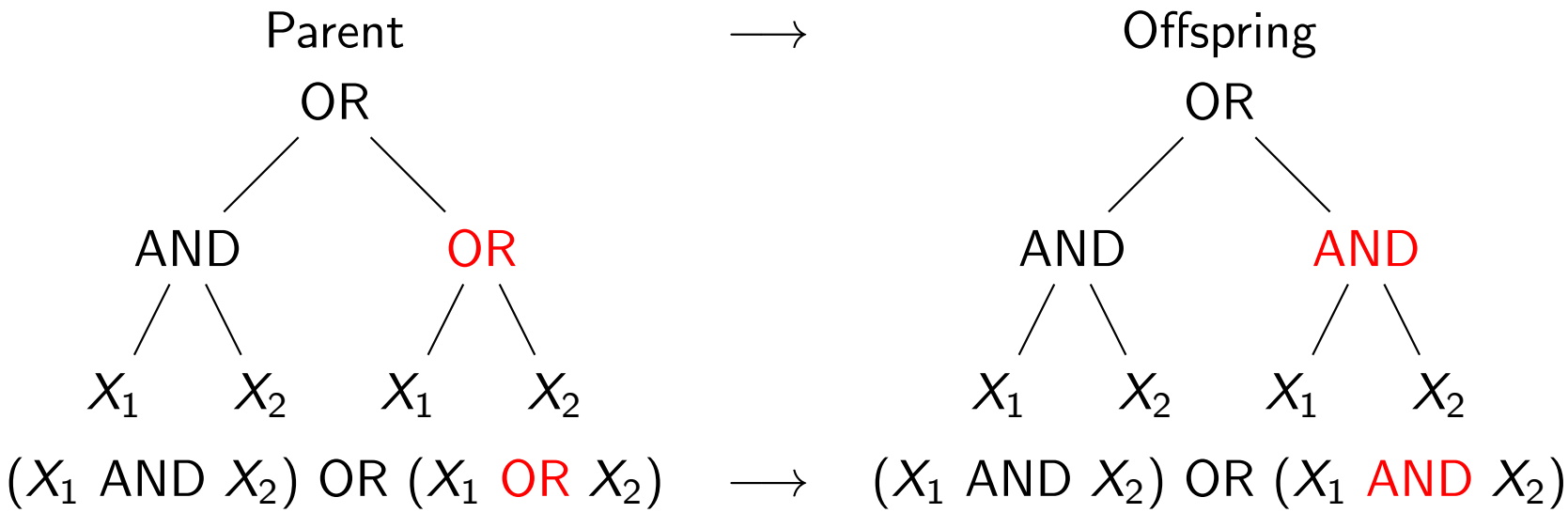
Traditional Genetic Programming



X_1	X_2	Output	Parent	Offspring
1	1	1	1	1
1	0	0	1	1
0	1	1	1	1
0	0	0	0	0

In this case the variation operator **hasn't produced any change** in the semantic of the function.

Traditional Genetic Programming



X_1	X_2	Output	Parent	Offspring
1	1	1	1	1
1	0	0	1	0
0	1	1	1	0
0	0	0	0	0

In this case it has produced a **big changes** on the truth table.

OneMax Problem

- box-constrained binary problem
- fitness of a solution (to maximise): number of matching bits with a particular fixed bit-string (traditionally a string consisting of all 1s)

Example

- Size of the problem n=5
- Target string $\overline{X} = 11111$
- fitness of an individual X: $\sum_{i=1}^{5} 1 - HD(x_i, \overline{x}_i)$ (e.g. $X = 10110$ has fitness equal to 3).

RLS Algorithm

Algorithm - Random Local Search (RLS)

1. Inizialise P_0 with a bitstring $x \in \{0,1\}^n$
2. Create the offspring x' by flipping one bit at random
3. Select the fittest between x' and x for survival
4. Repeat from point 2 until stopping condition applies

Runtime of RLS on OneMax:

- An individual with fitness k (thus having k 1s) is mutated to a better individual if one zero is flipped. This happens with probability $p = \frac{n-k}{n}$.
- The expected time to get to the optimal solution is thus: $E[T] = \sum_{k=1}^{n} \frac{1}{p} = \sum_{k=1}^{n} \frac{n}{n-k} = O(n \log n)$.

Why so easy? Because it is clear how a mutation affect the fitness.

Can we do the same with GP?

Semantic Genetic programming

Semantic variation operator

- acts on the syntax of an expression (given a parent expression produce as offspring another expression)
- **guarantee** some semantic features for the offspring (e.g. a semantic mutation might produce an offspring whose truth table always differs just by one row from the parent)

How to implement a semantic operator?

By trial & error use standard operators and reject offspring that do not conform to the semantic requirement → wasteful

By construction[1] : Operate on the syntax of the expression in a way that by construction the semantic criterion is satisfied

[1] A. Moraglio, K. Krawiec, C. Johnson – *Geometric Semantic Genetic Programming* – 5th Workshop on Theory of Randomized Search Heuristics, 2011

A semantic mutation operator[2]

Definition

Bit-flip point mutation: Given a parent function $P : \{0,1\}^n \to \{0,1\}$ the mutation returns the offspring boolean function $P' = \left(P \wedge \overline{M}\right) \vee \left(M \wedge \overline{P}\right)$, where M is a random minterm of all input variables.

Random minterm: $\mathrm{M} = X_1 \wedge \overline{X_2}$

Example: Parent: $\mathrm{P} = (X_1 \wedge X_2) \vee (X_1 \vee X_2)$

Offspring: $O = (P \wedge \overline{M}) \vee (\overline{P} \wedge M)$

X_1	X_2	Output	M	Parent	Offspring
1	1	1	0	1	1
1	0	0	1	1	0
0	1	1	0	1	1
0	0	0	0	0	0

[2] A. Moraglio, A. Mambrini, and L. Manzoni – *Runtime Analysis of Mutation-Based Geometric Semantic Genetic Programming on Boolean Functions* – Foundations of Genetic Algorithms (FOGA 2013)

Semantic Bit-flip point mutation

X_1	X_2	Output	M	Parent	Offspring
1	1	1	0	1	1
1	0	0	1	1	0
0	1	1	0	1	1
0	0	0	0	0	0

Effect on the output vector (semantic):

- Select a row at random (randomly generating M)
- Flip the output of the function on that row (the row where M is true)

Basically at each generation we flip one bit from a bitstring (the output vector) aiming to reach a desired bit configuration...

that reminds RLS on OneMax!

Runtime

Search equivalence

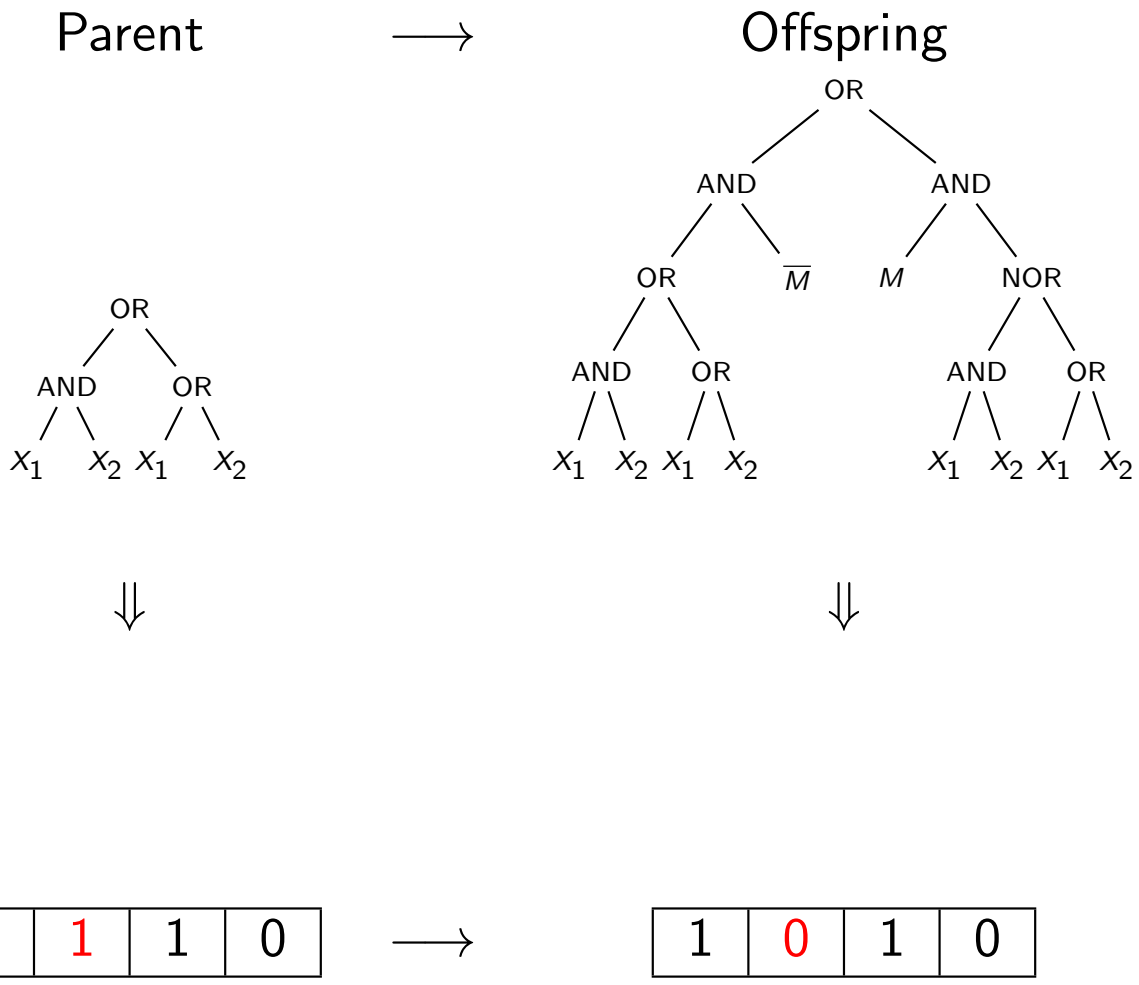
Genetic Programming using semantic operators solving **ANY** black-box boolean function learning problem

$\Updownarrow$ **is equivalent to** $\Updownarrow$

a Genetic Algorithm (evolving bitstrings) solving OneMax

The search outputs a tree (i.e., a boolean expression), but the runtime analysis can be done on the GA!

Equivalence



Runtime

Search equivalence

Genetic Programming using semantic operators solving **ANY** black-box boolean function learning problem

$\Updownarrow$ **is equivalent to** $\Updownarrow$

a Genetic Algorithm (evolving bitstrings) solving OneMax

The search outputs a tree (i.e., a boolean expression), but the runtime analysis can be done on the GA!

Theorem

Geometric Semantic Genetic Programming solves the black box boolean function learning problem in time $T = O(N \log N)$, where N is the length of the output vector (i.e. the number of rows of the truth table).

Issues

The proposed operator still suffers from the following issues:

1. In the way we designed the semantic mutation operator ($O = (P \wedge \overline{M}) \vee (\overline{P} \wedge M)$), the length of the offspring grows exponentially.
2. A fair choice of the problem size is the number of input variables n. Since $N = 2^n$, the runtime (which is $O(N \log N)$) is actually exponential in the problem size.

A shorter operator

Problem Since $O = (P \wedge \overline{M}) \vee (\overline{P} \wedge M)$, the length of the offspring grows exponentially.
Solution:

Definition

Forcing point mutation: Given a parent function $P : \{0,1\}^n \to \{0,1\}$:

- generate a random minterm M
- return as offspring $P' = P \vee M$ with probability 0.5 and $P' = P \wedge \overline{M}$ with probability 0.5

Effect on the output vector: forcing a random bit (corresponding to the row that makes M true) to a random value (1 with probability 0.5, and 0 with probability 0.5).

Polynomial-sized training set

Problem The runtime $O(N \log N)$ is exponential in the problem size n (since $N = 2^n$).
Actually, in practice, the training set is *small*. Particularly we can assume it to have polynomial size w.r.t. the number of input variables.
$\tau = \text{poly}(n)$.

X_1	X_2	X_3	O
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	1

This transforms the problem seen by the EA on output vectors into a "sparse" version of OneMax. **Unfortunately, using bit-flip or forcing point mutation operator, the time to reach the optimum is exponential.**

Still Exponential?

Why is it slow?

Most of the times the bit-flip operator flips bits that are not in the training set $\rightarrow$ fitness does not increase $\rightarrow$ slow runtime.

- How can we increase the chances to flip a bit of the training set at each generation? **flipping more bits!**

How many bits?

Sparse OneMax with τ out of N active bits can be solved in $O(\tau \log \tau)$ by a GA flipping at each generation each bit with probability $1/\tau$.

Proof idea: in expectation at each generation one active bit is flipped, thus the runtime is that of OneMax on a bitstring of length τ.

A bitwise mutation operator

How to implement a semantic operator inducing in the output space bitwise mutation with probability $p = 1/\tau$?

Definition

Bitwise mutation: Given a parent function $P : \{0,1\}^n \to \{0,1\}$ the mutation does the following:

- Sample an integer number x from $x \sim \text{Bin}(p, N)$
- Generate x minterms uniformly at random without repetitions $\{M_1, \cdots, M_x\}$
- The offspring is $P' = P \vee M$ with probability 0.5, and $P' = P \wedge \overline{M}$, where $M = M_1 \vee \cdots \vee M_x$.

Problem: Setting $p = 1/\tau$ (in order to solve the problem in $O(\tau \log \tau)$), the offspring takes exponential time to be generated (since M has expected length $pN = \frac{N}{\tau}$ which is exponential in the size of the problem n).

What are we looking for?

What we want is a mutation operator that:

- gives rise to a polynomial runtime on the sparse OneMax (thus when the size of the training set τ is polynomial in the number of variables)
- can be implemented efficiently (sampling an offspring takes polynomial time, so exponentially long midterms is not a feasible choice)
- makes the offspring growing polynomially with the number of generations (we are looking for short boolean expressions).

Block mutation

Why is it slow?

Most of the times the bit-flip operator flips bits that are not in the training set $\rightarrow$ fitness does not increase $\rightarrow$ slow runtime.

- How can we increase the chances to flip a bit of the training set at each generation? **flipping more bits!**
- How can we flip more bits keeping the size of the offspring small? **incomplete minterms!**

Incomplete minterms

Incomplete minterms force mutation on blocks of the output vector instead of single bits

X_1	X_2	X_3	$X_2 \wedge X_3$	$\overline{X_2} \wedge X_3$	$X_2 \wedge \overline{X_3}$	$\overline{X_2} \wedge \overline{X_3}$
1	1	1	**1**	0	0	0
1	1	0	0	0	**1**	0
1	0	1	0	**1**	0	0
1	0	0	0	0	0	**1**
0	1	1	**1**	0	0	0
0	1	0	0	0	**1**	0
0	0	1	0	**1**	0	0
0	0	0	0	0	0	**1**

We flip more bits at each generation keeping the length of the expression short.

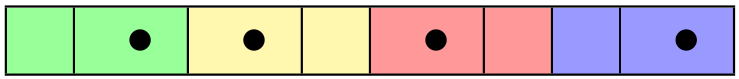
A block mutation operator

Definition

Fixed Block Mutation (FBM):

- Select randomly $v < n$ variables
- At each generation draw an incomplete minterm M comprising all the fixed v variables
- Use M for the forcing mutation

The output vector is partitioned into $b = 2^v$ blocks. At each generation a whole block is forced to the same value (0 or 1).



N = 8, b = 4

If each training point lays in a different block it is possible to reach the optimum in time $O(b \log b)$.

Polynomial runtime with high probability

We say that an operator solve a problem in polynomial time with high probability if:

- Choosing uniformly at random a **truth table** from all the possible truth tables using n boolean variables
- Choosing uniformly at random τ rows out of $N = 2^n$ to form the **training set**

$$\Downarrow$$

the probability that the problem chosen is one that can be solved in polynomial time using that operator is $1 - \Omega(\frac{1}{n})$.

Probability of success

Given a random truth table, a random training set of size τ, and a fixed set of v variables. The training set can be fit by FBM (in time $O(b \log b)$) if each training point lays in a different block.
This happens with probability

$$P(b, \tau) = \prod_{i=1}^{\tau} \frac{b - (i-1)}{b} \approx e^{-\frac{\tau^2}{2b}}$$

Choosing v $[b = 2^v]$

v must be:

- **Big enough** to guarantee that a solvable problem is selected with high probability
- **Small enough** to guarantee, when the problem is solvable, to solve it in polynomial time.

Solution (assuming $\tau = O(n^c)$)

$$v = 2(c+1)\log_2(n) \Rightarrow \text{runtime:} O(n^{2c} \log n), p = O(1 - 2/n)$$

Conclusion

Take home messages:

- Geometric Semantic Genetic Programming makes the search performed by GP on the space of functions (trees) equivalent to that performed by a GA on the output space.
- It is thus possible to easily get runtime results for GP reusing runtime results for GA thus:
  - Get guarantees of the convergence time of the algorithm
  - Improve the operators' design getting insight from the theory (theory-laden design)
  - Particularly an operator solving the black-box boolean function learning problem in polynomial time with high probability has been designed

Thank you!

The presentation will be available at `http://www.cs.bham.ac.uk/~axm322/`