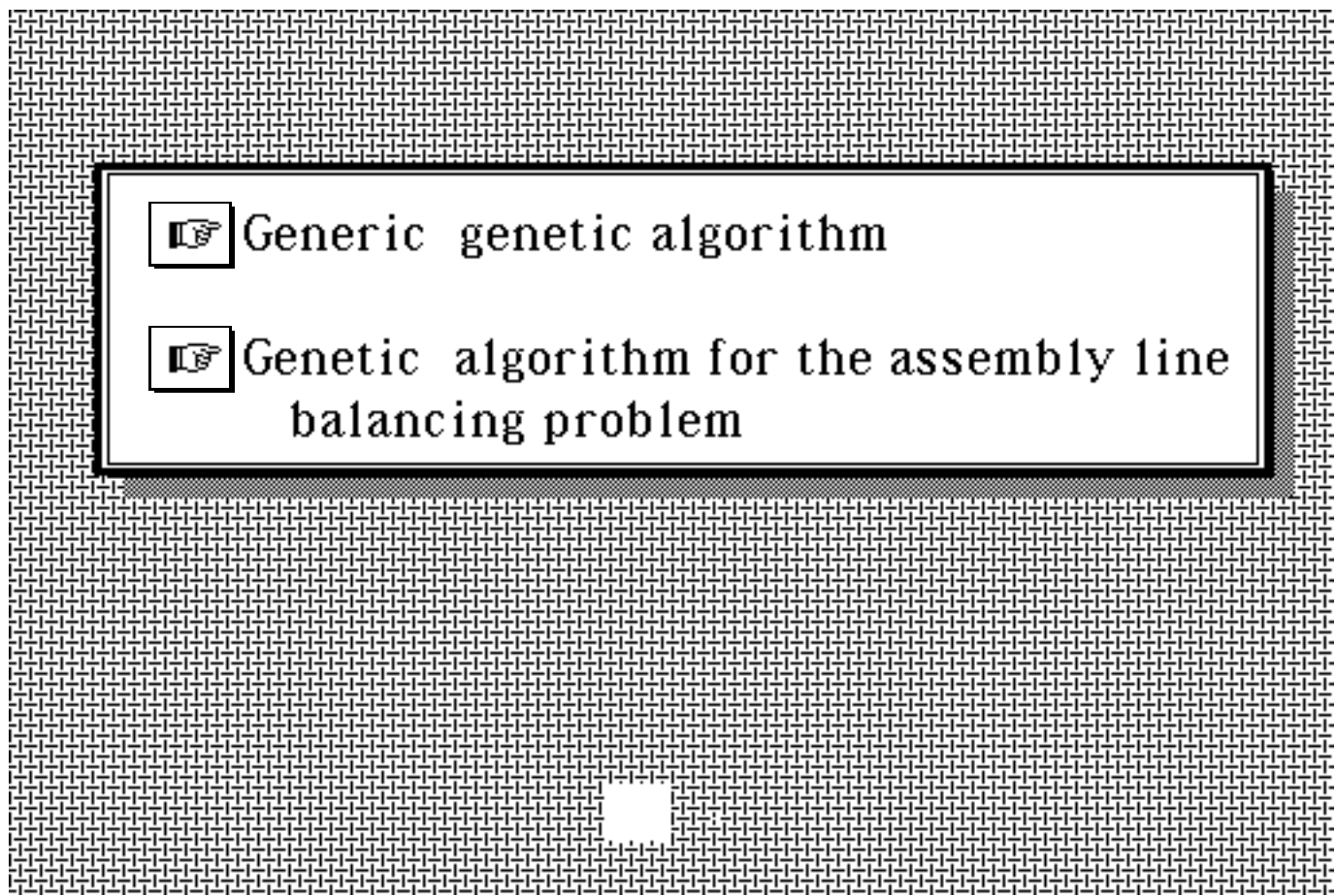# Genetic Algorithm for Line Balancing

author

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

☞ Generic genetic algorithm

☞ Genetic algorithm for the assembly line balancing problem

Genetic Algorithms (GAs)

- heuristic methods--do <u>not</u> guarantee optimality

- iterative methods

- simulate biological evolution, which has enabled "nature" to develop species remarkably well-adapted to their environment.

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.

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 $p_i = \dfrac{f(i)}{\sum\limits_{j=1}^{N} f(j)}$

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.

0   S

| f(1) f(2)   f(3)   f(4)   f(N−2) f(N−1) f(N)
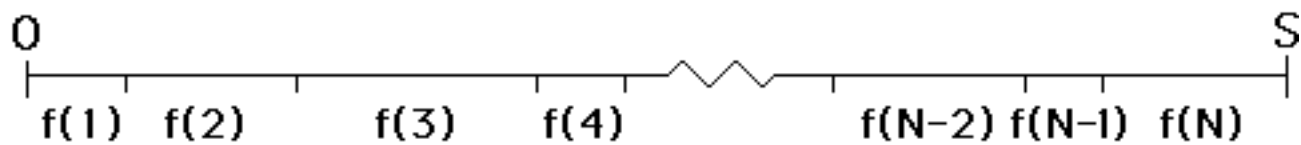
Stochastic Universal Sampling

©Dennis Bricker, U. of Iowa, 1997

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

$$s_0$$

0   S/N
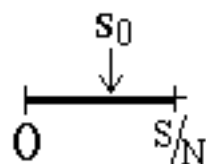
4. Compute the N equally-spaced numbers

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

0   s_0 s_1 s_2 s_3 s_{N-4} s_{N-3} s_{N-2} s_{N-} S

| f(1) f(2)   f(3)   f(4)   f(N−2) f(N−1) f(N)
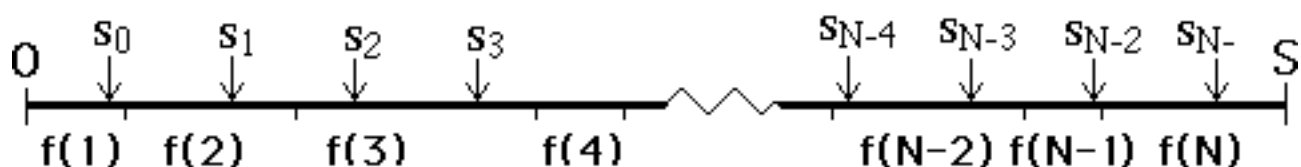
©Dennis Bricker, U. of Iowa, 1997

5. Let n_k be the number of values in the set $\{s_i\}$ which fall within interval k.
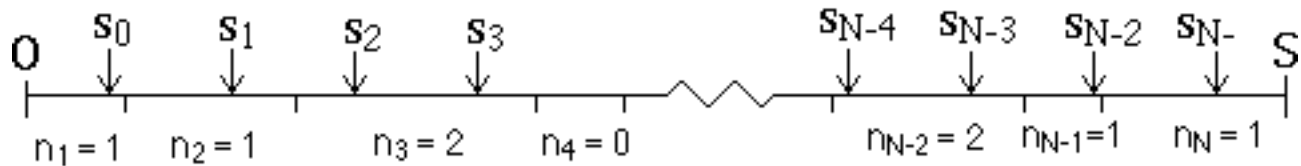
6. For each k (k=1,2,...N), create n_k copies of individual k and put into the mating pool.

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 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

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

and

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

Solution: Choose a & b so that

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

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

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

CROSSOVER

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

| ABCAA BBCC | → | ABCAA | BBCC | → | ABCAA | a c b a |
| c b a c b a c b a | | c b a c b | a c b a | | c b a c b | BBCC |

break point
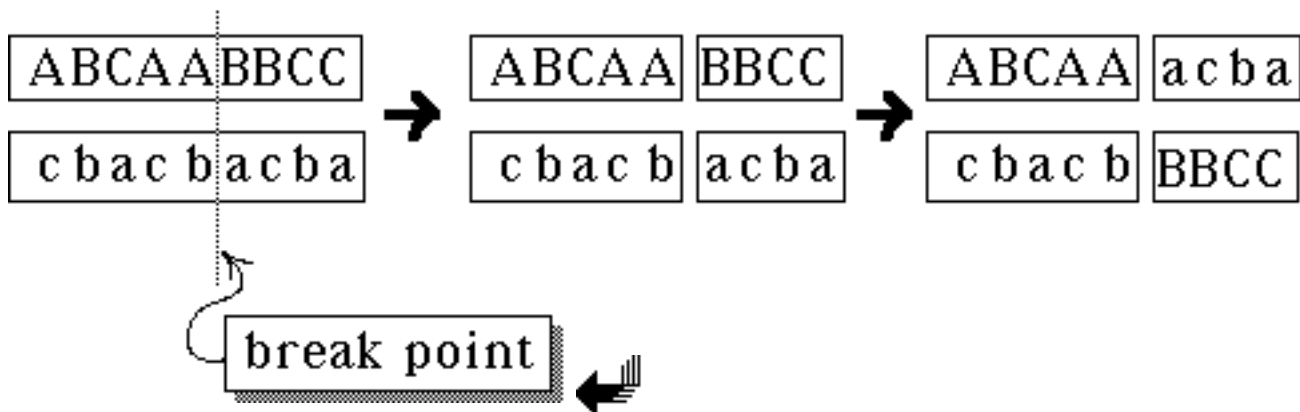
©Dennis Bricker, U. of Iowa, 1997

MUTATION

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

| ABCAABBCC | → | A**B**CAABBCC | → | A**C**CAABBCC |

A gene is selected at random

New value of gene is selected at random
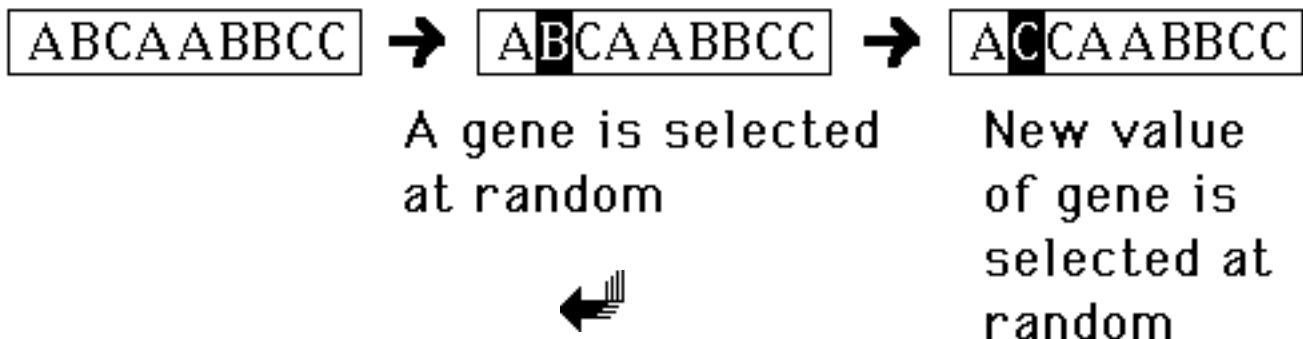
©Dennis Bricker, U. of Iowa, 1997

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.

↵

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 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 Implemen-
tation 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.
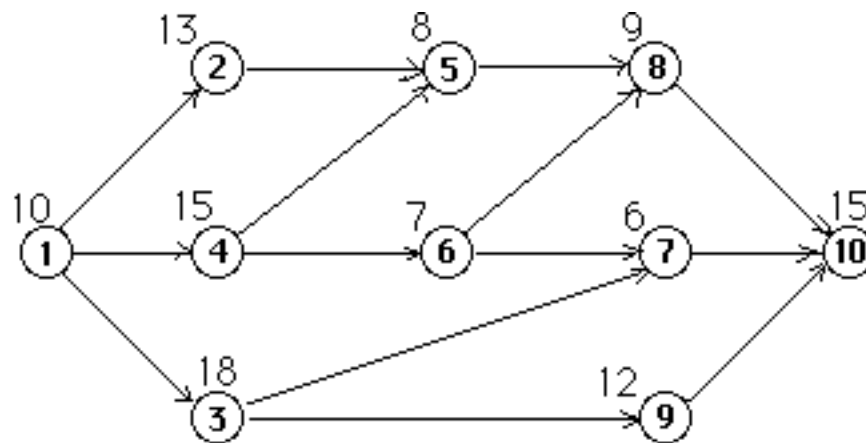
EXAMPLE

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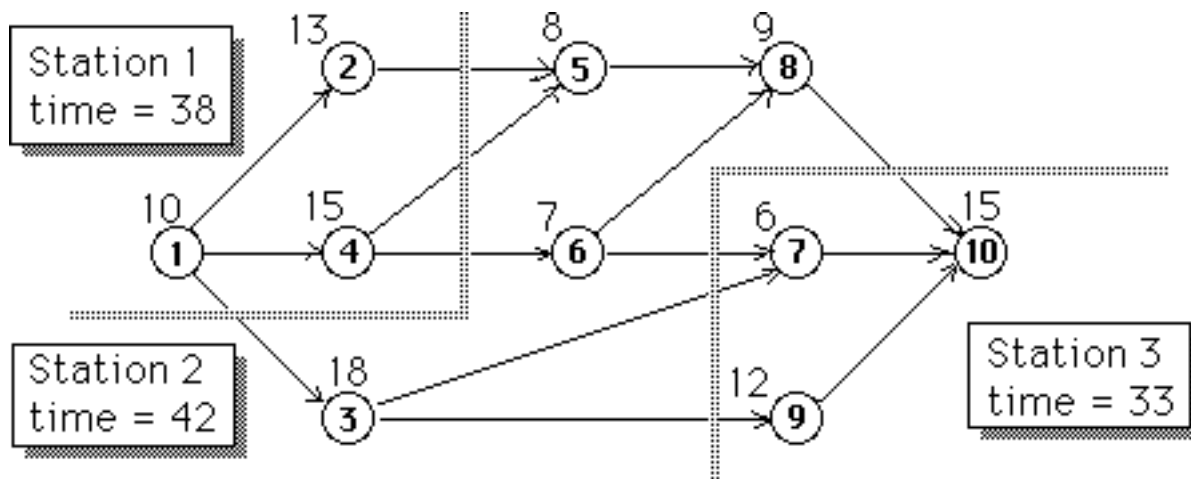
The assignment of the ten tasks to the three stations shown above results in a cycle time equal to $T_{max} = \max\{38, 42, 33\} = 42$.

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:

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

Cycle time = 43
= max{43, 30, 40}

1	2	1	1	2	3	3	2	3	3
1	2	3	4	5	6	7	8	9	10
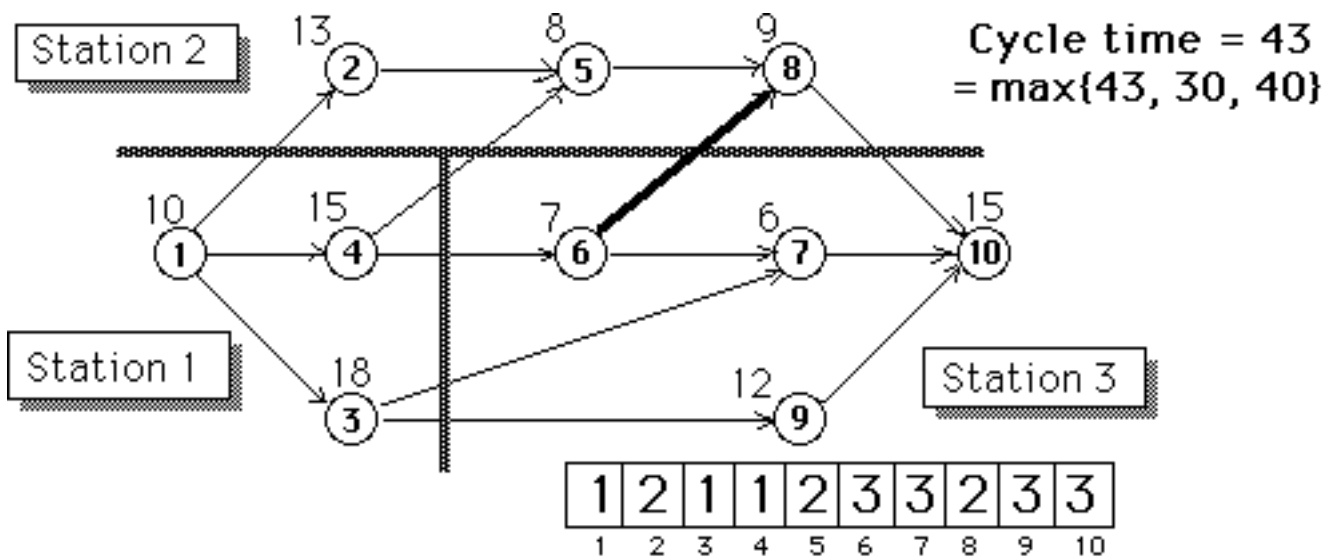
In the assignment shown, task 6 does not
precede task 8 as required, and so we reduce
the fitness by a penalty P=10:

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

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:
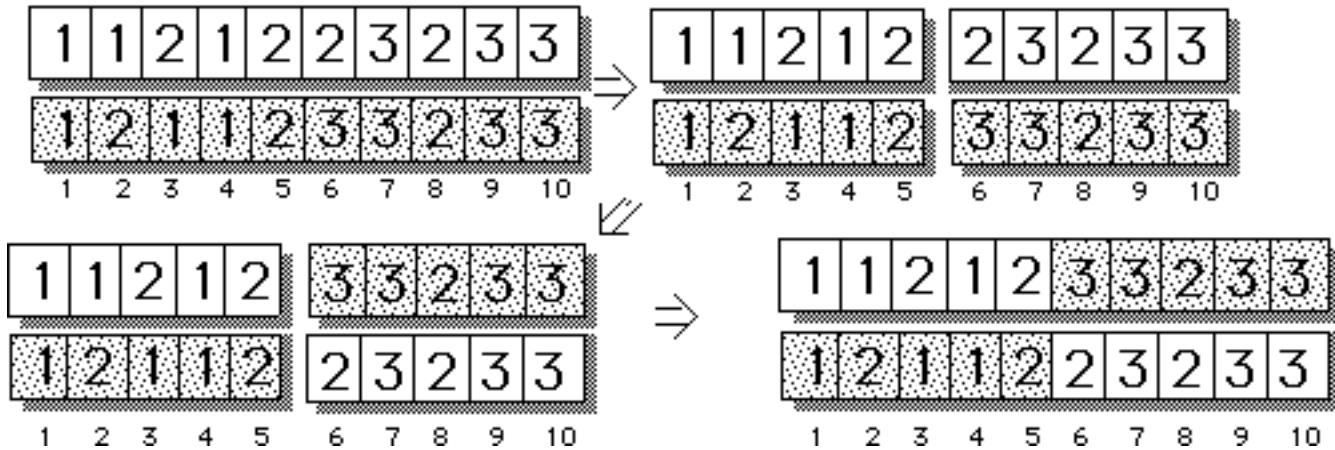
©Dennis Bricker, U. of Iowa, 1997

The two parents "mate" and pro-duce....

©Dennis Bricker, U. of Iowa, 1997

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

two "hybrids" of their parents!

1	2	1	1	2	2	3	2	3	3
1	2	3	4	5	6	7	8	9	10
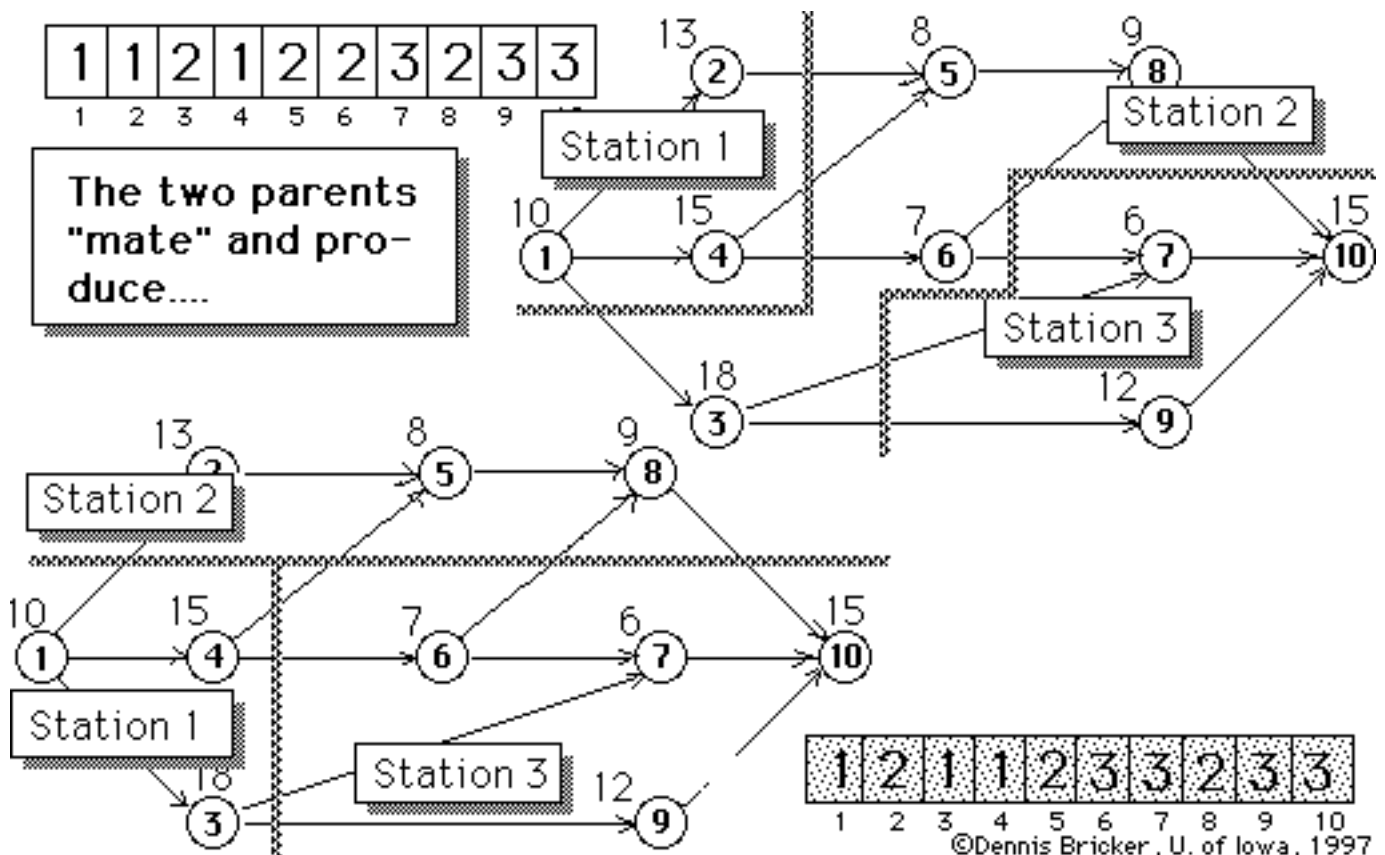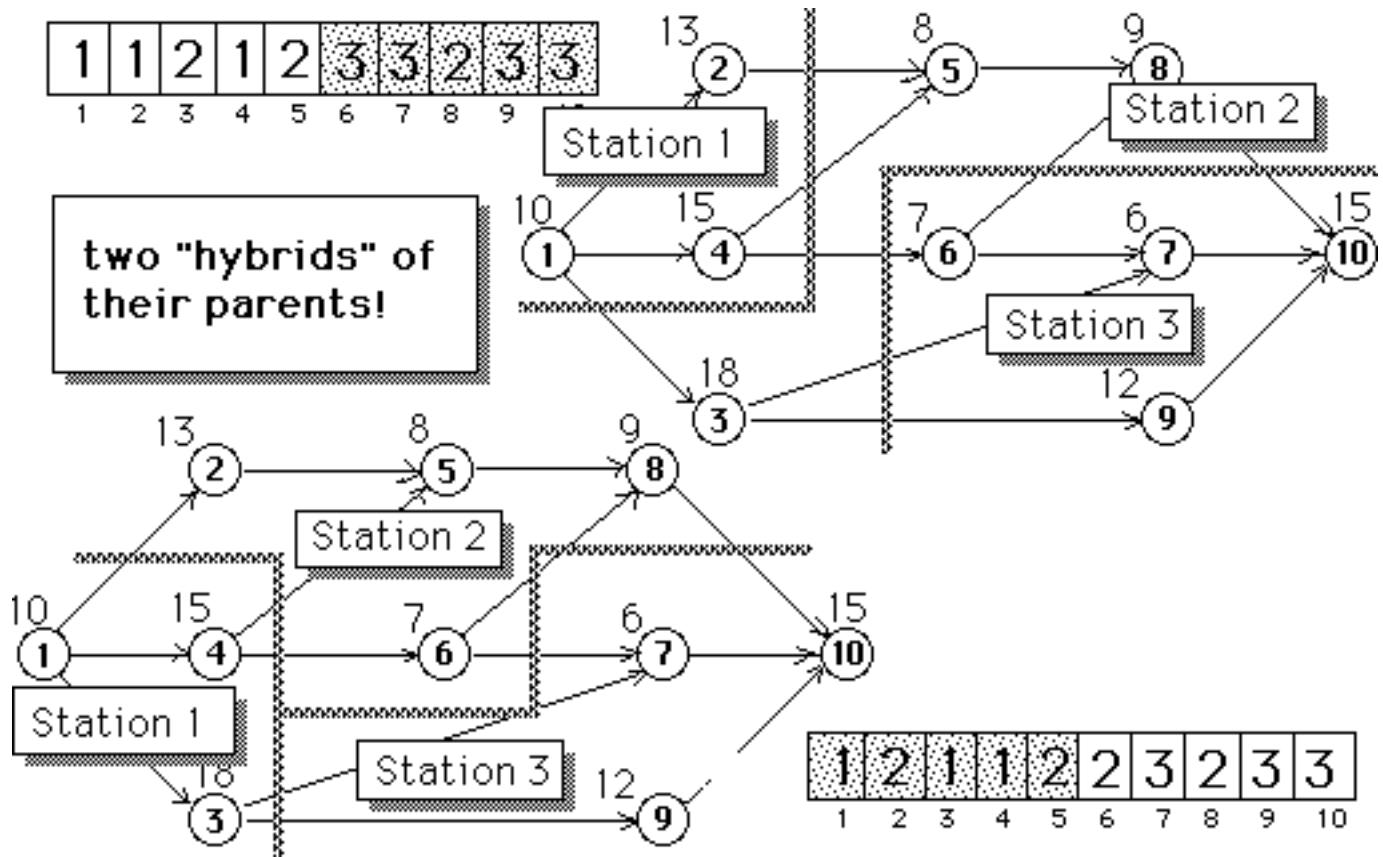
©Dennis Bricker, U. of Iowa, 1997

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.

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

➔

1	2	3	1	2	2	3	2	3	3
1	2	3	4	5	6	7	8	9	10
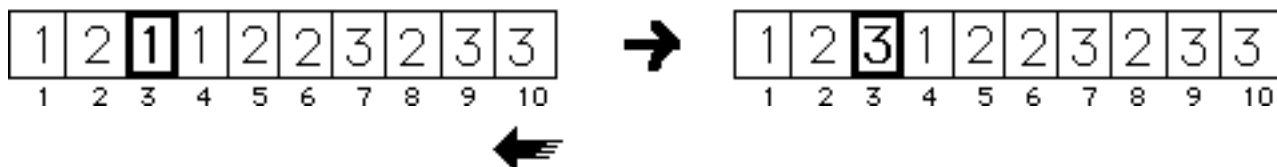
©Dennis Bricker, U. of Iowa, 1997

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      ⇐
```

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 2 3 2 3 3 3 3

None in final generation is feasible!

Station 1

(23)

13 ②   8 ⑤   9 ⑧

Station 3

(50)

10 ①   15 ④   7 ⑥   6 ⑦   15 ⑩

18 ③   12 ⑨

Station 2   (40)

Cycle time is 50
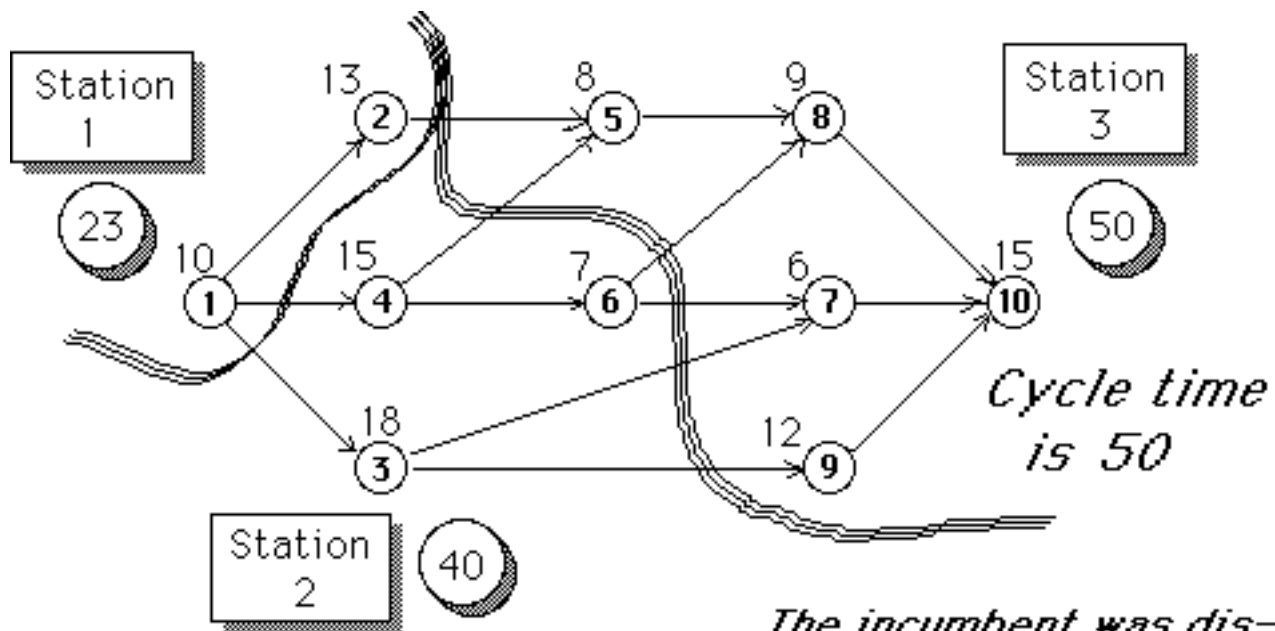
The incumbent was dis-covered in an earlier generation, but then disappeared!

Incumbent:

1	1	2	2	3	2	3	3	3	3

©Dennis Bricker, U. of Iowa, 1997

AVERAGE FITNESS

©Dennis Bricker, U. of Iowa, 1997

MAXIMUM FITNESS
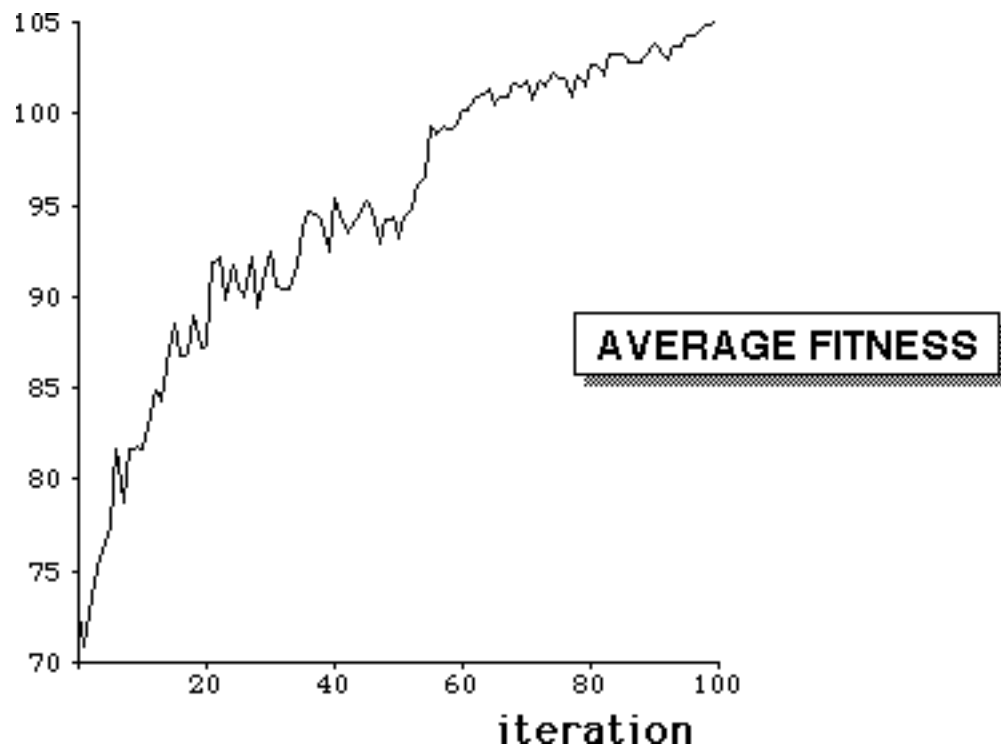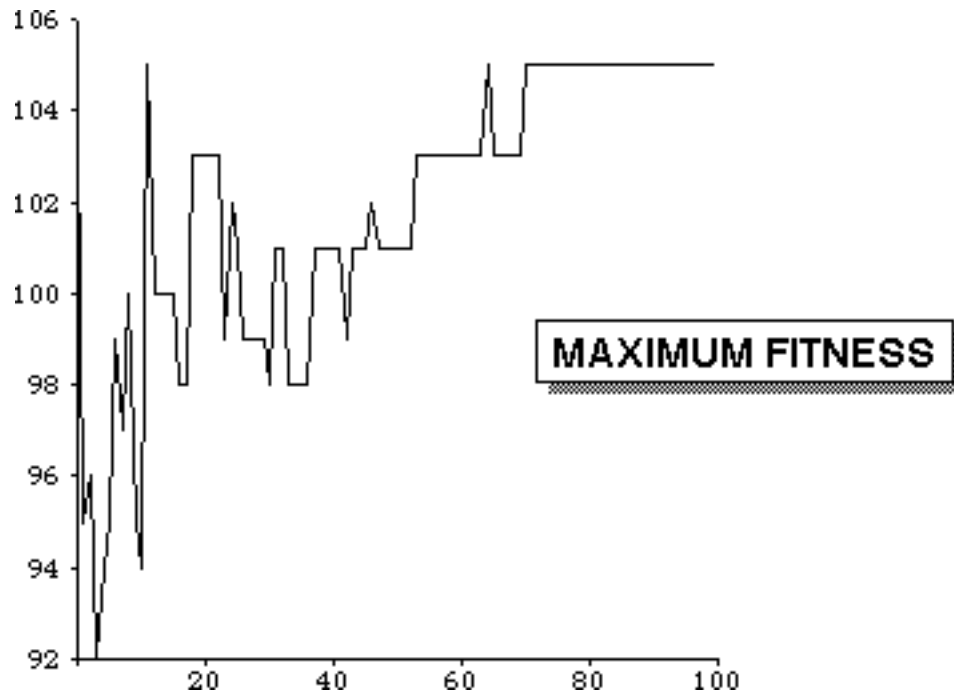
©Dennis Bricker, U. of Iowa, 1997

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	
3	117	45	1	
4	117	45	1	
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	
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	
12	117	45	1	
13	117	45	1	
14	117	45	1	
15	117	45	1	
16	117	45	1	
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	

©Dennis Bricker, U. of Iowa, 1997

Incumbent:

1	3	1	2	3	2	2	3	1	3

Cycle time is 45

MAX FITNESS

iteration #

AVERAGE FITNESS

©Dennis Bricker, U. of Iowa, 1997

MINIMUM FITNESS

©Dennis Bricker, U. of Iowa, 1997

# of Feasible Individuals in Population

iteration #

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

generation		Fitness		#feasible
	min	max	average	
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!

Incumbent

1	2	1	1	3	1	3	3	3	3

The incumbent was dis-covered in an earlier generation, and then disappeared!

Cycle time is 50

INCUMBENT

MAX FITNESS

AVERAGE FITNESS

©Dennis Bricker, U. of Iowa, 1997

MINIMUM FITNESS
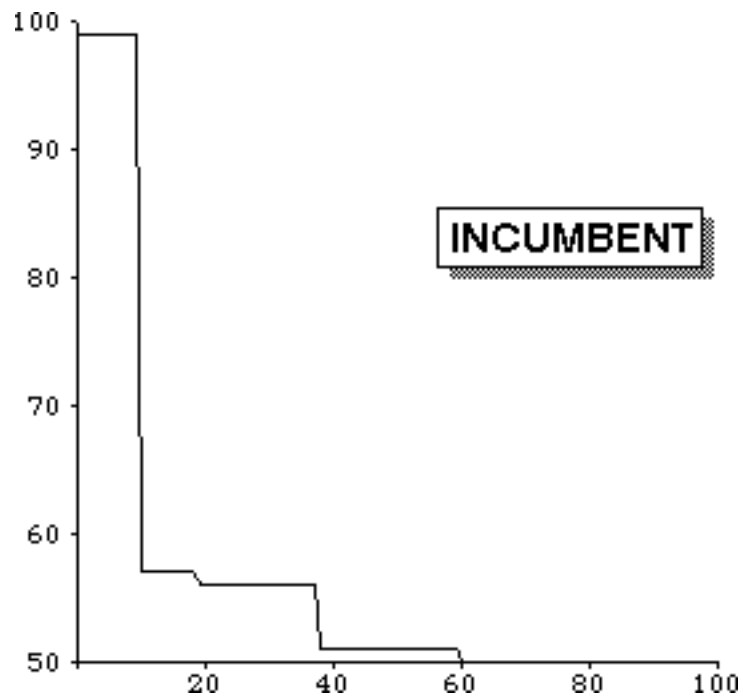
©Dennis Bricker, U. of Iowa, 1997

# of Feasible individuals in the population

©Dennis Bricker, U. of Iowa, 1997

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

Converges in 76th genera-tion

| | | Fitness | | |
generation	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!***

©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
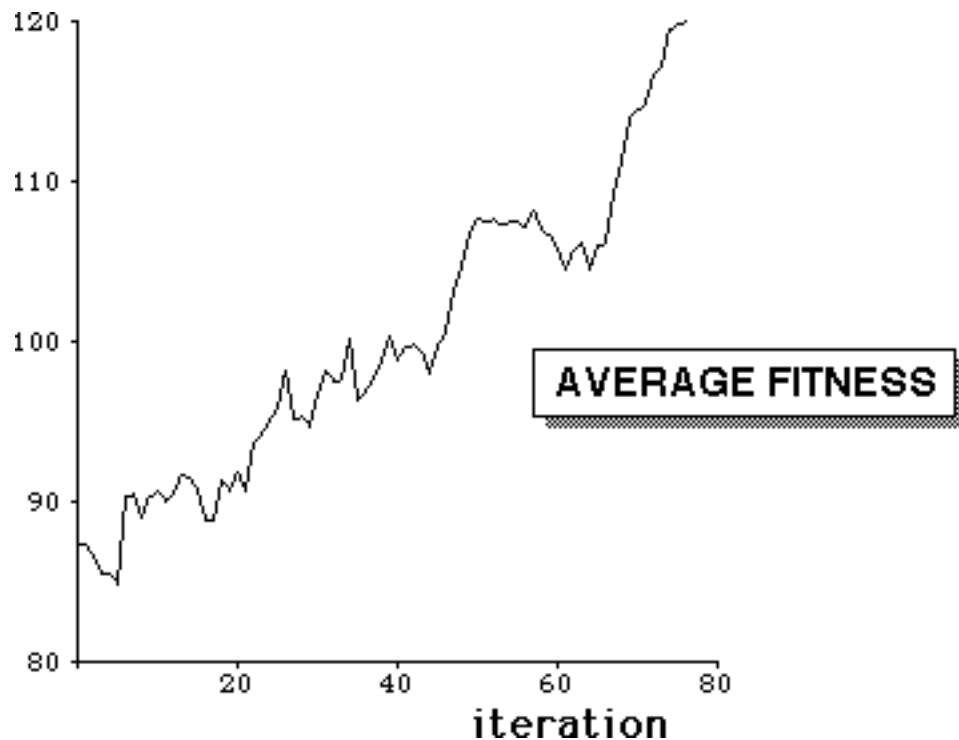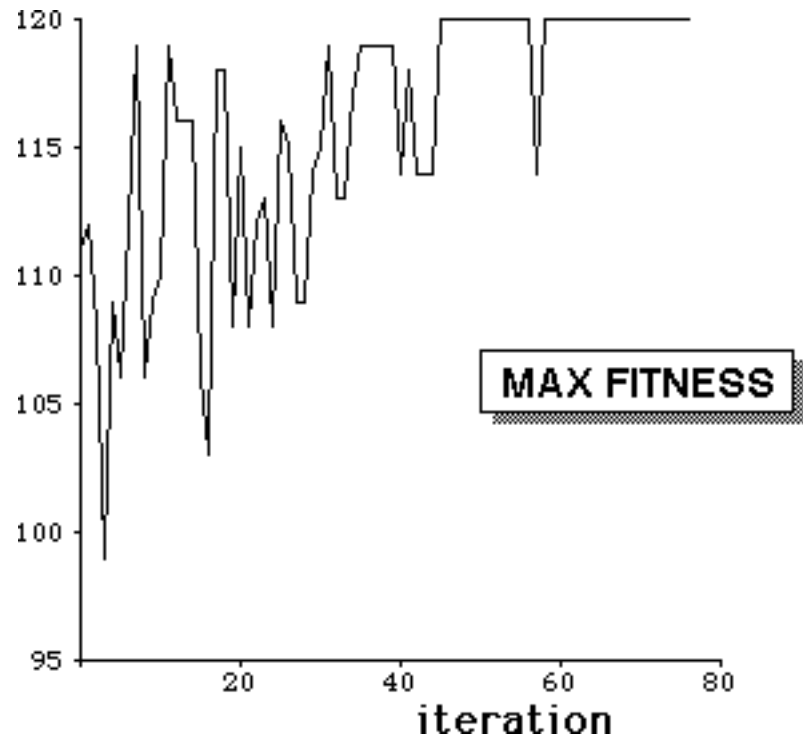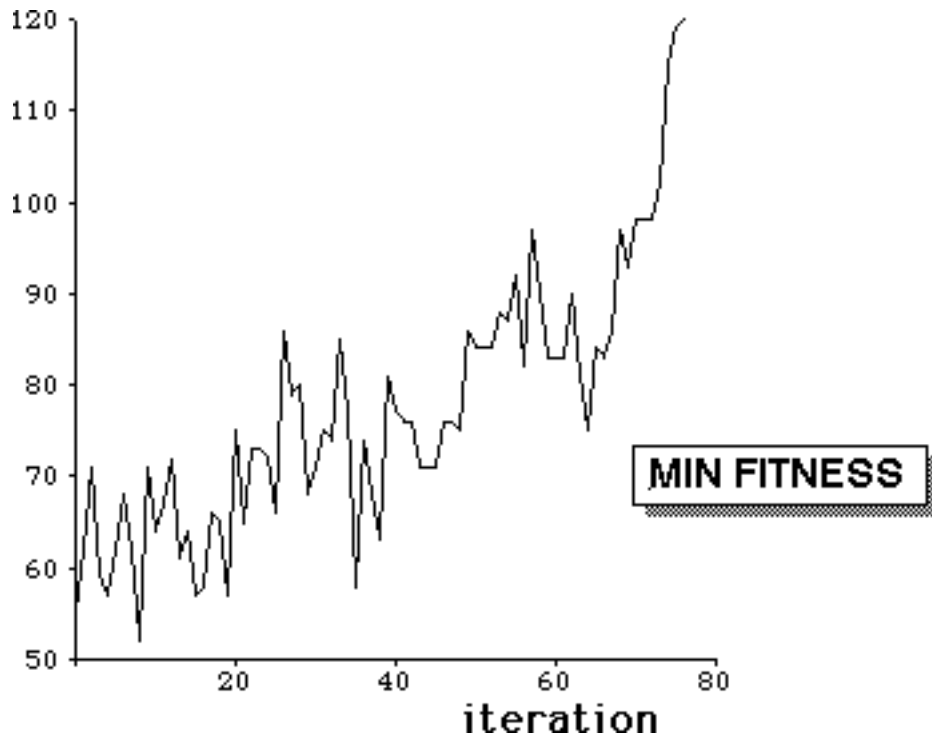
AVERAGE FITNESS

©Dennis Bricker, U. of Iowa, 1997

MAX FITNESS

©Dennis Bricker, U. of Iowa, 1997

MIN FITNESS

©Dennis Bricker, U. of Iowa, 1997

Number of Feasible Individuals in the generation
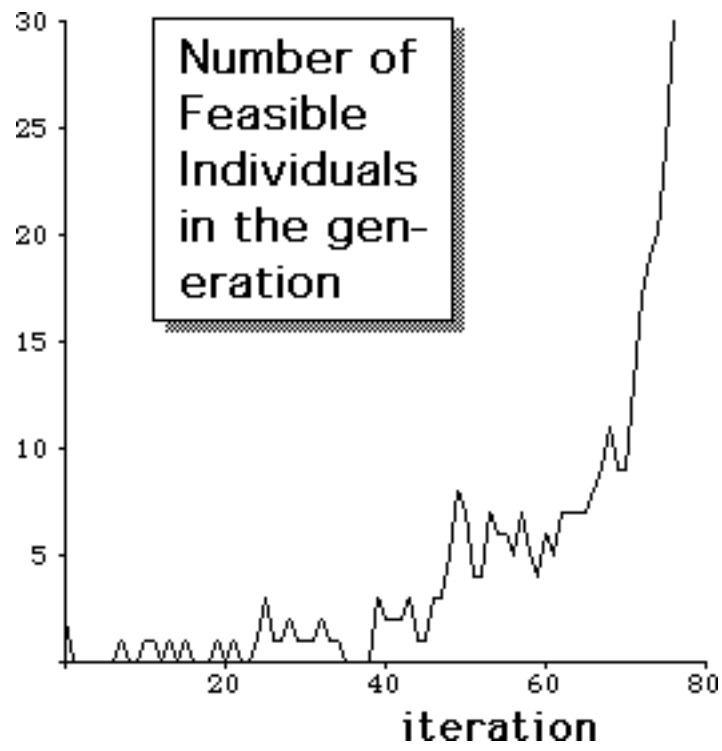
©Dennis Bricker, U. of Iowa, 1997
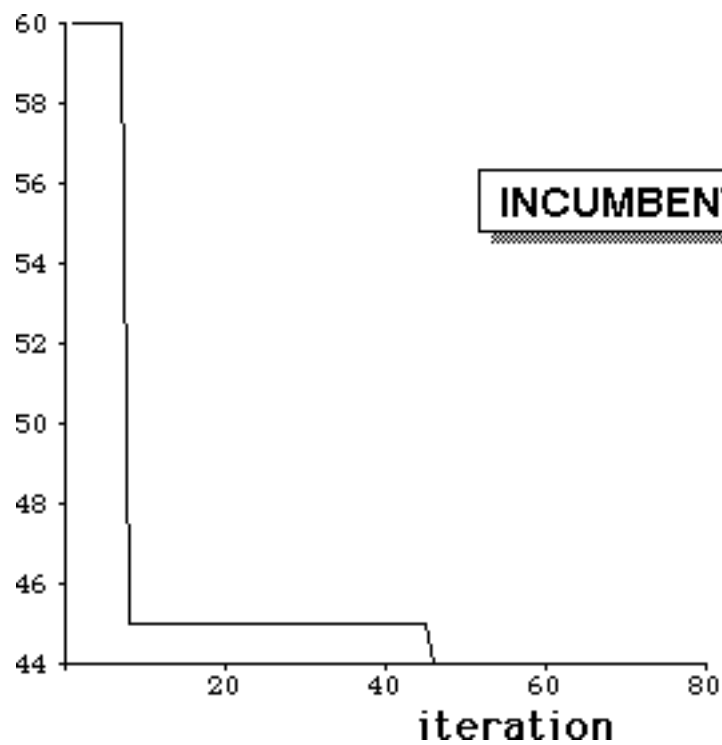
INCUMBENT

©Dennis Bricker, U. of Iowa, 1997
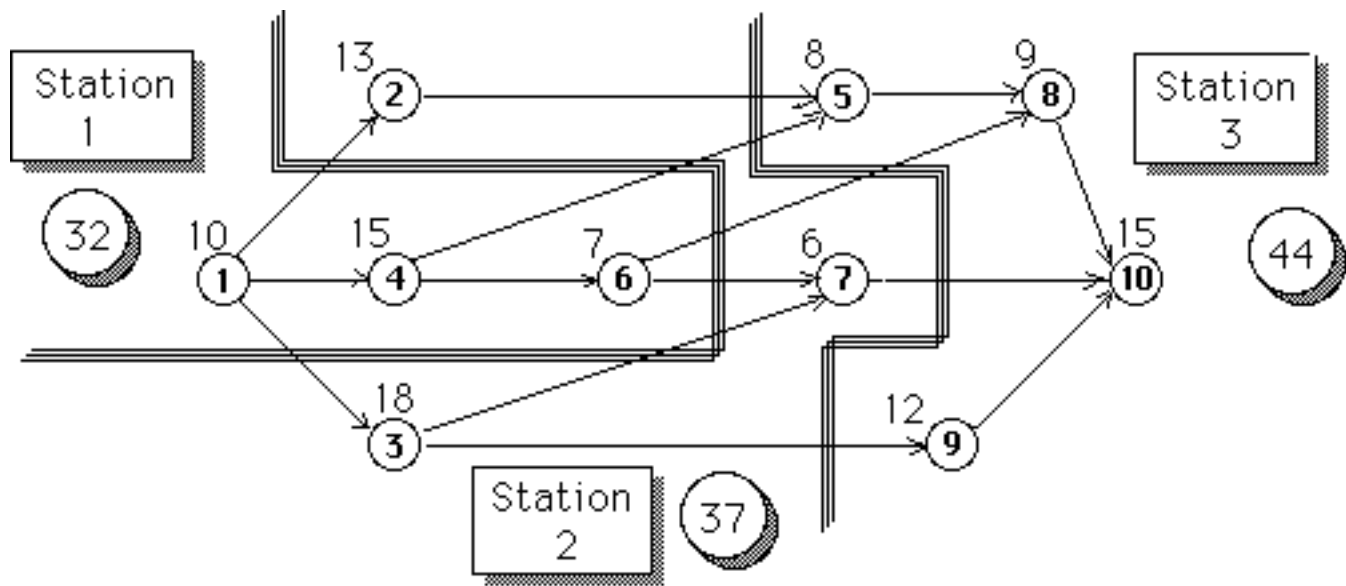
Incumbent:

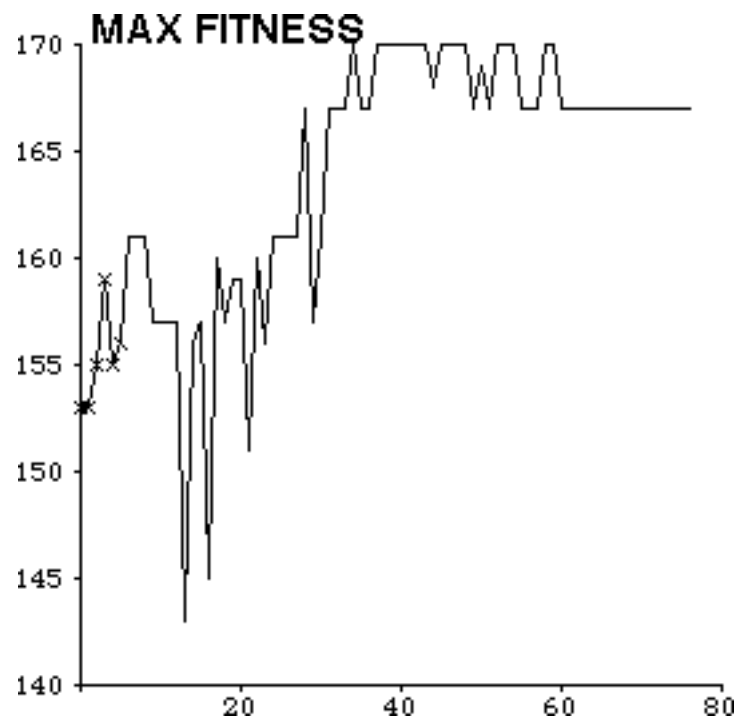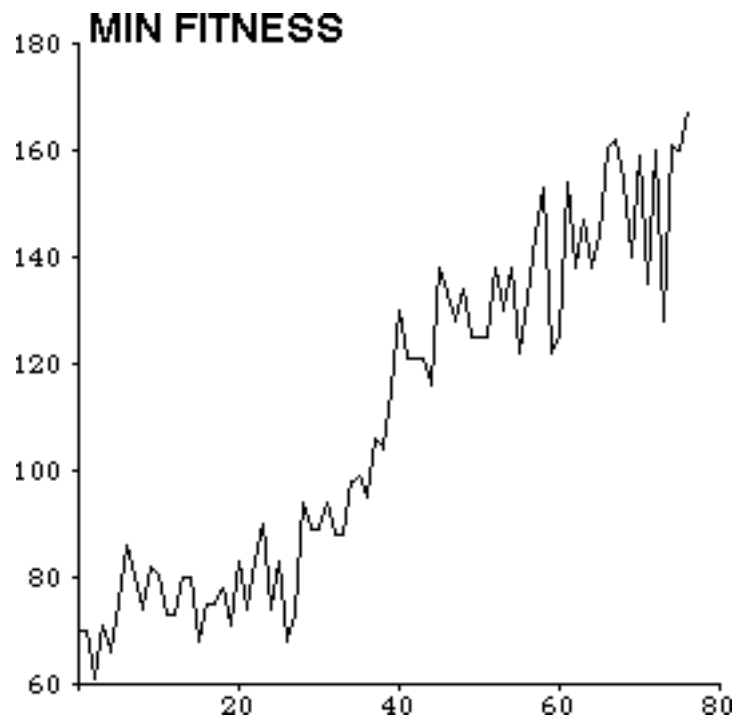| 1 | 2 | 2 | 1 | 3 | 1 | 2 | 3 | 3 | 3 |

Cycle time is 44

©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.

MAX FITNESS

©Dennis Bricker, U. of Iowa, 1997

MIN FITNESS

©Dennis Bricker, U. of Iowa, 1997

AVERAGE FITNESS
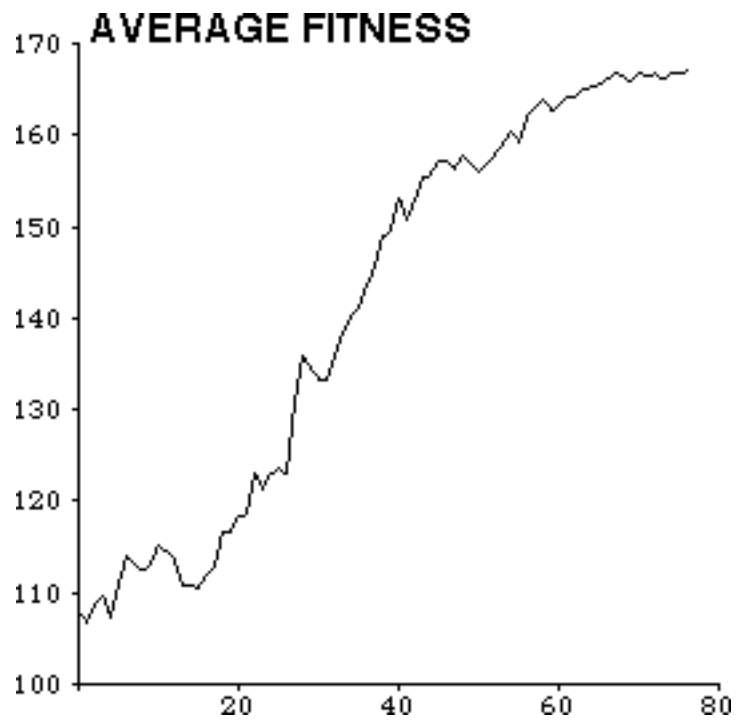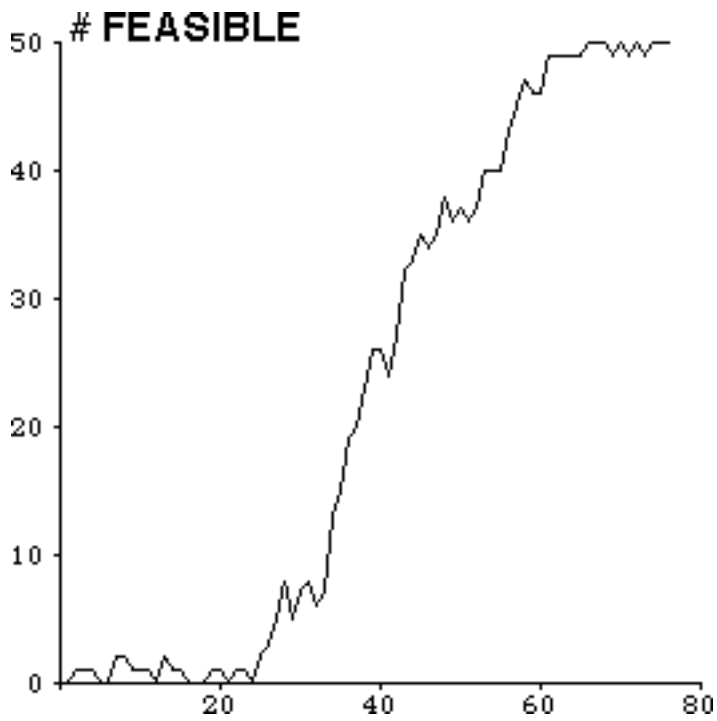
FEASIBLE

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.

Incumbent Cycle Time

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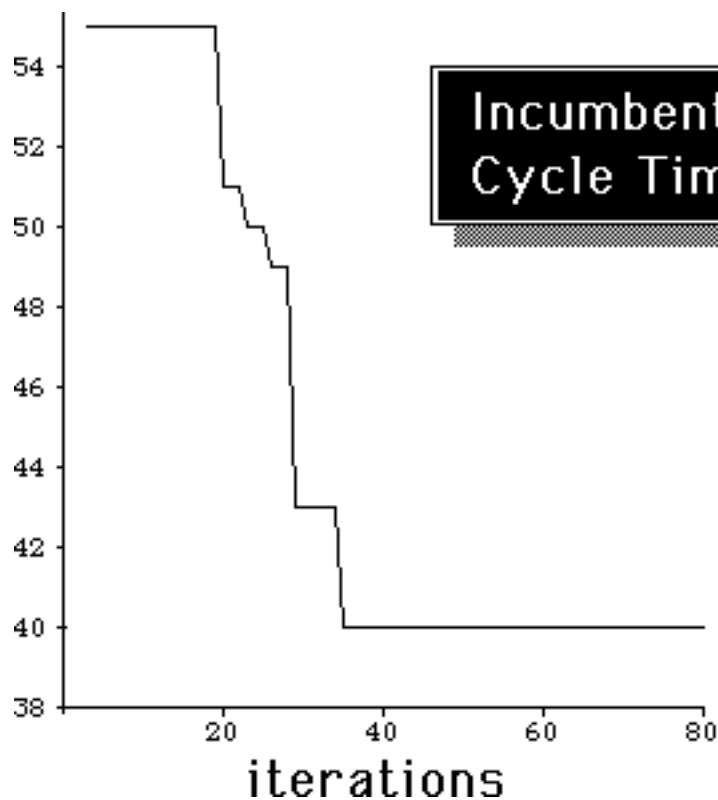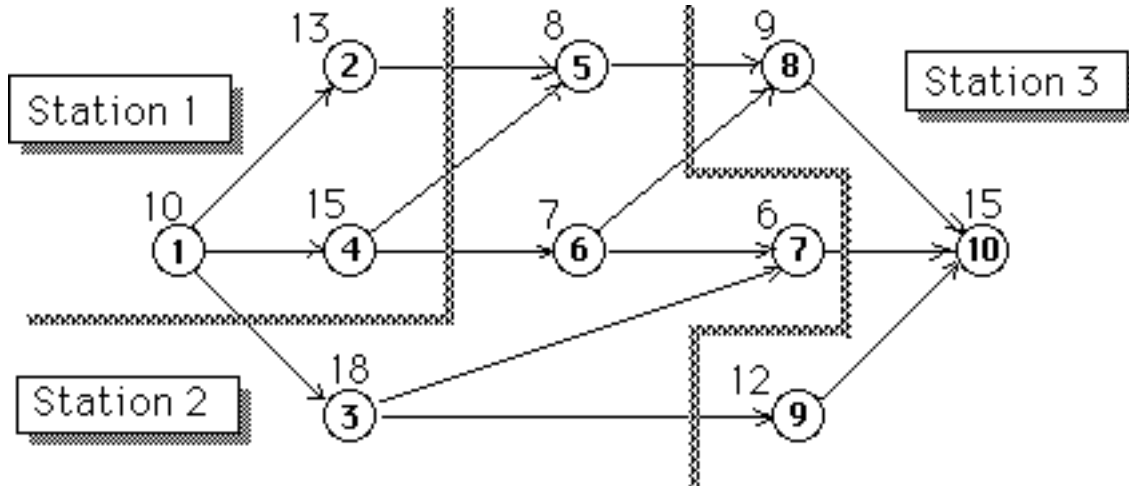
Cycle time = Minimum{38, 39, 37} = 39

1	1	2	1	2	2	2	3	3	3

Optimal Solution

The optimal cycle time is 39

←

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

```
[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]   ⍝
[21]    POP←Initial_Pool(Popsize,Nstations,⍴P)
[22]    CycleTime←Evaluate_Pool POP
[23]    Fitness←(⌈kkk×UB←⌈/CycleTime)-CycleTime
[24]    Incumbent←⍳0 ◇ MinCycleTime←BIG
[25]    MaxVals←MinVals←AvgVals←NumFeas←⍳0 ◇ iteration←0
[26]    Record_Status
[27]    'Random Seed: ',⍕Set_Random_Seed
```

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

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

```
        ∇POOL←Initial_Pool X;msg;rl
[1]    A
[2]    A        Get initial population for genetic algorith
[3]    A
[4]    A        X[1]= population size
[5]    A        X[2]= # of stations
[6]    A        X[3]= # of tasks
[7]    A
[8]     →Random ∆if ASKYN 'Random initial genetic pool?'
[9]    Enter:POOL←⍕30 Alpha_Request_Box 'name of initial
[10]    →OK ∆if^/(X[1]=1↑ρPOOL),(X[3]=¯1↑ρPOOL),,POOL∊⍳X
[11]   Message_Box '∙Error!∙Please try again'
[12]    →Enter
[13] Random:rl←Set_Random_Seed
[14]    POOL←?X[1 3]ρX[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↑ρPOP
[7]    →(~show_pop)/Summary
[8]    table←'I2,< | >,I6,< | >' ⎕FMT(⍉(2,M)ρ(⍳M),Fitness)
[9]    table←table,'I2' ⎕FMT POP
[10]   ⎕L,' #   Fitness   ---mating pool----'
[11]   table ◇ PAUSE ◇ →End
[12] Summary:
[13]   output←(iteration,(¯1↑MinVals),(¯1↑MaxVals),(¯1↑AvgVals)
[14]   'I7,2I10,F10.2,I10' ⎕FMT 1 5ρoutput
[15] End:
        ∇
```

```
        ∇Record_Status;b;i
[1]   ⍝
[2]   ⍝         Record status of the population
[3]   ⍝         at the end of each generation
[4]   ⍝
[5]    MinVals←MinVals,⌊/Fitness
[6]    MaxVals←MaxVals,⌈/Fitness
[7]    AvgVals←AvgVals,AVERAGE Fitness
[8]    NumFeas←NumFeas,+/0=nv←Number_Violations POP
[9]    →End ⍝if~∨/b←(nv=0)^CycleTime<¯1↑MinCycleTime
[10]   MinCycleTime←MinCycleTime,CycleTime[i←1↑b/⍳Popsize
[11]   Incumbent←POP[i;]
[12]   ⎕CC 35 ◇ →0
[13] End:MinCycleTime←MinCycleTime,¯1↑MinCycleTime
        ∇
```

```
        ∇z←Evaluate_Pool POP;i;M
[1]   ⍝
[2]   ⍝         Evaluate the individuals in the pool
[3]   ⍝         (cycle time of assignment)
[4]   ⍝
[5]    i←0 ◇ M←(⍴POP)[1] ◇ z←M⍴0
[6]   Next:→End ⍝if M<i←i+1
[7]    z[i]←Ctime POP[i;]
[8]    →Next
[9]   End:
        ∇
```

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

```
      ∇z←Number_Violations pop;i
[1]   ⍝
[2]   ⍝        Compute number of violations of
[3]   ⍝        precedence restrictions
[4]   ⍝
[5]   ⍝        Global variable:  A = precedence matrix
[6]    →Single ⍙if 1=⍴⍴pop
[7]    z←(1↑⍴⍴pop)⍴0 ◇ i←0
[8]  Next:→End ⍙if(⍴pop)[1]<i←i+1
[9]    z[i]←+/,A>pop[i;]∘.≤pop[i;]
[10]   →Next
[11] Single:z←+/,A>pop∘.≤pop
[12] End:
      ∇
```

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

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

```
        ∇Q←Crossover Pop;M;N;PAIR;I
[1]   ⍝          For each pair of individuals
[2]   ⍝          in the mating pool, either mate
[3]   ⍝          them & put the 2 offspring into
[4]   ⍝          the pool, or else copy them into
[5]   ⍝          the new pool directly
[6]     N←(⍴Pop)[1] ◇ M←(⍴Pop)[2]
[7]     Q←(0,⁻1↑⍴Pop)⍴0
[8]   ⍝          Randomly shuffle the pool
[9]     Pop←Pop[N?N;]
[10] Next:→End ⍙if 2>(⍴Pop)[1]
[11] ⍝          Remove pair from the pool
[12]   PAIR←(2,M)↑Pop ◇ Pop←2 0↓Pop
[13] ⍝
[14]   →Copy ⍙if Pcross<(?1000)÷1000
[15] ⍝          Choose crossover point
[16]   Q←Q,[1]((2,I)↑PAIR),⌽[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]   ⍝
[2]   ⍝          Perform mutation of the individuals in pool
[3]   ⍝
[4]    Q←,Pop
[5]    I←(Pmutate≥(?(⍴Q)⍴10000)÷10000)/⍳⍴Q
[6]    Q[I]←?(⍴I)⍴Nstations
[7]    Q←(⍴Pop)⍴Q
        ∇
```