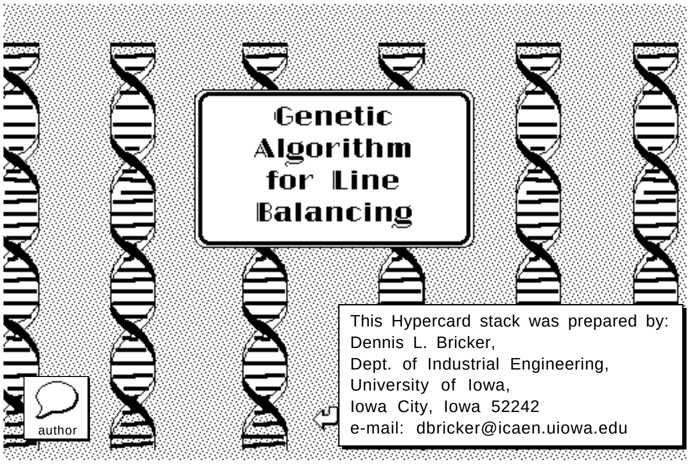


- Generic genetic algorithm
- Genetic algorithm for the assembly line balancing problem

Genetic Algorithms (GAs)

- heuristic methods--do not guarantee optimality
- iterative methods
- simulate biological evolution, which has enabled "nature" to develop species remarkably well-adapted to their environment.

©Dennis Bricker, U. of Iowa, 1997

GAs

- development & theoretical foundation provided by:
  John Holland, *Adaption in Natural and Artificial Systems,* The University of Michigan Press, 1975.
- further developed & analyzed by Holland's student:
  David Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, 1989.

©Dennis Bricker, U. of Iowa, 1997

GAs

- operate on a population of *individuals* which represent potential solutions to the problem
- from the population, a set of "good" individuals are selected to *mate* & form a new *generation*
- the quality of subsequent generations will (it is hoped!) gradually improve and approach optimality

©Dennis Bricker, U. of Iowa, 1997

- Encoding
- Fitness measure
- Selection
- Scaling fitness
- Crossover
- Mutation
- Generic GA Algorithm

©Dennis Bricker, U. of Iowa, 1997

ENCODING

Each individual in the population consists of a string, which "resembles" a biological *chromosome* composed of *genes*.

These genes may take on values from a finite set of digits or characters, called *alleles*. (Most frequently, the alleles are binary digits.)

Example: from the set of alleles {A,B,C} we might produce the individual ABCABBCABBC

©Dennis Bricker, U. of Iowa, 1997

FITNESS

For each individual i, we define a *fitness* value f(i) to measure how good it is.

Individuals representing good solutions will have a larger fitness value than those which correspond to poor solutions.

Used to determine which individuals *survive* to produce the next generation.

"Survival of the fittest..."

©Dennis Bricker, U. of Iowa, 1997

SELECTION

From the population at each generation, a GA randomly selects the fittest individuals to survive and mate to produce the next generation.

The simplest method for doing this is "stochastic sampling with replacement", in which individual i is selected with probability $p_i = f(i) \Big/ \sum_{j=1}^{N} f(j)$

This is repeated until N individuals have been selected.

©Dennis Bricker, U. of Iowa, 1997

In sampling with replacement, the more fit individuals are likely to be selected, and may be selected several times. However, there is not guarantee that an individual whose fitness is above the population average will be selected

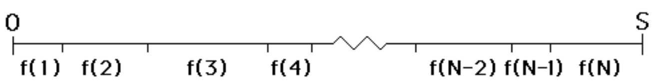
"Stochastic universal sampling" is a scheme which will guarantee this!

J.E. Baker, "Reducing bias and inefficiency in the selection algorithm", in ***Genetic Algorithms & Their Applications: Proceedings of the 2nd Int'l Conf. on Genetic Algorithms***, J.J. Grefenstette (ed.), 1987.

©Dennis Bricker, U. of Iowa, 1997

1. Compute S, the sum of the fitness values for the population.

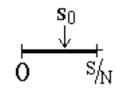
2. Map each fitness value in random order to contiguous segments of the interval [0,S] on the real line, such that each segment has length equal to its corresponding fitness value.



Stochastic Universal Sampling

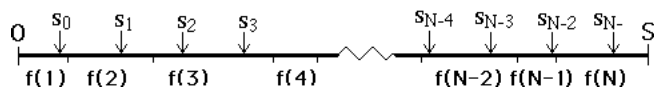
©Dennis Bricker, U. of Iowa, 1997

3. Generate a random number s_0 within the interval $[0, S/N]$, the "average" interval.



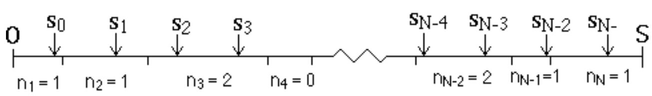
4. Compute the N equally-spaced numbers

$$s_i = s_0 + i \times \frac{S}{N},\ i = 0, 1, 2, \cdots N-1$$



©Dennis Bricker, U. of Iowa, 1997

5. Let n_k be the number of values in the set $\{s_i\}$ which fall within interval k.



6. For each k (k=1,2,...N), create n_k copies of individual k and put into the mating pool.

©Dennis Bricker, U. of Iowa, 1997

SCALING

The distribution of fitness values may strongly affect premature convergence of a genetic algorithm.

If a few individuals have much larger fitness values than the others, they will generate many more offspring and quickly dominate the generations which follow, and the algorithm will prematurely converge to a population of identical "clones" which are not optimal.

©Dennis Bricker, U. of Iowa, 1997

A way to prevent this is to scale the fitness values so as to maintain a constant "best fitness to average fitness" ratio, the scaling factor.

Given scaling factor λ and a set of fitness values $\{f_1, f_2, \ldots f_N\}$, we scale them to $F_i = a f_i + b$ where a & b are chosen so that

$$\boxed{\sum_{i=1}^{N} F_i = 1} \quad \text{and} \quad \boxed{\lambda = \frac{\max_i F_i}{\sum_{i=1}^{N} F_i / N}}$$

©Dennis Bricker, U. of Iowa, 1997

Solution: Choose a & b so that

$$\boxed{\begin{aligned} a &= \frac{\lambda - 1}{N f_{max} - \sum_{i=1}^{N} f_i} \\ b &= \lambda/N - a f_{max} \end{aligned}}$$

where $f_{max} \equiv \max_i \{f_i\}$

©Dennis Bricker, U. of Iowa, 1997

CROSSOVER

Two individuals can mate to generate two offspring, by randomly selecting a break point and exchanging substrings:

ABCAA|BBCC → ABCAA BBCC → ABCAA acba
cbacb|acba → cbacb acba → cbacb BBCC

break point

©Dennis Bricker, U. of Iowa, 1997

MUTATION

With relatively low probability, a gene on a chromosome might change its value from one allele to another.

ABCAABBCC → ABCAABBCC → ACCAABBCC

A gene is selected at random

New value of gene is selected at random

©Dennis Bricker, U. of Iowa, 1997

The GA is initiated with a random population of N individuals

At each iteration, the GA does the following:

1. Evaluates the fitness of each individual in the population
2. Based on fitness, chooses individuals from the population based on fitness, to form a *mating pool*, and pairs these individuals randomly.

©Dennis Bricker, U. of Iowa, 1997

3. For each pair of individuals in the mating pool,
   either mates them & puts the two offspring into the new population
   or copies them directly into the new population
4. Carries out mutation on the new population
5. Checks whether to terminate.

Usually a genetic algorithm will terminate after a given number of iterations, or when nearly all of the individuals in the population are identical.

©Dennis Bricker, U. of Iowa, 1997

There are numerous decisions which much be made in the design of a genetic algorithm:

- population size
- probability of crossover
- probability of mutation
- whether to "seed" the initial population with "good" solutions
- scaling factor (best to average ratio) for fitness

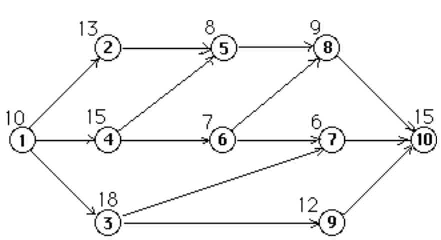
See sample Pascal code in Chapter 3: "Computer Implementation of a Genetic Algorithm", in book by D. Goldberg.

©Dennis Bricker, U. of Iowa, 1997

GENETIC ALGORITHM FOR LINE BALANCING

Consider the Assembly Line Balancing (ALB) Problem in which the number of stations n is fixed, and the tasks are to be assigned to the stations so as to minimize the cycle time.

©Dennis Bricker, U. of Iowa, 1997



EXAMPLE

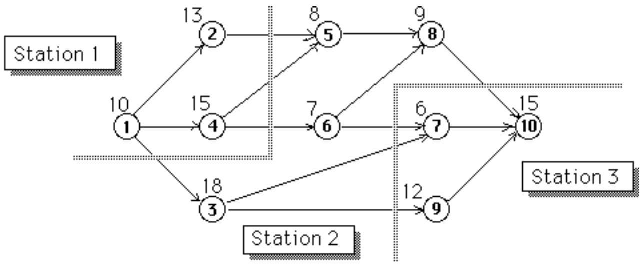
n = 3 stations
10 tasks

©Dennis Bricker, U. of Iowa, 1997

ENCODING

For the ALB problem, a natural encoding would be, using the station numbers as alleles, to identify the station to which task i is assigned by the number in the i^{th} position on the string.

©Dennis Bricker, U. of Iowa, 1997



The assignment shown above would be coded by the string:

1	1	2	1	2	2	3	2	3	3
1	2	3	4	5	6	7	8	9	10

ENCODING

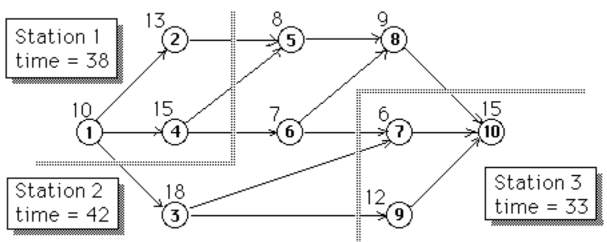
©Dennis Bricker, U. of Iowa, 1997

FITNESS

Our objective is to
- minimize the cycle time, while
- satisfy the precedence restrictions

The "fitness" measure of a solution should be a positive value which reflects both of these considerations, with a larger value for those solutions which better satisfy the objective & restrictions.

©Dennis Bricker, U. of Iowa, 1997



The assignment of the ten tasks to the three stations shown above results in a cycle time equal to $T_{max} = \max\{38, 42, 33\} = 42$.

©Dennis Bricker, U. of Iowa, 1997

The cycle time is to be minimized, and so is an indication of "unfitness", i.e., the larger the cycle time, the more unfit the solution is.

If T(i) is the cycle time of individual i, then the fitness could be measured by

$$f(i) = k \times M - T(i)$$

where

$$M = \underset{i}{\text{maximum}} \{T(i)\}$$

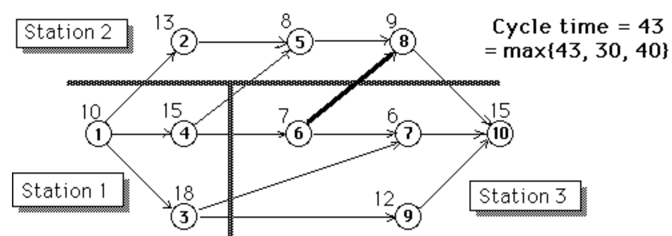
and k>1. With this definition, f(i)>0 for all i.

©Dennis Bricker, U. of Iowa, 1997

However, because individuals which violate one or more precedence restrictions might appear in the population, we will include a penalty P times the number V of such violations:

$$T(i) = T_{max} + P \times V$$

©Dennis Bricker, U. of Iowa, 1997



1	2	1	1	2	3	3	2	3	3
1	2	3	4	5	6	7	8	9	10

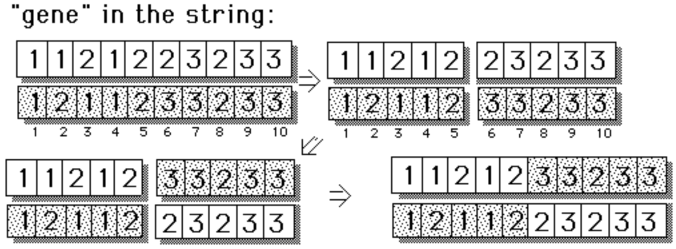
In the assignment shown, task 6 does not precede task 8 as required, and so we reduce the fitness by a penalty P=10:

$$T(i) = 43 + 10 \times 1 = 53$$

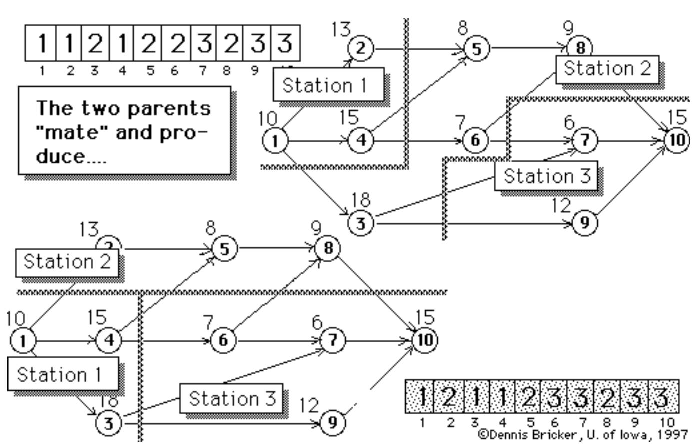
©Dennis Bricker, U. of Iowa, 1997

CROSSOVER

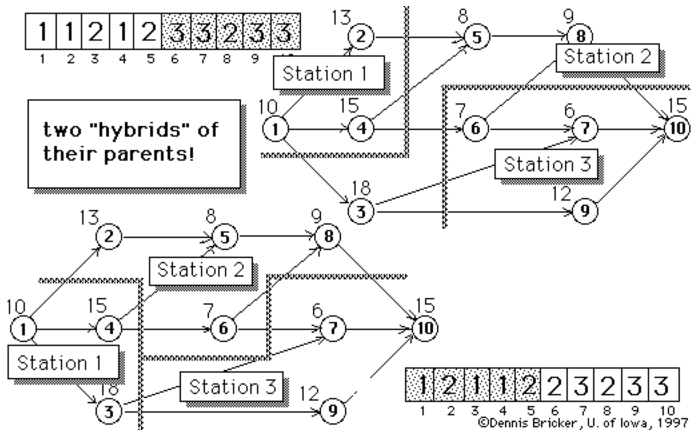
Suppose that the two individuals below have been paired for "mating", and the "break point" has been randomly chosen to be after the 5 "gene" in the string:



©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997

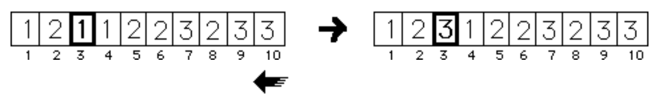


©Dennis Bricker, U. of Iowa, 1997

MUTATION

Mutation will be performed on each "gene" with some probability P_{mutate}, which is relatively small.

If a gene is randomly selected for mutation, it is assigned an allele selected uniformly from the set of alleles.



©Dennis Bricker, U. of Iowa, 1997

EXAMPLE

Consider the ALB problem used as an example earlier....

```
Genetic Algorithm Parameters

08:21:59 pm december 2, 1995
Nstations = 3
     = # of stations
Popsize = 20
     = # of individuals in population
Pcross = 0.8
     = P{crossover}
Pmutate = 0.005
      = P{mutation}
Penalty = 5
      = penalty per precedence violation
kkk = 1.5
```

©Dennis Bricker, U. of Iowa, 1997

At the 100th iteration, the algorithm converged:

the population is completely uniform!

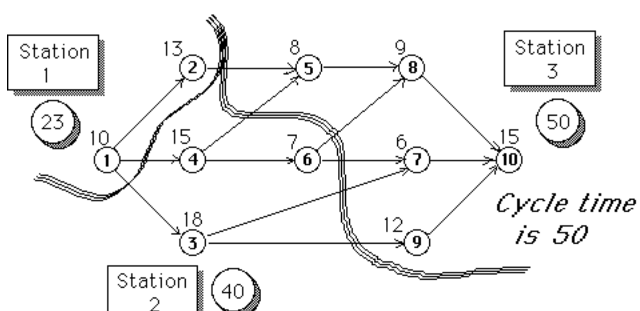
#	Fitness	Cycle time	# violations	Population
1	105	50	2	1 3 2 1 3 2 2 2 1 3
2	105	50	2	1 3 2 1 3 2 2 2 1 3
3	105	50	2	1 3 2 1 3 2 2 2 1 3
4	105	50	2	1 3 2 1 3 2 2 2 1 3
5	105	50	2	1 3 2 1 3 2 2 2 1 3
6	105	50	2	1 3 2 1 3 2 2 2 1 3
7	105	50	2	1 3 2 1 3 2 2 2 1 3
8	105	50	2	1 3 2 1 3 2 2 2 1 3
9	105	50	2	1 3 2 1 3 2 2 2 1 3
10	105	50	2	1 3 2 1 3 2 2 2 1 3
11	105	50	2	1 3 2 1 3 2 2 2 1 3
12	105	50	2	1 3 2 1 3 2 2 2 1 3
13	105	50	2	1 3 2 1 3 2 2 2 1 3
14	105	50	2	1 3 2 1 3 2 2 2 1 3
15	105	50	2	1 3 2 1 3 2 2 2 1 3
16	105	50	2	1 3 2 1 3 2 2 2 1 3
17	105	50	2	1 3 2 1 3 2 2 2 1 3
18	105	50	2	1 3 2 1 3 2 2 2 1 3
19	105	50	2	1 3 2 1 3 2 2 2 1 3
20	105	50	2	1 3 2 1 3 2 2 2 1 3

Incumbent:

1 1 2 2 3 2 3 3 3 3

None in final generation is feasible!

©Dennis Bricker, U. of Iowa, 1997



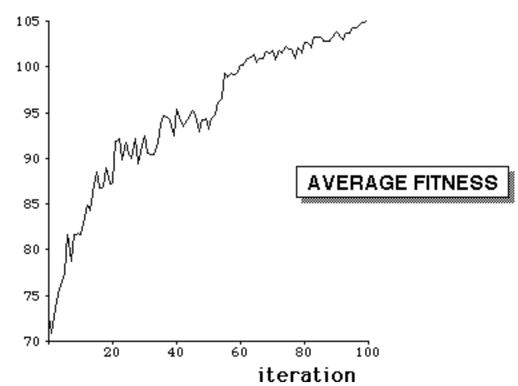
Cycle time is 50

Incumbent:

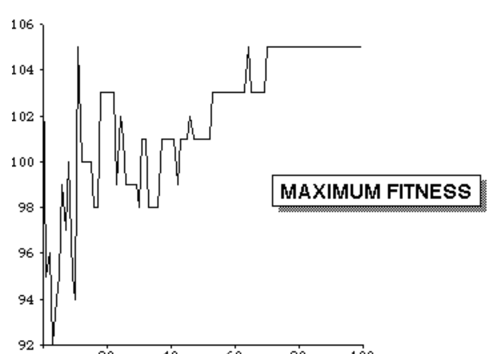
1 1 2 2 3 2 3 3 3 3

The incumbent was discovered in an earlier generation, but then disappeared!

©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997

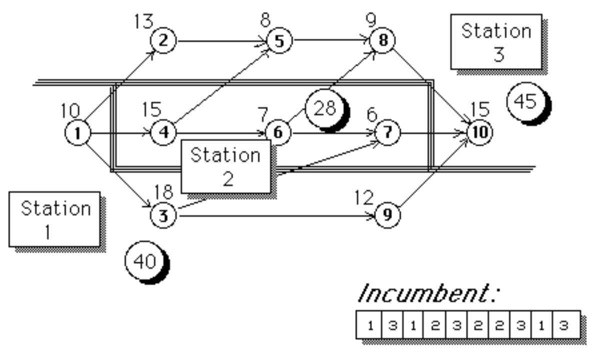
Running again, with a higher probability of crossover (90%)

Again, the algorithm converged

(population is not uniform, but has uniform fitness!)

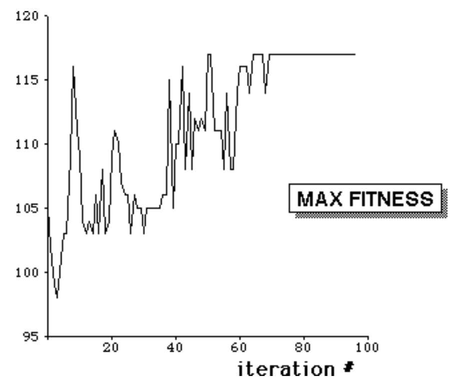
#	Fitness	Cycle time	# violations	Population
1	117	45	0	1 3 1 2 3 2 2 3 1 3
2	117	45	1	[illegible]
3	117	45	1	[illegible]
4	117	45	1	[illegible]
5	117	45	0	1 3 1 2 3 2 2 3 1 3
6	117	45	0	1 3 1 2 3 2 2 3 1 3
7	117	45	1	[illegible]
8	117	45	0	1 3 1 2 3 2 2 3 1 3
9	117	45	0	1 3 1 2 3 2 2 3 1 3
10	117	45	0	1 3 1 2 3 2 2 3 1 3
11	117	45	1	[illegible]
12	117	45	1	[illegible]
13	117	45	1	[illegible]
14	117	45	1	[illegible]
15	117	45	1	[illegible]
16	117	45	1	[illegible]
17	117	45	0	1 3 1 2 3 2 2 3 1 3
18	117	45	0	1 3 1 2 3 2 2 3 1 3
19	117	45	0	1 3 1 2 3 2 2 3 1 3
20	117	45	1	[illegible]

©Dennis Bricker, U. of Iowa, 1997

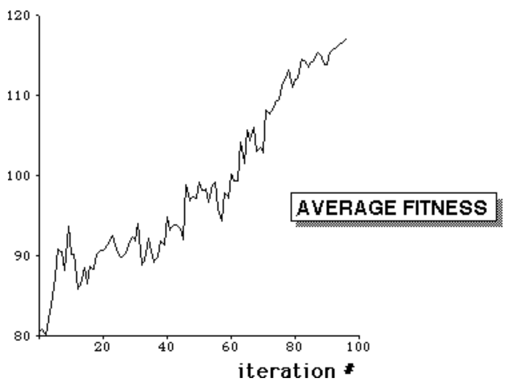


Cycle time is 45

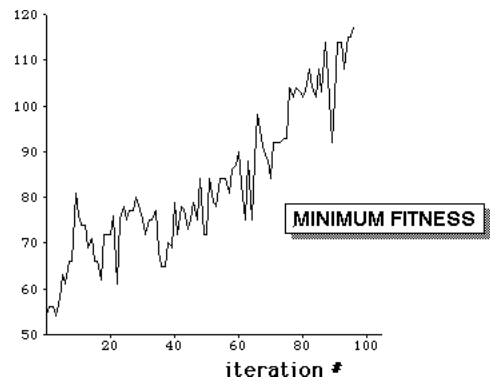
©Dennis Bricker, U. of Iowa, 1997



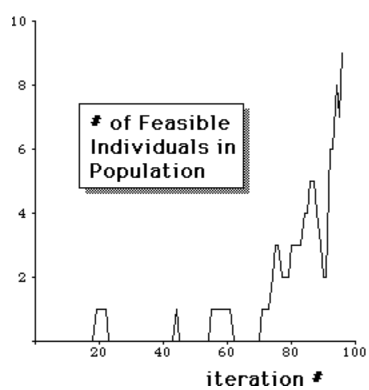
©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997

Running the GA again with Pcross=90%, it converged in 88th generation:

generation	Fitness min	Fitness max	Fitness average	#feasible
72	91	110	101.90	1
73	92	111	101.15	0
74	86	111	102.20	0
75	92	111	101.75	0
76	88	111	103.70	0
77	92	111	104.90	0
78	92	111	105.45	1
79	89	111	104.95	1
80	99	111	107.40	0
81	101	111	108.75	0
82	102	111	108.70	0
83	102	111	109.05	0
84	103	111	109.40	0
85	91	111	109.20	0
86	106	111	110.20	0
87	108	111	110.60	0
88	110	111	110.95	0

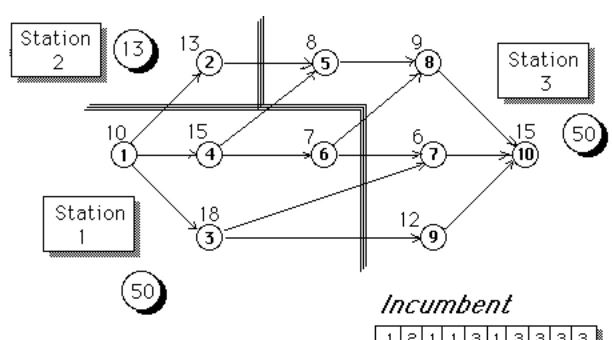
*** Converged!***

©Dennis Bricker, U. of Iowa, 1997

#	Fitness	Cycle time	# violations	Population
1	111	50	1	1 3 1 2 3 1 1 3 2 3
2	111	50	1	1 3 1 2 3 1 1 3 2 3
3	111	50	1	1 3 1 2 3 1 1 3 2 3
4	111	50	1	1 3 1 2 3 1 1 3 2 3
5	111	50	1	1 3 1 2 3 1 1 3 2 3
6	111	50	1	1 3 1 2 3 1 1 3 2 3
7	111	50	1	1 3 1 2 3 1 1 3 2 3
8	111	50	1	1 3 1 2 3 1 1 3 2 3
9	111	50	1	1 3 1 2 3 1 1 3 2 3
10	111	50	1	1 3 1 2 3 1 1 3 2 3
11	111	50	1	1 3 1 2 3 1 1 3 2 3
12	111	50	1	1 3 1 2 3 1 1 3 2 3
13	111	50	1	1 3 1 2 3 1 1 3 2 3
14	111	50	1	1 3 1 2 3 1 1 3 2 3
15	111	50	1	1 3 1 2 3 1 1 3 2 3
16	111	50	1	1 3 1 2 3 1 1 3 2 3
17	111	50	1	1 3 1 2 3 1 1 3 2 3
18	111	50	1	1 3 1 2 3 1 1 3 2 3
19	111	50	1	1 3 1 2 3 1 1 3 2 3
20	111	50	1	1 3 1 2 3 1 1 3 2 3

Converged to nonfeasible solution!

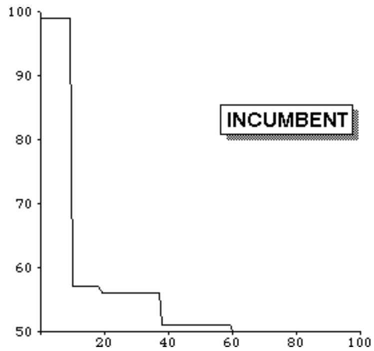
©Dennis Bricker, U. of Iowa, 1997



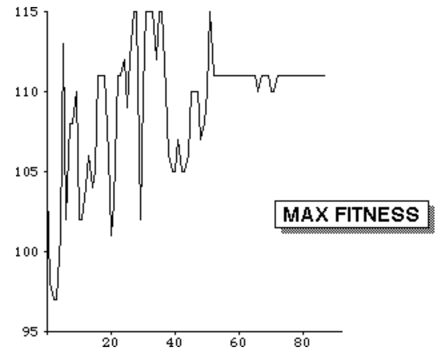
The incumbent was discovered in an earlier generation, and then disappeared!

Cycle time is 50

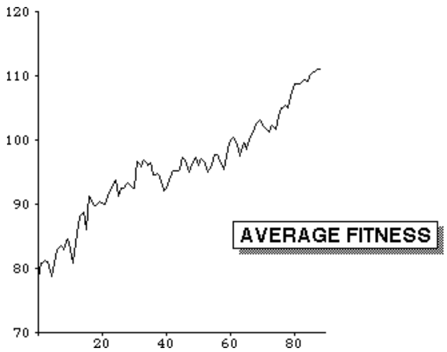
©Dennis Bricker, U. of Iowa, 1997



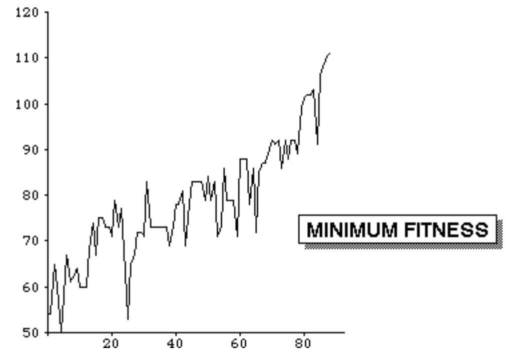
©Dennis Bricker, U. of Iowa, 1997



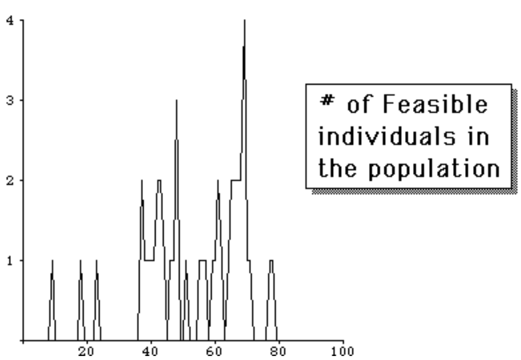
©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997

Running the GA again, but with population size = 30, not 20

generation	Fitness min	max	average	#feasible
60	83	120	106.63	4
61	83	120	105.60	6
62	83	120	104.47	5
63	90	120	105.63	7
64	81	120	106.20	7
65	75	120	104.50	7
66	84	120	106.03	7
67	83	120	106.10	8
68	86	120	109.17	9
69	97	120	111.10	11
70	93	120	113.90	9
71	98	120	114.47	9
72	98	120	114.83	13
73	98	120	116.63	17
74	102	120	117.17	19
75	115	120	119.50	20
76	119	120	119.80	24

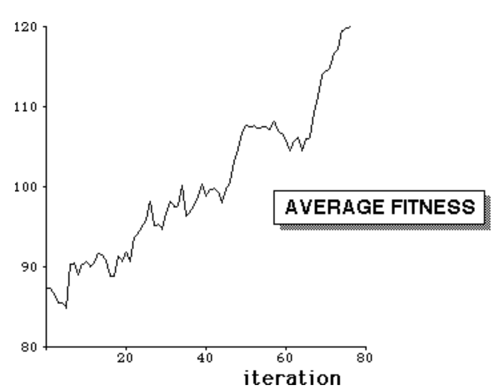
Converged!***

Converges in 76th generation

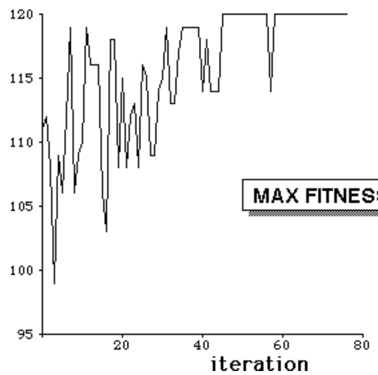
©Dennis Bricker, U. of Iowa, 1997

#	fitness	Cycle time	# violations	POPULATION
1	120	44	0	1 2 2 1 3 2 2 3 3 3
2	120	44	0	1 2 2 1 3 2 2 3 3 3
3	120	44	0	1 2 2 1 3 2 2 3 3 3
4	120	44	0	1 2 2 1 3 2 2 3 3 3
5	120	44	0	1 2 2 1 3 2 2 3 3 3
6	120	44	0	1 2 2 1 3 2 2 3 3 3
7	120	44	0	1 2 2 1 3 1 2 3 3 3
8	120	44	0	1 2 2 1 3 2 2 3 3 3
9	120	44	0	1 2 2 1 3 2 2 3 3 3
10	120	44	0	1 2 2 1 3 2 2 3 3 3
11	120	44	0	1 2 2 1 3 2 2 3 3 3
12	120	44	0	1 2 2 1 3 2 2 3 3 3
13	120	44	0	1 2 2 1 3 2 2 3 3 3
14	120	44	0	1 2 2 1 3 1 2 3 3 3
15	120	44	0	1 2 2 1 3 1 2 3 3 3
16	120	44	0	1 2 2 1 3 1 2 3 3 3
17	120	44	0	1 2 2 1 3 1 2 3 3 3
18	120	44	0	1 2 2 1 3 2 2 3 3 3
19	120	44	0	1 2 2 1 3 2 2 3 3 3
20	120	44	0	1 2 2 1 3 2 2 3 3 3
21	120	44	0	1 2 2 1 3 2 2 3 3 3
22	120	44	0	1 2 2 1 3 1 2 3 3 3
23	120	44	0	1 2 2 1 3 2 2 3 3 3
24	120	44	0	1 2 2 1 3 2 2 3 3 3
25	120	44	0	1 2 2 1 3 2 2 3 3 3
26	120	44	0	1 2 2 1 3 2 2 3 3 3
27	120	44	0	1 2 2 1 3 2 2 3 3 3
28	120	44	0	1 2 2 1 3 2 2 3 3 3
29	120	44	0	1 2 2 1 3 2 2 3 3 3
30	120	44	0	1 2 2 1 3 1 2 3 3 3

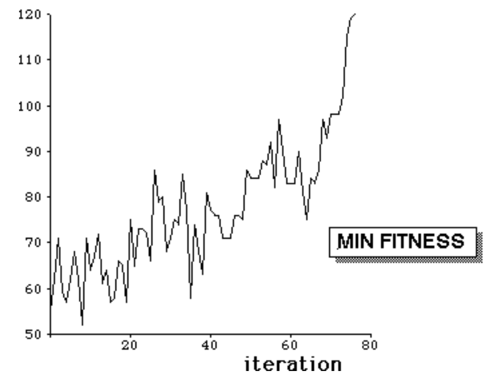
©Dennis Bricker, U. of Iowa, 1997



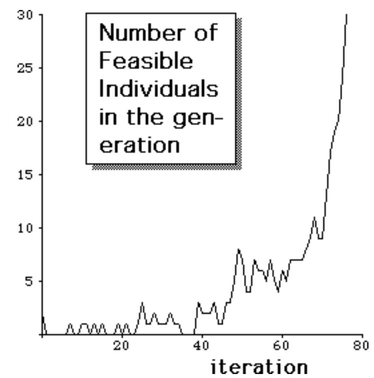
©Dennis Bricker, U. of Iowa, 1997



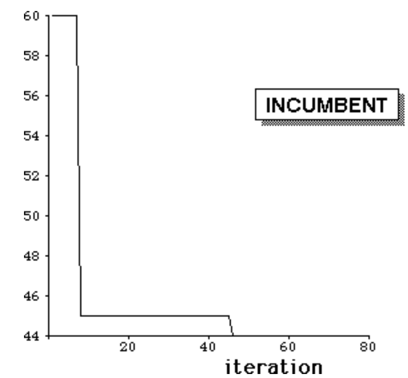
©Dennis Bricker, U. of Iowa, 1997



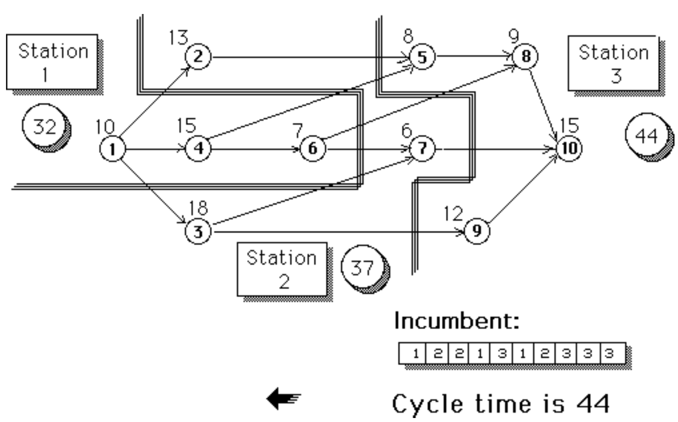
©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



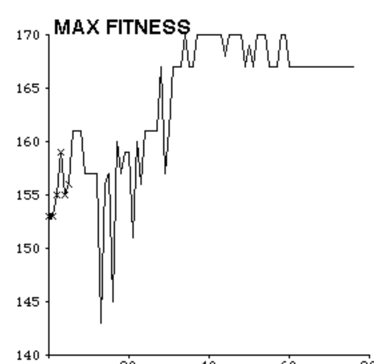
©Dennis Bricker, U. of Iowa, 1997



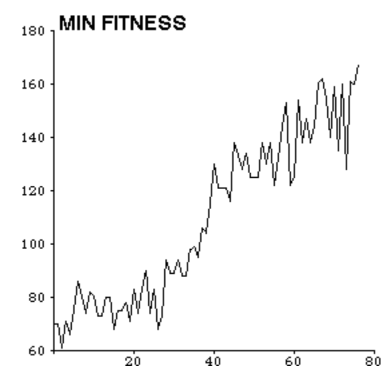
©Dennis Bricker, U. of Iowa, 1997

Finally, the GA was run with a population size of 50, and a probability of crossover equal to 75%.
Convergence occurred on the 77th iteration.

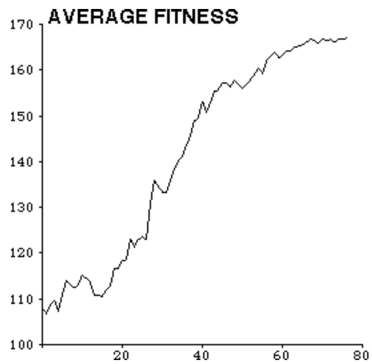
©Dennis Bricker, U. of Iowa, 1997



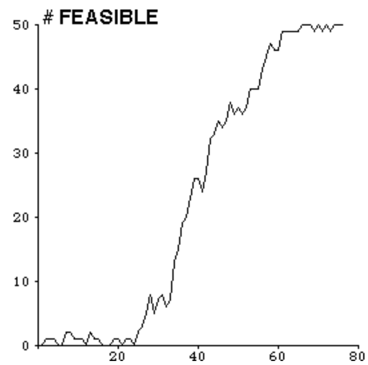
©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997

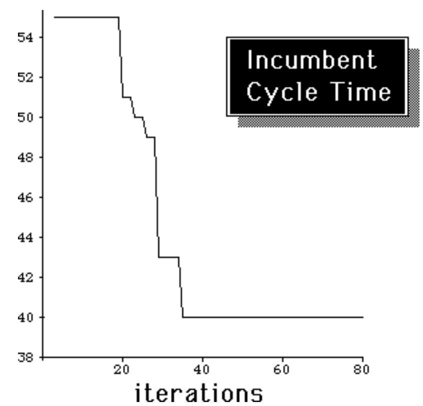
All solutions in the final population are feasible, with fitness = 167 and cycle time = 43

There are four distinct solutions in the population:

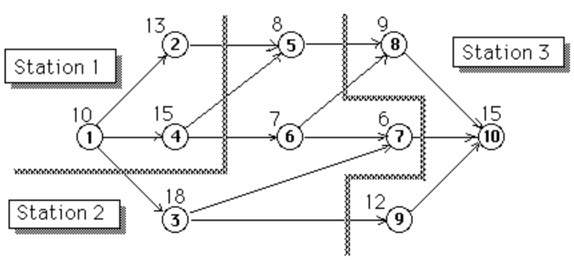
1	2	1	1	3	2	3	3	2	3
1	2	1	1	2	2	3	3	2	3
1	2	1	1	2	2	3	3	3	3
1	2	1	1	2	3	3	3	2	3

The best solution was found in the 35th generation, with a cycle time equal to 40.

©Dennis Bricker, U. of Iowa, 1997



©Dennis Bricker, U. of Iowa, 1997



Cycle time = Minimum{38, 39, 37} = 39

1	1	2	1	2	2	2	3	3	3

Optimal Solution

The optimal cycle time is 39

©Dennis Bricker, U. of Iowa, 1997

APL Code

The following screens contain listings of some APL functions written to implement GA for the Assembly Line Balancing problem.

©Dennis Bricker, U. of Iowa, 1997

```
     ∇GA;detail;toprinter;Max_iterations;I;detail;sho
 ⍝
 ⍝    Genetic Algorithm
 ⍝    for Assembly Line Balancing Problem
 ⍝    namely, minimize Cycle Time, s.t. #stations
 ⍝
 ⍝    Global variables used:
 ⍝       Nstations = # of stations to be used
 ⍝       P         = vector of task times
 ⍝       A         = adjacency matrix of precedenc
 ⍝       Popsize   = population size
 ⍝       kkk       = scalar > 1
 ⍝       Pcross    = probability of crossover
 ⍝       Pmutate   = probability of mutation
 ⍝
```

©Dennis Bricker, U. of Iowa, 1997

```
 show_pop←detail←ASKYN 'Show details?'
 ⍎detail/'show_pop←ASKYN ''Show pools?'' '
 Max_iterations←Request_Box 'max # iterations'
 toprinter←AskPrint ◇ PrSelect
 Prologue
⍝
 POP←Initial_Pool(Popsize,Nstations,⍴P)
 CycleTime←Evaluate_Pool POP
 Fitness←(⌈kkk×UB←⌈/CycleTime)-CycleTime
 Incumbent←⍳0 ◇ MinCycleTime←BIG
 MaxVals←MinVals←AvgVals←NumFeas←⍳0 ◇ iteration←0
 Record_Status
 'Random Seed: ',⍕Set_Random_Seed
```

©Dennis Bricker, U. of Iowa, 1997

```
 ⍝
 ⍎(~detail)/'⎕L ⋄ UNDERLINE ''-Generation--Minimu
 easible-'''
Next:→Stop ⍙if Max_iterations<iteration←iteration
 ⍝
 →Converged ⍙if(⌈/Fitness)=⌊/Fitness
 ⍝
 ⍝      Select pool of individuals who will survive
 ⍝      & be candidates for mating (& mutation)
 I←Select_Pool Fitness
 Report_GA
```

```
 ⍝
 ⍝      Mate pairs in the selected pool
 POP←Crossover POP[I;]
 ⍝
 ⍝      Mutate individuals in the pool
 POP←Mutate POP[I;]
 ⍝
 Fitness←(⌈kkk×UB)-CycleTime←Evaluate_Pool POP
 Record_Status
 ⍝      End of iteration
 →Next
 '*** Maximum # of iterations performed! ***'
 →Stop
Converged:'*** Converged!***'
 ⍝
Stop:'*** terminated ***'
 ∇
```

©Dennis Bricker, U. of Iowa, 1997

©Dennis Bricker, U. of Iowa, 1997

```
 ∇POOL←Initial_Pool X;msg;rl
 ⍝
 ⍝      Get initial population for genetic algorith
 ⍝
 ⍝      X[1]= population size
 ⍝      X[2]= # of stations
 ⍝      X[3]= # of tasks
 ⍝
 →Random ⍙if ASKYN 'Random initial genetic pool?'
Enter:POOL←⍎30 Alpha_Request_Box 'name of initial
 →OK ⍙if^/(X[1]=1↑⍴POOL),(X[3]=¯1↑⍴POOL),,POOL∊⍳X
 Message_Box '•Error!•Please try again'
 →Enter
Random:rl←Set_Random_Seed
 POOL←?X[1 3]⍴X[2]
OK:
 ∇
```

```
 ∇Report_GA;table;M;output
 ⍝
 ⍝      Report results in iteration of
 ⍝      Genetic Algorithm for ALB problem
 ⍝
 ⍎detail/'SHIFT⍙CTR UNDERLINE ''Generation #'',⍕iteration
 M←1↑⍴POP
 →(~show_pop)/Summary
 table←'I2,< | >,I6,< | >' ⎕FMT(⍉(2,M)⍴(⍳M),Fitness)
 table←table,'I2' ⎕FMT POP
 ⎕L,' #   Fitness   ---mating pool----'
 table ⋄ PAUSE ⋄ →End
Summary:
 output←(iteration,(¯1↑MinVals),(¯1↑MaxVals),(¯1↑AvgVals)
 'I7,2I10,F10.2,I10' ⎕FMT 1 5⍴output
End:
 ∇
```

©Dennis Bricker, U. of Iowa, 1997

©Dennis Bricker, U. of Iowa, 1997

```
 ∇Record_Status;b;i
 ⍝
 ⍝      Record status of the population
 ⍝      at the end of each generation
 ⍝
 MinVals←MinVals,⌊/Fitness
 MaxVals←MaxVals,⌈/Fitness
 AvgVals←AvgVals,AVERAGE Fitness
 NumFeas←NumFeas,+/0=nv←Number_Violations POP
 →End ⍙if~∨/b←(nv=0)^CycleTime<¯1↑MinCycleTime
 MinCycleTime←MinCycleTime,CycleTime[i←1↑b/⍳Popsize
 Incumbent←POP[i;]
 ⎕CC 35 ⋄ →0
End:MinCycleTime←MinCycleTime,¯1↑MinCycleTime
 ∇
```

```
 ∇z←Evaluate_Pool POP;i;M
 ⍝
 ⍝      Evaluate the individuals in the pool
 ⍝      (cycle time of assignment)
 ⍝
 i←0 ⋄ M←(⍴POP)[1] ⋄ z←M⍴0
Next:→End ⍙if M<i←i+1
 z[i]←Ctime POP[i;]
 →Next
End:
 ∇
```

©Dennis Bricker, U. of Iowa, 1997

©Dennis Bricker, U. of Iowa, 1997

```
 ∇t←Ctime pop
 ⍝
 ⍝      Compute Cycle time for solution "pop"
 ⍝      where pop[i]= station to which task i is
 ⍝      assigned.  Include penalty times number
 ⍝      of precedence violations.
 ⍝
 ⍝      Global variable: Penalty
 ⍝                       P = vector of task times
 ⍝
 t←⌈/((⍳Nstations)∘.=pop)+.×P   ⍝ Cycle time
 t←t++/Penalty×,A>pop∘.≤pop  ⍝ Violation penalty
 ∇
```

```
 ∇z←Number_Violations pop;i
 ⍝
 ⍝      Compute number of violations of
 ⍝      precedence restrictions
 ⍝
 ⍝      Global variable:  A = precedence matrix
 →Single ⍙if 1=⍴⍴pop
 z←(1↑⍴pop)⍴0 ⋄ i←0
Next:→End ⍙if(⍴pop)[1]<i←i+1
 z[i]←+/,A>pop[i;]∘.≤pop[i;]
 →Next
Single:z←+/,A>pop∘.≤pop
End:
 ∇
```

©Dennis Bricker, U. of Iowa, 1997

©Dennis Bricker, U. of Iowa, 1997

```
    ∇I←Select_Pool F;S;N;s0;s;f;n;a;b
 ⍝
 ⍝      Select individuals from pool according
 ⍝      to their fitness values F
 ⍝      Fitness values are scaled so as
 ⍝           to sum to 1.0 and
 ⍝              Max Fitness = scalefactor÷N
 ⍝      Global variable:  GA_Scale_Factor
  a←(GA_Scale_Factor-1)÷((Popsize×⌈/F)-+/F)
  b←(-a×⌊/F)⌈(GA_Scale_Factor÷Popsize)-a×⌈/F
  F←b+a×F
 ⍝
  S←+/F ⋄ f←+\F
  s0←(S÷N←⍴F)×0.001×?1000
  s←s0+(S÷N)×0,⍳N-1
  n←n-0,¯1↓n←+/(f∘.≥s)
  →0 ⍙if Popsize=⍴I←((n>0)/n)\(n>0)/⍳N
    ∇
```

©Dennis Bricker, U. of Iowa, 1997

```
    ∇Q←Sample Fitness;N;S;s0;s;F
 ⍝
 ⍝    Select individuals from the population,
 ⍝    who will survive to the next generation.
 ⍝    Argument: Fitness = vector of fitness values
 ⍝       of the individuals in the population,
 ⍝    Result:  Q = vector of number of copies of
 ⍝       individuals to be included in the next ge
 ⍝
 ⍝    Method used is "stochastic universal samplin
 ⍝       due to J.E. Baker
 ⍝
  S←+/Fitness ⋄ N←⍴Fitness ⋄ F←+\Fitness
  s0←(?1000)×N÷1000×S
  s←s0+(S÷N)×0,⍳N-1
  Q←(+/F∘.≥s)-(+/(0,¯1↓F)∘.≥s)
    ∇
```

©Dennis Bricker, U. of Iowa, 1997

```
    ∇Q←Crossover Pop;M;N;PAIR;I
 ⍝       For each pair of individuals
 ⍝       in the mating pool, either mate
 ⍝       them & put the 2 offspring into
 ⍝       the pool, or else copy them into
 ⍝       the new pool directly
  N←(⍴Pop)[1] ⋄ M←(⍴Pop)[2]
  Q←(0,¯1↑⍴Pop)⍴0
 ⍝       Randomly shuffle the pool
  Pop←Pop[N?N;]
Next:→End ⍙if 2>(⍴Pop)[1]
 ⍝       Remove pair from the pool
  PAIR←(2,M)↑Pop ⋄ Pop←2 0↓Pop
 ⍝
  →Copy ⍙if Pcross<(?1000)÷1000
 ⍝       Choose crossover point
  Q←Q,[1]((2,I)↑PAIR),⊖[1](0,I←1+?M-2)↓PAIR
  →Next
Copy:Q←Q,[1]PAIR ⋄ →Next
End:Q←Q,[1]Pop
    ∇
```

©Dennis Bricker, U. of Iowa, 1997

```
    ∇Q←Mutate Pop;I
 ⍝
 ⍝       Perform mutation of the individuals in pool
 ⍝
  Q←,Pop
  I←(Pmutate≥(?(⍴Q)⍴10000)÷10000)/⍳⍴Q
  Q[I]←?(⍴I)⍴Nstations
  Q←(⍴Pop)⍴Q
    ∇
```

©Dennis Bricker, U. of Iowa, 1997