Genetic Algorithms

Presented by Chen Shan-Tai

Reference:

1. Machine Learning, Chapter 9
2. Data Mining, Chapter 10
3. Tom M. Mitchell's teaching materials
4. Genetic algorithms and engineering design

Outline

- Fundamentals of Genetic Algorithms
- GAs for TSP
- Schema Theorem
- GAs for Machine Learning
  - Classification Problems
  - Concept Learning
- Genetic Programming
- Our Story (if time permits)

GAs

- GAs are derivative-free, stochastic-optimization methods based loosely on the concepts of natural selection and evolutionary processes.
- Properties of GAs:
  - *approximate* not *complete* methods
  - emphasizing *crossover* as the key operation
  - maintaining a population of potential solutions (beam search) while other methods process a single point of the search space.
- Problems GAs apply to:
  - No reasonably fast algorithms for the problems have been developed
  - NP-hard, hard-optimization problems, and learning tasks.

Genetic Algorithm Vocabulary

- **Population**: initial set of random *feasible solutions*
- **Chromosome** (individual, string): a solution to the problem
- **Crossover**: operators to merge two chromosomes
- **Mutation**: operators to modify a chromosome
- **Elitism**: a strategy that ensures the propagation of the elite member, requiring that
  - the elite member selected
  - a copy of it does not become disrupted by crossover or mutation.
- A new generation formed by ***Selecting*** some of parents and offspring and *Rejecting* others so as to keep the population size constant.

Exploitation and Exploration

- Two important issues in search strategies
- The tradeoff between *solution quality* and *convergent speed*
  - Hill-climbing: Exploiting the best solution
  - Random search: Exploring the search space
- GAs: a general-purpose search method
  - Exploration: initial population, similarity control, crossover, mutation,
  - Exploitation: fitness function, crossover, mutation

Factors on the Convergent Speed

- Population size
- Selection scheme
- Crossover operator applying
- Mutation rate
- Forbidding of replicates
- Scaling procedures:
  - adjust objective function values to avoid rapid convergence
- ...

Procedure: Genetic Algorithms

begin
  $t \leftarrow 0$; // *t*: *i-th* generations
  initialize $P(t)$; // *P*: parents
  evaluate $P(t)$;
  while (not termination condition) **do**
    recombine $P(t)$ to yield $C(t)$; // crossover, mutation
    evaluate $C(t)$; // *C*: children
    select $P(t+1)$ from $P(t)$ and $C(t)$; // selection
    $t \leftarrow t + 1$;
  end
end

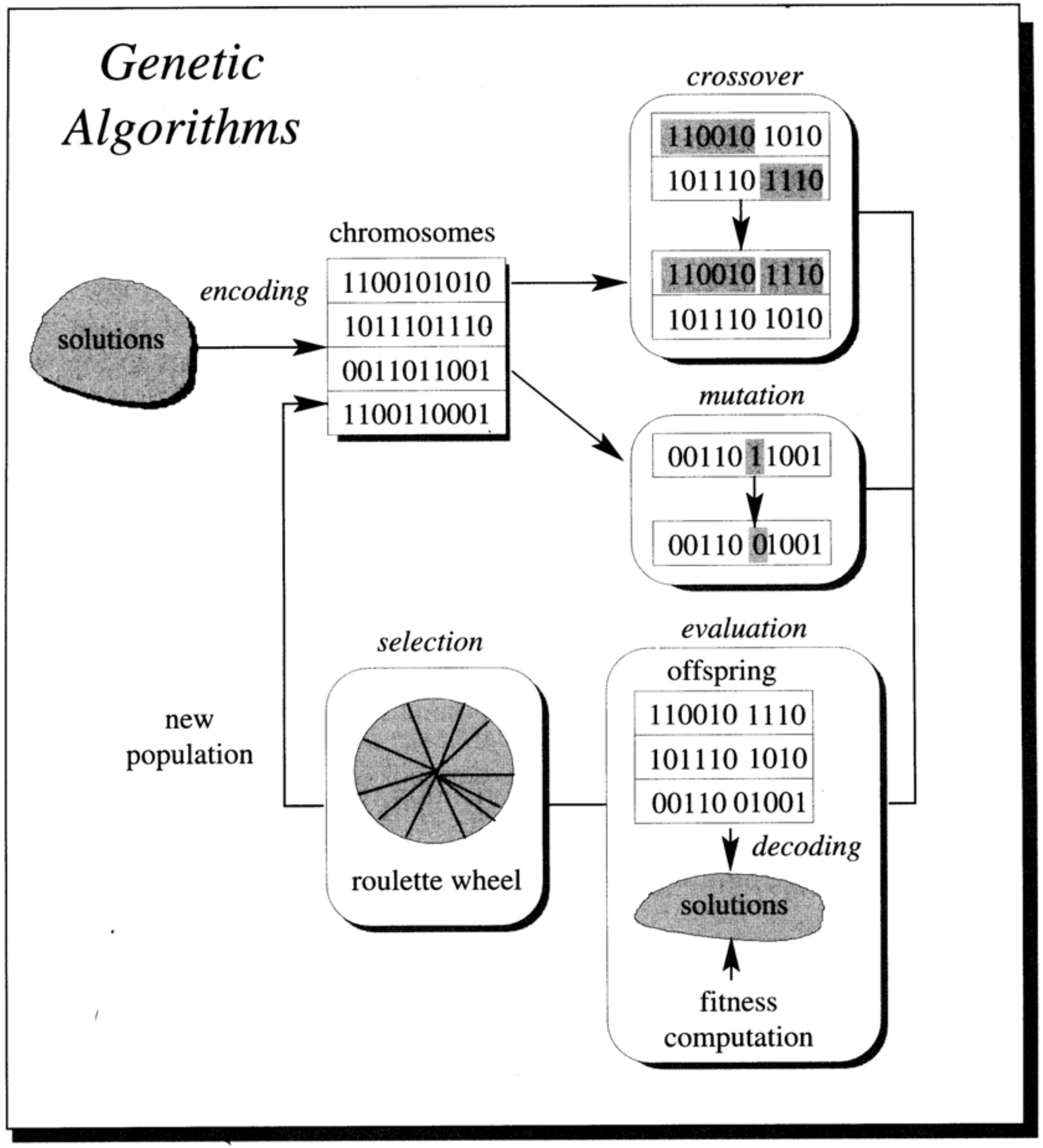


Figure 1.1. The general structure of genetic algorithms.

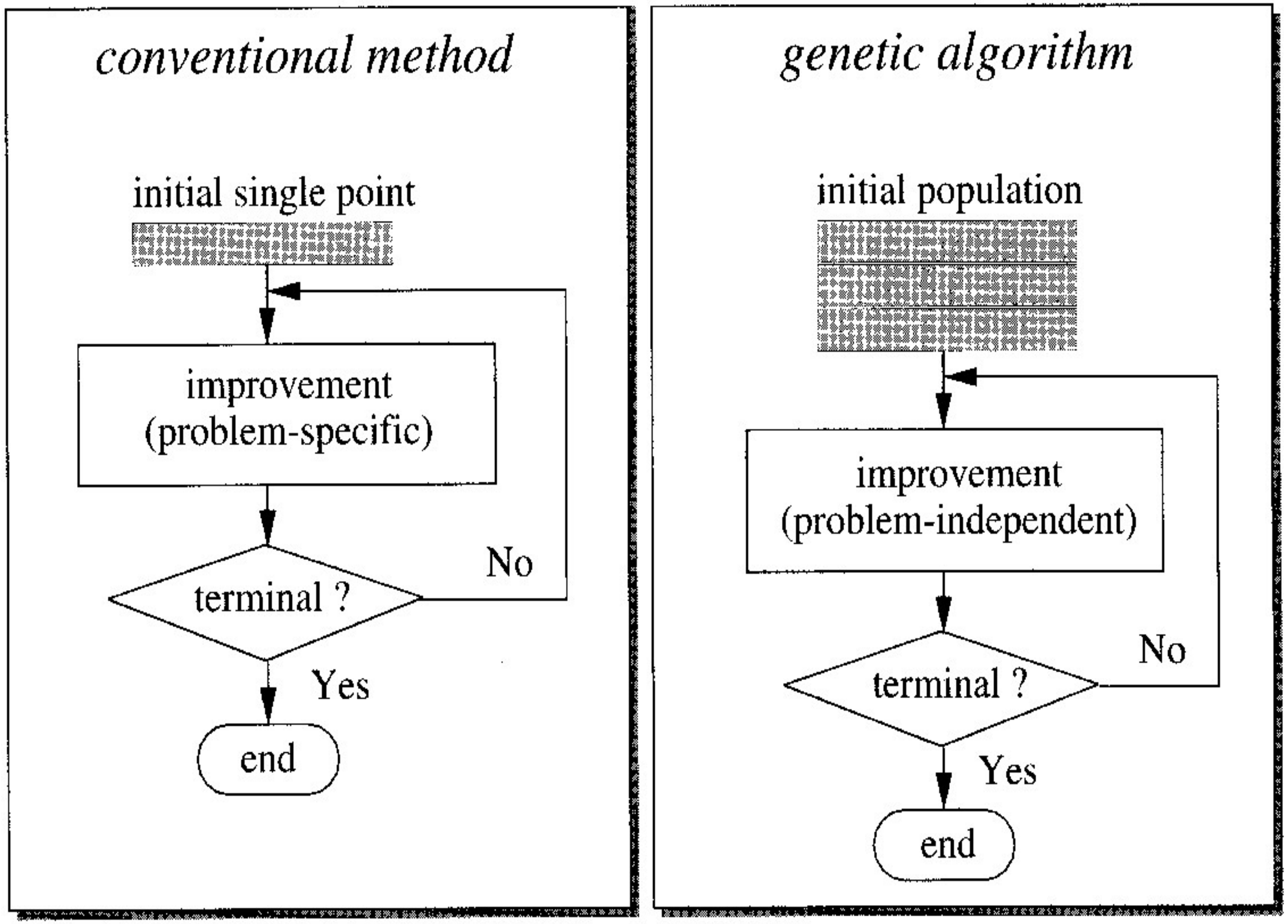


Figure 1.2. Comparison of conventional and genetic approaches.

Selection

- Based on Darwinian natural selection
- High selection pressure will lead to the search terminating prematurely.
- Low selection pressure will cause the process slower than necessary
- A possible solution
  - Low selection pressure at the start of genetic search, for exploration, high selection pressure at the end, for exploitation.

Two phases of selection

- Reproduction-selection
  - Select candidates (from parents) for mating (reproduction)
- Survival-selection
  - Select candidates to form the next generation
  - Normally, selection refers this phase

Selection Methods

- Proportional (roulette) selection:
  - Probability of selection is proportional to the individual's fitness.

Fitness proportionate selection:

$$\Pr(h_i) = \frac{Fitness(h_i)}{\Sigma_{j=1}^{p} Fitness(h_j)}$$

- Ranking method:
  - All Individuals are sorted, and probabilities of their selection are according to their ranking rather than their fitness.
- Tournament selection:
  - Some number, e.g., 2, of individuals compete for selection
  - The competition step is repeated *popsize* times for each generation.
  - More diverse

Issues for Selection phase

- Sampling space:
  - *Regular sampling space* and *Enlarged Sampling space*
- Sampling mechanism: how to select chromosomes from sampling space
  - *Stochastic sampling*: based on its survival probability,
    - Ex: roulette wheel selection
  - *Deterministic sampling*: select the best k individuals
  - Mixed sampling
- Selection probability
  - Scaling (or ranking) mechanisms: to maintain a reasonable differential so as to prevent a too-rapid premature
  - Static scaling and Dynamic scaling, to adjust the selective pressure.

Enlarged sampling space

- Both parents and children have the same chance of competing for survival
- $(\mu+\lambda)$ selection:
  - μ *parents* and λ *offspring* compete for survival and the μ best out of offspring and old parents are selected as parents of the next generation.
- Vs. (μ, λ) selection: regular sampling
  - Select μ best offspring as parents of the next generation, where $\mu < \lambda$.

The word-matching problem: "tobeornottobe"

The representation: a~z➔97~122

[116, 111, 98, 101, 111, 114, 110, 111, 116, 116, 111, 98, 101]

Generate an initial population of 10 random phrases as follows:

[114, 122, 102, 113, 100, 104, 117, 106, 97, 114, 100, 98, 101]
[110, 105, 101, 100, 119, 118, 121, 118, 106, 97, 104, 102, 106]
[115, 99, 121, 117, 101, 105, 115, 111, 115, 113, 118, 99, 98]
[102, 98, 102, 118, 114, 97, 109, 116, 101, 107, 117, 118, 115]
[107, 98, 117, 113, 114, 116, 106, 116, 106, 101, 110, 115, 98]
[102, 119, 121, 113, 121, 107, 107, 116, 122, 121, 111, 106, 104]
[116, 98, 120, 98, 108, 115, 111, 105, 122, 103, 103, 119, 109]
[101, 111, 111, 117, 114, 104, 100, 120, 98, 118, 116, 120, 97]
[100, 116, 114, 105, 117, 111, 115, 114, 103, 107, 109, 98, 103]
[106, 118, 112, 98, 103, 101, 109, 116, 112, 106, 97, 108, 113]

Now, we convert this population to string to see what they look like:

rzfqdhujardbe
niedwvyvjahfj
scyueisosqvcb
fbfvramtekuvs
kbuqrtjtjensb
fwyqykktzyojh
tbxblsoizggwm
dtriuosrgkmbg
jvpbgemtpjalq

- Fitness function: # of matched letters
- Selection: the top 50% better individuals
- Search space: 26^{13}
- # of generations to match: 23

The best String for Each Generation

Generation	String	Fitness	Generation	String	Fitness
1	rzfqdhujardbe	2	16	rzbwornottobe	10
2	rzfqdhuoardbe	3	17	rzbwornottobe	10
3	rzfqghuoatdbe	4	18	rzbwornottobe	10
4	rzfqghuoztobe	5	19	rzbwornottobe	10
5	rzfqghhottobe	6	20	robwornottobe	11
6	rzfqohhottobe	7	21	tobwornottobe	12
7	rzfqohnottobe	8	22	tobwornottobe	12
8	rzfqohnottobe	8	23	tobeornottobe	13
9	rzfqohnottobe	8	24	tobeornottobe	13
10	rzfqohnottobe	8	25	tobeornottobe	13
11	rzfqornottobe	9	26	tobeornottobe	13
12	rzfqornottobe	9	27	tobeornottobe	13
13	rwfwornottobe	9	28	tobeornottobe	13
14	rwcwornottobe	9	29	tobeornottobe	13
15	rzcwornottobe	9	30	tobeornottobe	13

Hybrid GAs

- *Genetic algorithms* are used to perform global exploration among population, while *heuristic methods,* e.g. hill_climbing*,* are used to perform local exploitation around chromosomes.
- Try to inject some “smarts” into the offspring before returning it to be evaluated.

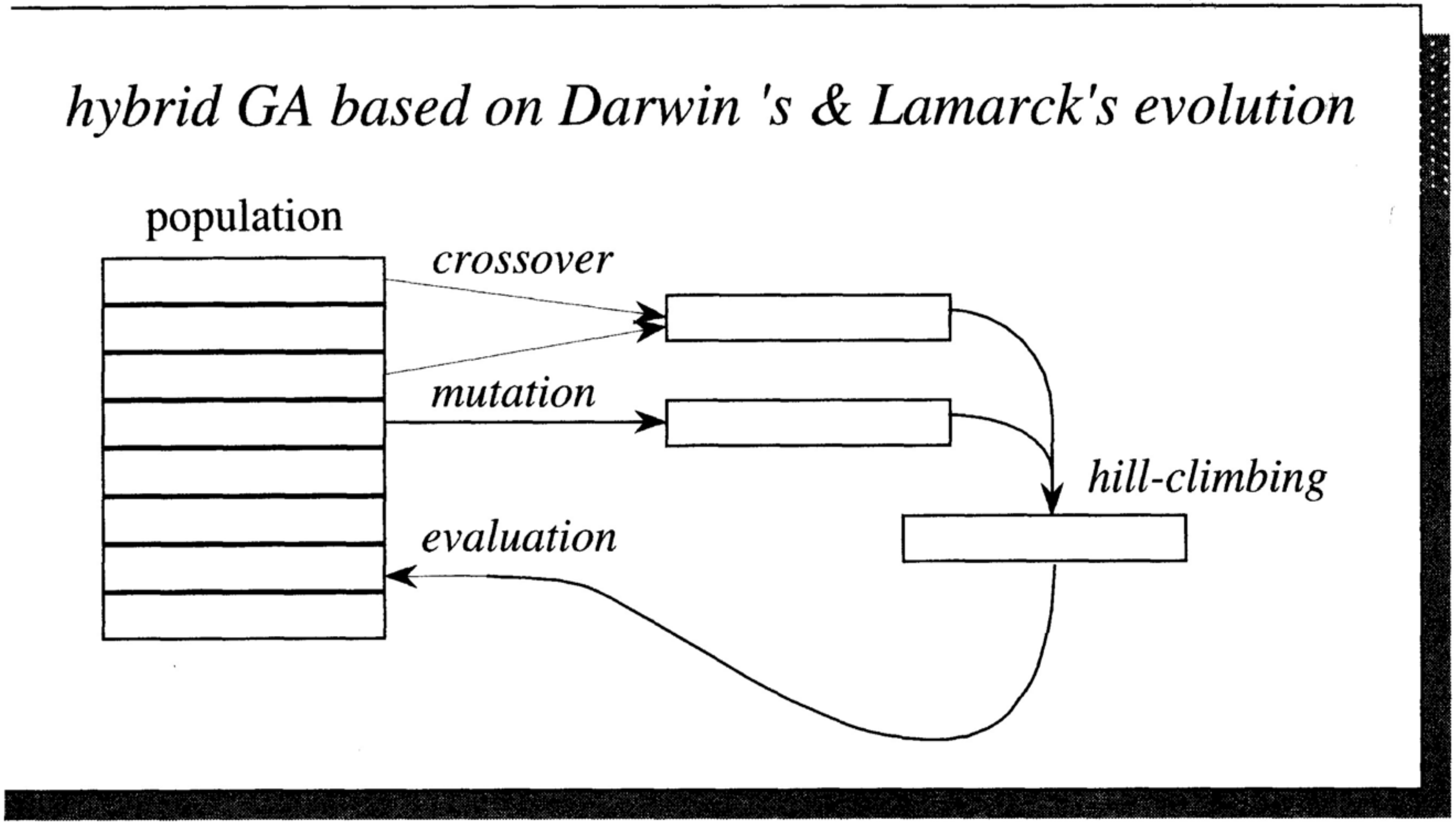


Figure 1.10. General structure of hybrid genetic algorithms.

GAs for TSP

A GA for TSP

- Representations:
  - *permutations* instead of binary strings
- Crossover:
  - OX, CX, PMX, …
- Mutation:
  - hill-climbing
- Selection:
  - The lower cost individuals

Algorithm 5.26: GENETICTSP $(popsize, c_{max})$

external SELECT(), REC()

$c \leftarrow 1$

$[P_0, \ldots, P_{popsize-1}] \leftarrow$ SELECT$(popsize)$

Sort $P_0, P_1, \ldots, P_{popsize}$ in increasing order of cost

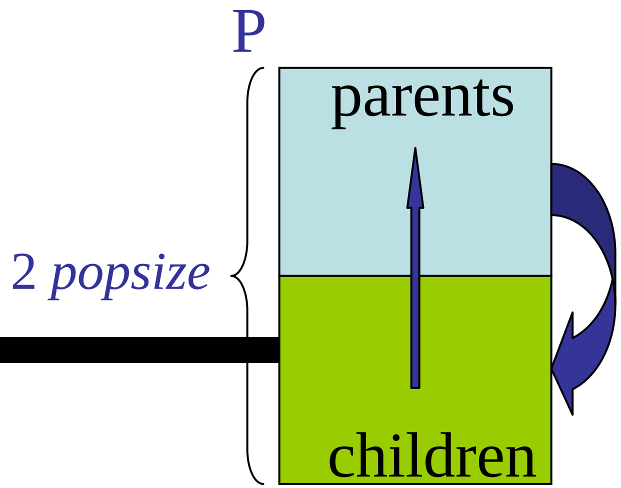
$X_{best} \leftarrow P_0$

$BestCost \leftarrow \mathsf{C}(P_0)$

while $c \leq c_{max}$

do
- **for** $i \leftarrow 0$ **to** $popsize/2 - 1$ // crossover, mutation
  - **do** $(P_{popsize+2i}, P_{popsize+2i+1}) \leftarrow \mathsf{Rec}(P_{2i}, P_{2i+1})$
- Sort $P_0, P_1, \ldots, P_{2 \cdot popsize-1}$ in increasing order of cost //sort
- $CurCost \leftarrow \mathsf{C}(P_0)$
- **if** $CurCost < BestCost$
  - **then** $X_{best} \leftarrow P_0$; $BestCost \leftarrow CurCost$
- $c \leftarrow c + 1$

return (X_{best})



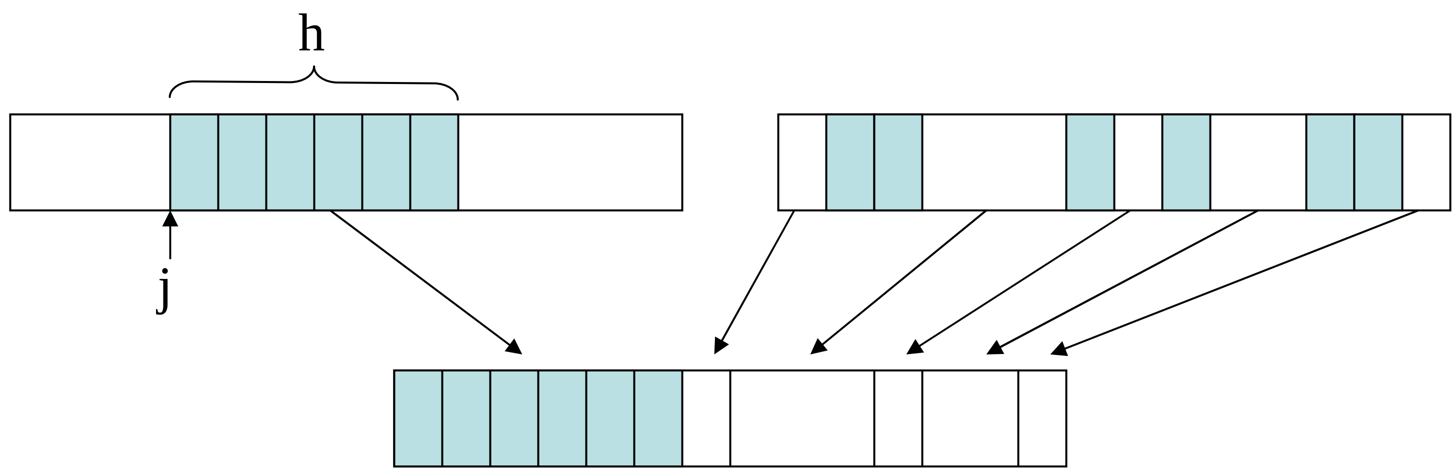
Variant Representations and Recombination solutions for TSP

- The binary string representations➔ permutations
- The typical one-point crossover may lead to infeasible solutions, e.g., not permutations
- A Solution: -- the PMRec algorithm
  - Choose two crossover points, 2, 5, randomly
  - Parents: A=[3,1,4,7,6,5,2,8], B=[8,6,4,3,7,1,2,5]
  - Transpositions of symbols:
  - 4⇔4, A=[3,1,4,7,6,5,2,8], B=[8,6,4,3,7,1,2,5]
  - 7⇔3, A=[7,1,4,3,6,5,2,8], B=[8,6,4,7,3,1,2,5]
  - 6⇔7, A=[6,1,4,3,7,5,2,8], B=[8,7,4,6,3,1,2,5]
  - 5⇔1, **C=[6,5,4,3,7,1,2,8], D=[8,7,4,6,3,5,2,1]**
- Other better solutions?

Algorithm 5.25: MGKREC (A, B)

external RandomInteger(), STEEPESTASCENTTWOOPT()
$h \leftarrow \text{RandomInteger}(10, \frac{n}{2})$
$j \leftarrow \text{RandomInteger}(0, n-1)$
$T \leftarrow \emptyset$
for $i \leftarrow 0$ **to** $h - 1$ **do** $\begin{cases} D[i] \leftarrow B[(i+j) \bmod n] \\ T \leftarrow T \cup \{D[i]\} \end{cases}$
for $j \leftarrow 0$ **to** $n - 1$
  do if $A[j] \notin T$
  then $\begin{cases} D[i] \leftarrow A[j] \\ i \leftarrow i + 1 \end{cases}$
STEEPESTASCENTTWOOPT(D)
$j \leftarrow \text{Random}(0, n-1)$
$T \leftarrow \emptyset$
for $i \leftarrow 0$ **to** $h - 1$ **do** $\begin{cases} C[i] \leftarrow A[(i+j) \bmod n] \\ T \leftarrow T \cup \{C[i]\} \end{cases}$
for $j \leftarrow 0$ **to** $n - 1$
  do if $B[j] \notin T$
  then $\begin{cases} C[i] \leftarrow B[j] \\ i \leftarrow i + 1 \end{cases}$
STEEPESTASCENTTWOOPT(C)
return (C, D)

A Better Crossover for TSP-- MGKRec



- Generate j, h randomly, where 0≤j≤n-1, 4≤h≤n/2
- EX: (j, h) = (5,4)
  - A=[3,1,4,7,6,5,2,8], B=[8,6,4,3,7,1,2,5]
  - C=[5,2,8,3,6,4,7,1]
- In a similar way, D=[2,5,8,6,3,1,4,7] for (j,h)=(6,4)
- Two strategies: remains the green part in the middle part or two ends.

Schema Theorem

Schema Theorem (1975,1989)

- *Schemata that are short, low-order, and above-average are given exponentially increasing numbers of trials in subsequent generations of a genetic algorithm.*
- theoretical foundations of genetic algorithms
- Introduced by *Holland* and popularized by *Goldberg*

Schema(ta)

- A schema describes a subset of strings with similarities at some string positions; i.e., it defines a subset of the search space.

- A template allowing exploration of similarities among chromosomes

Number of Schemata

- A given binary *real string* of length L: 2^L
  - E.g., 2^3 schemata for the string 101
- An alphabet of distinct characters k: $(k+1)^L$
  - E.g., 3^2 schemata for binary strings (k=2) of length 2
  - 00, 01, 0#, 10, 11, 1#, #0, #1, ##
- A population of N real strings: Nk^L
  - Actually, it will be always $< Nk^L$ because of sharing

Properties of Schemata

E.g., H_a = 1###, H_b = 1##0, H_c=#001

- Order(O): number of non-#,
  - reflecting how large the covering regions of space
  - the probability of a schema destroyed by a *mutation*
  - $O(H_a)=1$, $O(H_b)=2$, $O(H_c)=3$
- Length(L):
  - the difference between the first and the last non-# symbols,
  - the probability of a schema destroyed by a *crossover*
  - $O(H_a)=0$, $O(H_b)=3$, $O(H_c)=2$
- Fitness(F): the average fitness of all strings in the population matched by a schema, S, at time t

$$F(S, t) = \left[\sum_{i=1}^{p} f(v_i)\right] / p$$

$\{v_1, v_2, \ldots, v_p\}$: p strings in a population matched by S

Schema Growth Equation (consider just selection)

- $\bar{f}(t)$ = average fitness of pop. at time t
- $m(s,t)$ = instances of schema s in pop at time t
- $\hat{u}(s,t)$ = ave. fitness of instances of s at time t

Probability of selecting h in one selection step

$$\Pr(h) = \frac{f(h)}{\sum_{i=1}^{n} f(h_i)} = \frac{f(h)}{n\bar{f}(t)}$$

Probabilty of selecting an instance of s in one step

$$\Pr(h \in s) = \sum_{h \in s \cap p_t} \frac{f(h)}{n\bar{f}(t)} = \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)$$

Expected number of instances of s after n selections

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Schema Theorem

selection crossover mutation

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \left(1 - p_c \frac{d(s)}{l-1}\right) (1-p_m)^{o(s)}$$

- $m(s,t)$ = instances of schema s in pop at time t
- $\bar{f}(t)$ = average fitness of pop. at time t
- $\hat{u}(s,t)$ = ave. fitness of instances of s at time t
- p_c = probability of single point crossover operator
- p_m = probability of mutation operator
- l = length of single bit strings
- $o(s)$ number of defined (non "*") bits in s
- $d(s)$ = distance between leftmost, rightmost defined bits in s

GAs for Machine Learning

Part I: Classification

Machine Learning Using GAs

- To discover input-output mapping for a given, usually complex, *system* (a set of input-output samples)
  - To come up with an appropriate form of a function or a model, simpler than the given system
- Expect that the population of *classifiers* converges to some rules with very high *strength*.
- Successful applications:
  - Pattern classification, control, and prediction

Classification Model

- The description of the model (a conjunction normal form)

  $((A_1=x) \wedge (A_5=s)) \vee ((A_1=y) \wedge (A_4=n)) \Rightarrow C_1$

  $((A_3=y) \wedge (A_4=n)) \vee (A_1=x) \Rightarrow C_2$

- The training or learning data set is described with a set of attributes where each attribute has its categorical *range* (a set of possible values)
- Table 10.4

Classification rules (classifiers)

- The general form of each classifier

  $$(p_1, p_2, \ldots, p_{\#\text{ of attributes}}): d$$

- Generate classifiers from models
  - $((A_1=x) \wedge (A_5=s)) \vee ((A_1=y) \wedge (A_4=n)) \Rightarrow C_1$
    - (x***s*): C_1
    - (y**n**): C_1
  - $((A_3=y) \wedge (A_4=n)) \vee (A_1=x) \Rightarrow C_2$
    - (**yn**): C_2
    - (x*****): C_2

Fitness of classifiers: Strength

- Strengths S_i are proportional to the percentage of the data set supported by the classifier (rule).
- GAs try to optimize *the set of rules* with respect to the fitness function of the rules to training data set.

Mutation

- Randomly choose a position *i*, e.g., 2,
- Randomly choose a value from the domain of p_i, e.g., *
- The strength of the offspring is usually the same as that of its parents.

- E.g., (xy****):1, s=8.7
  ➔(x*****):1, s=8.7

Crossover

- Select two parents.
- Generate a random crossover-position point, e.g., 3.
- The strength of the offspring is an average of the parents' strengths.
- E.g.,
  - Parents: (*** ms*):1; (**y **n):0
  - Offspring: (*** **n):0; (**y ms*):1

GAs for Machine Learning

Part II: Concept Learning

Fitness Functions

- To learn classification rule
  - The classification accuracy of the individual (rule, hypothesis) over a set of provided training examples
- To learn a program for solving *block problems*
  - The number of training examples the individual can solve (*Genetic Programming*)
- To learn a strategy for playing a game
  - The number of games won by the individual (strategy) when playing against other individuals in the current population. (*Genetic Programming*)

Representing Hypotheses—*Bit String Representation*

Represent

$$(Outlook = Overcast \vee Rain) \wedge (Wind = Strong)$$

by

Outlook	*Wind*
011	10

Outlook = {sunny, overcast, rain}

Wind = {strong, weak}

PlayTennis = {yes, no}

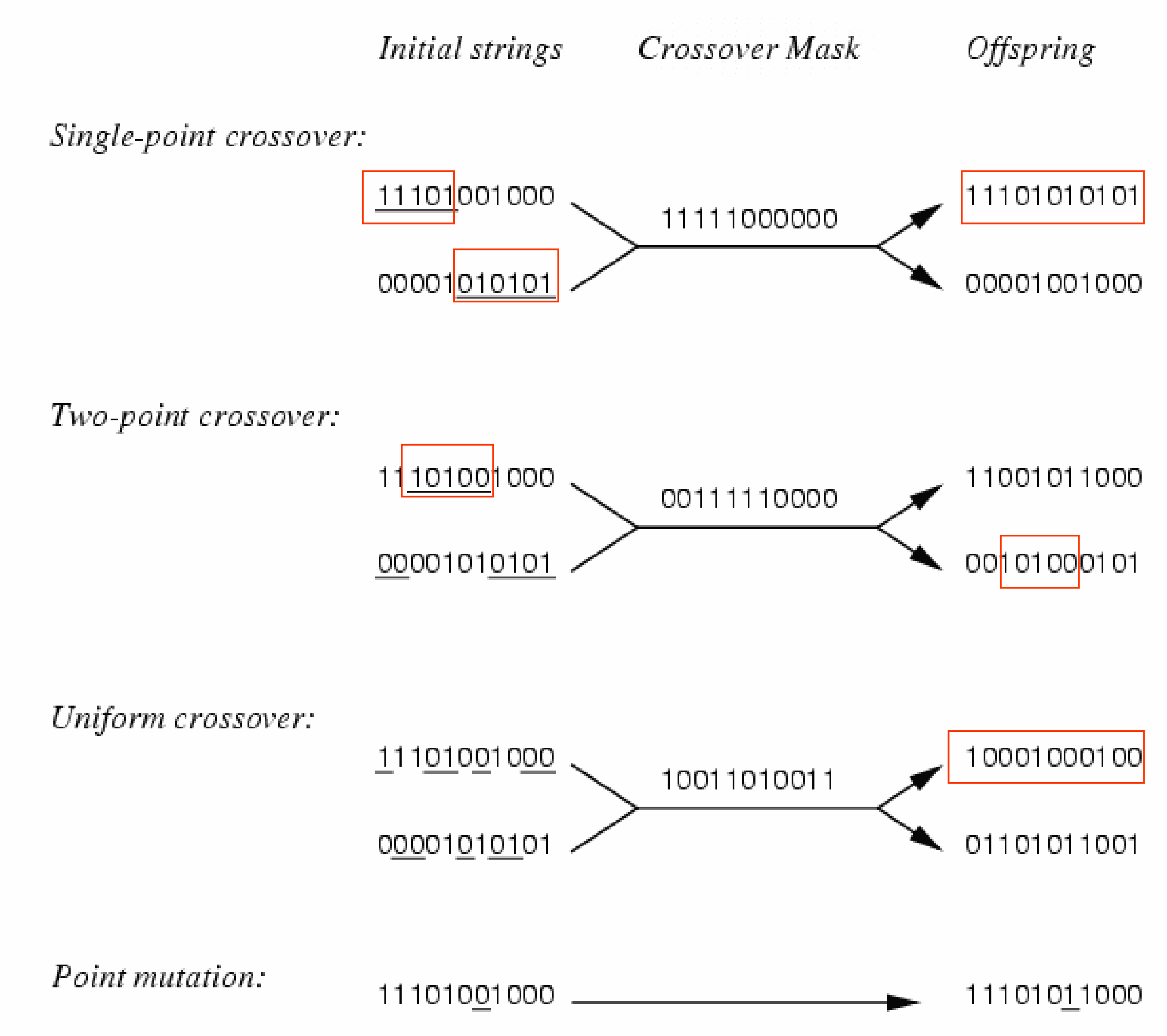
Represent

$$\text{IF} \;\; Wind = Strong \quad \text{THEN} \;\; PlayTennis = yes$$

by

Outlook	*Wind*	*PlayTennis*
111	10	10

Operators for GAs



Example for Concept Learning *GABIL (1993)*

Fitness:

$$Fitness(h) = (correct(h))^2$$

the percent of all training examples correctly classified by *h*

Representation:

IF $a_1 = T \wedge a_2 = F$ THEN $c = T$; IF $a_2 = T$ THEN $c = F$

represented by

a_1	a_2	c	a_1	a_2	c
10	01	1	11	10	0

Crossover with variable-length bit string

Start with

$$\begin{array}{rccc@{\qquad}ccc} & a_1 & a_2 & c & a_1 & a_2 & c \\ h_1: & 10 & 01 & 1 & 11 & 10 & 0 \\ & & & & & & \\ h_2: & 01 & 11 & 0 & 10 & 01 & 0 \end{array}$$

1. choose crossover points for h_1, e.g., after bits 1, 8
2. now restrict points in h_2 to those that produce bitstrings with well-defined semantics, e.g., $\langle 1, 3 \rangle$, $\langle 1, 8 \rangle$, $\langle 6, 8 \rangle$.

if we choose $\langle 1, 3 \rangle$, result is

$$\begin{array}{rccc} & a_1 & a_2 & c \\ h_3: & 11 & 10 & 0 \end{array}$$

$$\begin{array}{rccc@{\qquad}ccc@{\qquad}ccc} & a_1 & a_2 & c & a_1 & a_2 & c & a_1 & a_2 & c \\ h_4: & 00 & 01 & 1 & 11 & 11 & 0 & 10 & 01 & 0 \end{array}$$

Enable offspring to contain a different number of rules than their parents

GABIL Extensions

- Add two GA operators
  - AddAlternative (AA):
    - Generalize constraint on a_i by changing **a** 0 to 1 in a substring.
  - DropCondition (DC):
    - Completely dropping the constraint on a_i by replacing all bits for a_i by a 1.
- Furthermore, add two bits to determine which of the operators can be applied to the hypotheses.

a_1	a_2	c	a_1	a_2	c	AA	DC
01	11	0	10	01	0	1	0

GABIL Results

Performance of GABIL comparable to symbolic rule/tree learning methods C4.5, ID5R, AQ14

Average performance on a set of 12 synthetic problems:

- GABIL without AA and DC operators: 92.1% accuracy
- GABIL with AA and DC operators: 95.2% accuracy
- symbolic learning methods ranged from 91.2 to 96.6

Genetic Programming

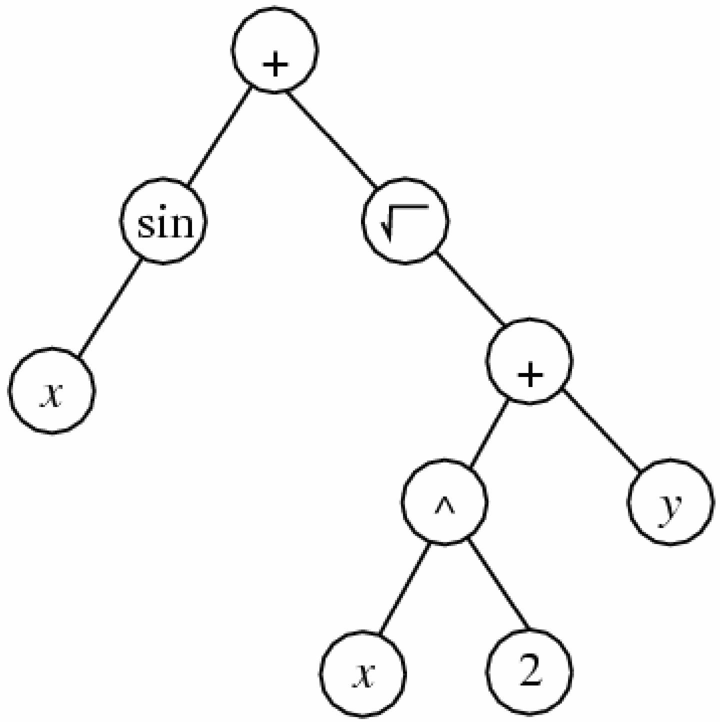
Genetic Programming (GP)

- The individuals are *computer programs* rather than bit strings.
- The programs are typically represented by trees corresponding to the parse tree of the program.
- The fitness of a individual program is determined by executing the program on a set of training data.
- Program tree representation.#36

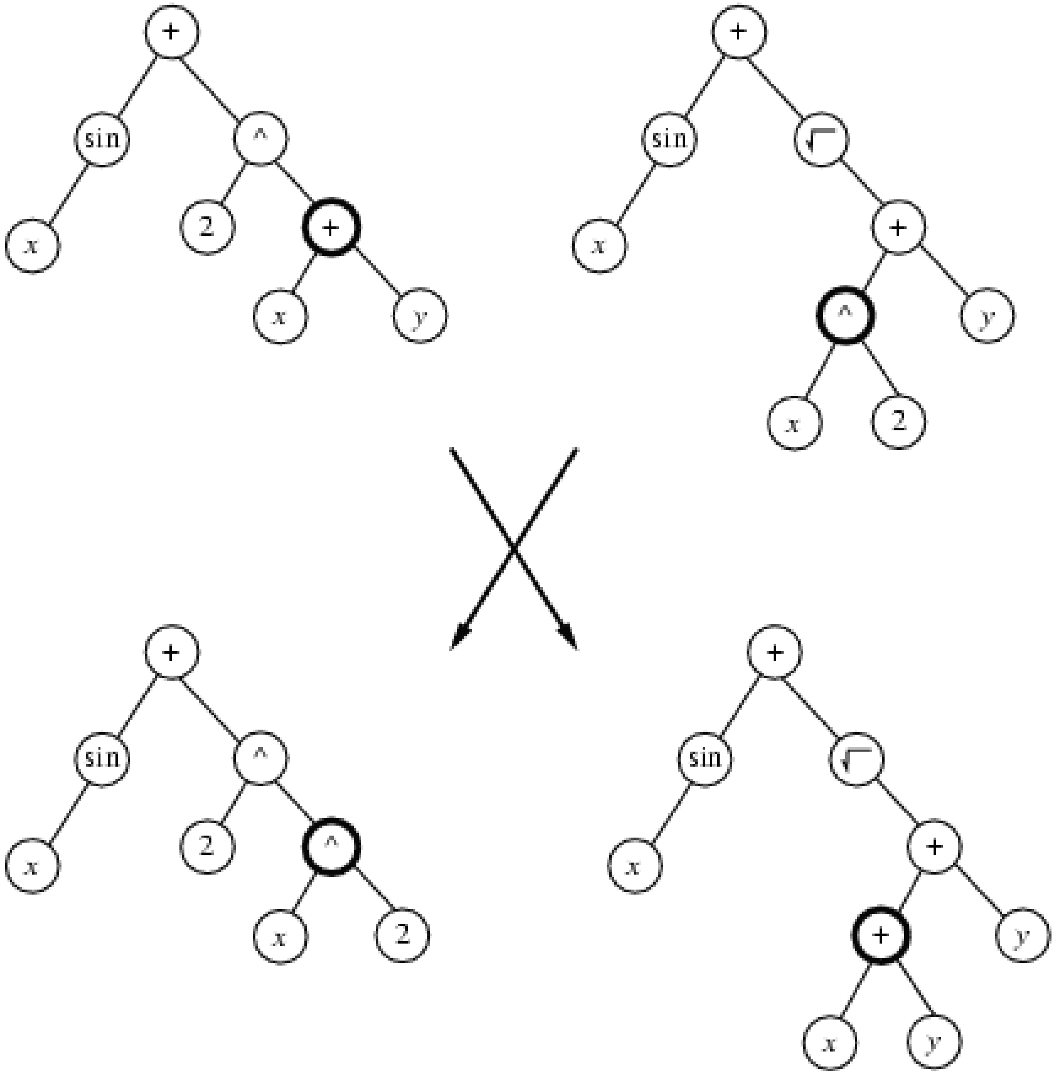
Program tree representation

Population of programs represented by trees

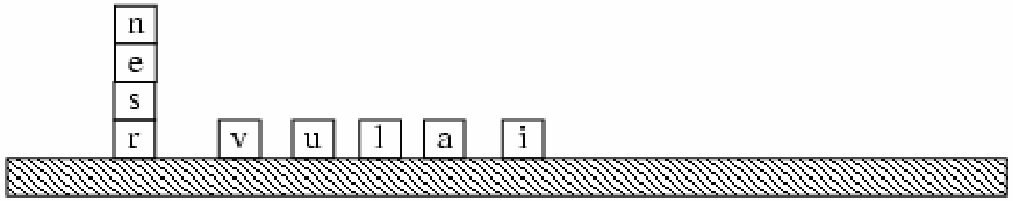
$$\sin(x) + \sqrt{x^2 + y}$$



Crossover Operation



Block Problems



Goal: spell UNIVERSAL

Terminals:

- CS ("current stack") = name of the top block on stack, or F.
- TB ("top correct block") = name of topmost correct block on stack
- NN ("next necessary") = name of the next block needed above TB in the stack

Primitive functions

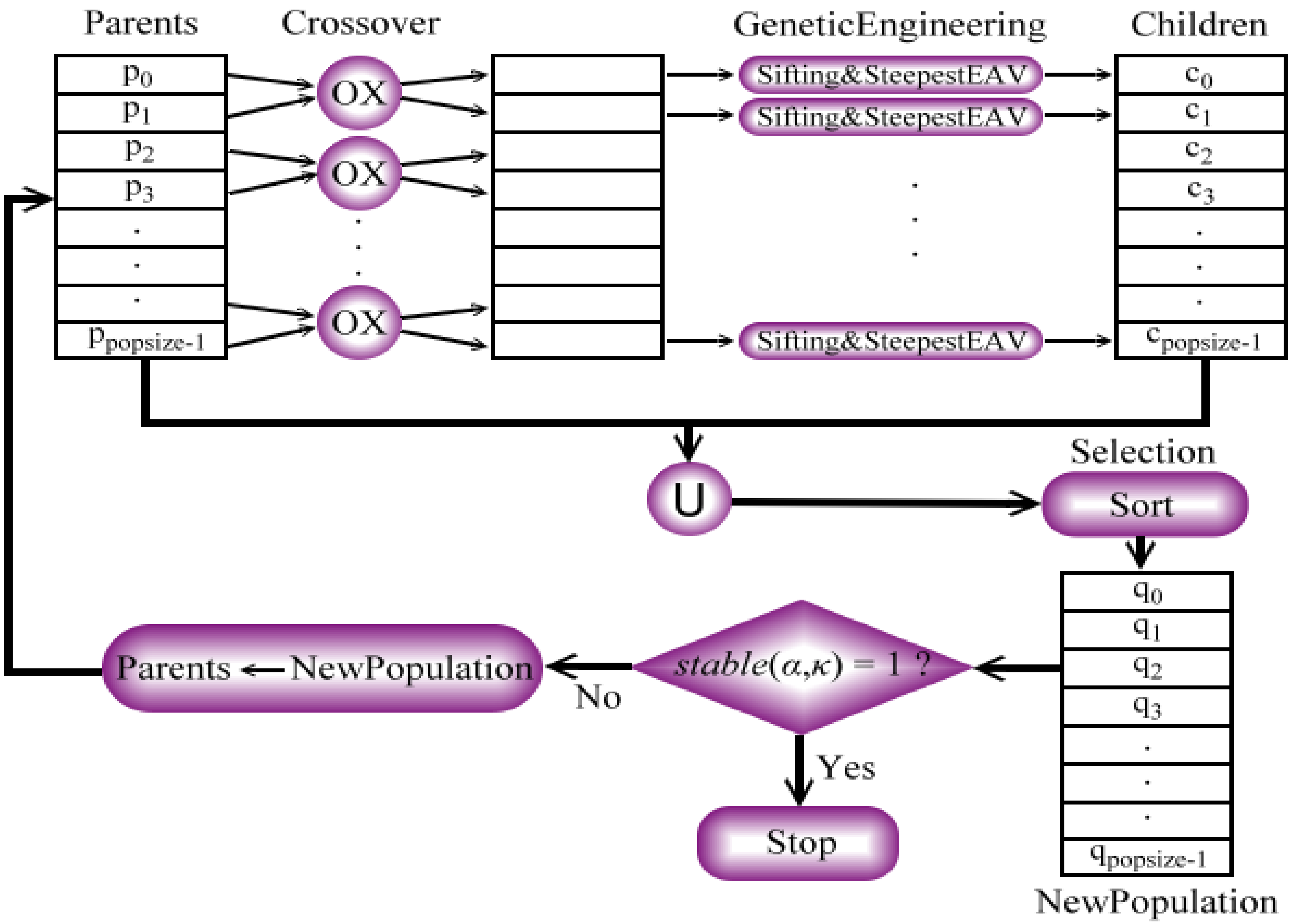
- (MS x): ("move to stack"), if block x is on the table, moves x to the top of the stack and returns the value T. Otherwise, does nothing and returns the value F.
- (MT x): ("move to table"), if block x is somewhere in the stack, moves the block at the top of the stack to the table and returns the value T. Otherwise, returns F.
- (EQ x y): ("equal"), returns T if x equals y, and returns F otherwise.
- (NOT x): returns T if $x = F$, else returns F
- (DU x y): ("do until") executes the expression x repeatedly until expression y returns the value T

Learned Program

- Goal: train an individual (program) to fit 166 training examples (block problems)
- Fitness: The number of training examples the individual can solve
- Initialize 300 random programs
- After 10 generations, the system discovered the following program, which can solve all 166 problems.

```
(EQ (DU (MT CS)(NOT CS)) (DU (MS NN)(NOT NN)) )
```

EBEA



Parents
Crossover
GeneticEngineering
Children
p0
p1
p2
p3
ppopsize-1
OX
OX
OX
Sifting&SteepestEAV
Sifting&SteepestEAV
Sifting&SteepestEAV
c0
c1
c2
c3
cpopsize-1
U
Selection
Sort
q0
q1
q2
q3
qpopsize-1
NewPopulation
stable(α,κ) = 1 ?
No
Yes
Stop
Parents ← NewPopulation

Operators for GAs