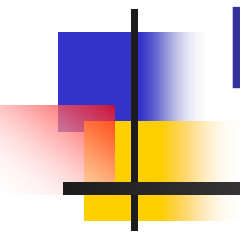
Probabilistic Incremental Program Evolution (PIPE)



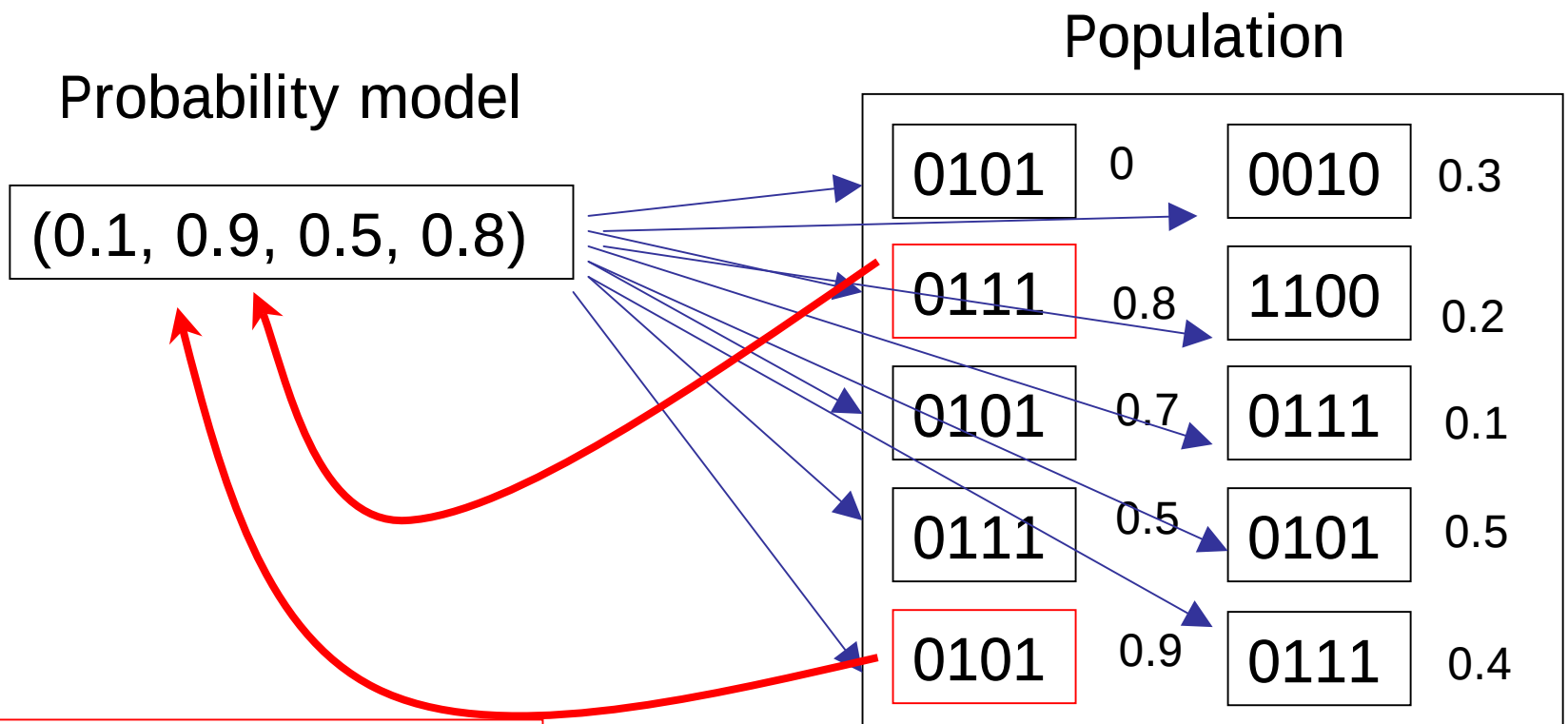
Population Based Incremental Learning [Baluja, 94]

- It learns **explicitely** a probabilistic model of the interesting regions of the search space
- If points in the search space are bitstrings (like “0101110”), the probabilistic model to be learned is:
  - a vector $p=(p_1, p_2, ..., p_n)$
  - p_i is the probability of generating a 1 in the i position of the bitstring $x=(x_1, x_2, ..., x_n)$
  - Initially $p=(0.5, 0.5, ..., 0.5)$

Population Based Incremental Learning [Baluja, 94]

- A population of x, is generated, their fitness is computed, and p is updated by using the M best individuals
- Update rule: $p_i' = p_i.(1\text{-}LR) + LR.x_i^*$
- Eventually, p should converge to a solution like:
  - p=(0.99, 0.001, …, 0.99)

Population Based Incremental Learning [Baluja, 94]



Update model with best individuals

$$p_i' = p_i.(1\text{-}LR) + LR.x_i^*$$

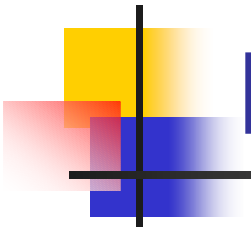
Estimation of Distribution Algorithms (EDA's)

1. Generate an initial population and evaluate them
2. Select *M* best individuals
3. Estimate the probability distribution
4. Generate a new population (sometimes, the old population is mixed with the new one)
5. If not-termination, go to 2

State of the Art for EDA's applied to GP

- Yin Shan, Robert McKay, Daryl Essam, Hussein Abbass. 2005. **"A Survey of Probabilistic Model Building Genetic Programming"**. *TR-ALAR-200510014*

Probabilistic Incremental Program Evolution (PIPE)

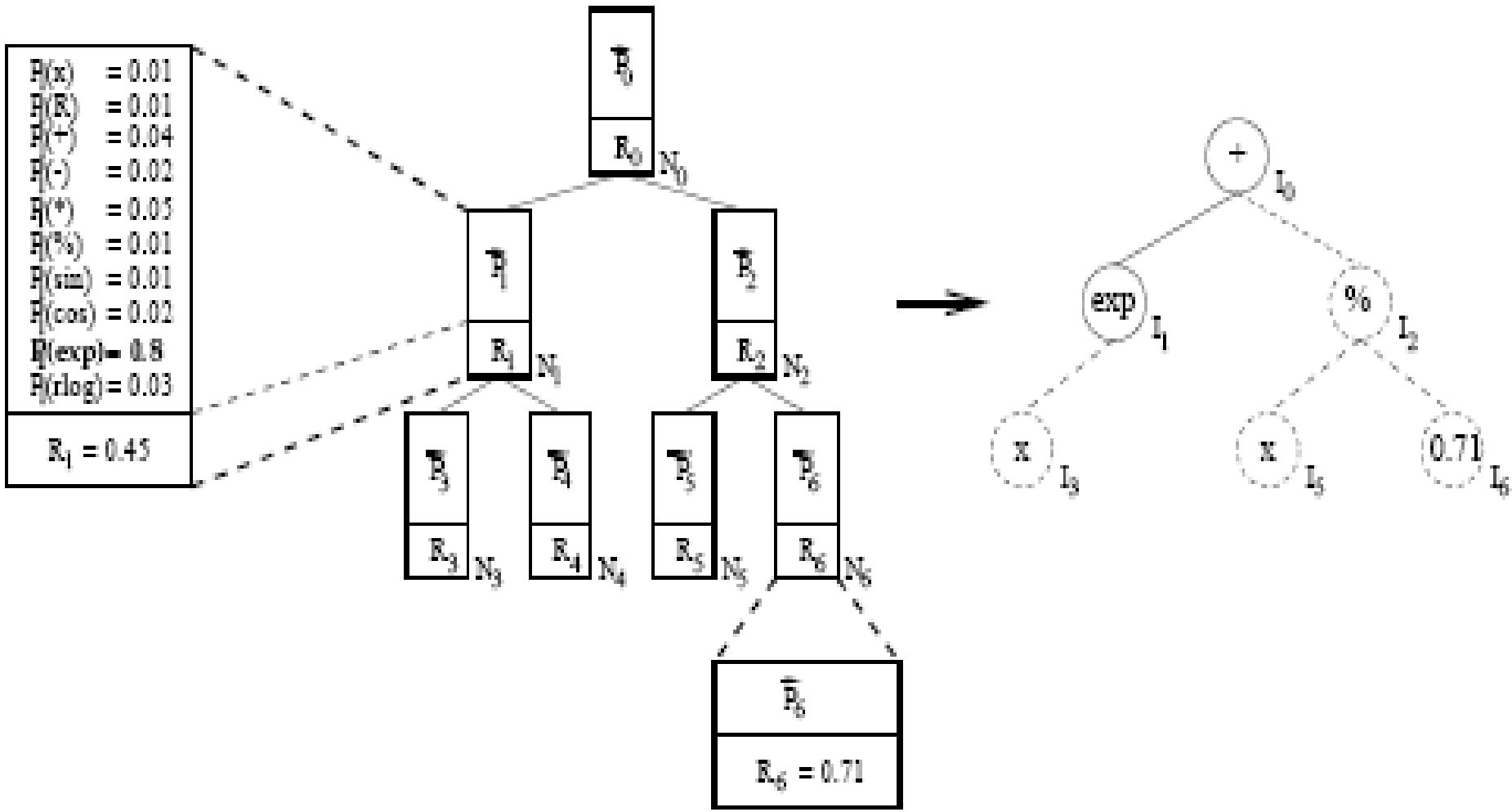


- EDA's applied to the evolution of parse trees (hierarchical programs)
- PIPE [Salustowicz, Schmidhuber, 97]
- Search in the space of tree-shaped probability distributions
- Tries to find a distribution that generates good programs (trees)

Probabilistic Incremental Program Evolution (PIPE)

- Programs made of:
  - Functions: $F = \{F_1, ..., F_k\}$
  - Terminals: $T = \{T_1, ..., T_l\}$
- The Generic Random Constant (GRC):
  - Similar to Ephemeral Random Constant (ERC)
- *Closure* required

Probabilistic Prototype Tree (PPT)



An Initial Node of the PPT

Initially:

$$P_j(I) := \frac{P_T}{l}, \forall I : I \in T$$

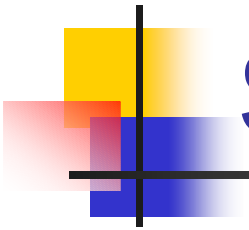
$$P_j(I) := \frac{1 - P_T}{k}, \forall I : I \in F,$$

$P_j(x)$	$= 0.3$
$P_j(R)$	$= 0.3$
$P_j(+)$	$= 0.05$
$P_j(-)$	$= 0.05$
$P_j(*)$	$= 0.05$
$P_j(\%)$	$= 0.05$
$P_j(\sin)$	$= 0.05$
$P_j(\cos)$	$= 0.05$
$P_j(\exp)$	$= 0.05$
$P_j(\mathrm{rlog})$	$= 0.05$
R_j	$= 0.45$

N_j

Creation of the Initial Population

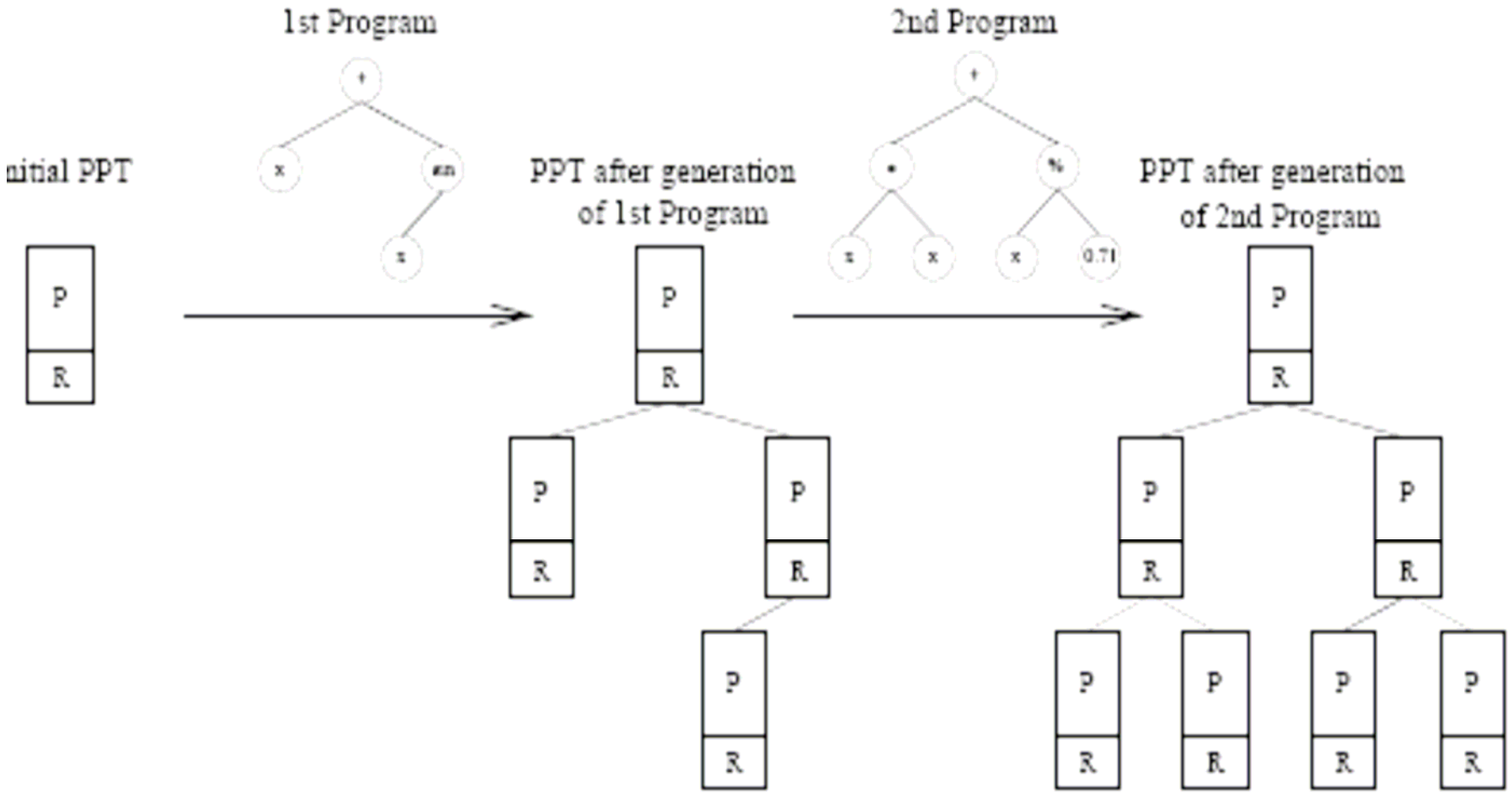
- The PPT is parsed top-down, left-to-righ
- A function or terminal is selected according to its probability
- If R is selected (the GRC), then:
  - If prob(R) > threshold, Then fixed value R
  - Else, generate a random value for R



Size of the PPT

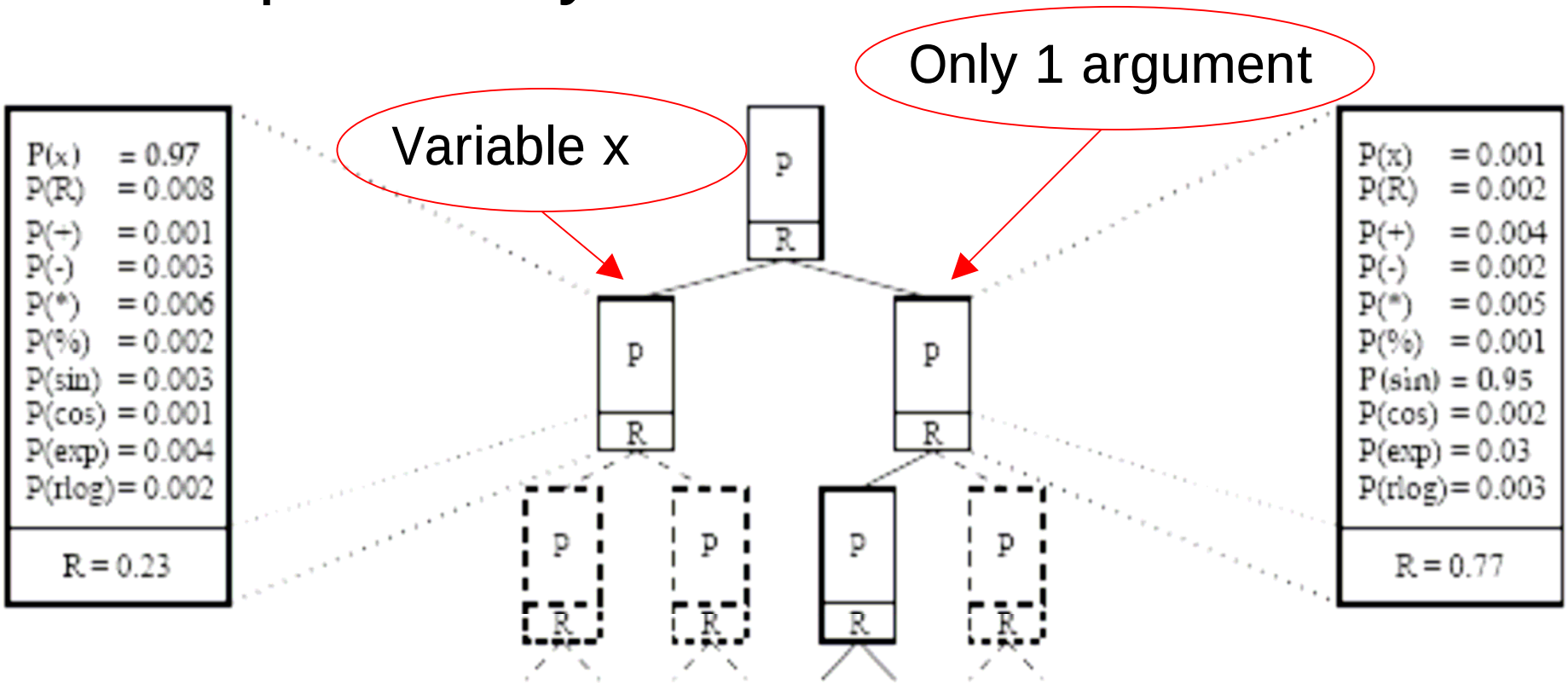
- Empirically, it is enough with PPT three times the best solution found so far
- Initially, the PPT contains only the root node
- Nodes are created on demmand (if in a leave node, a function is selected, it is necessary to create the arguments)

PPT Growth

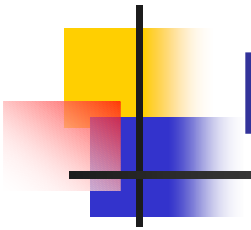

1st Program
2nd Program
nitial PPT
PPT after generation of 1st Program
PPT after generation of 2nd Program
P
R
x
sin
x
*
%
x
x
0.71

Pruning the PPT

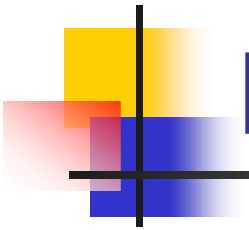
- In case a symbol has a very large probability



Probabilistic Incremental Program Evolution (PIPE)



- Every generation, the PPT is updated towards the best individual (just one), so that it becomes more likely to generate similar individuals
- Minimization
- If *fitness* are equal, the smaller solution is preferred



PIPE's Algorithm

- Two modes of learning:
  - *Generation based learning (GBL):* Updates the PPT towards the best individual in that generation. Increases the probability that similar individuals will be generated by the PPT
  - *Elitist learning (EL):* Updates the PPT towards the best program **found so far** (it's a kind of long term memory)

PIPE's Algorithm

1. Initial Population
2. Fitness computation
3. PPT update: with probability P_{el}
    1. Generation Based Learning
    2. Elitist Learning
4. PPT mutation (exploration)
5. PPT prunning

PPT Update

1. Let $PROG_b$ be the target program, used to update the PPT
2. Let $P(PROG_b)$ be the probability that the PPT currently generates $PROG_b$

$$P(\text{PROG}_b) = \prod_{j:N_j \text{ used to generate PROG}_b} P_j(I_j(\text{PROG}_b))$$

PPT Update

3. The desired (target) probability for $PROG_b$ is computed

$$P_{TARGET} = P(\text{PROG}_b) + (1 - P(\text{PROG}_b)) \cdot lr \cdot \frac{\varepsilon + FIT(\text{PROG}^{el})}{\varepsilon + FIT(\text{PROG}_b)}$$

- *lr* = learning rate
- The quotient controls fitness-dependent learning (minimization)
- If large epsilon, learning is independent of the fitness (quotient = 1)

PPT Update

4. PPT is modified until $P(PROG_b) = P_{TARGET}$
5. Probabilities are updated in parallel.
6. c^{lr} (0.1): tradeoff between good accuracy and fast update

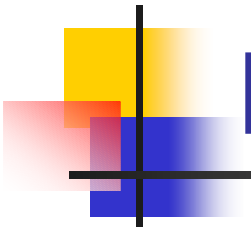
$$\text{REPEAT UNTIL } P(\text{PROG}_b) \geq P_{TARGET} :$$

$$P_j(I_j(\text{PROG}_b)) := P_j(I_j(\text{PROG}_b)) + c^{lr} \cdot lr \cdot (1 - P_j(I_j(\text{PROG}_b)))$$

PPT Update

5. Normalization of PPT (instructions not in $PROG_b$, get decreased proportionally to their current value. Everything must add to 1.0)

$$P_j(I) := P_j(I) \cdot \left(1 - \frac{1 - \sum_{I^* \in S} P_j(I^*)}{P_j(I_j(\text{PROG}_b)) - \sum_{I^* \in S} P_j(I^*)} \right) \quad \forall P_j(I) : I \neq I_j(\text{PROG}_b)$$



PPT Update

6. Finally, R constants are copied from $PROG_b$ to PPT

Mutating the PPT

- To explore around $PROG_b$
- Pj(Ij($PROG_b$) of instructions in $PROG_b$ get mutated
- P_M= mutation probability per program
- P_{Mp} = mutation probability per node and instruction
- z = number of possible instructions
- Dividing by $|PROG_b|$ avoids larger programs having more mutations (the square root gives more mutations to large programs. Empirical reasons)

$$P_{M_p} = \frac{P_M}{z \cdot \sqrt{|\mathrm{PROG}_b|}}$$

Mutating the PPT

- Mutation:

$$P_j(I) := P_j(I) + mr \cdot (1 - P_j(I))$$

- Normalization:

$$P_j(I) := \frac{P_j(I)}{\sum_{I^* \in S} P_j(I^*)} \qquad \forall P_j(I) : I \in S$$

Results PIPE

- Symbolic regression: PIPE better than GP in 24% of runs, and worse in 33%. Larger variance
- 6-bit parity problem:
  - More successful runs (70% vs. 60%)
  - Faster (52476 vs. 120000 evaluations)
  - Smaller (61 vs. 90 nodes)
- R. P. Salustowicz, M. A. Wiering, J. Schmidhuber. 1998. **“Learning Team Strategies: Soccer Case Studies”**. *Machine Learning Journal*

PIPE Applied to Soccer

- Domain similar to Robosoccer
- Actions:
  - Simple: go_forward, turn_to_ball, turn_to_goal, shoot
  - Complex: goto_ball, goto_goal, goto_own_goal, goto_player, goto_opponent, pass_to_player, shoot_to_goal

PIPE Applied to Soccer

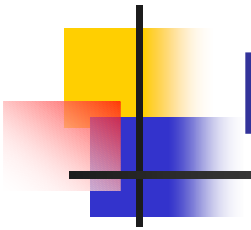
- BRO: Biased Random Opponent. Biased random player (scores 75 goals to an static opponent)
- GO: Good Opponent. Scores 417 goals to BRO
- PIPE: fitness computed by playing against BRO
- COPIPE: co-evolution fitness
- TD-Q: Reinforcement learning
- A PPT is learned for every action

Results PIPE vs. BRO (simple actions)

Team size		*GO*	PIPE	CO-PIPE	TD-Q
	Maximum score difference	417	310	192	42
1	Average goals ± st.d.	417 ± 6	320 ± 42	212 ± 97	52 ± 14
	Average *BRO* goals ± st.d.	0 ± 0	10 ± 7	20 ± 10	10 ± 3
	Achieved after games	n.a.	3300	3000	1700
	Maximum score difference	481	359	310	70
3	Average goals ± st.d.	481 ± 8	373 ± 86	324 ± 62	102 ± 14
	Average *BRO* goals ± st.d.	0 ± 1	14 ± 6	14 ± 11	32 ± 8
	Achieved after games	n.a.	3300	3200	1700
	Maximum score difference	364	481	357	154
11	Average goals ± st.d.	367 ± 18	512 ± 129	393 ± 53	212 ± 84
	Average *BRO* goals ± st.d.	3 ± 1	31 ± 23	36 ± 27	58 ± 23
	Achieved after games	n.a.	3100	1900	2500

Results PIPE vs. BRO (complex actions)

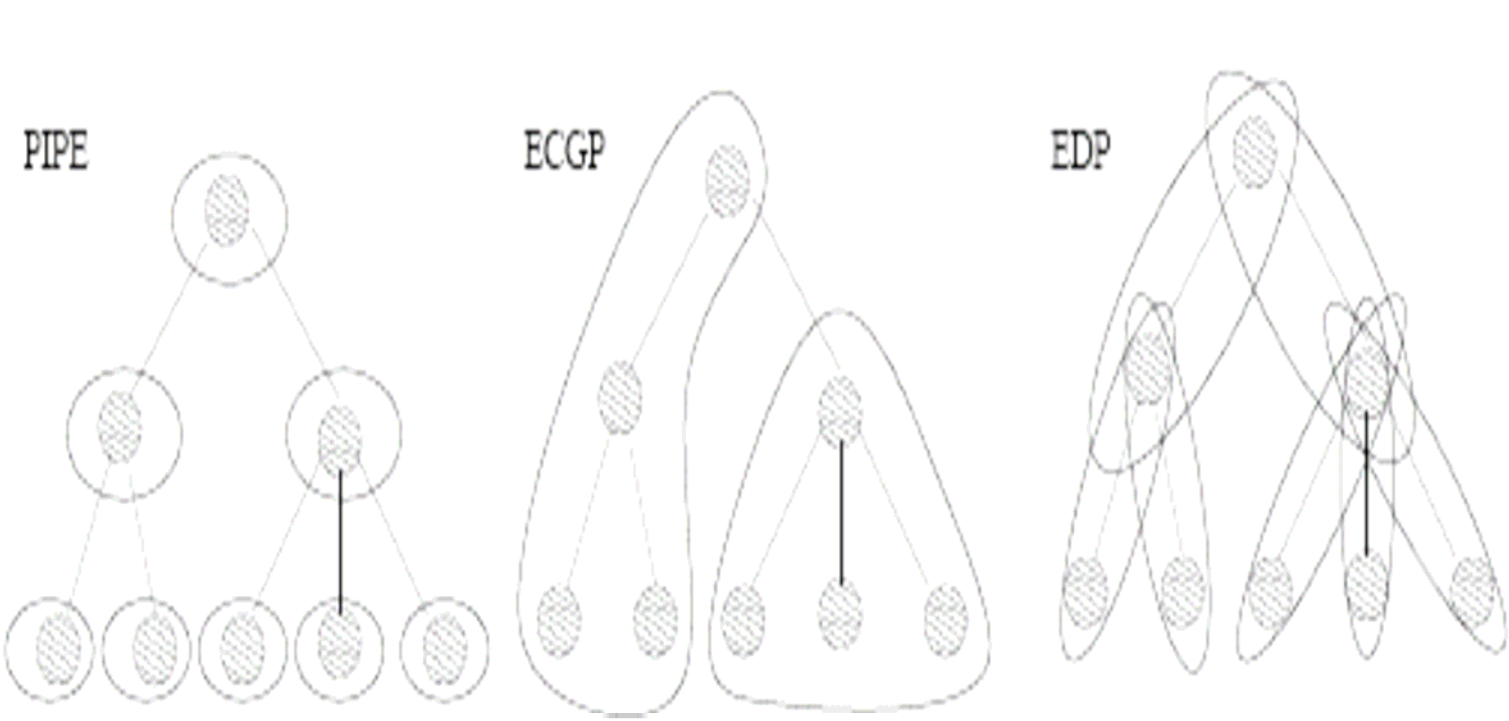
	GO	PIPE	CO-PIPE	TD-Q
Maximum score difference	364	530	536	46
Average goals ± st.d.	367 ± 18	551 ± 215	539 ± 220	76 ± 140
Average *BRO* goals ± st.d.	3 ± 1	21 ± 35	3 ± 4	30 ± 29
Achieved after games	n.a.	1200	1200	900



PIPE Limitations

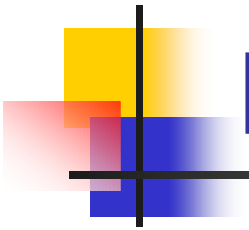
- Probability distribution in a node is independent of other nodes

PIPE, ECJP, EDP



Estimation of Distribution Programming (EDP)

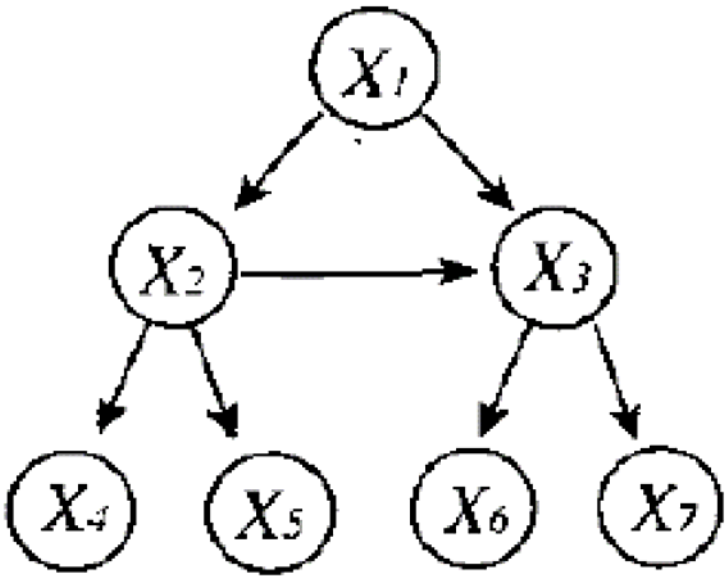
- Yanai, Iba. 2003. **"Estimation of distribution programming based on Bayesian network"**. CEC 2003.
- Joint probability of father and children nodes



EDP Algorithm

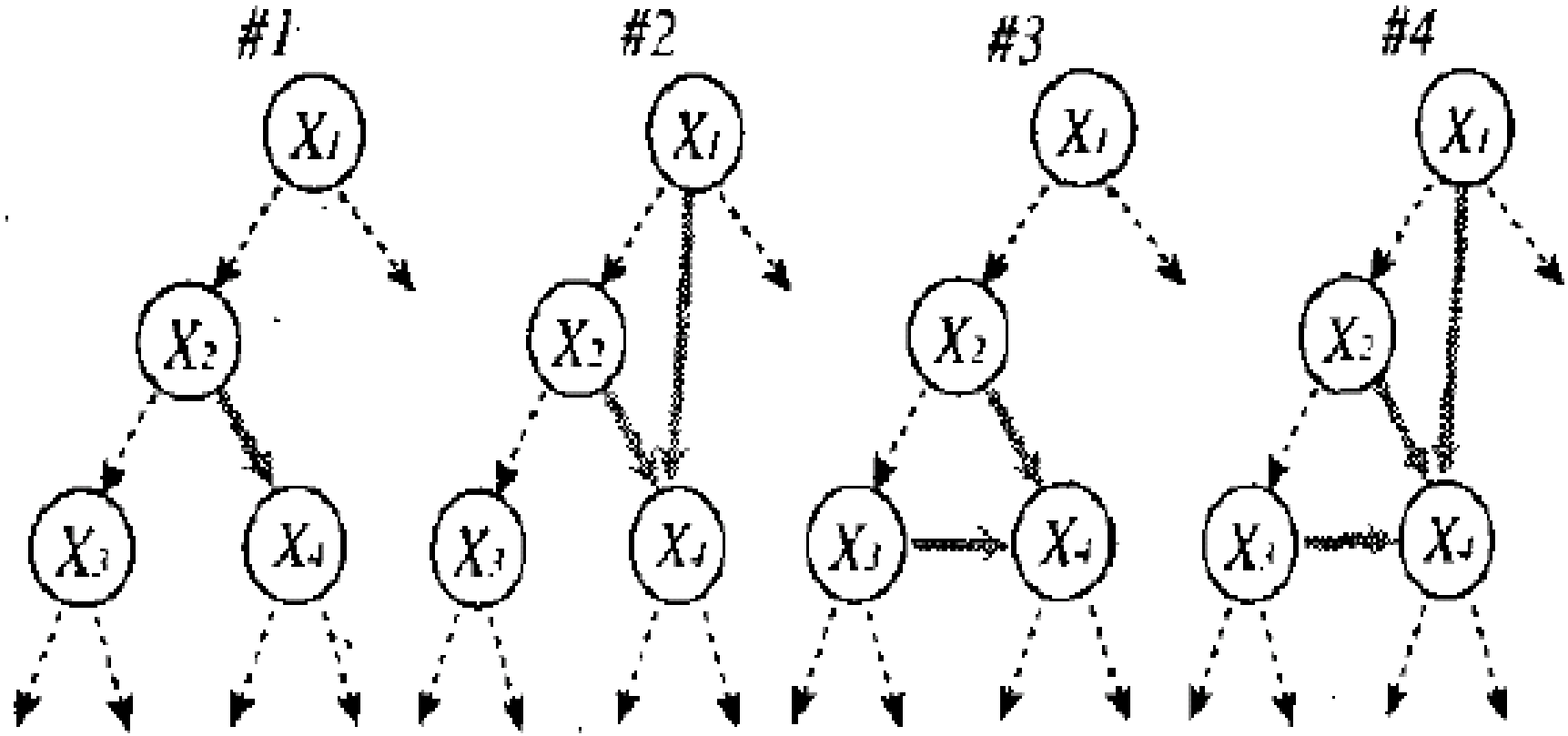
1. Create population
2. Eval individuals
3. Estimate distribution
4. If termination, go to 7
5. Generate a new population
6. Replace old population
7. Return best individual

Bayesian Network



$$P(X_3 = x_3 | X_1 = x_1, X_2 = x_2, \cdots, X_6 = x_6, X_7 = x_7,)$$
$$= P(X_3 = x_3 | X_1 = x_1, X_2 = x_2). \quad (1)$$

Possible Bayesian Networks



Cost of Different Bayesian Networks

m=núm. possible symbols

n=núm. nodes in the tree

i=núm. dependencies

$$n \cdot m^{i+1} \times (memory\ size\ of\ int).$$

#	Graph topology		Memory size	Adequate sampling size
	Dependency relationship	Level		
1	Only parent node	One	$n \cdot m^2 \times INT$	m^2
2	Only parent node	Up to two	$n \cdot m^3 \times INT$	m^3
3	Parent and brother nodes	One	$n \cdot m^4 \times INT$	m^4
4	Parent and brother nodes	Up to two	$n \cdot m^5 \times INT$	m^5

Computing Probabilities

$$P(X_i = x|C_i = c) = \frac{\sum_{j=1}^{N} \delta(j, X_i = x|C_i = c)}{N};$$

where

$$\delta(j, X_i = x|C_i = c) = \begin{cases} 1 & \text{if } X_i = x \text{ and } C_i = c \\ & \text{at the individual } j \\ 0 & \text{else} \end{cases}$$

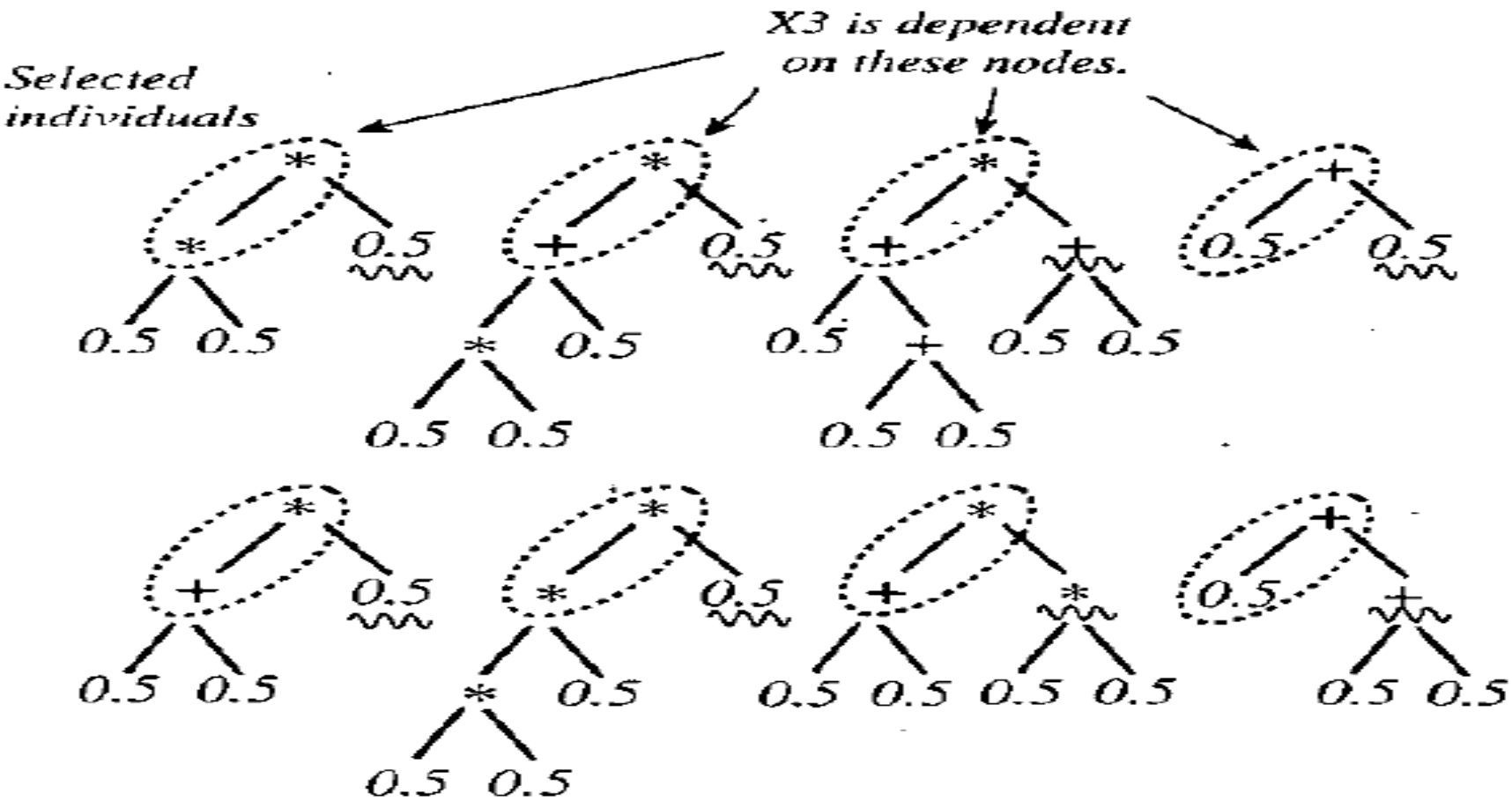
j: individual j

Fitness Weighted Probabilities

$$P(X_i = x | C_i = c) = \frac{\sum_{j=1}^{N} F_j \delta(j, X_i = x | C_i = c)}{\sum_{j=1}^{N} F_i}.$$

Unlike PIPE, the probability distribution is generated directly from the population, withouth considering the previous distribution

Selected Individuals



Note: only father and brother nodes are considered

Distribution from the Selected Individuals

Table 2: Estimating $P(X_3 = x_3 | X_1 = x_1, X_2 = x_2)$

X_3	X_1	X_2	Frequency	Probability (= P)	Modified probability (= P')
+	+	0.5	1	0.5	0.49
*	+	0.5	0	0	0.02
0.5	+	0.5	1	0.5	0.49
+	*	+	1	0.25	0.26
*	*	+	1	0.25	0.26
0.5	*	+	2	0.5	0.48
+	*	*	0	0	0.03
*	*	*	0	0	0.03
0.5	*	*	**2**	**1.0**	**0.94**

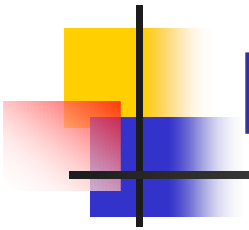
Distribution Adjustement

$$P' = (1 - \alpha)P(X_i = x|C_i = c) + \alpha P_{default}(X_i = x|C_i = c).$$

$$P_{default}(X_i = x|C_i = c) = \frac{1}{node\ symbol\ number}.$$

It is some sort of a-priory probability. Thus, all probabilities become different than 0

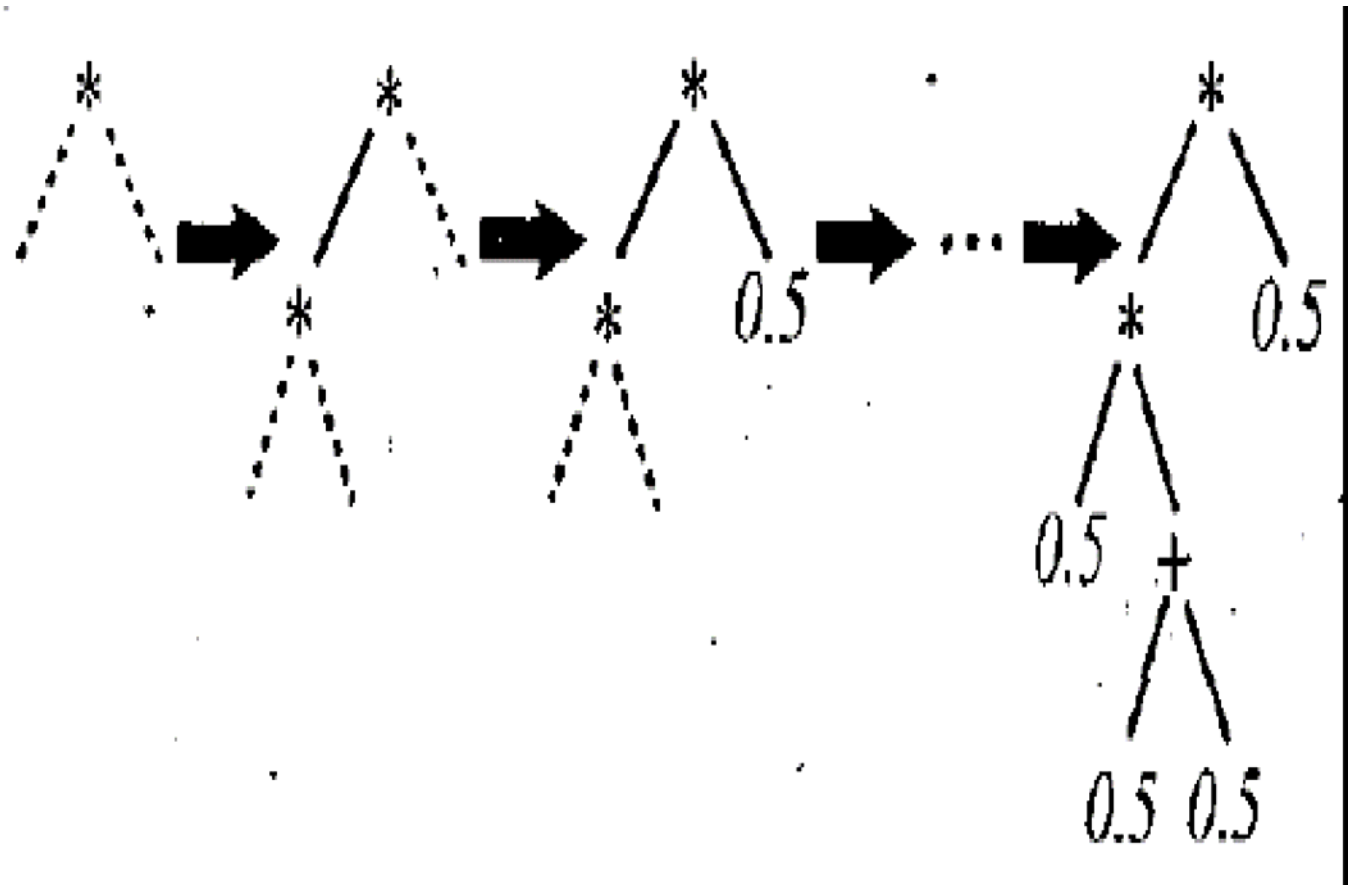
(number of symbols)



Program Generation

- First, the best k individuals are selected from the previous generation

Individual Generation



The “Max Problem”

Obtain the maximum value using +, * y 0.5

Easy for EDP-GP because 0.5 must be at the bottom, + in the middle and * at the top

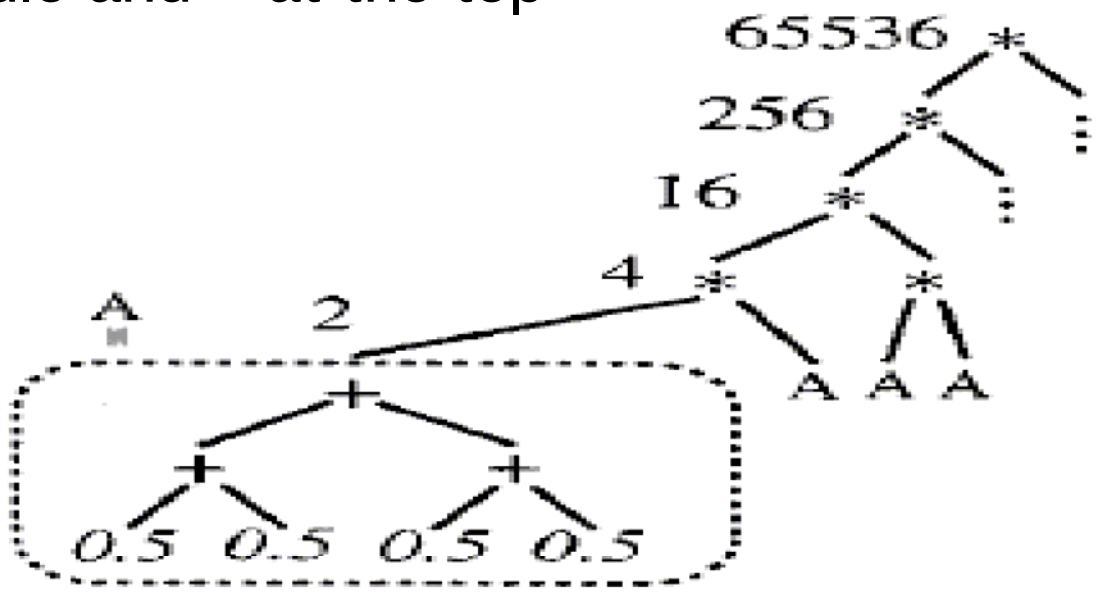
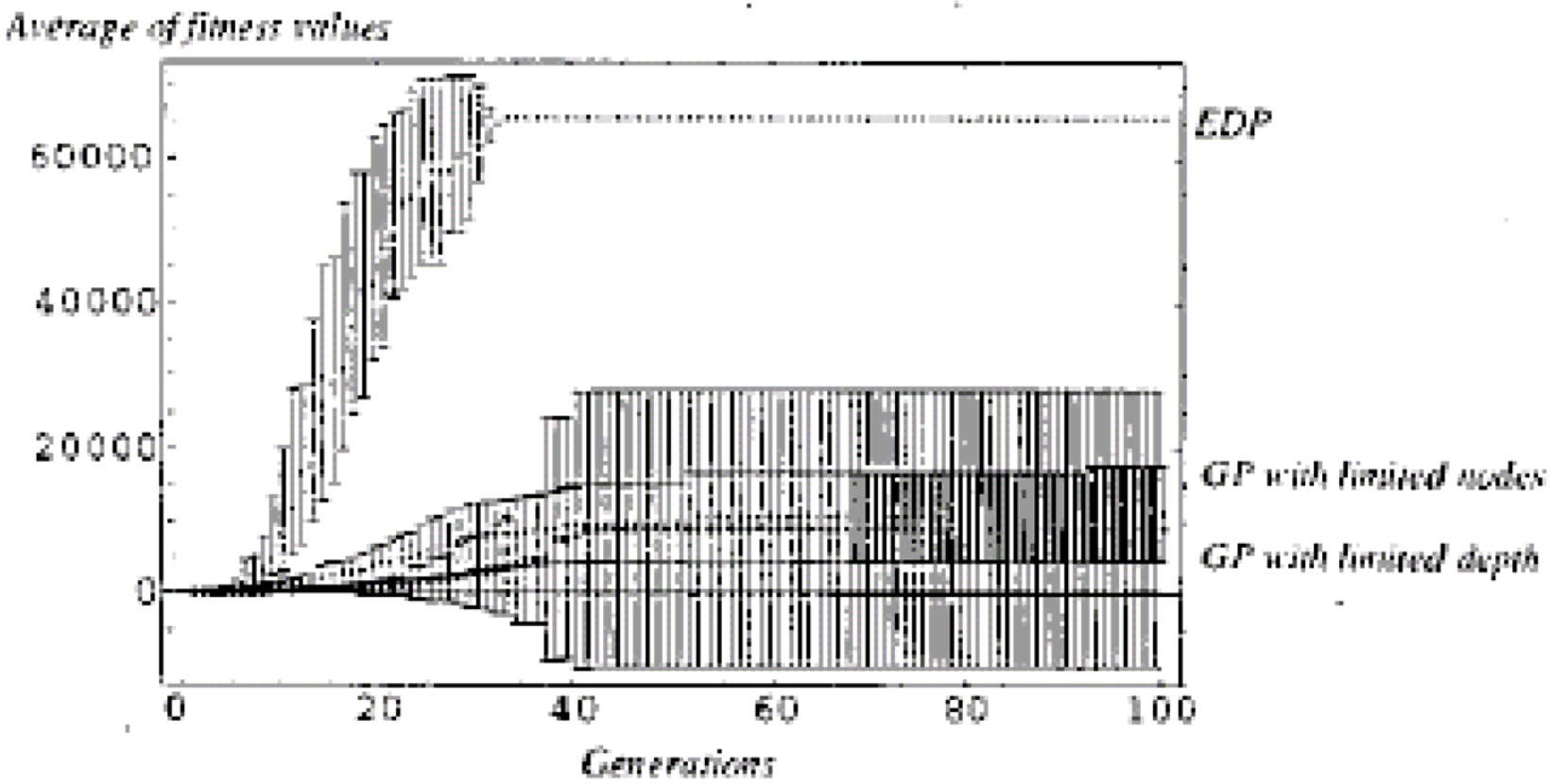


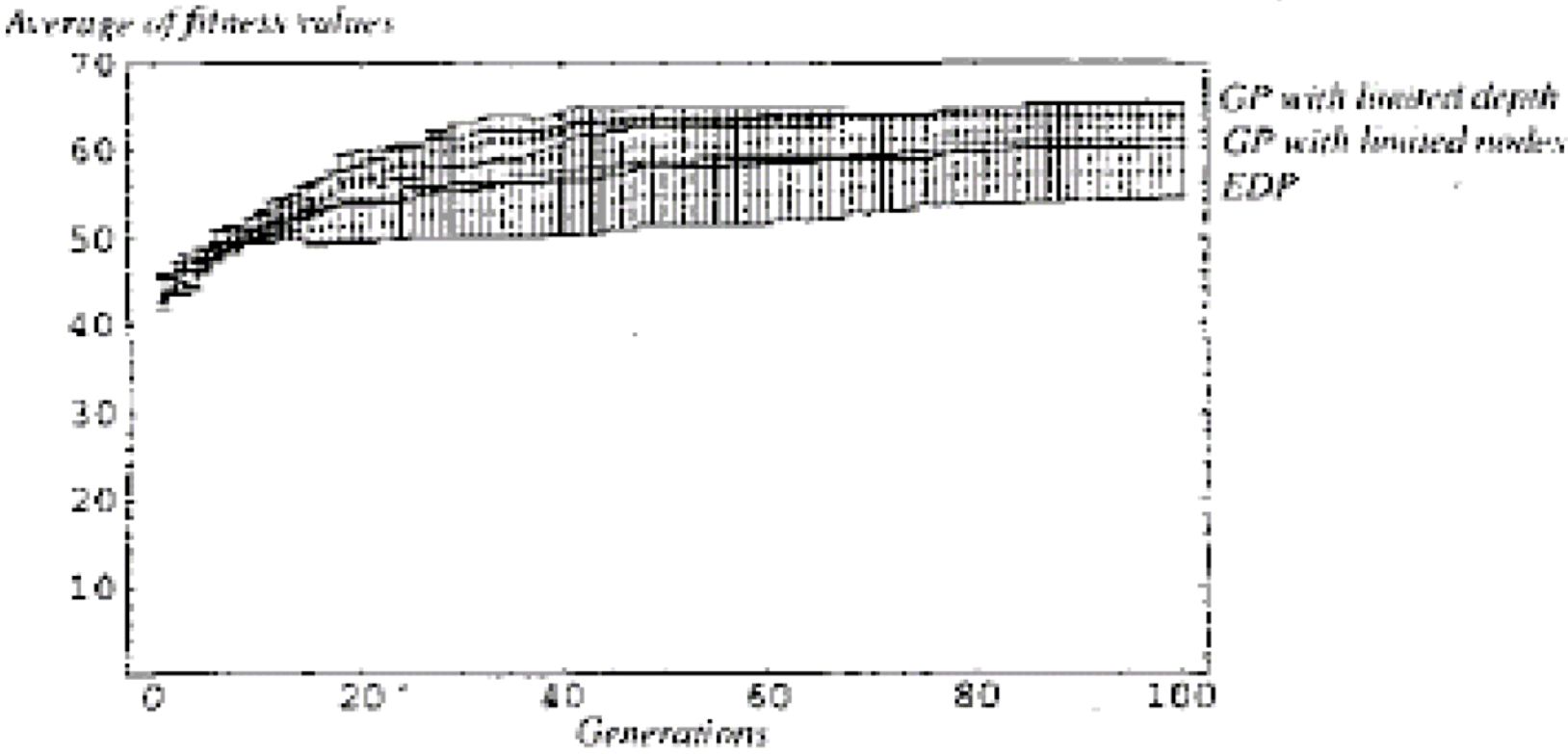
Figure 6: The maximum value by a limited depth tree.

Results. Max Problem



Results. 6-Multiplexer

Here EDP works slightly worse



Recursive Bayesian Networks

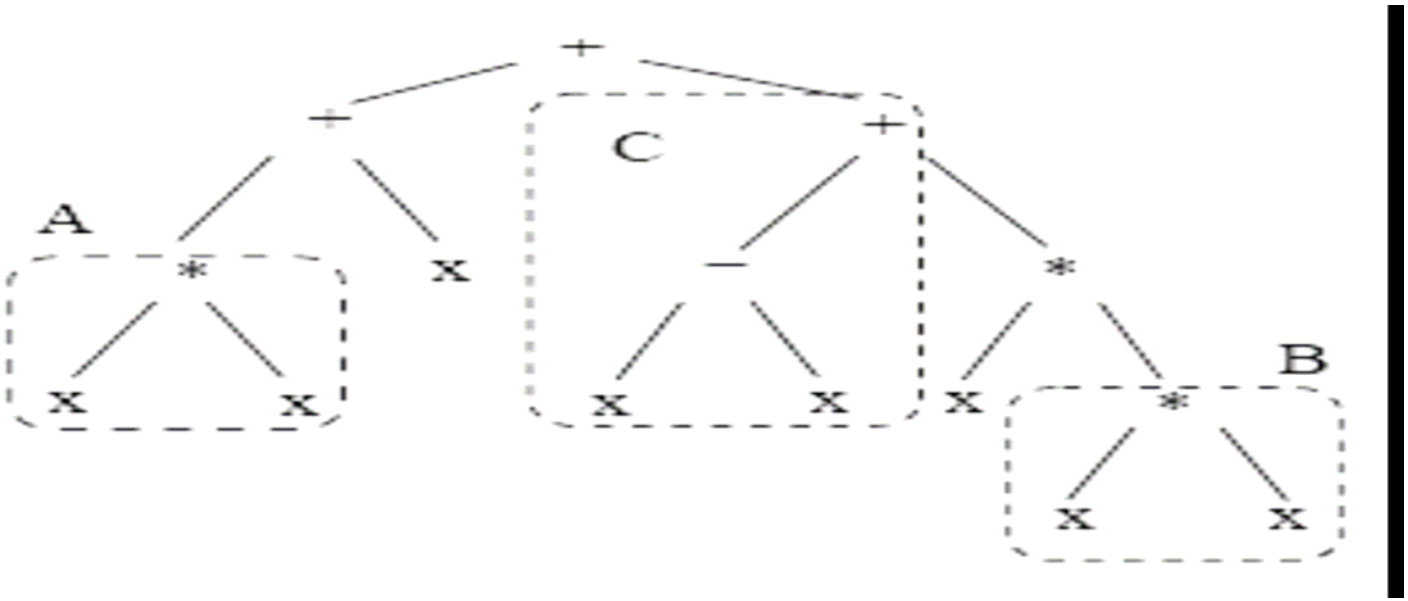
Figure 10: The method used in experiments

Figure 11: Recursive single Bayesian network

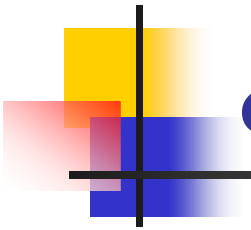
Figure 12: Recursive multi-Bayesian network

Limitations. Building Blocks at Different Places

$$f(x) = x^3 + x^2 + x.$$



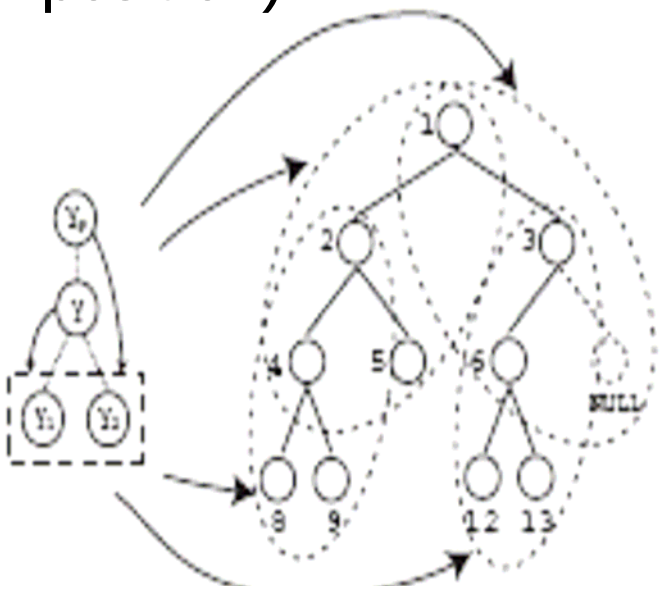
Extended EDP (XEDP) [Yanai & Iba, 05]



- It is like EDP, but in addition, it uses a recursive (position independent) bayesian network

XEDP. Redes bayesianas

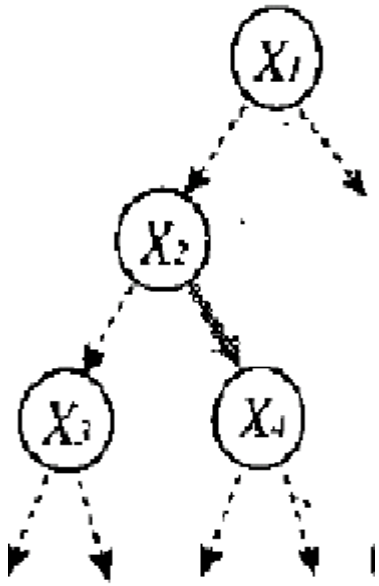
Dist. Recursiva (relative position)



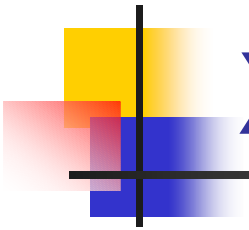
$$P(Y_1, Y_2, \cdots, Y_{a_{max}} | Y, Y_p)$$

P(Children|father,grandfather)

Conditional Distribution (absolute position)



P(children|father)



XEDP. Generating Individuals

- It combines the two distributions (absolute and relative)

1. Generate a program T using the absolute distribution
2. Generate a subtree S using the relative distribution
3. Replace a subtree of T by S

XEDP. Experiments

- XEDP
- GP
- Type A: uses only thel absolute distribution
- Type B: uses only the relative distribution
- Type C: XEDP, but replaces the subtree by a random one
- Type D: XEDP, but only the absolute distribution without father-son dependencies (like PIPE)

Results Max Problem

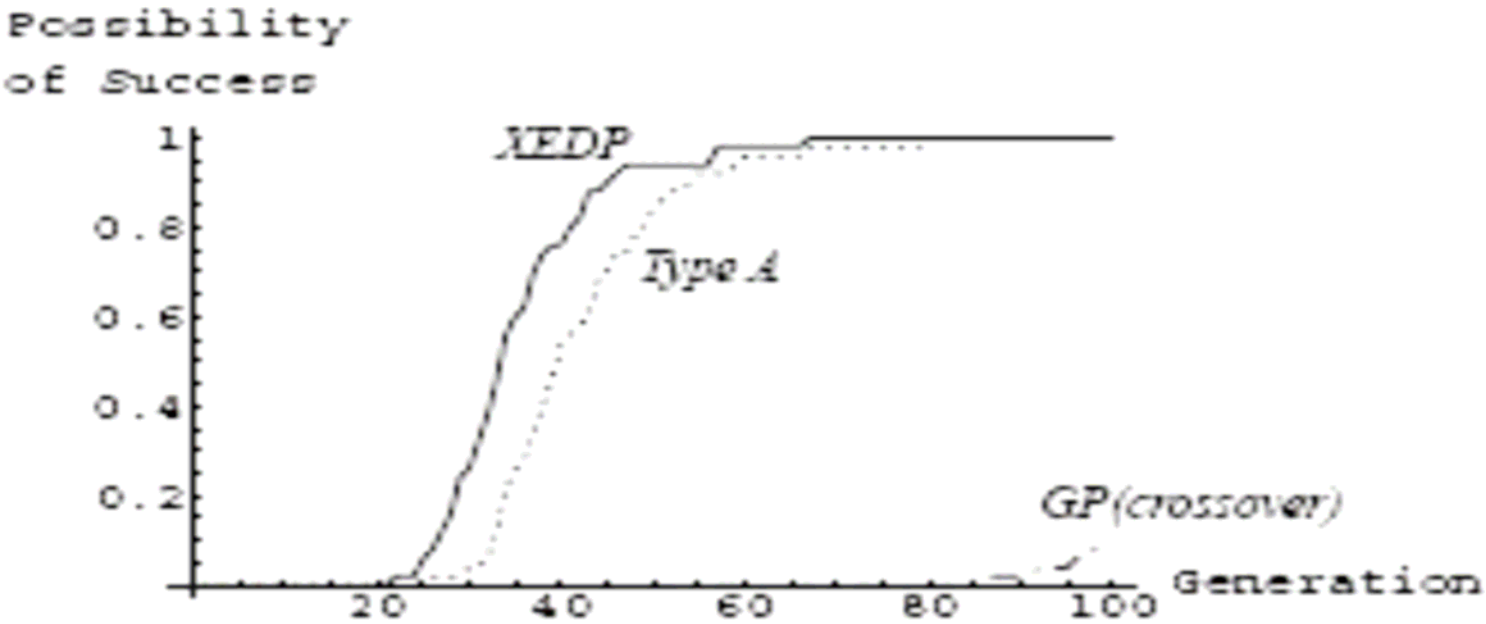


Figure 2: Cumulative probability of success for the Max problem.

Results 6-multiplexer

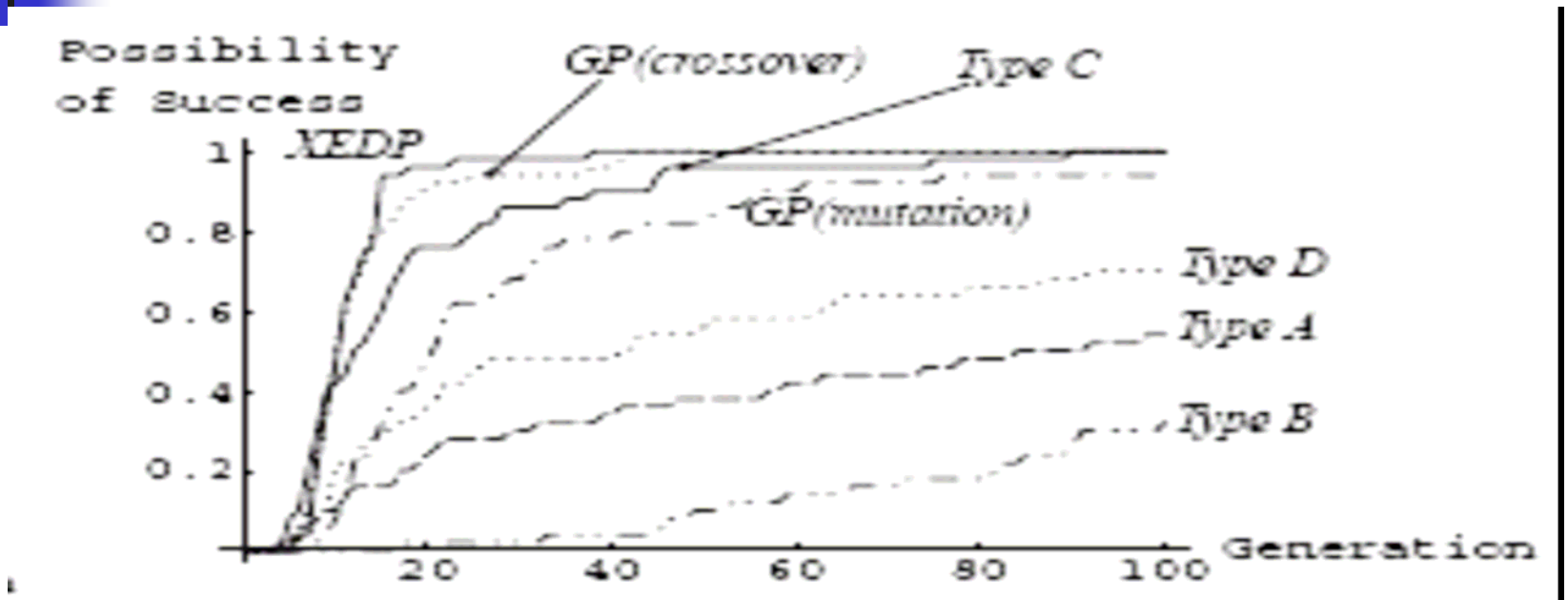


Figure 3: Cumulative probability of success for the Boolean 6-bit multiplexer problem.

Results Wall-following

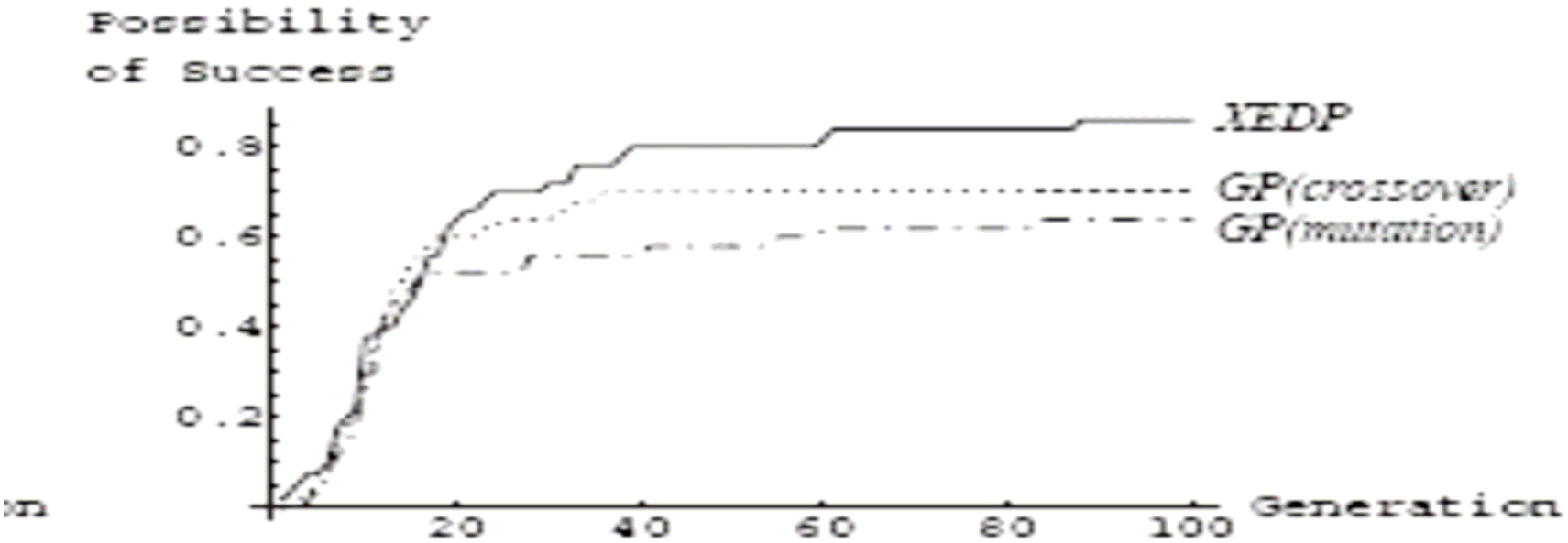


Figure 4: Cumulative probability of success for the Wall-following problem.

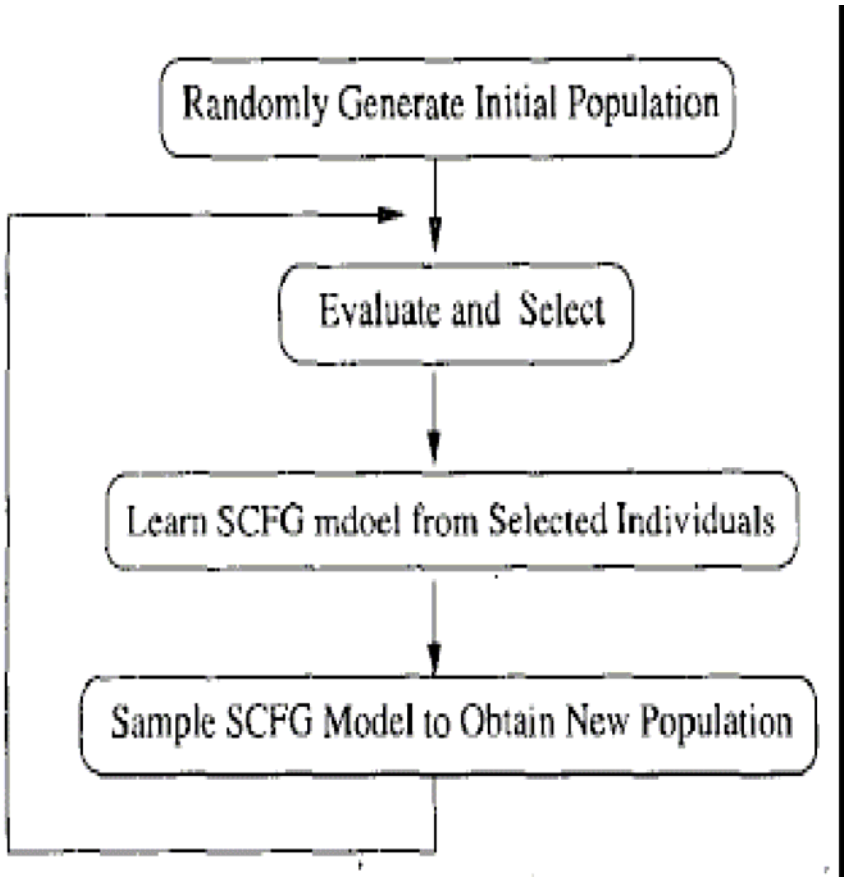
Extended Compact Genetic Programming (ECGP)

- K. Sastry, D.E. Goldberg. 2003. Probabilistic model building and competent genetic programming. Genetic Programming Theory and Practice
- It uses Marginal Product Models
- Divides the PPT into several subtrees, considered independent
- Joint probabilities are computed for eery subtree (no independence assumed within subtrees)

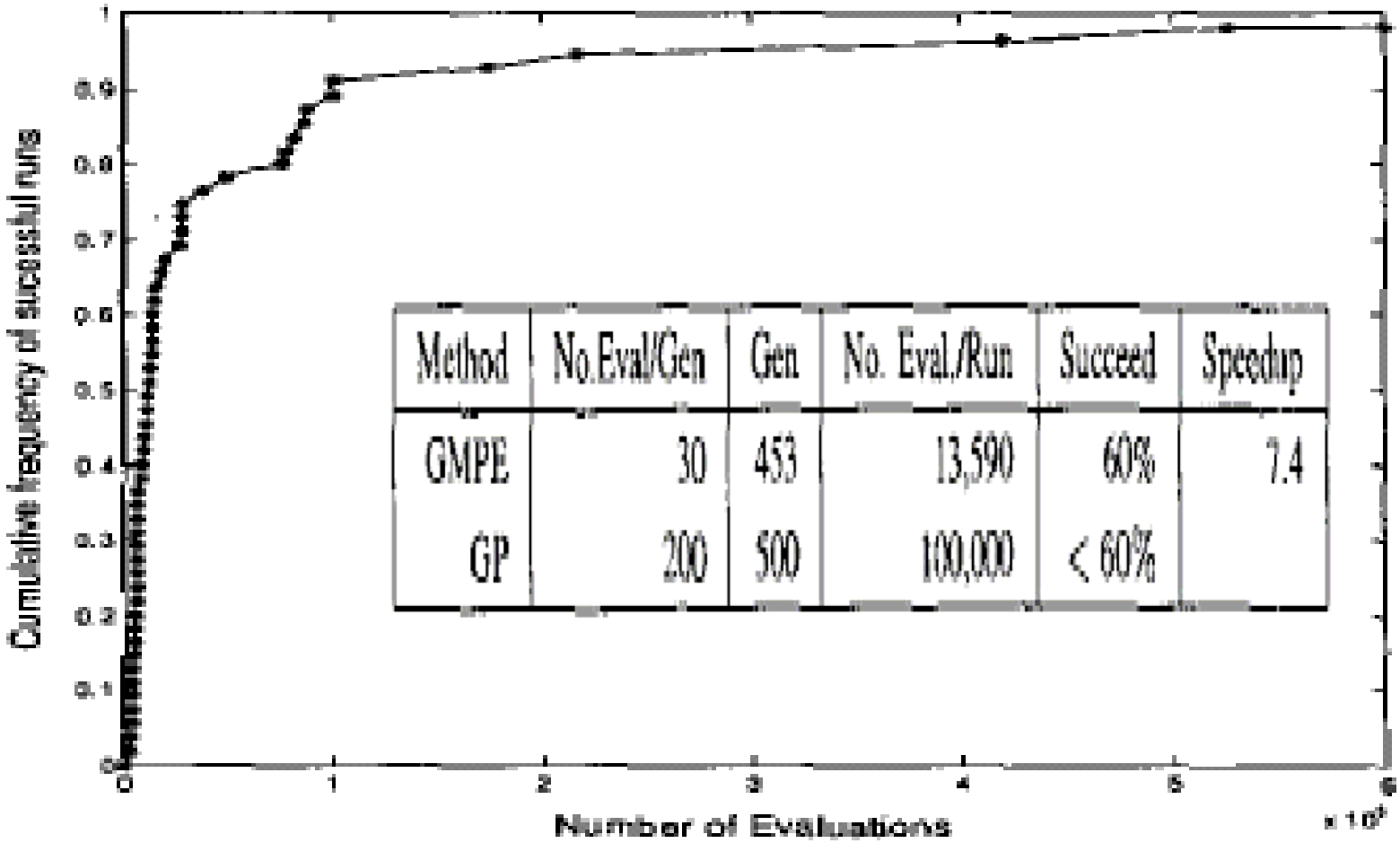
EDA's with Grammars

- Shan, McKay, Baxter, Abbass, Essam, Nguyen. 2004. Grammar Model-Based Program Evolution: **GMPE**
- Learning Stochastic Context Free Grammars (SCFG)
- Each rewriting rule has a weight that indicates the probability to be used

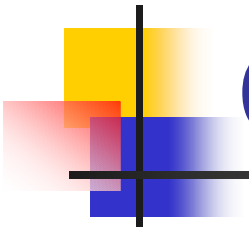
GMPE Algorithm



Results. Max Problem



Method	No.Eval/Gen	Gen	No. Eval./Run	Succeed	Speedup
GMPE	30	453	13,590	60%	7.4
GP	200	500	100,000	< 60%	



Conclusions EDA-GP

- Probability distributions are explored explicitely
- They can also learn stochastic context free grammars
- Not well tested yet, but it seems that they are equivalent or better than GP