

# Genetic Algorithm for Line Balancing

This Hypercard stack was prepared by:  
Dennis L. Bricker,  
Dept. of Industrial Engineering,  
University of Iowa,  
Iowa City, Iowa 52242  
e-mail: [dbricker@icaen.uiowa.edu](mailto:dbricker@icaen.uiowa.edu)



author



Generic genetic algorithm

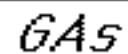


Genetic algorithm for the assembly line  
balancing problem

## 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.



A small graphic of a computer monitor with the letters "GAS" displayed on the screen.

- 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.



## 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

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

## 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



## 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..."*

## 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  $p_i = \frac{f(i)}{\sum_{j=1}^N f(j)}$

This is repeated until N individuals have been selected.

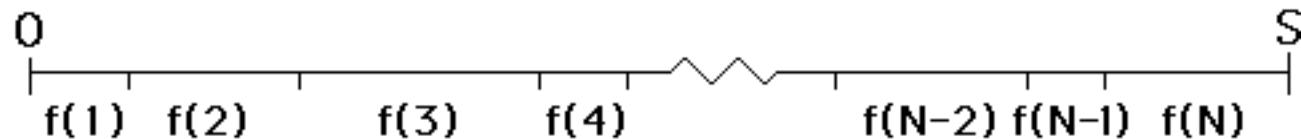


In sampling with replacement, the more fit individuals are likely to be selected, and may be selected several times. However, there is not guarantee that an individual whose fitness is above the population average will be selected

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

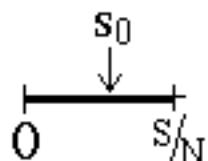
J.E. Baker, "Reducing bias and inefficiency in the selection algorithm", in *Genetic Algorithms & Their Applications: Proceedings of the 2nd Int'l Conf. on Genetic Algorithms*, J.J. Grefenstette (ed.), 1987.

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.



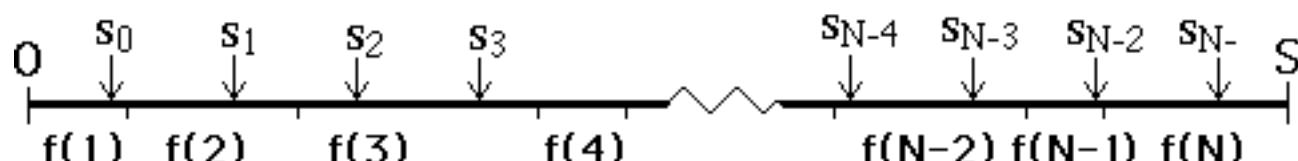
*Stochastic  
Universal  
Sampling*

3. Generate a random number  $s_0$  within the interval  $[0, \frac{S}{N}]$ , the "average" interval.

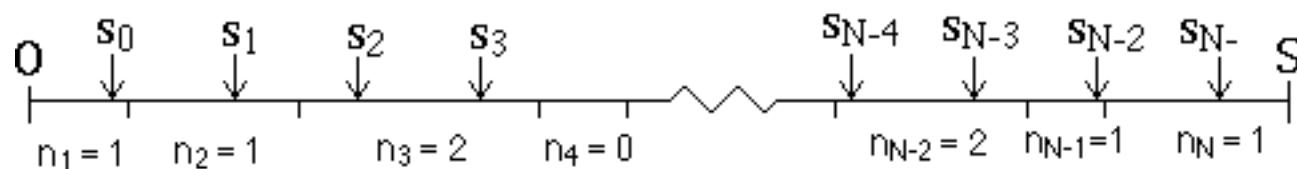


4. Compute the N equally-spaced numbers

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



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



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



**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.



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

$$\sum_{i=1}^N F_i = 1$$

and

$$\lambda = \frac{\max_i F_i}{\sum_{i=1}^N F_i / N}$$

Solution: Choose a & b so that

$$a = \frac{\lambda - 1}{N f_{\max} - \sum_{i=1}^N f_i}$$

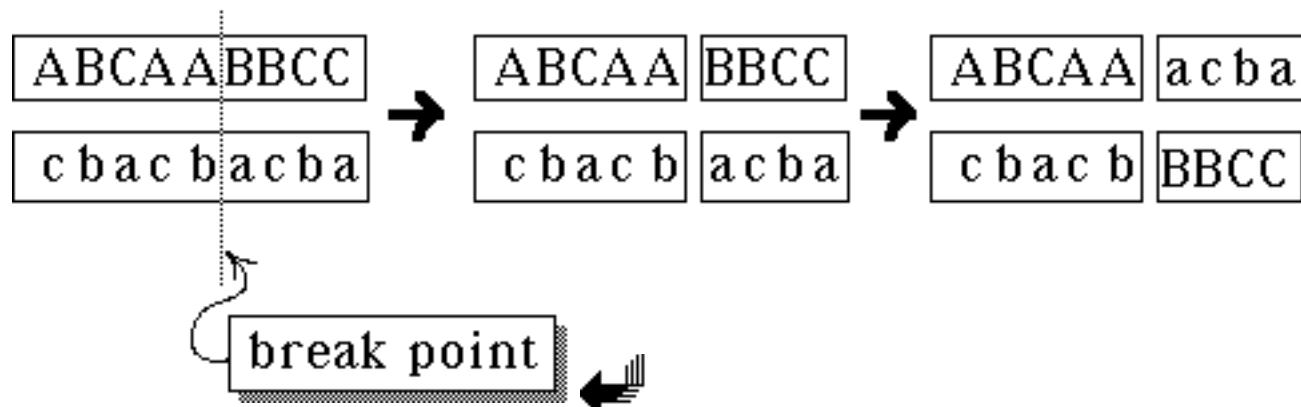
$$b = \lambda/N - a f_{\max}$$

where

$$f_{\max} = \max_i \{f_i\}$$

## Crossover

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



## 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

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.



- 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.*

There are numerous decisions which must 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

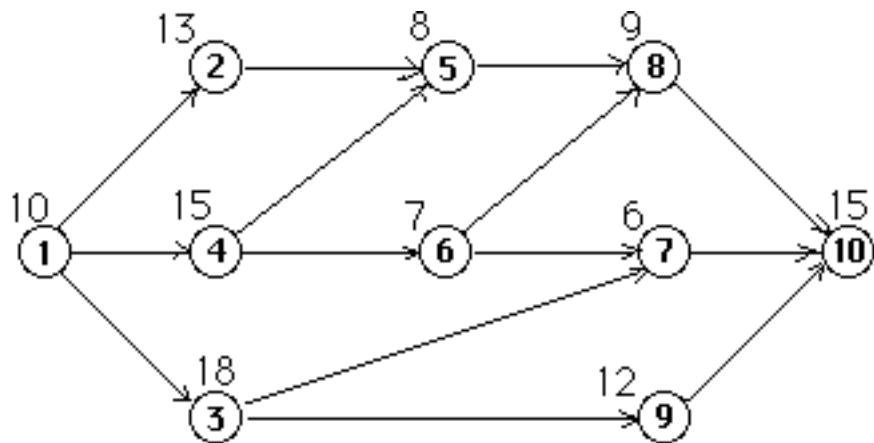
See sample Pascal code in Chapter 3: "Computer Implementation of a Genetic Algorithm", in book by D. Goldberg.



## 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.



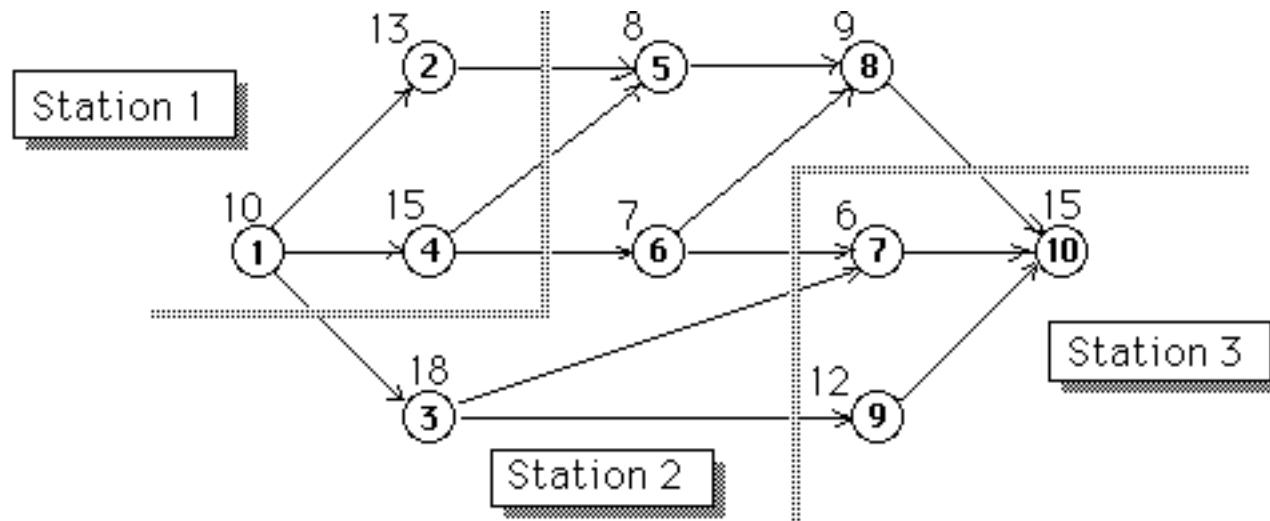


EXAMPLE

$n = 3$  stations  
10 tasks

## 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  $i^{\text{th}}$  position on the string.



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

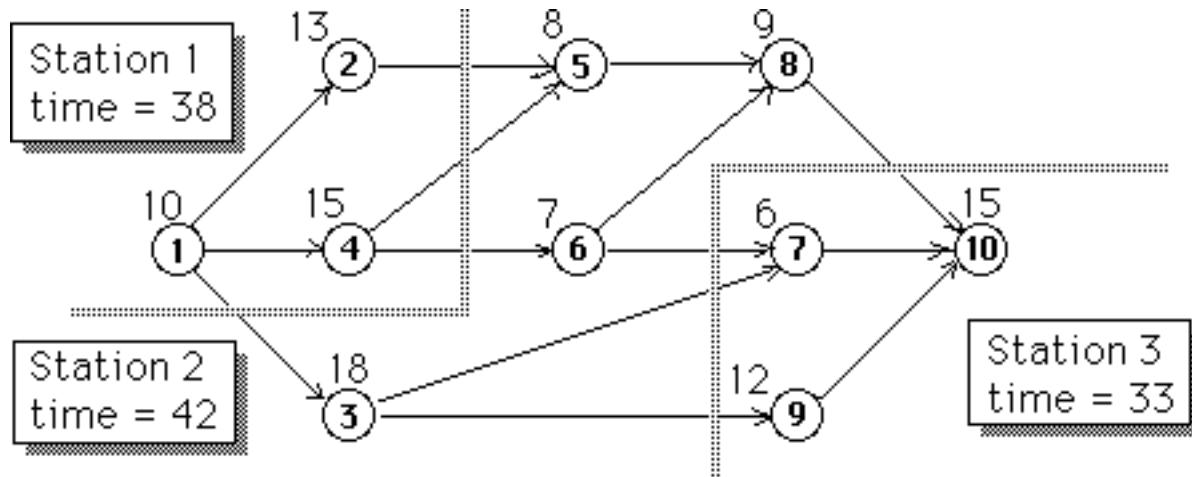
**ENCODING**

## 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.



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

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

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

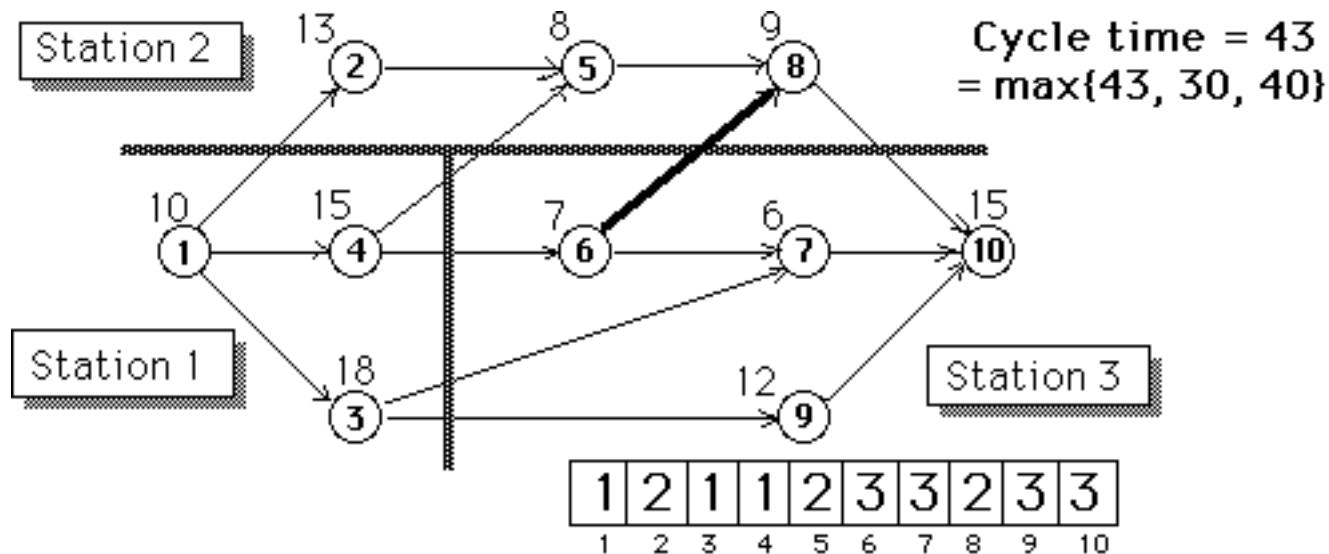
where

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

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

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:

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

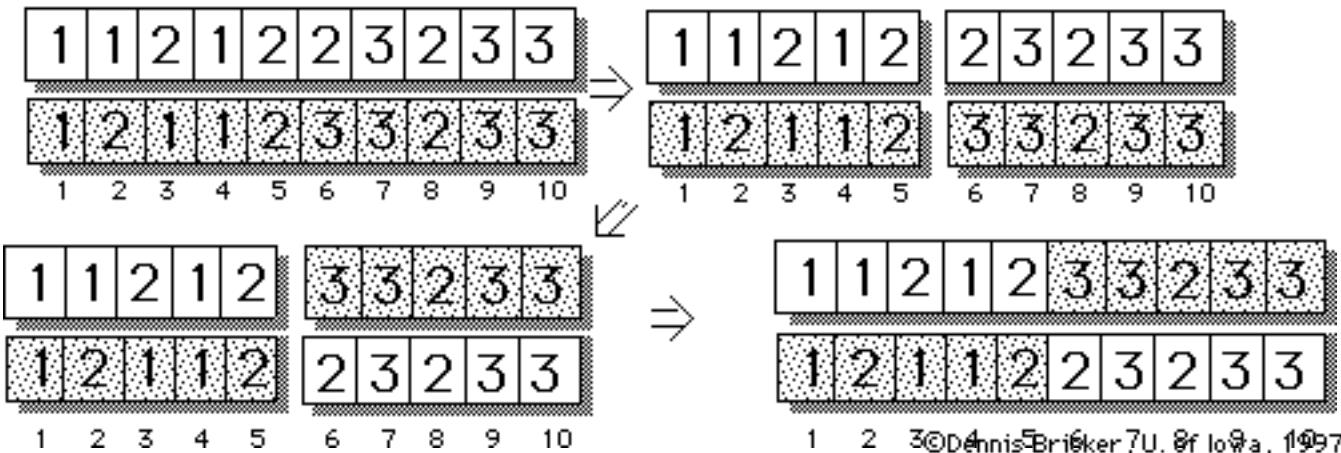


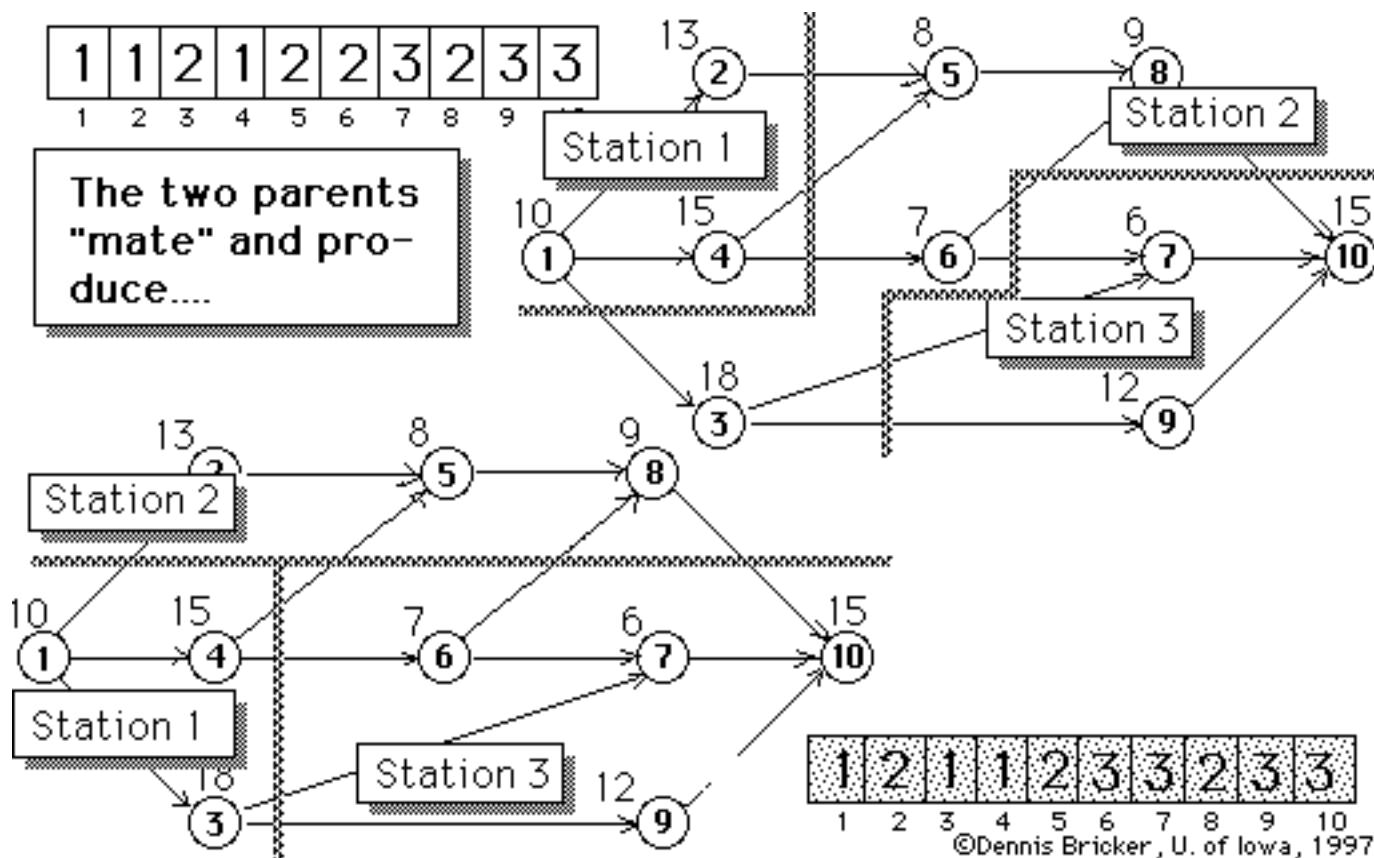
In the assignment shown, task 6 does not precede task 8 as required, and so we reduce the fitness by a penalty  $P=10$ :

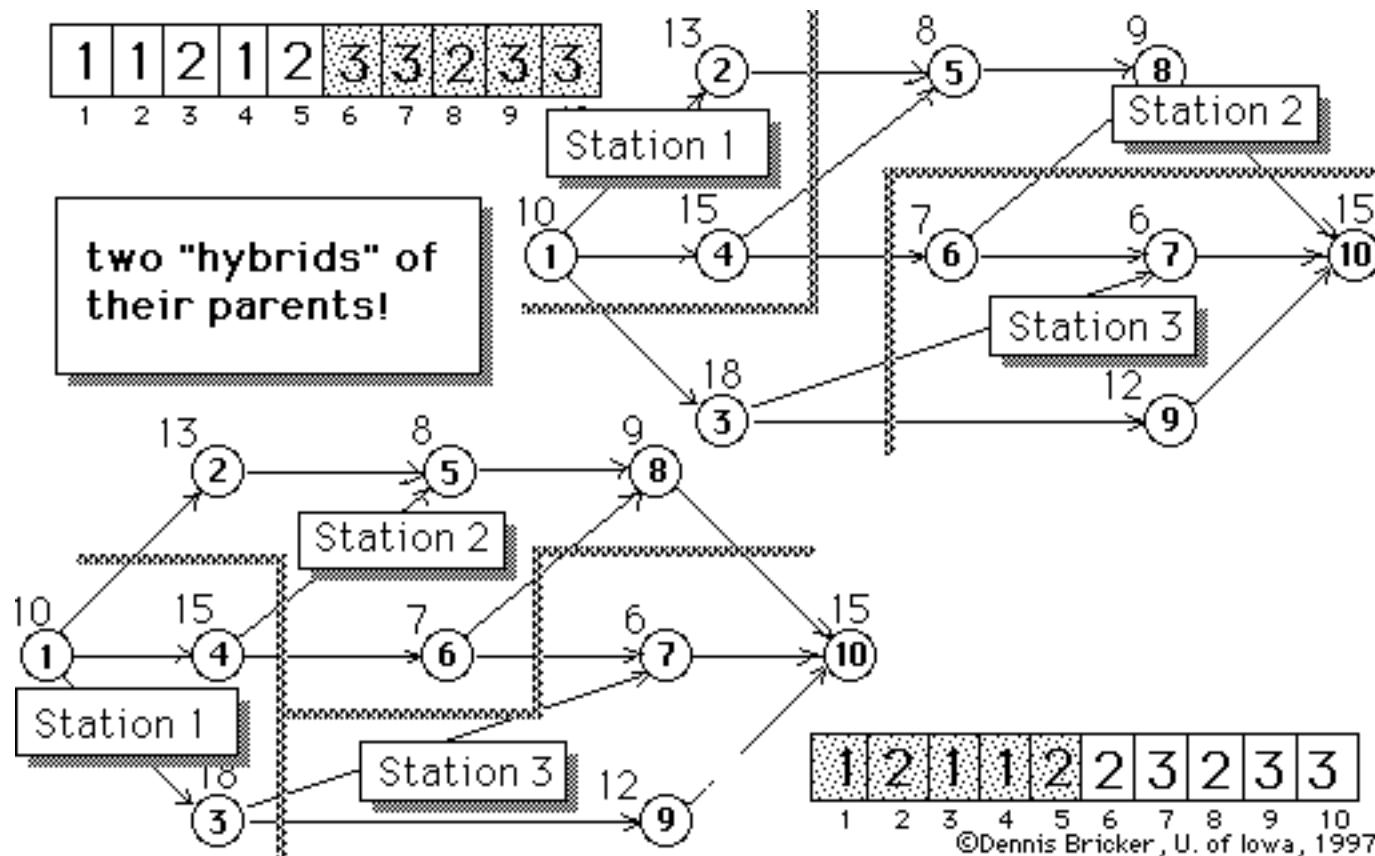
$$T(i) = 43 + 10 \times 1 = 53$$

## 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:



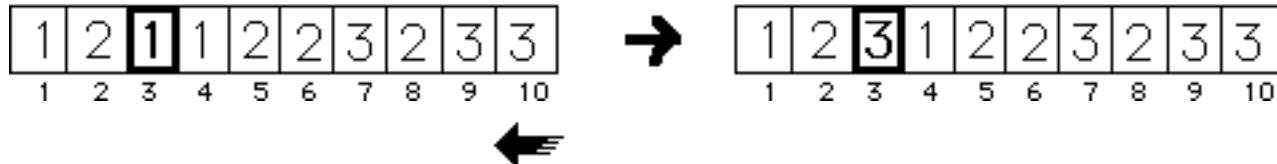




## MUTATION

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

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



**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
```



At the 100th iteration, the algorithm converged:

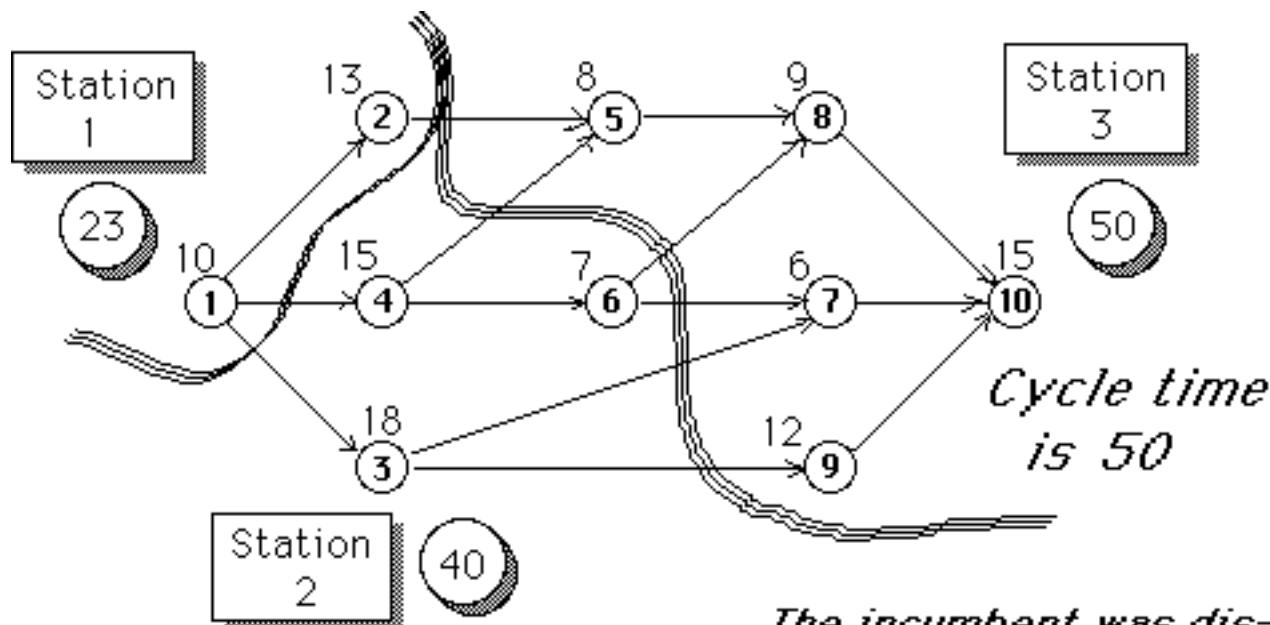
*the population is completely uniform!*

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

Incumbent:

1 1 2 2 3 2 3 3 3 3

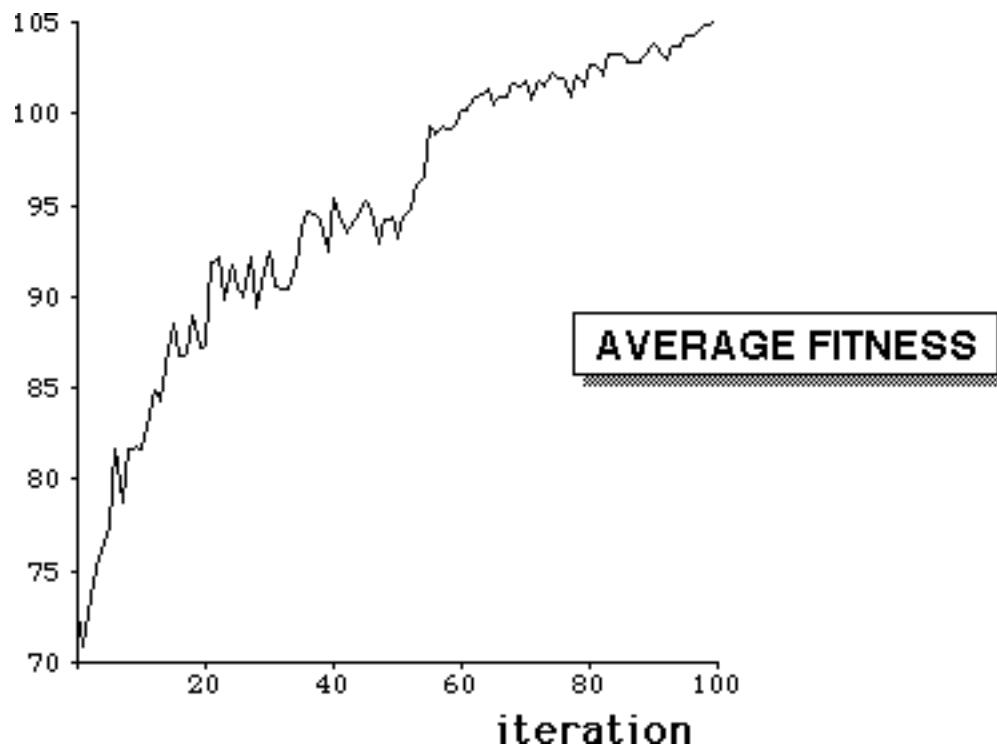
*None in final generation is feasible!*  
©2000 DEKKER, D. DE, IOWA, 1221

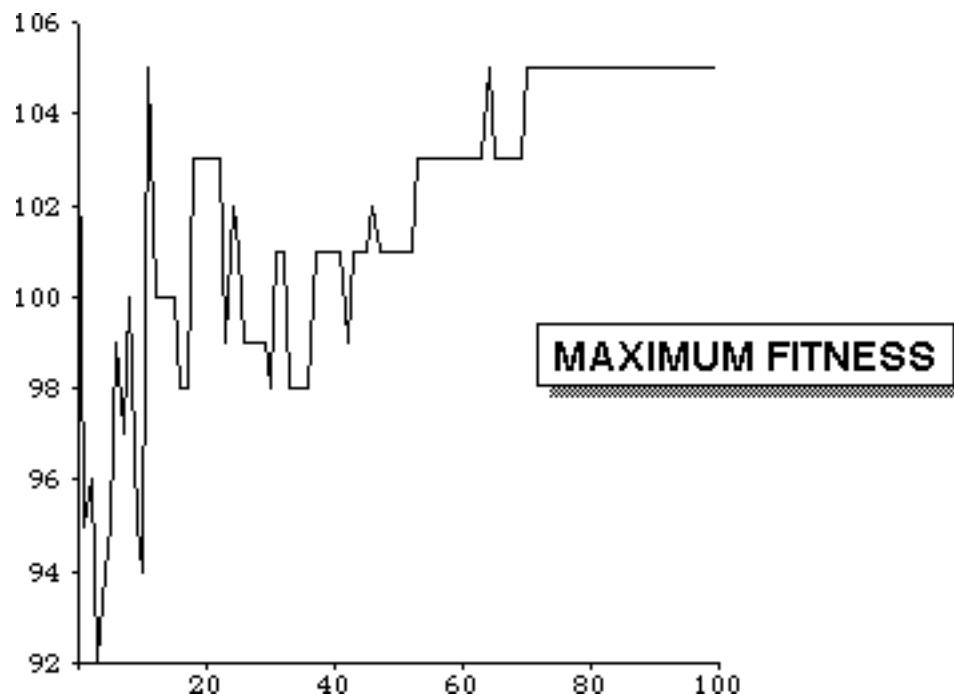


Incumbent:

1	1	2	2	3	2	3	3	3	3
---	---	---	---	---	---	---	---	---	---

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

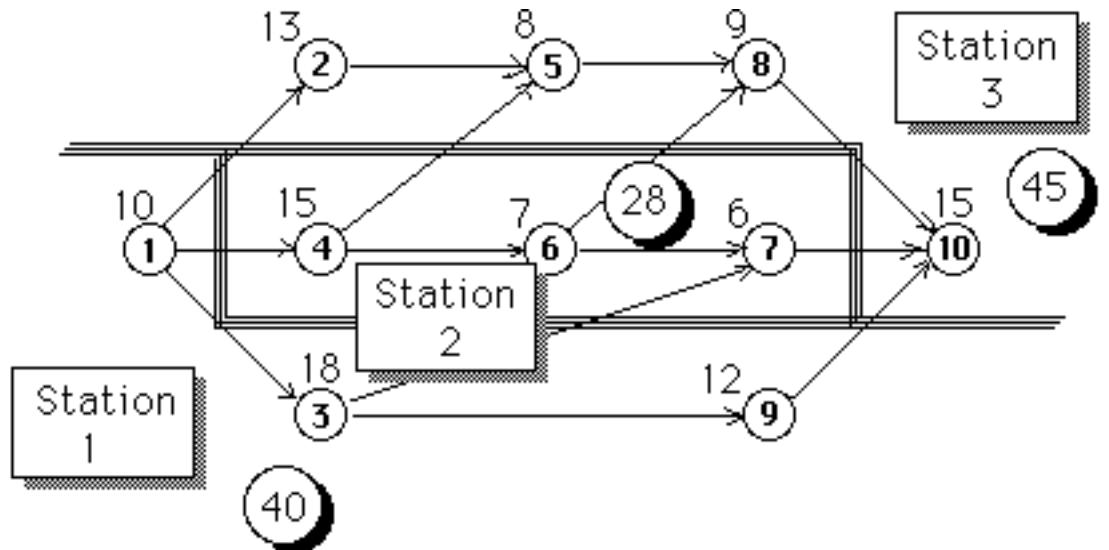




*Running again,  
with a higher  
probability of  
crossover  
(90%)*

*Again, the  
algorithm  
converged  
(population is  
not uniform, but  
has uniform  
fitness!)*

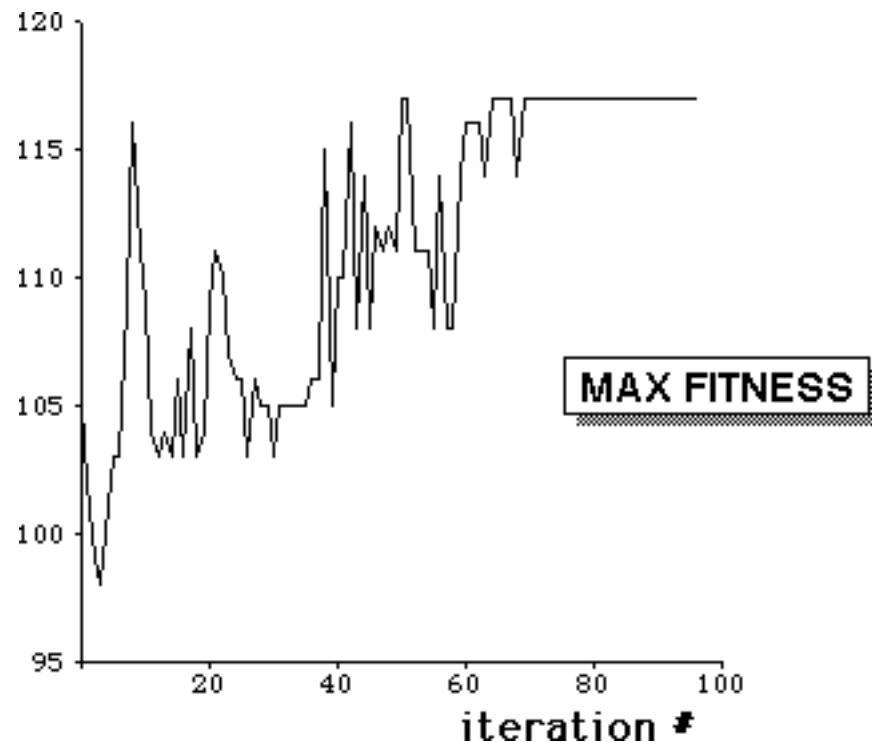
*	Fit ness	Cycle time	# viola tions	Population
1	117	45	0	1 3 1 2 3 2 2 3 1 3
2	117	45	1	1 3 1 2 3 2 2 3 1 3
3	117	45	1	1 3 1 2 3 2 2 3 1 3
4	117	45	1	1 3 1 2 3 2 2 3 1 3
5	117	45	0	1 3 1 2 3 2 2 3 1 3
6	117	45	0	1 3 1 2 3 2 2 3 1 3
7	117	45	1	1 3 1 2 3 2 2 3 1 3
8	117	45	0	1 3 1 2 3 2 2 3 1 3
9	117	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	1 3 1 2 3 2 2 3 1 3
12	117	45	1	1 3 1 2 3 2 2 3 1 3
13	117	45	1	1 3 1 2 3 2 2 3 1 3
14	117	45	1	1 3 1 2 3 2 2 3 1 3
15	117	45	1	1 3 1 2 3 2 2 3 1 3
16	117	45	1	1 3 1 2 3 2 2 3 1 3
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	1 3 1 2 3 2 2 3 1 3

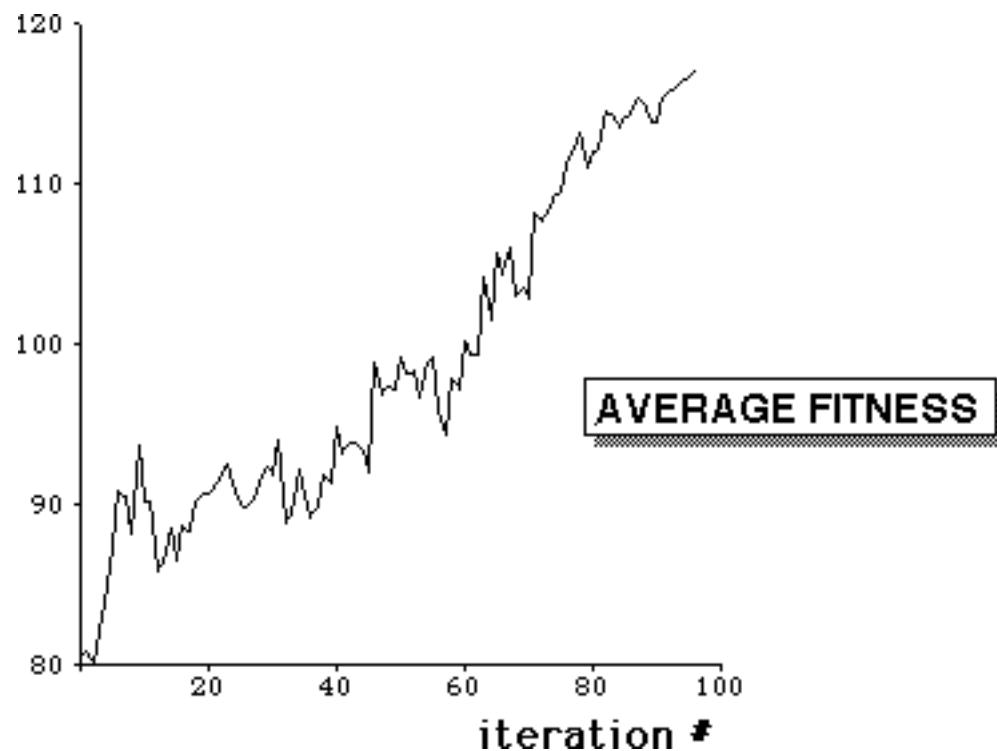


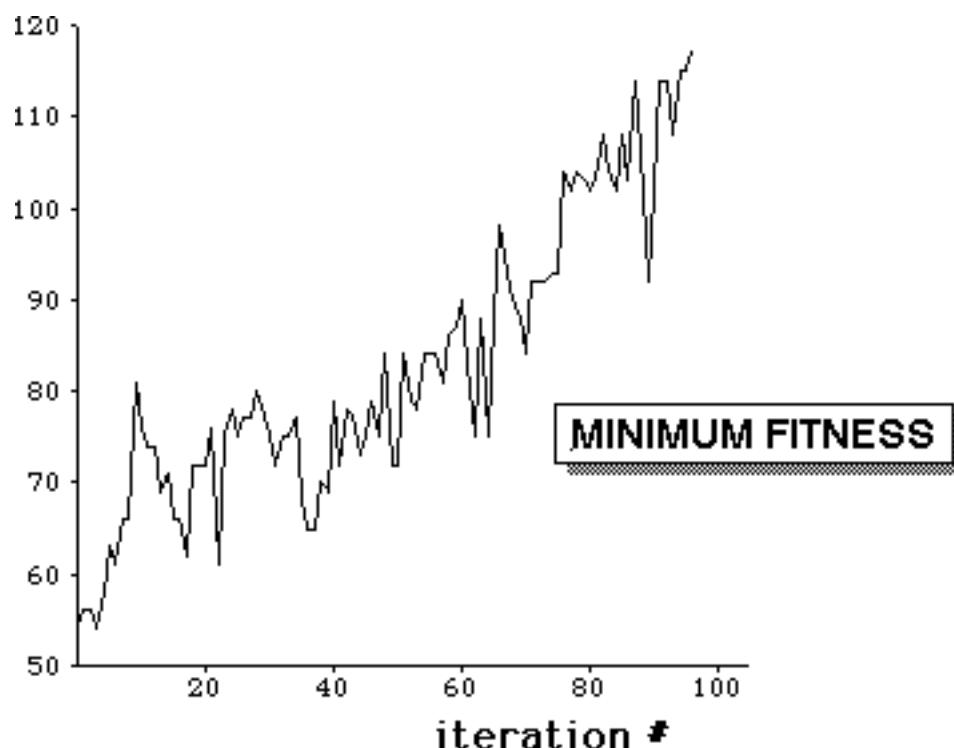
*Incumbent:*

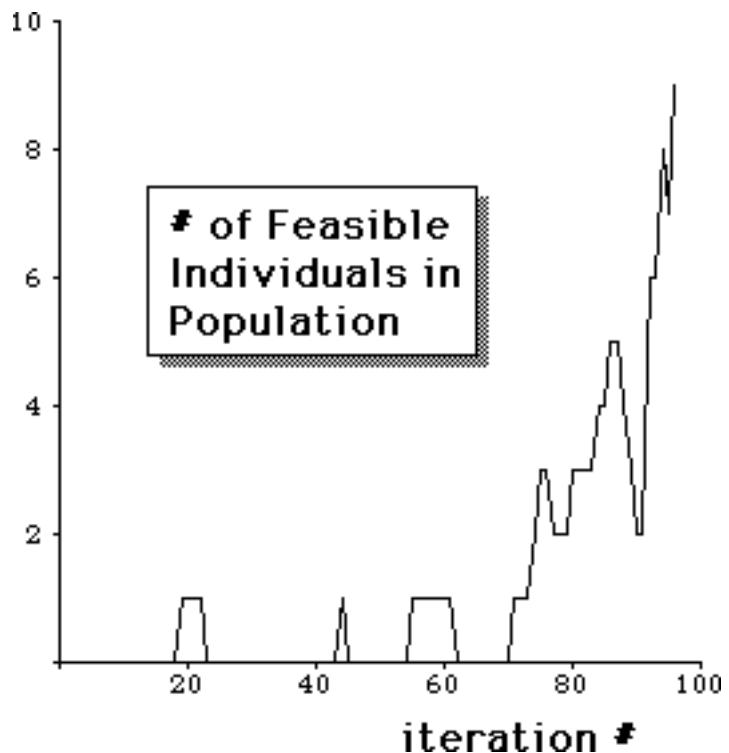
1	3	1	2	3	2	2	3	1	3
---	---	---	---	---	---	---	---	---	---

*Cycle time is 45*









Running the GA again with  $P_{cross}=90\%$ , it converged in 88th generation:

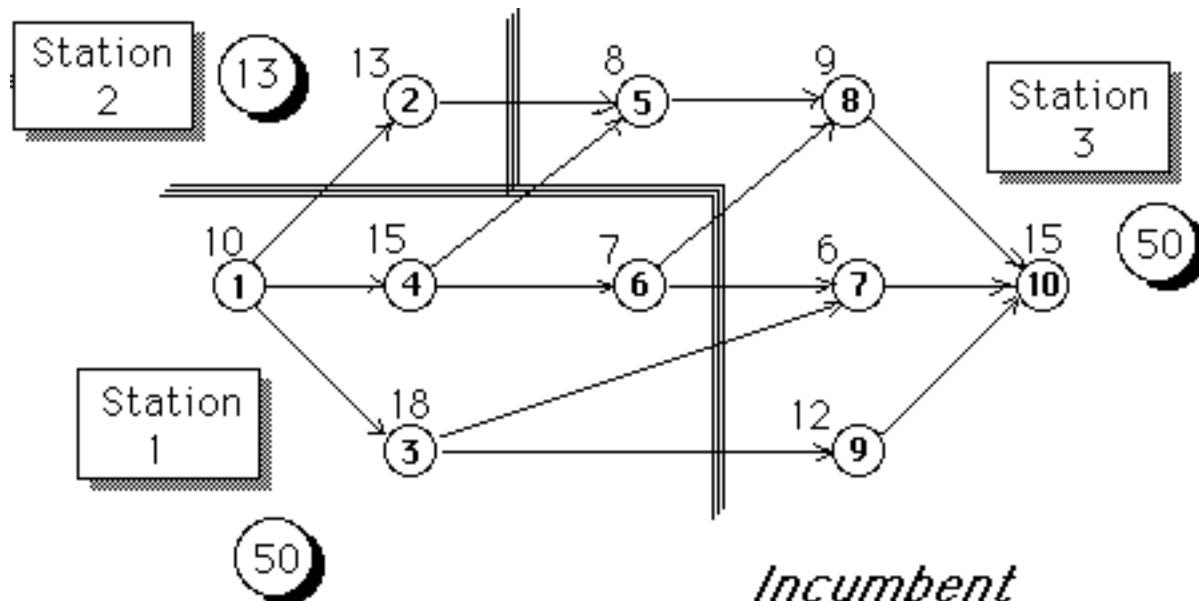
generation	Fitness			
	min	max	average	#feasible
72	91	110	101.90	1
73	92	111	101.15	0
74	86	111	102.20	0
75	92	111	101.75	0
76	88	111	103.70	0
77	92	111	104.90	0
78	92	111	105.45	1
79	89	111	104.95	1
80	99	111	107.40	0
81	101	111	108.75	0
82	102	111	108.70	0
83	102	111	109.05	0
84	103	111	109.40	0
85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

\*\*\* Converged! \*\*\*

#	Fit ness	Cycle time	* viola- tions	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!**

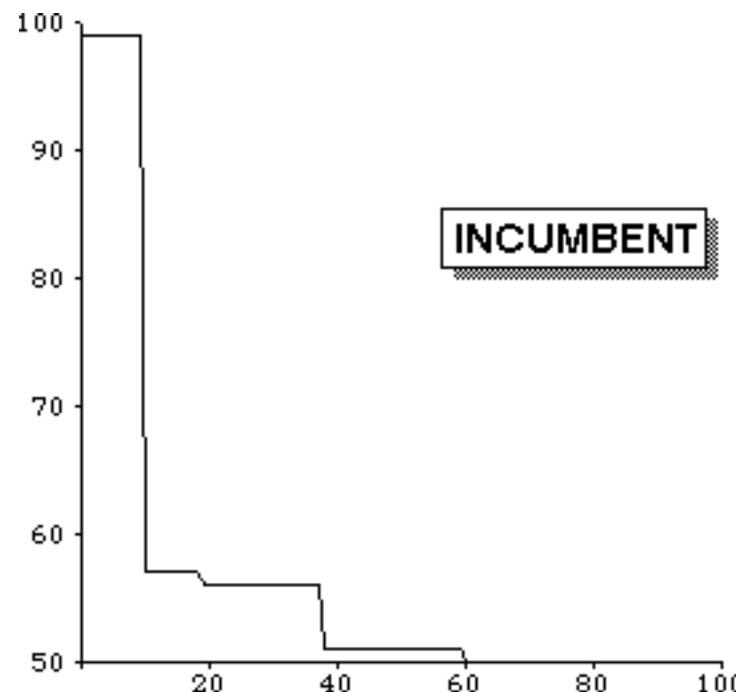
© Dennis Bricker, U. of Iowa, 1997

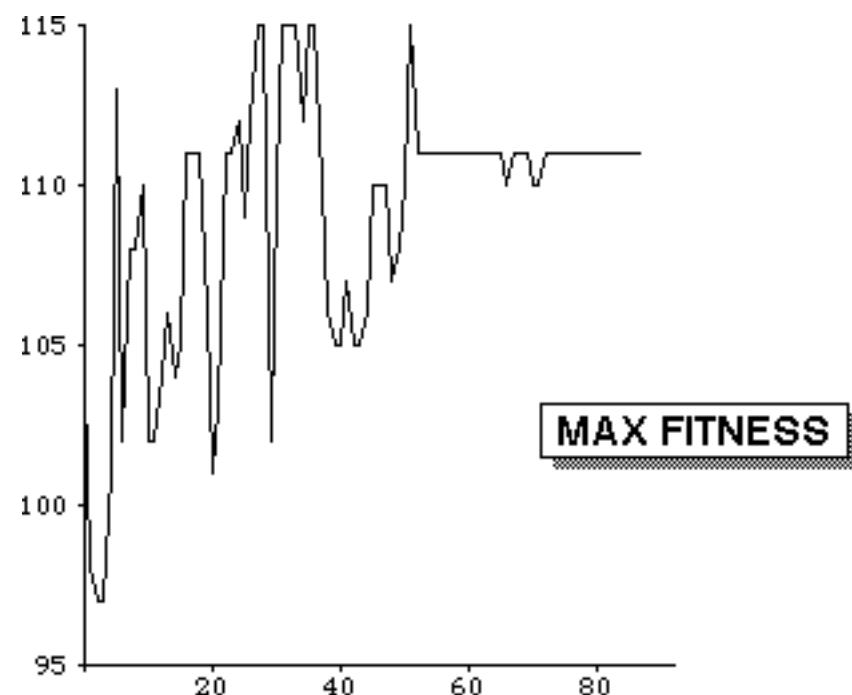


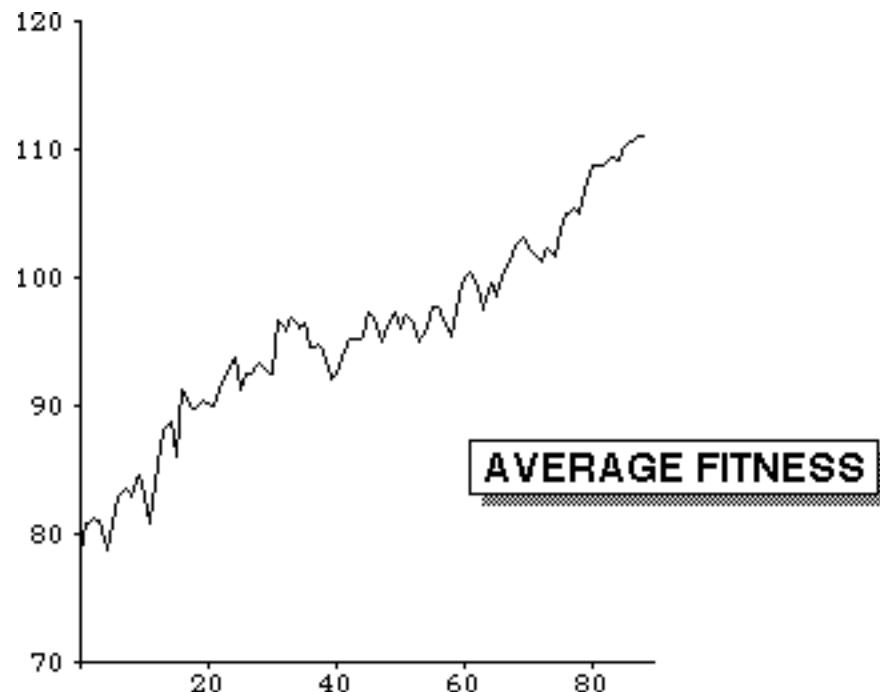
*The incumbent was discovered in an earlier generation, and then disappeared!*

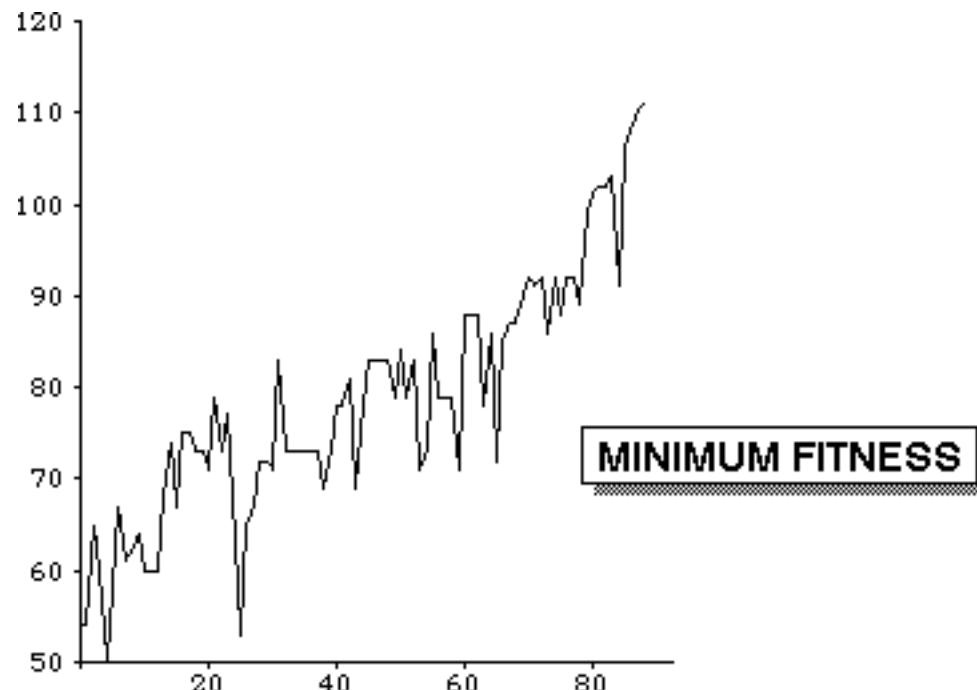
*Cycle time is 50*

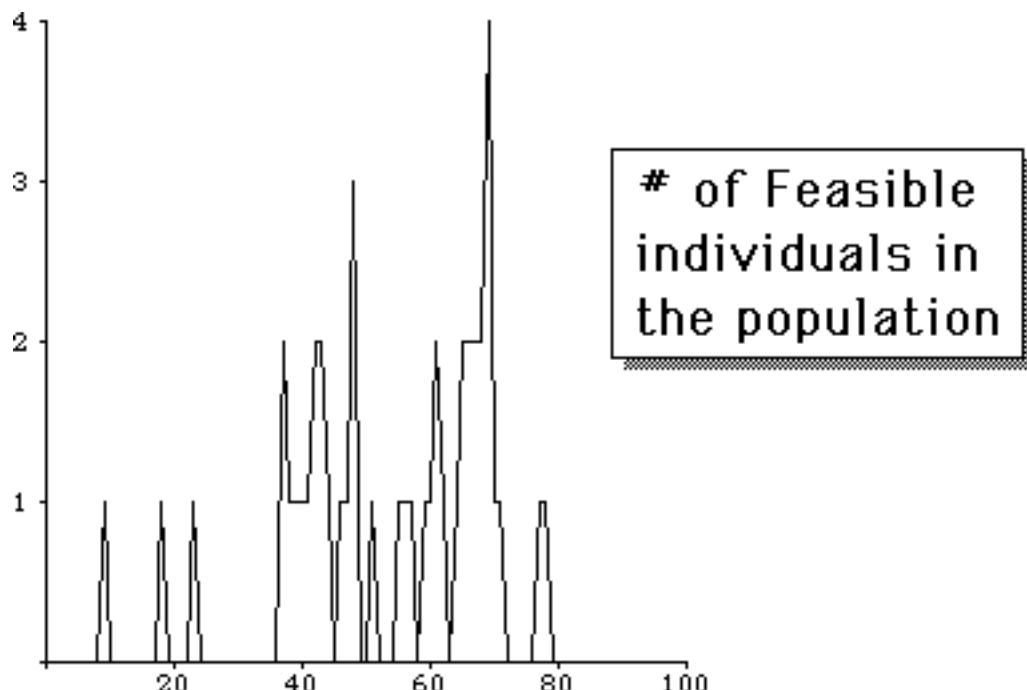
©Dennis Bricker, U. of Iowa, 1997











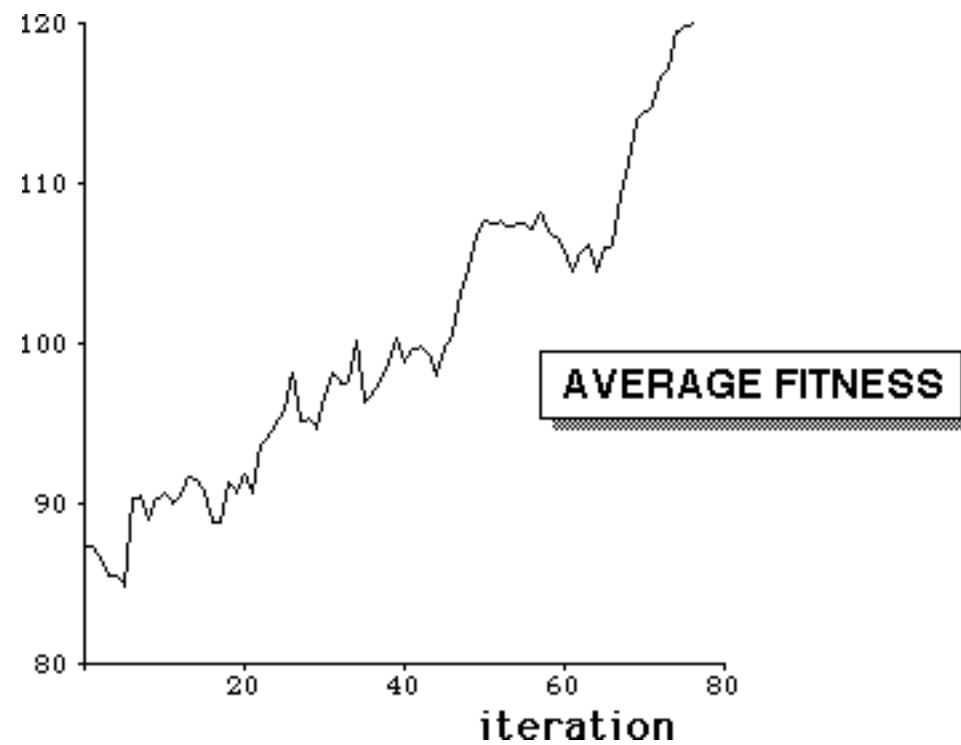
*Running the  
GA again, but  
with popula-  
tion size =  
30, not 20*

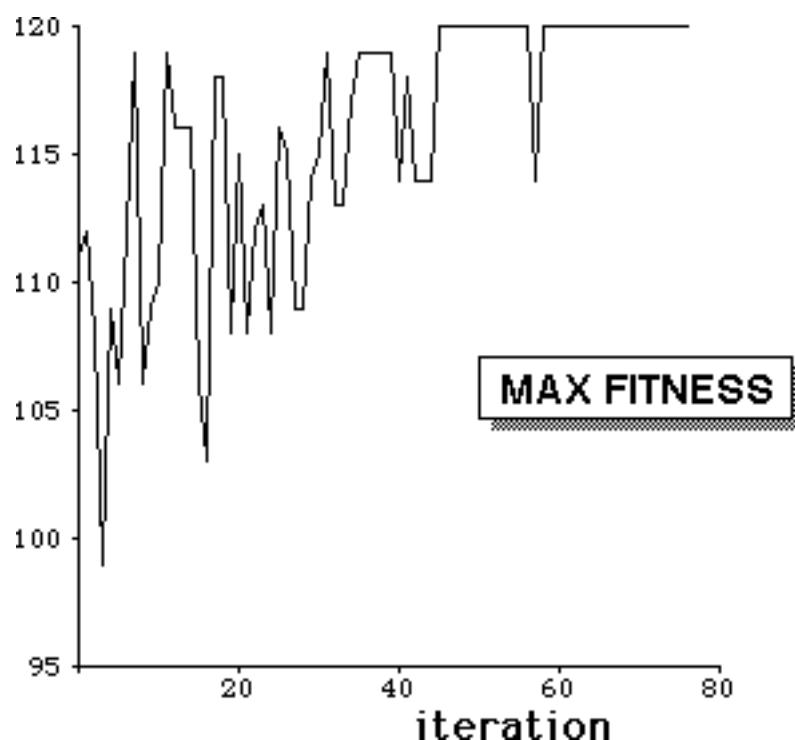
*Converges in  
76th genera-  
tion*

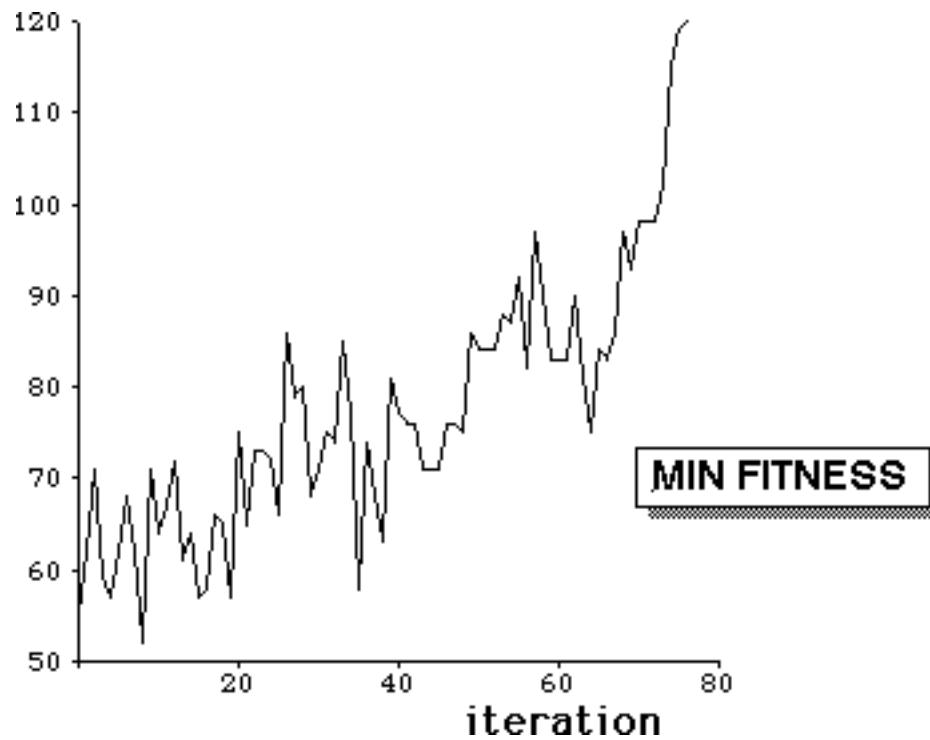
generation	Fitness			
	min	max	average	#feasible
60	83	120	106.63	4
61	83	120	105.60	6
62	83	120	104.47	5
63	90	120	105.63	7
64	81	120	106.20	7
65	75	120	104.50	7
66	84	120	106.03	7
67	83	120	106.10	8
68	86	120	109.17	9
69	97	120	111.10	11
70	93	120	113.90	9
71	98	120	114.47	9
72	98	120	114.83	13
73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

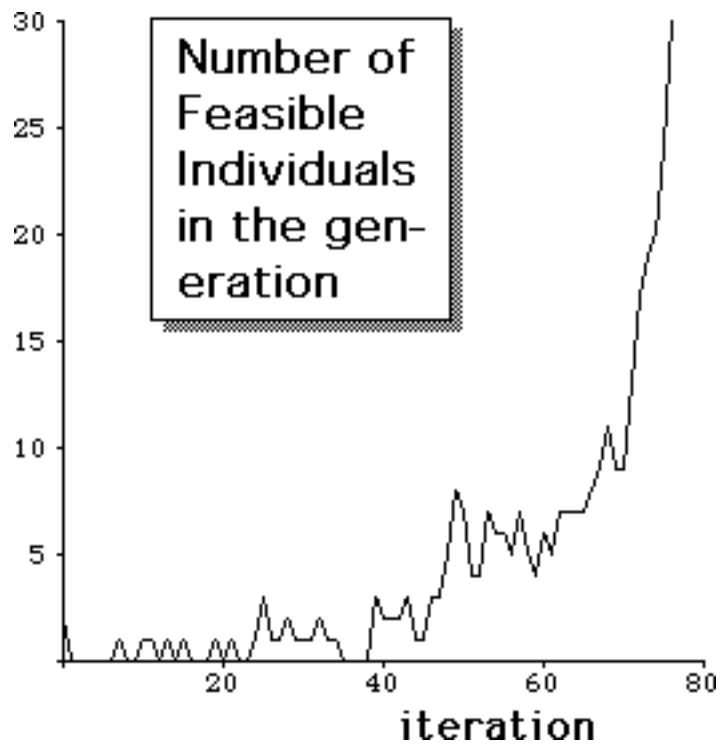
Converged!\*\*\*

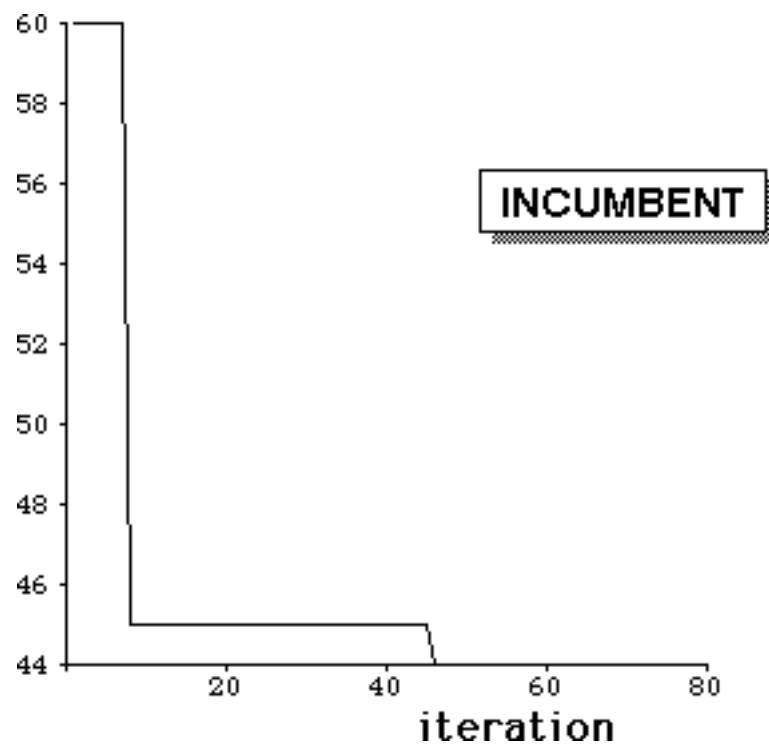
#	fitness	Cycle time	# violations	POPULATION											
1	120	44	0	1	2	2	1	3	2	2	3	3	3	3	3
2	120	44	0	1	2	2	1	3	2	2	3	3	3	3	3
3	120	44	0	1	2	2	1	3	2	2	3	3	3	3	3
4	120	44	0	1	2	2	1	3	2	2	3	3	3	3	3
5	120	44	0	1	2	2	1	3	2	2	3	3	3	3	3
6	120	44	0	1	2	2	1	3	2	2	3	3	3	3	3
7	120	44	0	1	2	2	1	3	1	2	3	3	3	3	3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

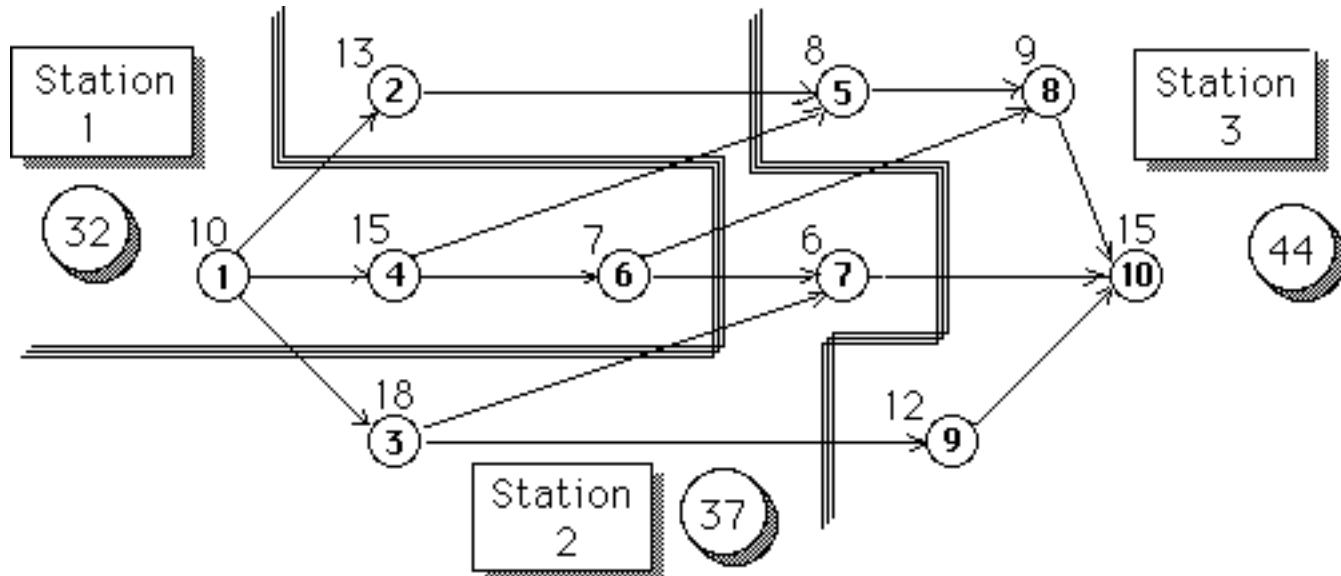












Incumbent:

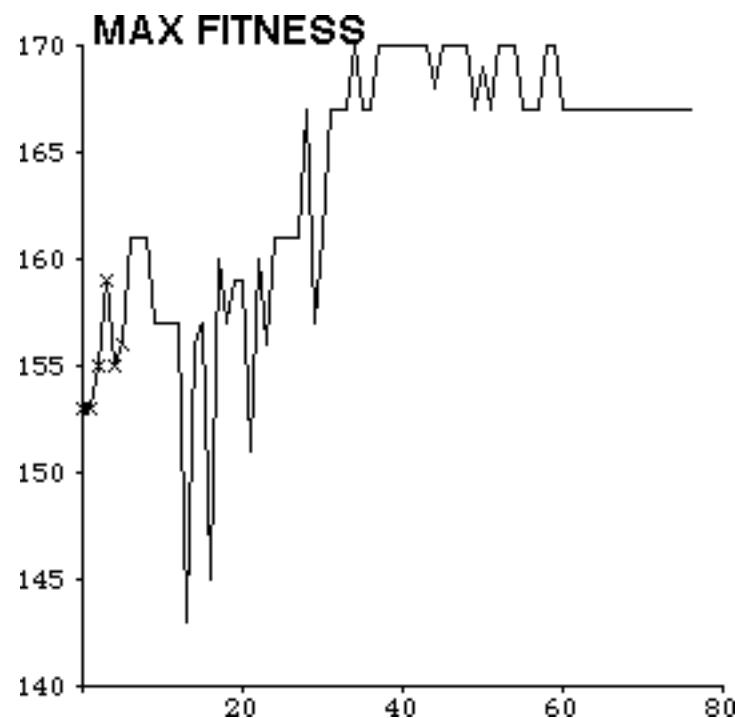
1	2	2	1	3	1	2	3	3	3
---	---	---	---	---	---	---	---	---	---

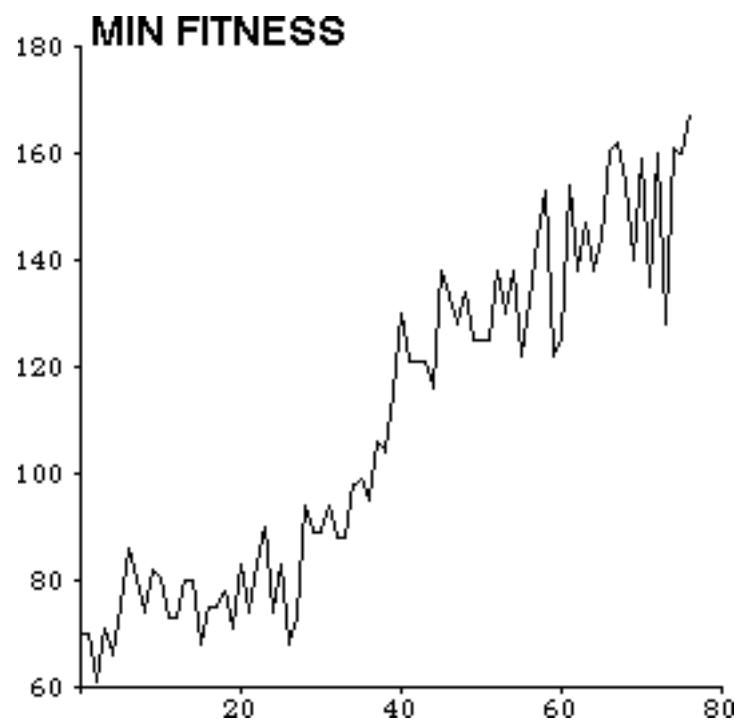


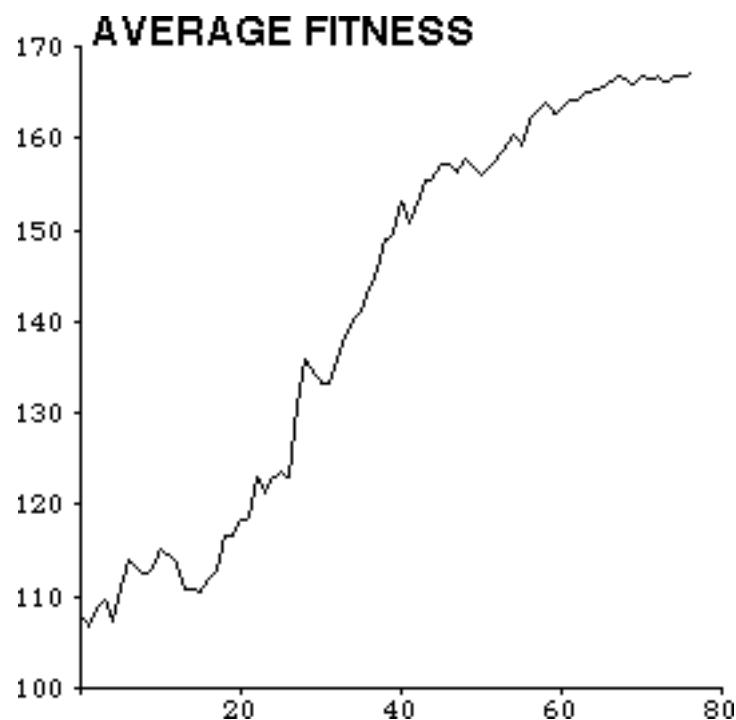
Cycle time is 44

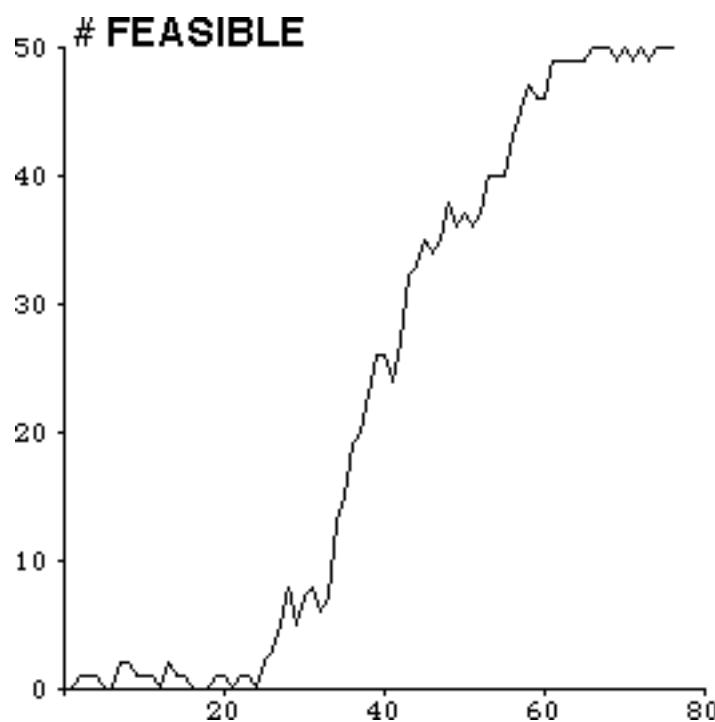
Finally, the GA was run with a population size of 50, and a probability of crossover equal to 75%.

Convergence occurred on the 77th iteration.







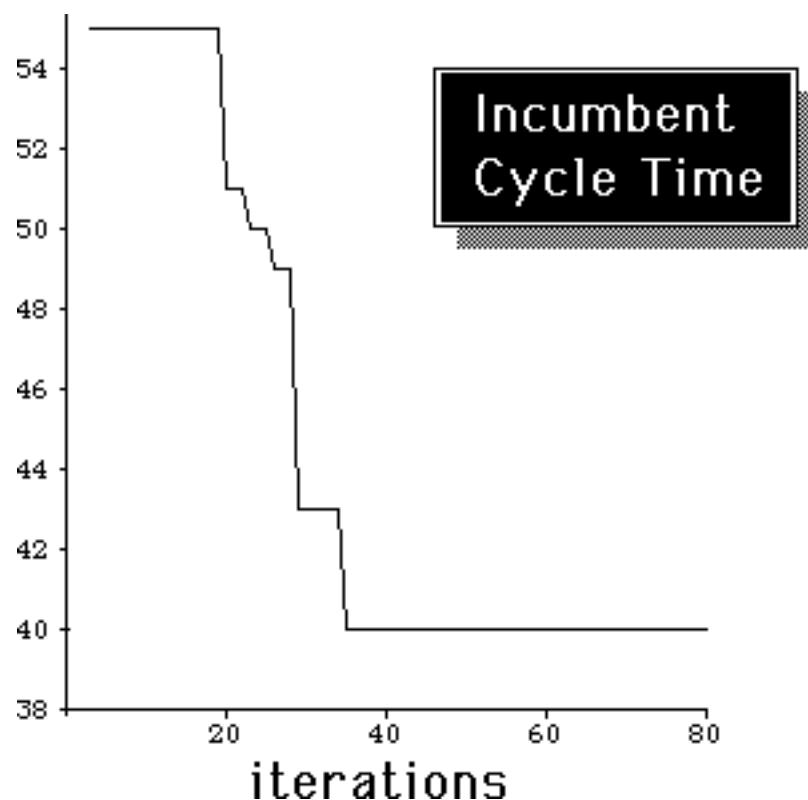


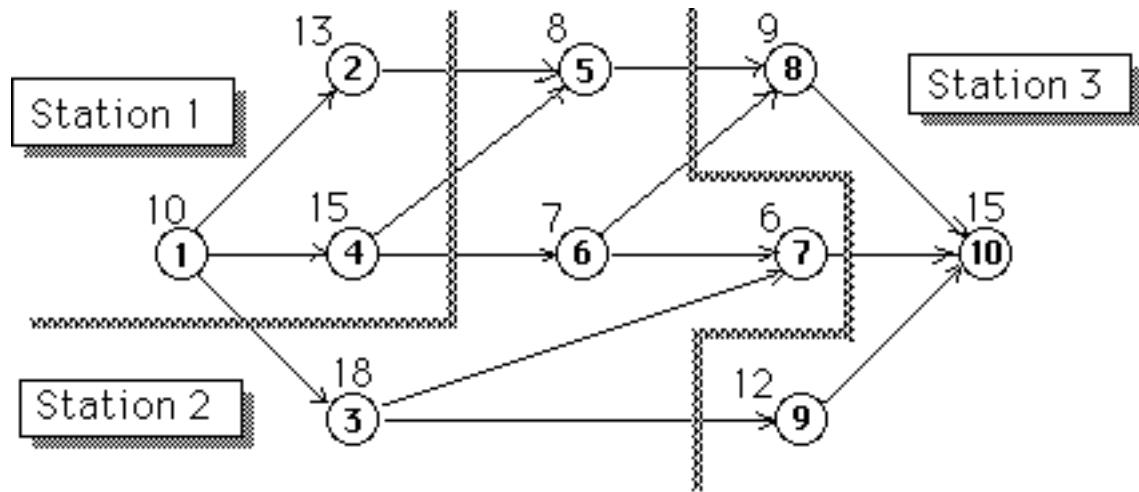
All solutions in the final population  
are feasible, with fitness = 167  
and cycle time = 43

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.





Cycle time =  $\text{Minimum}\{38, 39, 37\} = 39$

Optimal  
Solution

1 1 2 1 2 2 2 3 3 3

The optimal cycle time is 39



## APL Code

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



```
    ^GA;detail;toprinter;Max_iterations;I;detail;sho
[1] A
[2] A      Genetic Algorithm
[3] A      for Assembly Line Balancing Problem
[4] A      namely, minimize Cycle Time, s.t. #stations
[5] A
[6] A      Global variables used:
[7] A          Nstations = # of stations to be used
[8] A          P          = vector of task times
[9] A          A          = adjacency matrix of precedenc
[10] A         Popsize   = population size
[11] A         kkk        = scalar > 1
[12] A         Pcross    = probability of crossover
[13] A         Pmutate   = probability of mutation
[14] A
```

```
[15]  show_pop<-detail<-ASKYN 'Show details?'
[16]  $detail/'show_pop<-ASKYN ''Show pools?'' '
[17]  Max_iterations<-Request_Box 'max # iterations'
[18]  toprinter<-AskPrint ◊ PrSelect
[19]  Prologue
[20] A
[21] POP<-Initial_Pool(Popsize,Nstations,ρP)
[22] CycleTime<-Evaluate_Pool POP
[23] Fitness<-(Γkkk×UB<-Γ/CycleTime)-CycleTime
[24] Incumbent<-10 ◊ MinCycleTime<-BIG
[25] MaxVals<-MinVals<-AvgVals<-NumFeas<-10 ◊ iteration<-0
[26] Record_Status
[27] 'Random Seed: ',$Set_Random_Seed
```

```
[28] A
[29]  $(~detail) /'OL ◊ UNDERLINE ''-Generation--Minimu
      easible-'''
[30] Next:=>Stop ∆if Max_iterations<iteration+iteration
[31] A
[32] →Converged ∆if(Γ/Fitness)=L/Fitness
[33] A
[34] A      Select pool of individuals who will survive
[35] A      & be candidates for mating (& mutation)
[36] I←Select_Pool Fitness
[37] Report_GA
```

```
[38] A
[39] A      Mate pairs in the selected pool
[40] POP←Crossover POP[1;]
[41] A
[42] A      Mutate individuals in the pool
[43] POP←Mutate POP[1;]
[44] A
[45] Fitness←(F_kkkxUB)−CycleTime←Evaluate_Pool POP
[46] Record_Status
[47] A      End of iteration
[48] →Next
[49] '*** Maximum # of iterations performed! ***'
[50] →Stop
[51] Converged:'*** Converged! ***'
[52] A
[53] Stop:'*** terminated ***'
      ▽
```

```
    ▽POOL←Initial_Pool X;msg;r1
[1] A
[2] A      Get initial population for genetic algorithm
[3] A
[4] A      X[1]= population size
[5] A      X[2]= # of stations
[6] A      X[3]= # of tasks
[7] A
[8] →Random Δif ASKYN 'Random initial genetic pool?'
[9] Enter:POOL←‡30 Alpha_Request_Box 'name of initial
[10] →OK Δif^(X[1]=1↑pPOOL),(X[3]=‐1↑pPOOL),,POOL∈iX
[11] Message_Box '•Error!•Please try again'
[12] →Enter
[13] Random:r1←Set_Random_Seed
[14] POOL←?X[1] 3]pX[2]
[15] OK:
    ▽
```

```
    ^Report_GA;table;M;output
[1] A
[2] A      Report results in iteration of
[3] A      Genetic Algorithm for ALB problem
[4] A
[5]  !detail/'SHIFT^CTR UNDERLINE ''Generation #'',#iteration
[6]  M+1↑pPOP
[7]  →(~show_pop)/Summary
[8]  table←'I2,< | >,I6,< | >'  DFMT(4(2,M)P(iM),Fitness)
[9]  table←table,'I2' DFMT POP
[10] DL,' #  Fitness  ---mating pool----'
[11] table ◊ PAUSE ◊ →End
[12] Summary:
[13] output←(iteration,(-1↑MinVals),(-1↑MaxVals),(-1↑AvgVals)
[14] 'I7,2I10,F10.2,I10' DFMT 1 5poutput
[15] End:
  ↴
```

```
    ▽Record_Status;b;i
[1] A
[2] A      Record status of the population
[3] A      at the end of each generation
[4] A
[5] MinVals←MinVals,L/Fitness
[6] MaxVals←MaxVals,Γ/Fitness
[7] AvgVals←AvgVals,AVERAGE Fitness
[8] NumFeas←NumFeas,+/0=nv←Number_Violations POP
[9] →End Δif~^/b←(nv=0)^CycleTime<~1↑MinCycleTime
[10] MinCycleTime←MinCycleTime,CycleTime[i←1↑b/iPopszie
[11] Incumbent←POP[i;]
[12] OCC 35 ◊ →0
[13] End:MinCycleTime←MinCycleTime,~1↑MinCycleTime
    ▽
```

```
    ▽z←Evaluate_Pool  POP;i;M
[1]  A
[2]  A      Evaluate the individuals in the pool
[3]  A      (cycle time of assignment)
[4]  A
[5]  i←0  ◊ M←(ρPOP)[1]  ◊ z←Mρ0
[6]  Next:→End  △if  M<i←i+1
[7]  z[i]←Ctime  POP[i;]
[8]  →Next
[9] End:
    ▽
```

```
    ▽t←Ctime pop
[1] A
[2] A      Compute Cycle time for solution "pop"
[3] A      where pop[i]= station to which task i is
[4] A      assigned.  Include penalty times number
[5] A      of precedence violations.
[6] A
[7] A      Global variable: Penalty
[8] A                  P = vector of task times
[9] A
[10] t←r/((iNstations)++.=pop)+.xP    A Cycle time
[11] t←t++/Penaltyx,A>pop++.≤pop    A Violation penalty
    ▽
```

```
     $\triangleright z \leftarrow \text{Number\_Violations pop}; i$ 
[1] A
[2] A      Compute number of violations of
[3] A      precedence restrictions
[4] A
[5] A      Global variable: A = precedence matrix
[6]    $\rightarrow$  Single  $\triangleleft$  if  $1 = pppop$ 
[7]    $z \leftarrow (1 \uparrow ppop) \rho 0 \diamond i \leftarrow 0$ 
[8] Next:  $\rightarrow$  End  $\triangleleft$  if ( $\rho pop$ ) $[1] < i \leftarrow i + 1$ 
[9]    $z[i] \leftarrow +/ , A > pop[i;] \dots \downarrow pop[i;]$ 
[10]   $\rightarrow$  Next
[11] Single:  $z \leftarrow +/ , A > pop \dots \downarrow pop$ 
[12] End:
       $\triangleright$ 
```

```
    ▽I←Select_Pool F;S;N;s0;s;f;n;a;b
[1]  A
[2]  A      Select individuals from pool according
[3]  A      to their fitness values F
[4]  A      Fitness values are scaled so as
[5]  A          to sum to 1.0 and
[6]  A          Max Fitness = scalefactor+N
[7]  A      Global variable:  GA_Scale_Factor
[8]  a←(GA_Scale_Factor-1)÷((PopsizexF)-+/F)
[9]  b←(-axL/F)F(GA_Scale_Factor+Popsize)-axF/F
[10] F←b+axF
[11] A
[12] S←+/F ◇ f←+/F
[13] s0←(S÷N+ρF)×0.001×?1000
[14] s←s0+(S÷N)×0,1N-1
[15] n←n-0,-1↓n←+/(f*.≥s)
[16] →0 ◇if Popsize=ρI←((n>0)/n)\(n>0)/1N
    ▽
```

```
    ▽Q←Sample_Fitness;N;S;s0;s;F
[1] A
[2] A      Select individuals from the population,
[3] A      who will survive to the next generation.
[4] A      Argument: Fitness = vector of fitness values
[5] A          of the individuals in the population,
[6] A      Result: Q = vector of number of copies of
[7] A          individuals to be included in the next ge
[8] A
[9] A      Method used is "stochastic universal samplin
[10] A          due to J.E. Baker
[11] A
[12] S←+/Fitness ◊ N←pFitness ◊ F←\Fitness
[13] s0←(?1000)×N÷1000×S
[14] s←s0+(S÷N)×0,1N-1
[15] Q←(+/F◦.2s)−(+/(0,¬1↓F)◦.2s)
    ▽
```

```
    ▽Q←Crossover Pop;M;N;PAIR;I
[1] A      For each pair of individuals
[2] A      in the mating pool, either mate
[3] A      them & put the 2 offspring into
[4] A      the pool, or else copy them into
[5] A      the new pool directly
[6] N←(pPop)[1] ◊ M←(pPop)[2]
[7] Q←(0,¬1↑pPop)ρ0
[8] A      Randomly shuffle the pool
[9] Pop←Pop[N?N;]
[10] Next:→End △if 2>(pPop)[1]
[11] A      Remove pair from the pool
[12] PAIR←(2,M)↑Pop ◊ Pop←2 0↓Pop
[13] A
[14] →Copy △if Pcross<(?1000)÷1000
[15] A      Choose crossover point
[16] Q←Q,[1]((2,I)↑PAIR),Φ[1](0,I←1+?M-2)↓PAIR
[17] →Next
[18] Copy:Q←Q,[1]PAIR ◊ →Next
[19] End:Q←Q,[1]Pop
    ▽
```

```

    ▽Q←Mutate_Pop;I
[1] A
[2] A      Perform mutation of the individuals in pool
[3] A
[4] Q←,Pop
[5] I←(Pmutate≥?(ρQ)ρ10000)÷10000)/iρQ
[6] Q[I]←?(ρI)ρNstations
[7] Q←(ρPop)ρQ
    ▽

```

