APPLICATION OF GENETIC ALGORITHM TO MIXED-MODEL ASSEMBLY LINE BALANCING

by

Jonathan D. Evans

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Industrial and Systems Engineering

APPROVED:

O. K. Eyada, Chairman

S. C. Sarin

R. T. Sumichrast

March 1996

Blacksburg, Virginia

Keywords: Genetic Algorithm, Mixed-Model Assembly Line Balancing

Application of Genetic Algorithm to Mixed-Model Assembly Line Balancing

by

Jonathan D. Evans

Dr. Osama K. Eyada, Chairman

Industrial and Systems Engineering

(ABSTRACT)

The demand for increased diversity, reduced cycle time, and reduced work-in-process has caused increased popularity of mixed-model assembly lines. These lines combine the productivity of an assembly line and the flexibility of a job shop. The mixed-model assembly line allows setup time between models to be zero. Large lines mixed-model assembly lines require a timely, near-optimal method. A well balanced line reduces worker idle time and simplifies the mixed-model assembly line sequencing problem.

Prior attempts to solve the balancing problem have been in-adequate. Heuristic techniques are too simple to find near-optimal solutions and yield only one solution. An exhaustive search requires too much processing time. Simulated Annealing works well, but yields only one solution per run and the solutions may vary because of the random nature of the Simulated Annealing process. Multiple runs are required to get more than one solution, each run requiring some amount of time which depends on problem size. If only one run is performed, the solution achieved may be far from optimal. In addition, Simulated Annealing requires different parameters depending on the size of the problem.

The Genetic Algorithm (GA) is a probabilistic heuristic search strategy. In most cases, it begins with a population of random solutions. Then the population is reproduced using crossover and mutation with the fittest solutions having a higher probability of being parents. The idea is survival of the fittest, poor or unfit solutions do not reproduce and are replaced by better or fitter solutions. The final generation should yield multiple near-optimal solutions.

The objective of this study is to investigate the Genetic Algorithm and its performance compared to Simulated Annealing for large mixed-model assembly lines. The results will show that the Genetic Algorithm will perform comparably to the Simulated Annealing. The Genetic Algorithm will be used to solve various mixed-model assembly line problems to discover the correct parameters to solve any mixed-model assembly line balancing problem.

ACKNOWLEDGEMENTS

I would like to thank my chairman, Dr. Eyada, for his guidance and support. Also, I would like to thank my other committee members, Drs. Sarin and Sumichrast, for their time and advice. Also, I would like to thank Andre Ramos and Patrick Evans for their help in programming my thesis code. Their advice helped me through several problems.

My parents need to be acknowledged for their help and support, both emotional and financial, which made it possible to focus on my school work. This thesis would not have been possible without them.

I would like to thank Alicia DeMartino for the love and support she offered as I completing my graduate degree. Finally, I would like to thank Bertram Chase, Patrick Evans, and Christoffe Moreau for their friendship while at Virginia Polytechnic Institute and State University.

Table of Contents

List of Figures

List of Tables

CHAPTER 1 : INTRODUCTION

1.1 ASSEMBLY LINES

A common configuration in manufacturing is the product layout, more commonly referred to as a product line or assembly line. This type of layout is designed for a specific product or a family of similar products. This configuration is also called a flow line because equipment is set up such that the product flows, while abiding by the precedence diagram [10], from the first machine or station to the second, and so on until the product is complete at the final machine or station. Precedences are requirements which state which work element must be done before a certain other work element while assembling a product. For example, in baking a cake, the ingredients must be mixed before baking. Therefore, mixing would have precedence over baking. Equipment is typically used in only one line. Therefore, product layouts require high volume to justify the cost of dedicating the equipment. "Product lines are unquestionably the most effective and efficient arrangement when justified by product mix and volume." [1]

The two main advantages of flow lines are low work-in-process (WIP) and high production rates. Merchant [2] points out that, in a typical metal machining non-product layout factory, the product spends 95% of production time waiting or moving and only

5% of production time processing. Flow lines reduce the 95% waiting and moving time, thus decreasing throughput time. Today, the various strategies by which companies manage their time offer "the most powerful new sources of competitive advantage." [3] Low WIP keeps a company from tying up its money and allows it to use that money to invest or pay bills. Other advantages of assembly lines include a reduction in both material handling and the use of skilled labor.

Assembly lines take three forms: single-model, batch, and mixed-model. A single-model assembly line produces only one product, and thus needs no sequencing. In a single-model assemble line, the time needed for a work element does not change because, again, only one part is produced. There has to be high demand for that one product to justify the single-model assembly line. Both batch and mixed-model assembly lines yield multiple products, but each suits a different situation. A batch assembly line produces each model for a long-enough period of time to assure excess inventory while other batches of other models are produced. The production of excess inventory demands that the company accurately predict demand. If they overestimate demand and are left with excess stock, they lose money. But the risk might be worthwhile when the cost and time of changeover is high. On a mixed-model assembly line, the products can be changed on a daily basis according to customer demands; this decreases response time and alleviates the need for excess inventory.

If the demand for a model requires constant high volume production, then the single-model assembly line is the most efficient way to manufacture it. More often,

demand for a single-model is not sufficient to justify the expense of establishing a single-model assembly line, but there is sufficient demand for a family of products. A mixed-model assembly line is designed for just this case: the production of two or more types of products without changeover time between models. In addition to the increased flexibility that mixed-model lines add, Samitt and Barry [4], both consulting managers, say the following about mixed-model assembly lines:

> The mixed model environment combines the strengths of both JIT and MRP II approaches and forms an operations flow that is better than either approach by itself. Companies can achieve all of the following benefits by implementing mixed model operation:
> - High levels of customer service;
> - Short cycle time;
> - Decreased introduction time for new products;
> - Increased product variety;
> - Efficient operations; and
> - Low inventory levels.

There are important issues to consider with mixed-model assembly lines. There must be no setup time when switching from one model to the next, unless the process includes excess time that can be used for setup. If there is not excess time and setup time cannot be reduced, then a batch assembly line may be called for although, as discussed above, batch assembly increases inventory and decreases flexibility. Ideally, the family of parts will have similar work elements and require no setup between models.

1.2 BALANCING MIXED-MODEL ASSEMBLY LINES

Balancing assembly lines involves assigning work elements or tasks to stations. Each model's work should be spread evenly over the stations in the assembly line in order

to minimize idle time and maximize output. Mixed-model assembly lines are a challenge to balance because the work element times vary from model to model. The work elements for each model may differ in duration and precedence, but are considered to require similar skill and tooling such that they should always be in the same station [6][7][9][12][18].

Each model has certain work elements that must be done before others. In other words, there are precedence relationships which must be adhered to for each model and the precedence diagram may be different from model to model. The precedence diagram is explained in more detail by Prenting and Battaglia [10]. An example of combining, the precedence diagram for each model can be found in Thomopoulos [7]. Occasionally the process of combining precedence diagrams becomes complicated if different models require work elements to be done in opposite order, but generally this problem can be solved by adding a dummy task to represent the different order. The single-model line does not have this problem, but it is one which is not difficult to overcome for the mixed-model assembly line.

Salveson [5] uses the example of getting dressed to explain that certain tasks must be done before others; such as socks before shoes. Let me extend this analogy to a mixed-model situation, Monday requires the task of putting on a tie and Saturday usually does not. Therefore, mixed-model product lines put on Monday's, Saturday's and possibly other day's clothes every day of production. Balancing this variability among models is one of the problems with mixed-model assembly lines.

The next problem is both more important and more difficult to solve. This is the model sequence [12][13][14] problem. If models are sent through the system arbitrarily, then the effort of balancing is wasted. Bad sequencing of the models can cause a tremendous amount of excess idle time and congestion. When designing a mixed-model assembly line, sequencing needs to be addressed. However, this problem will not be examined in this thesis; further investigation will be left to the reader.

1.3 PREVIOUS WORK

Early research attempted to balance one model at a time. Such a procedure may result in the assignment of similar tasks to different stations [8][16]. In other words, operators would be trained to perform similar work elements on different models. Also, tooling would have to be duplicated. Generally, this method of assigning one model at a time is not the most effective method [6][7][9][12][18]. The best balance occurs when all models are examined at once. Other early attempts assigned work elements to one station at a time [7]. This will work, but by doing this the optimal solution may be undiscovered because the first or other earlier station may be poorly balanced. In fact, when using an exhaustive search on each half of a large problem, the results are poor and slow as compared to using the Simulated Annealing on the whole problem. Edwards [23], to get solutions to compare her work to, performs the exhaustive search optimizing five stations at a time on a 25 to 30 station line. The results from optimizing five stations at a time are poor compared to her results obtained by Simulated Annealing.

In summary, the exhaustive search is too time consuming to solve large problems. The Simulated Annealing method works well, but only one solution per run can be obtained. Multiple runs have to be performed to ensure a good solution.

1.4 THE GENETIC ALGORITHM

The Genetic Algorithm (GA) was invented by John Holland. It is "a 'blind' probabilistic heuristic search strategy" [24]. The Genetic Algorithm originated from natural reproduction and nature's tendency to follow the survival of the fittest principle. The GA most commonly uses a series of zeros and ones to represent a solution. The entire series is called a chromosome with genes being the zero and one positions. The traditional GA begins with a population of random solutions. The initial population could be established using heuristics. Each individual solution is evaluated for fitness, and the chance of reproducing is directly related to the fitness. Then, through crossover and mutation, the population reproduces another population. Crossover is an operation in which two parents are used to produce two children. After the selection of two parents, a random point on the chromosome is selected. The first child gets the first half of the first parent and the second half of the second parent. The second child gets the first half of the second parent and the second half of the first parent. This type of crossover is called one-point crossover. Mutation is an operation in which each gene is tested to see if it will change. If the gene mutates, the value is reversed; a one would become a zero and a zero would become a one. The new population is evaluated and mixes with the original

population, discarding the least fit individuals. This process continues until the specified number of generations is performed. The Genetic Algorithm will yield multiple solutions in one run and is expected to provide solutions competitive with those provided by Simulated Annealing.

1.5 RESEARCH OBJECTIVE

The research objective is to determine if using the Genetic Algorithm is an effective method of solving the large mixed-model assembly line balancing problem. This requires experimenting with the Genetic Algorithm's parameters to provide near-optimal solutions quickly. The algorithm is used to solve problems with various characteristics to test it's ability to solve any type of problem. It is expected that there will be a set of parameters which can be used to solve the general mixed-model assembly line balancing problem.

The size of the population, the number of generations, and the probability of mutation will need to be experimented with to discover the best parameters for this application. The GA applied to large mixed-model assembly line balancing problems will be compared to the results of Edwards [23], who used Simulated Annealing. Simulated Annealing generates an initial solution and then manipulates that one solution. The process continues until the next manipulations do not yield a better solution. At this point, the solution is "frozen" and it is considered the best solution for that run. To ensure the quality of the solution, several runs are performed to see if better solutions can be

obtained. The GA has a population of solutions, therefore the need for multiple runs is reduced. This population may also give the user a chance to pick from a wide range of high quality solutions. The Genetic Algorithm may take longer than the Simulated Annealing, since the Genetic Algorithm generates many more solutions than the Simulted Annealing.

The Genetic Algorithm has been applied to the single-model deterministic balancing problem with promising results [28]. Some of the same constraints and representations used in that work can be applied to this study. However, the model formulation will be quite different. The Simulated Annealing method gives a good solution in minutes, but only gives one solution per run. Also, the Simulated Annealing has parameters which vary with problem characteristics. Edwards [23] does not suggest general parameters to solve any mixed-model assembly line problem.

1.6 THESIS OUTLINE

Chapter 2 reviews past work in mixed-model assembly line balancing and the Genetic Algorithm. Section 2.1 discusses mixed-model assembly line balancing. Section 2.2 reviews computerized mixed-model assembly line balancing. Section 2.3 discusses the Genetic Algorithm. Chapter 3 provides a description of the methodology. Section 3.1 shows the approach taken to the problem. It describes the objective function and constraints. In Section 3.2, the information needed to balance the line is discussed. Section 3.3 shows the precedence diagram and solution representation in computer

language. Section 3.4 discusses the methodology that will be used to find the parameters to solve the general mixed-model assembly line problem. Eight problems with varying parameters are discussed. These problems are used to determine the best parameters for the Genetic Algorithm. These parameters include population size, number of generations, and probability of mutation.

CHAPTER 2: LITERATURE REVIEW

This chapter contains the literature review. In 2.1, heuristic techniques which were used in the past are presented. Also, the objective function, which was used in the exhaustive search, Simulated Annealing and this research, is explained and a sample calculation from Thomopoulos [7] is shown. In 2.2, computerized techniques are introduced, including the Simulated Annealing which is used for comparison against this research. In 2.3, Genetic Algorithms are discussed further and some relevant books and articles are evaluated.

2.1 MIXED-MODEL ASSEMBLY LINE BALANCING

The first attempts to solve the mixed-model assembly line balancing problem balanced one model at a time and then combined them [8][16]. This leads to similar work elements being assigned to different stations which is not the best method to solve the problem. Assigning similar work elements to different stations causes extra training of the workers and extra tooling. This is one of the first methods of solving the mixed-model assembly line balancing problem. None of the following manual methods will separate the

work element into different stations; but they use heuristic techniques which do not necessarily lead to the optimal solution.

The next approach, developed by Thomopoulos [6], balances the line considering the amount of time available on a shift and the amount of time the model needs for production. In this method, the same amount of work is given to each station over the whole shift. The time available during the shift becomes the cycle time and then in Thomopoulos [6] the line is balanced using the heuristic for a single line in Kilbridge and Wester[15]. This method has merit, but it leaves each model with variable times at each station thus, making sequencing more difficult. The time for each model at each station should be even over the assembly line.

Thomopoulos developed another approach to solve the mixed-model assembly line balancing problem. In this approach, Thomopoulos accounts for work done during a shift, but the objective function minimizes the difference ([7] calls this delta) between the amount of time each model needs per station and how much time is actually used per station. In Thomopoulos [7], the available time per shift is 414 minutes which he translates into a range from 408 to 420 minutes. The range represents a minimum and maximum value for the amount of work which can be put in a station. Since the objective function by Thomopoulos [7] is what is used in the present research, it may be helpful to examine his example, in which, the objective function is to minimize Δ, such that

$$\Delta = \sum_{i=1}^{n} \sum_{j=1}^{J} \left| \frac{N_j}{n} \left(\sum_{k=1}^{K} t_{jk} \right) - N_j p_{ij} \right| \qquad (1)$$

where

J	= number of models;
N_j	= quantity to be produced of model j, j=1,2,... J;
K	= number of work elements or work tasks;
t_{jk}	= time to process element k, k=1,2, ... K, on model j, j=1,2, ... J;
n	= number of stations;
p_{ij}	= the time station i, i=1,2, ... n, is assigned work from model j, j=1,2, ... J.

This problem is shown and the answer he calculates is on the following three pages. Figure 2.1 shows the precedence diagram, while Tables 2.1, 2.2, and 2.3 show work element times, quantities needed per shift, and assignments of work elements to stations, respectively.

Using equation (1), Δ is calculated in the following way.

$$\Delta = \sum_{i=1}^{3} \sum_{j=1}^{3} \left| \frac{N_j}{3} \left(\sum_{k=1}^{K} t_{jk} \right) - N_j p_{ij} \right| =$$

$$\left| \frac{120}{3}(1.6+1.7+1.8) - 120(1.6) \right| + \left| \frac{60}{3}(2.0+1.8+1.9) - 60(2.0) \right| + \left| \frac{40}{3}(2.5+2.5+2.2) - 40(2.5) \right| +$$

$$\left| \frac{120}{3}(1.6+1.7+1.8) - 120(1.7) \right| + \left| \frac{60}{3}(2.0+1.8+1.9) - 60(1.8) \right| + \left| \frac{40}{3}(2.5+2.5+2.2) - 40(2.5) \right| +$$

$$\left| \frac{120}{3}(1.6+1.7+1.8) - 120(1.8) \right| + \left| \frac{60}{3}(2.0+1.8+1.9) - 60(1.9) \right| + \left| \frac{40}{3}(2.5+2.5+2.2) - 40(2.2) \right| =$$

= 12+6+4+0+6+4+12+0+8=

Δ=52

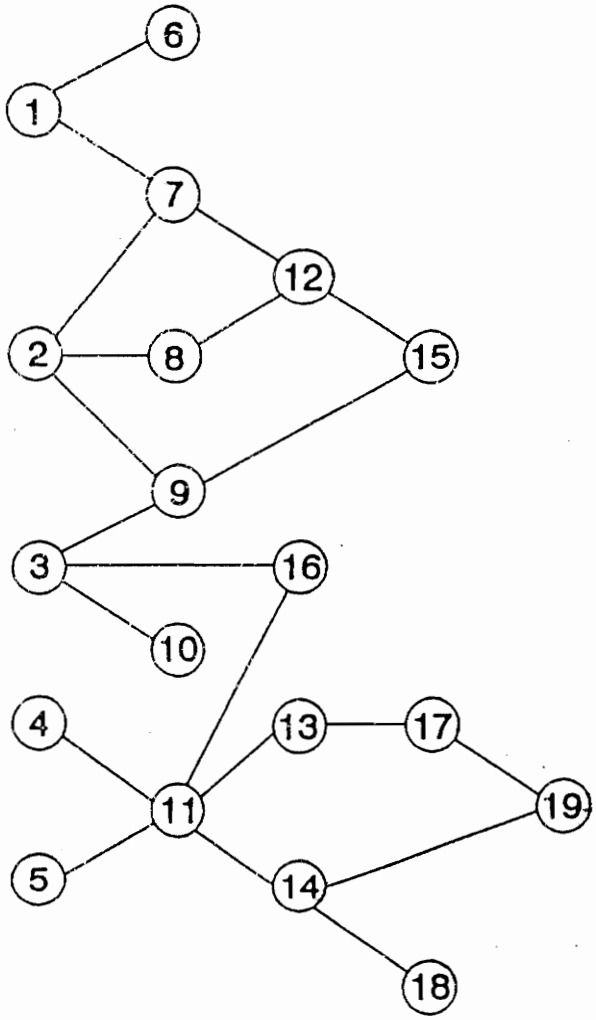


Figure 2.1: Precedence Diagram for the 19 Element Problem

Table 2.1: Work Element Data for Three Model and 19 Element Problem

work element	model 1	model 2	model 3	Total time/model
1	0.50	0.00	1.00	100
2	0.40	0.80	1.20	144
3	0.00	0.20	0.40	28
4	0.40	0.00	0.00	48
5	0.20	0.20	0.20	44
6	0.20	0.00	0.00	24
7	0.40	0.50	0.60	102
8	0.00	0.50	0.50	50
9	0.40	0.30	0.20	74
10	0.00	0.00	0.20	8
11	0.30	0.30	0.30	66
12	0.10	0.30	0.50	50
13	0.10	0.00	0.10	16
14	0.20	0.20	0.20	44
15	0.70	1.00	1.50	204
16	0.00	0.10	0.00	6
17	0.50	0.50	0.00	90
18	0.30	0.50	0.30	78
19	0.40	0.30	0.00	66
Total	5.10	5.70	7.20	1242

Table 2.2: Quantity/Shift needed per model for Three Model and 19 Element Problem

Model, J	Quantity/Shift, N_j
1	120
2	60
3	40

Table 2.3: Thomopoulos' [7] solution to Three Model and 19 Element Problem

Station	Elements, k	Total Elem. Times, t_k	Element Time/Model1 t_{1k}	Element Time/Model2 t_{2k}	Element Time/Model3 t_{3k}
1	2,4,5,8,11, 13,14	412	1.6	2.0	2.5
2	1,3,7,10,16,1 7,18	412	1.7	1.8	2.5
3	6,9,12,15,19	418	1.8	1.9	2.2

Johnson [17] solves the mixed-model assembly line balancing problem using stochastic work element times. Johnson [17] concludes that deterministic work element times will do if sufficient tolerance time is available or sufficient in-process inventory space is available. Next, Chakravarty and Shtub [18] demonstrate mixed-model line balancing while considering in-process inventory cost, setup cost, and station idle time cost. Inventory and setup costs should already be minimal, thus the problem is reduced to minimizing idle time, which is done by minimizing delta. In [14], Dar-El and Naidivi fix the number of stations and use parallel stations. In their example, the stations must be parallel because some of the task times are longer than the cycle time. Parallel stations require multiple tooling which, as mentioned earlier in section 2.1, is not desirable [6][7][9][12][18] and this research will not investigate parallel stations.

2.2 COMPUTERIZED MIXED-MODEL ASSEMBLY LINE BALANCING

COMSOAL [19] randomly generates solutions (usually 1000) and selects the one with the fewest stations. This program will generate quick solutions, but they are not guaranteed to be near-optimal. It would require a large number of random solutions to find a good solution, especially considering the fact that similar solutions might be generated. Also, the solution which has the fewest stations is not necessarily the best solution. NULISP [20] is similar to COMSOAL in that it generates random solutions, but NULISP gives some weights to work elements depending on characteristics like work

element time, number of work elements following, and time of work elements following. Even with weighted work elements, the optimal is not guaranteed.

Pieter Smith [21] developed a backtracking exhaustive search algorithm for the mixed-model, deterministic assembly line balancing problem with the objective of minimizing Thomopoulus' delta. The user can specify how full a station must be in order to be considered a feasible station, i.e. a minimum percent full. This value sets a lower limit on how full a station must be similar to the range Thomopoulos uses. Also, the software only considers solutions which are less than or equal to a predetermined maximum number of stations. The basic idea behind the program is to try all possible solution except those which do not meet two logical criteria, station length and minimum percent full. The first criteria sets a limit on the number of stations. A feasible solution might assign one task to each station, thus having as many stations as elements, but this would not be the best solution. The second criteria is assembly line length; it makes sense to make the line as short as possible so as to reduce both the number of operators and the chance for idle time. The best way to search for a solution would be to start with the minimum feasible number of stations which can be obtained from taking the next highest integer of the total time needed to complete the required products divided by the available shift time. The search begins with the minimum feasible number of stations and then increases by one until a solution can be found.

Once the number of stations is large enough to produce solutions, the next constraint will put a lower limit on the station time to ensure idle time is not high in one

particular station. It may be possible to reach an excellent balance with one task assigned to a station which leaves the station only 10% utilized, but this would not be practical. As stated earlier, Thomopoulos [7] has a cycle time of 414 minutes and a range of 408 to 420 minutes. This range gives a limit on how far from the cycle time the stations can be for this specific problem.

For small problems like Thomopoulos' 19 element, 3 model problem, Smith's [21] program performs well. It may take up to 20 hours, depending on the two parameters, (percent full and maximum number of stations); but the optimal solution is guaranteed, given the parameters are not too tight. The problem arises when big problems, similar to the one in [14], are attempted. Even if tight parameters are chosen for this large problem, the program will not finish in four weeks.

John Pantouvanos [22] generated a computerized exhaustive search program similar to Smith's and went beyond that to determine the sequence in which the models should be introduced to the line. His objective function is a cost model requiring data such as the labor rate, incompletion cost for each element, and zoning constraints. (Zoning constraints are restrictions on work elements being performed in the same station. For example, welding could not safely be performed near a painting operation.) Because this program is an exhaustive search, it does not find near-optimal solutions to large problems in a reasonable amount of time. Methods which take longer than a week to finish are considered to be not within a reasonable amount of time. Pantouvanos's method also requires a great deal of information about the line. For example, this method requires

the probability distribution function of the processing time for each work element of each model. All of these factors limit the program's practical applicability.

Edwards [23] uses Simulated Annealing to solve the mixed-model assembly line balancing problem using Thomopoulos' delta as an objective function. Edwards compared her results to the exhaustive search optimizing five stations at a time, and for the large problem in [14] she got considerably better results. Edwards' best delta was 3517.35 while the exhaustive search was 4449.47. Using the two methods on a Pentium, 90 Mhz computer, the exhaustive search's and Edwards' best deltas were 4286.91 and 3616.65 respectfully. The exhaustive search ran for eight days, while the Simulated Annealing ran in about two minutes. For reasons of accuracy as well as practicality, then, the results from the present research will be compared to the results of Edwards' [23] using the Simulated Annealing.

2.3 THE GENETIC ALGORITHM

John Holland is the founder of the Genetic Algorithm. In [25], Holland presents the Genetic Algorithm without practical applications. Two principle characteristics of any GA are schema and genetic operators. Schema is the way a problem is represented; this is usually done in binary. Care must be taken to ensure that the schema, which is dependent on the type of application, depicts the solution space accurately. The schema for this research is discussed in Chapter 3. Holland's three methods of reproduction are crossover, inversion, and mutation. To perform crossover, two parents are randomly selected with

the probability of selection directly related to a measure of fitness. Next, a random point (x) is selected. Values on the parent strings between x+1 and the end of the string (m) are interchanged as illustrated below.

Parent 1 $\mathbf{a_1}$ $\mathbf{a_2}$ ... $\mathbf{a_x}$ $\mathbf{a_{x+1}}$ ... $\mathbf{a_m}$

Parent 2 b_1 b_2 ... b_x b_{x+1} ... b_m

Child 1 $\mathbf{a_1}$ $\mathbf{a_2}$ ... $\mathbf{a_x}$ b_{x+1} ... b_m

Child 2 b_1 b_2 ... b_x $\mathbf{a_{x+1}}$ ... $\mathbf{a_m}$

To perform inversion, one parent and two random points (x1,x2) are selected where x1 is the smaller of the two numbers. The section of the chromosome between the two points is reversed.

Parent a_1 a_2 ... a_{x1} $\mathbf{a_{x1+1}}$ $\mathbf{a_{x1+2}}$ ... $\mathbf{a_{x2-2}}$ $\mathbf{a_{x2-1}}$ a_{x2} a_{x2+1} ... a_m

Child a_1 a_2 ... a_{x1} $\mathbf{a_{x2-1}}$ $\mathbf{a_{x2-2}}$ ... $\mathbf{a_{x1+2}}$ $\mathbf{a_{x1+1}}$ a_{x2} a_{x2+1} ... a_m

Mutation generally has a small probability of occurring. Each position has a chance, independent of other positions, to be inverted. In binary code, zero would become one and one would become zero.

In [26], Goldberg presents practical GA research tools such as computer techniques, mathematical tools, and results. Goldberg demonstrates that the Genetic Algorithm is a realizable, near-optimal strategy for searching alternatives.

Davis' [27] book, like Goldberg [26], describes the GA by giving examples and explanations which are focused toward the person who is interested in using the GA to solve problems. Davis points out that the inversion process described by Holland [25] has

not been found useful in practice. In this research, precedence would most likely be violated if this were performed. For these two reasons, inversion is not used.

In the next work [28], Leu, Matheson, and Rees use the Genetic Algorithm to solve the single-model assembly line balancing problem. The Genetic Algorithm shuffles and mutates the genes, for the assembly line problem this shuffling woud violate the precedence diagram. The work elements cannot be arranged in just any order. If a work element is already assigned, it cannot be assigned or moved to another station. There may not be room at the new station or the precedence diagram may be violated. The modified two point crossover and scramble mutation used in [28] solve this problem. The modified two point crossover picks two random points on the chromosome; the first section and last section stay the same and the middle work elements are replaced in the order occurring in the other parent. If the parents are in a feasible order (i.e. do not violate the precedence diagram), the modified two point crossover described above produces children which do not violate the precedence diagram. The following is an example of the modified two point crossover.

Parent 1	3	2	|	5	1	4	|	7	6	8
Parent 2	**1**	3	|	7	**4**	2	|	6	8	**5**
Child 1	3	2	|	**1**	**4**	**5**	|	7	6	8
Child 2	1	3	|	2	4	7	|	6	8	5

Scramble mutation is performed by selecting a parent and a random point on the chromosome. Then, the first section stays the same and the rest of the chromosome is

generated by selecting work elements at random among those feasible according to the precedence diagram. The authors [28] explain that the modified two point crossover is a neighborhood search and the scramble mutation is meant to keep the algorithm from getting caught in a local extreme. The process takes two steps. First, the tasks are put in feasible order, and then the tasks are assigned to stations.

The next article describes the search for optimal control parameters for the Genetic Algorithm [29]. Grefenstette [29] experiments with different values of six parameters that he considers necessary for the Genetic Algorithm. The first condition is population size, which is the number of individuals in a population. The second parameter is crossover rate, which controls the crossover frequency. The third parameter is mutation rate, which controls the mutation frequency. A high mutation rate is close to a random search process. It should be noted that the mutation described in Grefenstette [29] is different than scramble mutation in Leu, Matheson, and Rees [28], thus the mutation rate will be much different than Grefenstette's [29] mutation rate. The fourth parameter is generation gap. This parameter determines how much of the old population is retained in the next generation. A value of one would mean the entire population was replaced by the new generation. The fifth parameter is a scaling window. According to Grefenstette [29], after several generations the GA tends to focus on a range of values. This makes the selection of parents less dependent on fitness, since most of the population is close to the same fitness value. If the results are scaled, then this does not occur. The sixth parameter is selection strategy. Two selection strategies are tested, pure selection and elitist

strategy. The two selection strategies have nothing to do with one another; one determines how parents are picked, and the other determines how the old generation is combined with the new generation. Table 2.4 summarizes the six parameters, the range experimented with, the increments, and the best values where applicable. Pure selection or Roulette Wheel Selection is choosing parents with the probability of selection directly related to a measure of fitness. The elitist strategy picks the fittest individuals and ensures they are carried to the next generation.

Below is a typical GA procedure and the flow chart, Figure 2.2.

1. Generate an initial population.
2. Evaluate the individuals in the population.
3. Reproduce by crossover and mutation.
4. Replace less fit parents with better fit children.
5. If time is up, stop; else go to 2.

Table 2.4: Summary of Genetic Algorithm Parameter Settings [29]

Characteristic	Range or Values	Increment	Best Range
Population Size(N)	10 to 160	10	30<N<100
Crossover Rate(C)	0.25 to 1.00	0.05	
Mutation Rate(M)	0.0 to 1.0	exponential	0<M<0.5
Generation Gap(G)	0.30 to 1.00	0.10	large
Scaling Window	0, 1, and 7	N/A	small
Selection Strategy	pure and elitist	N/A	elitist

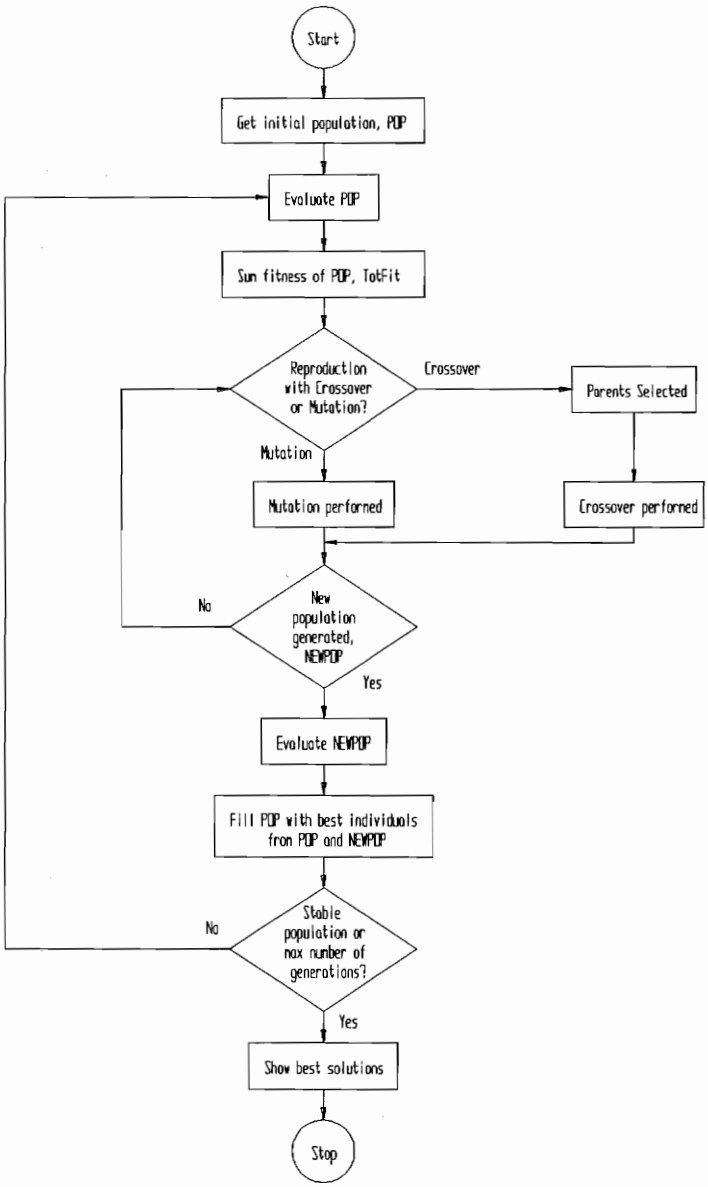


Figure 2.2: Flowchart of the Genetic Algorithm.

CHAPTER 3: RESEARCH METHODOLOGY

This chapter describes the methodology used to achieve the research objectives. Section 3.1 shows the approach taken to the problem. It describes the objective function and constraints. Section 3.2 discusses the information needed to balance the line. Section 3.3 shows the precedence diagram and solution representation in computer language. Section 3.4 explains the methodology used to find the parameters to solve the general mixed-model assembly line problem. Eight problems with varying parameters are discussed. These problems are used to determine the best parameters for the Genetic Algorithm. These parameters include population size, number of generations, and probability of mutation.

3.1 APPROACH

In order to use the Genetic Algorithm to solve the mixed-model assembly line balancing problem, there must be a means of evaluating the fitness of the population. In other words, there must be an objective function. Also, there are constraints which have to be met when balancing a line.

As described earlier, the objective function (Δ) originates from Thomopoulos [7]. This objective function rewards solutions that have each model's work balanced evenly over all stations. The objective function does not take into account the number of stations. For example, if a model requires 30 minutes of work, then the objective function considers two stations working for 15 minutes equivalent to six stations working for five minutes.

3.2 DATA NEEDED

The precedence diagram and the following data are needed:

t_{jk} = time required for work element k for model j;

TTc = shift duration;

N_j = quantity of model j to be assembled in TTc

3.3 REPRESENTATION IN C

The precedence diagram and the solutions are represented in binary. The exhaustive search and Simulated Annealing were both done this way, which is very convenient for application to the GA. As an added benefit, the computer can perform binary manipulation quickly. First, the precedence diagram is described. The precedence diagrams are accounted for using the 32 binary positions of a long integer. For example, if a certain work element requires work element 1 and 4 to be done beforehand, then the binary number would look like this: 00000000000000000000000000001001 and it would

be in the input file as 9. If the problem requires more than 32 elements, then a second number would be beside the first. The input line, 9 9 would mean that this element required element 1, 4, 33, and 36 to be done before it. An example of the input file for Problem 1 is provided in Appendix A.

The solution to the mixed-model assembly line balancing problem will be in binary, with a typical solution for stations 1, 2, and 3 looking like the following:

Variable	Value	Binary	Interpretation
solution[1]	9	01001	Station 1 Element 1 and 4
solution[2]	20	10010	Station 2 Element 2 and 5
solution[3]	4	00100	Station 3 Element 3

etc.

The values and the binary numbers will be larger, but this is the basic principle of how it will be represented.

3.4 METHODOLOGY

Eight problems are examined in this research to determine the parameters of the GA. Problem 1 originates from Thomopoulos [7]. Problems 2, 3, and 4 are modifications of Problem 1 and originate from Edwards [23]. Problems 2 has a different precedence diagram than 1, which will change the F-ratio. Problem 3 and 4 are variations of 1 and 2 with a shorter cycle time, which will change the WEST ratio. The F-ratio is a measure of the degree to which the problem is constrained by its precedence diagram. The less

constrained, the higher the F-ratio will be. Therefore, a problem which has a higher F-ratio will have more feasible solutions because it is less constrained. The WEST ratio, or the Work Elements to STations ratio, is simply the number of work elements divided by the number of stations. Typically a better balance is obtained when the WEST ratio is high, because the elements can shuffle around more and find stations which minimize Δ. The higher the WEST ratio, the fewer unique solutions. Further descriptions of the F-ratio and WEST ratio can be found in Dar-El [11]. In order to get as near-optimal a solution as possible, considerable experimentation with the exhaustive search method was performed to determine whether the F-ratio, WEST ratio, the Rank Positional Weight minimum percent full, the number of models, or the number of elements would give some clue about choosing a good lower limit, but no significant findings were discovered. If a lower limit is chosen, care must be taken; it is possible to eliminate near-optimal or optimal solutions just because they may include a station that does not meet the minimum time constraint. In fact, when performing experiments on Problem 8 with a high percent full, determined by the Simulated Annealing method, the exhaustive search did not yield a single solution after four days.

Problem 5 originates from Dar-El and Naidivi [14]. Some modifications were needed, as the article assumed parallel processing of some work elements because they are larger than the cycle time. Problems 6, 7, and 8 are modifications of Problem 5 and originate from Edwards [23]. Problem 6 has a different precedence diagram than 5. Problems 7 and 8 are similar to 5 and 6, but have longer cycle times.

The idea behind using the different type of problems is to find the parameters which will solve any mixed-model assembly line balancing problem. The eight problems represent three characteristics. The first is size. The first four problems are 19 element, three model problems. The last four problems are 50 element, eight model problems. The second characteristic is the amount of constraint due to the precedence diagram, the F-ratio. This characteristic will have the biggest influence on the processing time [23]. Problems 1 and 5 are taken from the literature as mentioned earlier. The F-ratio for the small and large problems are 0.74 and 0.41, respectively. Problems 2 and 6 have a different precedence diagram, and the F-ratio for problems 2 and 6 are 0.31 and 0.67, respectively. The next characteristic is the WEST ratio. Problems 1 and 5 have a WEST ratio of 6.3 and 2.1, respectively. The two variations, Problems 3 and 7, have a WEST ratio of 3.1 and 5.55, respectively. The difference is caused by a change in the cycle time. The small and large problems' cycle times vary from 414 to 205 and from 500 to 1308 minutes, respectively. See Table 3.3 for a summary of the eight problems.

The precedence diagrams, work element times, and quantity needed per shift are on the following pages. Problem 1 has already been shown in Section 2.1 in Figure 2.1, Table 2.1, and Table 2.2. The work element times and quantity needed per shift for Problems 1, 2, 3, and 4 are the same. Problems 1 and 3 have the same precedence diagrams. Please refer back to Section 2.1 for that information. The work element times and quantity needed per shift for Problems 5, 6, 7, and 8 are the same. Problems 5 and 7

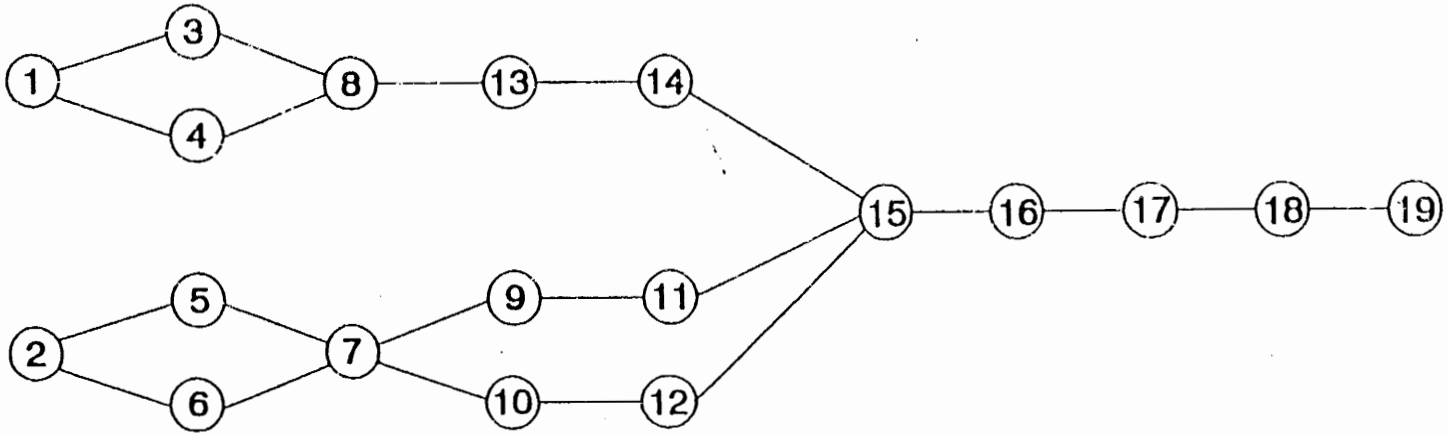


Figure 3.1: Precedence diagram for Problem 2 and Problem 4

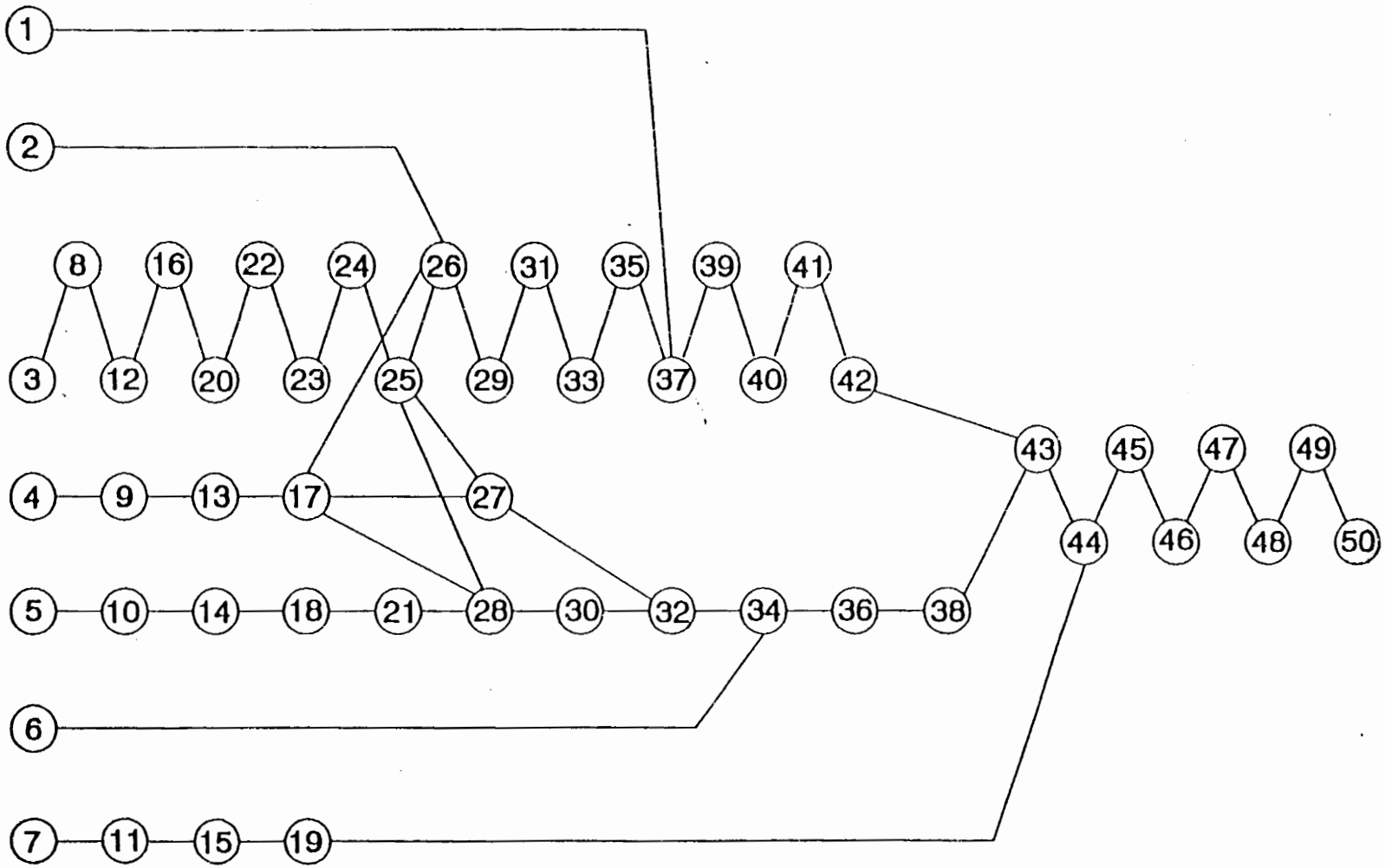


Figure 3.2: Precedence diagram for Problem 5 and Problem 7

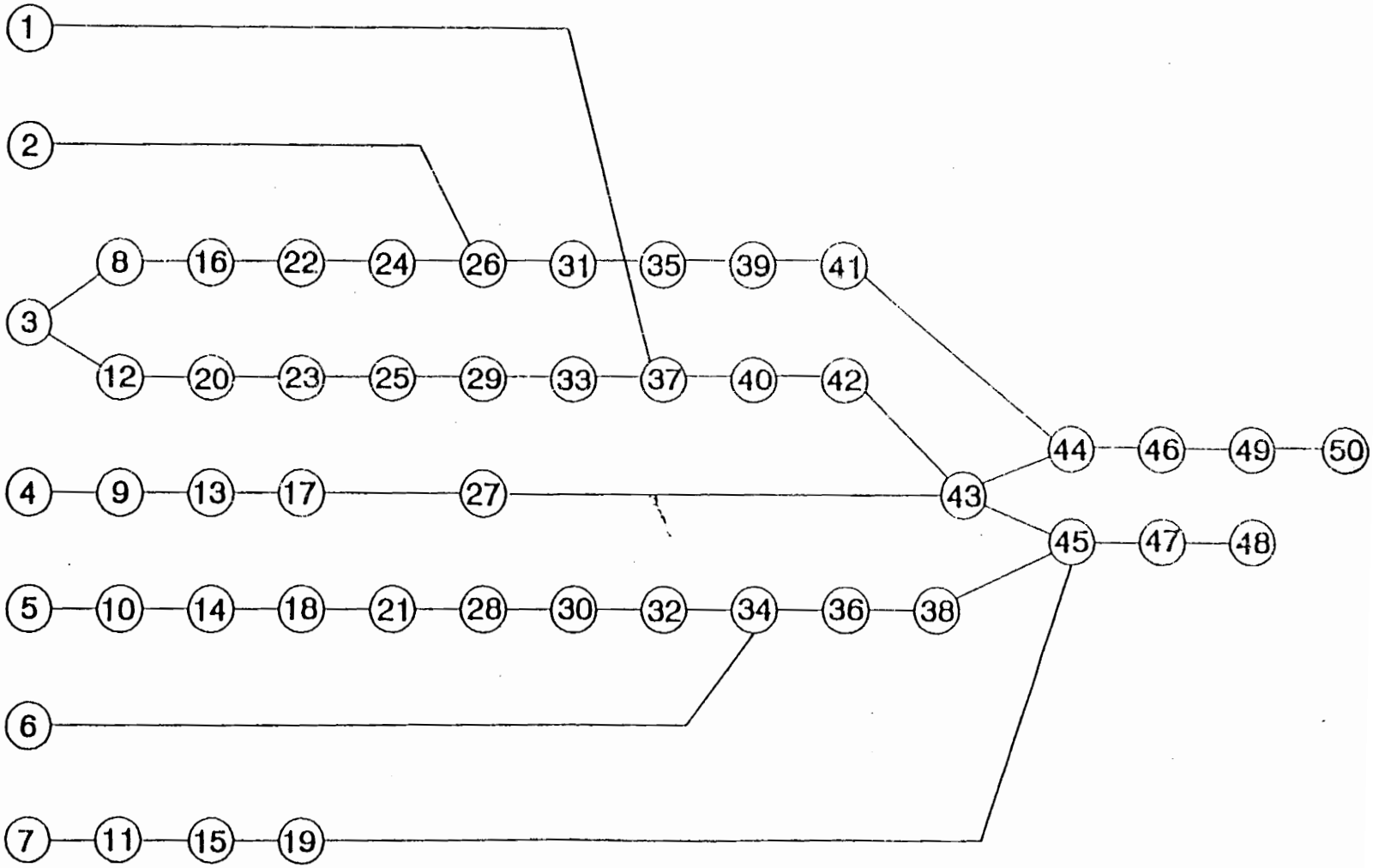


Figure 3.3: Precedence diagram for Problem 6 and Problem 8

Table 3.1: Work Element Times for Problem 5, 6, 7, and 8.

work el.	model 1	model 2	model 3	model 4	model 5	model 6	model 7	model 8	time/shift
1	12.61	12.37	10.43	14.55	15.16	12.73	8.97	7.28	207
2	35.00	35.00	10.00	8.00	10.00	44.00	42.00	38.00	447
3	27.13	26.74	13.37	10.22	13.37	23.20	20.45	21.63	324
4	13.73	13.56	8.64	6.34	8.17	16.24	14.61	12.51	193
5	38.00	33.00	52.00	33.00	52.00	0.00	0.00	0.00	458
6	6.00	6.00	6.00	6.00	6.00	0.00	0.00	0.00	72
7	14.00	14.00	14.00	14.00	14.00	0.00	0.00	0.00	168
8	12.59	12.45	6.87	7.01	7.16	13.31	9.16	10.73	167
9	12.89	12.75	5.95	4.25	5.95	14.59	12.61	11.90	165
10	20.00	18.00	28.00	18.00	28.00	0.00	0.00	0.00	248
11	14.00	14.00	28.00	14.00	28.00	0.00	0.00	0.00	210
12	2.09	2.05	1.25	1.02	1.33	4.51	1.77	3.23	34
13	24.15	23.03	26.83	22.81	26.83	0.00	0.00	0.00	288
14	6.00	5.00	8.00	5.00	8.00	0.00	0.00	0.00	70
15	14.00	14.00	14.00	14.00	14.00	0.00	0.00	0.00	168
16	38.00	37.00	13.00	9.00	15.00	35.00	23.00	45.00	424
17	31.00	30.00	37.00	30.00	37.00	0.00	0.00	0.00	382
18	4.04	3.54	5.55	3.54	5.55	0.00	0.00	0.00	49
19	14.00	14.00	14.00	14.00	14.00	0.00	0.00	0.00	168
20	7.77	7.40	5.69	3.03	4.93	14.98	4.93	11.38	117
21	9.00	8.00	12.00	8.00	12.00	0.00	0.00	0.00	109
22	17.69	17.50	9.04	6.98	8.94	16.81	13.66	14.45	218
23	11.00	11.00	13.00	11.00	13.00	0.00	0.00	0.00	138
24	28.58	27.76	22.86	18.78	22.86	13.88	11.02	11.84	345
25	34.00	34.00	16.00	14.00	18.00	32.00	28.00	29.00	425
26	20.19	20.19	5.92	4.53	5.92	30.63	22.63	23.67	267
27	6.00	6.00	6.00	6.00	6.00	0.00	0.00	0.00	72
28	5.77	5.04	7.98	5.04	7.98	0.00	0.00	0.00	70
29	16.07	16.07	6.37	4.31	5.70	24.20	21.35	18.86	226
30	32.21	28.01	44.25	28.01	44.25	0.00	0.00	0.00	389
31	9.27	9.27	4.20	2.97	3.71	14.83	14.21	12.60	141
32	37.00	34.00	46.00	34.00	46.00	0.00	0.00	0.00	447
33	21.00	21.00	7.00	6.00	7.00	34.00	25.00	29.00	297
34	23.12	20.94	29.45	20.94	29.45	0.00	0.00	0.00	279
35	10.00	10.00	4.00	8.00	4.00	24.00	20.00	18.00	200
36	24.00	21.00	34.00	21.00	34.00	0.00	0.00	0.00	294
37	16.62	16.62	7.70	7.30	7.70	55.14	33.25	28.79	341
38	12.00	12.00	12.00	12.00	12.00	0.00	0.00	0.00	144
39	13.00	13.00	9.00	7.00	8.00	42.00	34.00	36.00	307
40	21.32	21.32	10.93	9.90	11.46	41.57	27.95	12.45	331
41	10.00	10.00	4.00	4.00	5.00	16.00	17.00	0.00	145
42	24.00	24.00	13.00	10.00	11.00	49.00	44.00	16.00	399
43	24.00	24.00	7.00	7.00	7.00	47.00	33.00	39.00	368
44	9.32	8.76	11.21	8.76	11.21	0.00	0.00	0.00	113
45	25.00	25.00	35.00	25.00	35.00	0.00	0.00	0.00	330
46	34.00	30.00	44.00	30.00	44.00	0.00	0.00	0.00	406
47	17.09	16.83	16.29	12.38	17.00	6.63	16.92	10.37	241
48	0.95	0.95	1.13	1.13	0.95	1.19	1.78	1.58	20
49	16.61	16.04	12.79	10.88	12.79	17.57	12.03	14.13	236
50	4.59	4.59	4.59	4.59	4.59	4.59	4.59	4.59	78
Total	880.4	846.8	765.3	587.3	771.0	649.6	517.9	482.0	11735

Table 3.2: Quantity/Shift needed per model for Problems 5, 6, 7, and 8.

Model	Quantity/Shift
1	1
2	4
3	2
4	4
5	1
6	2
7	2
8	1

Table 3.3: Summary Table of the Eight Problems

Problem	1	2	3	4	5	6	7	8
Elements	19	19	19	19	50	50	50	50
Models	3	3	3	3	8	8	8	8
F-ratio	0.74	0.31	0.74	0.31	0.41	0.67	0.41	0.67
WEST	6.3	6.3	3.1	3.1	2.1	2.1	5.55	5.55
Cycle Time	414	414	205	205	500	500	1308	1308

have the same precedence diagrams. Problems 6 and 8 have the same precedence diagrams.

On page 20 and 21, the modified two point crossover and scramble mutation are explained. The following is an example of how this will be applied. The precedence diagram is on the following page, Figure 3.4.

P1	2	3	5	|	1	6	8	7	10	9	| 11	4
P2	3	**1**	4	|	2	5	**7**	**8**	**9**	**6**	| **10**	11
C1	2	3	5	|	**1**	**7**	**8**	**9**	**6**	**10**	| 11	4
C2	3	1	4	|	2	5	6	8	7	9	| 10	11

The following modified two point crossover features the same parents with different random positions and different children.

P1	2	3	5	1	|	6	8	7	10	|	9	11	4
P2	3	1	4	2	|	5	**7**	**8**	9	|	**6**	**10**	11
C1	2	3	5	1	|	**7**	**8**	**6**	**10**	|	9	11	4
C2	3	1	4	2	|	5	8	7	9	|	6	10	11

Notice that all precedences are met. Both children are feasible and can now be assigned to stations. If scramble mutation is performed it may look like the following:

P1	2	3	5	1	6	|	8	7	10	9	11	4
C1	2	3	5	1	6	|	8	7	9	10	4	11

The work elements assigned after the random point in the chromosome are selected randomly among the feasible work elements that have not been assigned.

From the parameters discussed in Grefenstette [29] and Leu, Matheson, and Rees [28], the following describes the parameters that are tested in this study. The population size will vary from 50 to 110 in increments of 20. The population size will remain constant from generation to generation, unlike in Leu, Matheson, and Rees [28] who allow the population to increase as the search proceeds. Because of the requirements of the assembly line problem, the crossover rate and the mutation rate will equal 100%. Leu, Matheson, and Rees [28] use a mutation rate of 2%, 10%, and 18% and conclude that these values may not be high enough; thus this research tests 10% to 40% mutation rates with 10% increments. Therefore, the crossover rate will have to vary from 90% to 60% with 10% increments. The stopping criteria, similar to Leu, Matheson, and Rees [28], is the number of generations or, in other words, the number of iterations. The number of generations is equal to the number of iterations divided by the population size. Five to 20 generations in increments of five are tested. Parents are selected by Roulette Wheel Selection, or pure selection, as Grefenstette [29] refers to it. The elitist strategy is not used directly, because the next generation consists of the best individuals from the old and new population. The best from the old population will be put in the next generation; thus

to use the elitist strategy would be to include individuals who will be in the next generation anyway. Each new generation is checked to ensure that are no twins. If there are identical solutions, one of the twins will be replaced by a randomly generated individual.

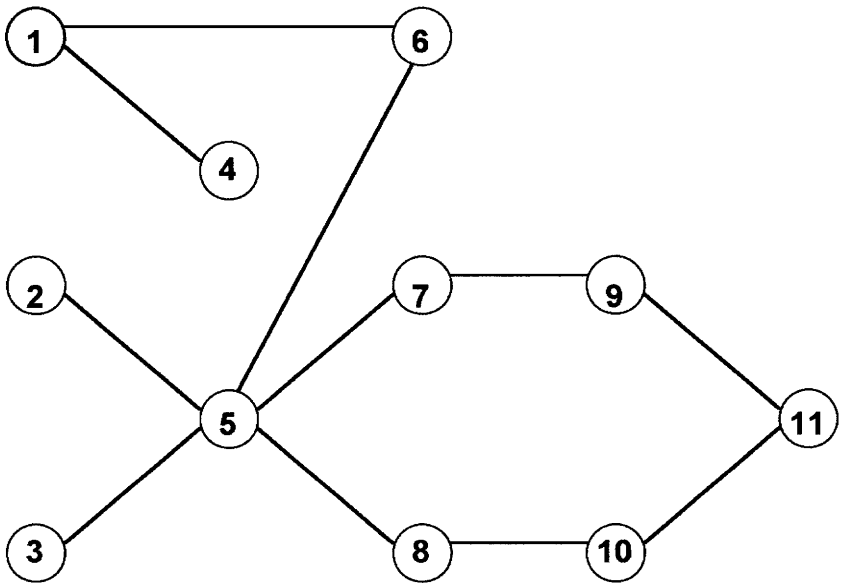


Figure 3.4: Precedence diagram for Example Problem

CHAPTER 4: EXPERIMENTAL RESULTS

This chapter shows the results of the experimentation with the software. First, the results of Smith's[21] exhaustive search for the four small problems will be presented. Then, the results of the SA search for the four large problems follows. Third, the GA search will be applied to all eight problems.

4.1 EXHAUSTIVE SEARCH AND SIMULATED ANNEALING RESULTS

4.1.1 Small Problems

The results of the exhaustive search are in Table 4.1. The problems were run on 486 33Mhz personal computers. All the problems were run with the station required to be 5% full, except for Problem 3 which required it to be 50% full. This problem requires a very long processing time if not limited in this way. The other three problems were run with a lower limit of five percent. The exhaustive search does not require that tasks be assigned to a station that may have room for it, unlike the GA search, which will be discussed later. The exhaustive search tries every combination, as one would expect from an exhaustive search procedure.

Table 4.1: Exhaustive Search Data

	Low WEST ratio	High WEST ratio
Low F-ratio	**Problem 4** Delta: 254.0 Time: 75 min.	**Problem 2** Delta: 164.0 Time: 1 min.
High F-ratio	**Problem 3** Delta: 161.7* Time: 1260 min.*	**Problem 1** Delta: 60.0 Time: 109 min.

* Run with lower limit at 50% of cycle time.

Table 4.2: Simulated Annealing Data

	Low WEST ratio	High WEST ratio
Low F-ratio	**Problem 5** Delta: 3531.46 Time: 90 min.	**Problem 7** Delta: 1189.75 Time: 41 min.
High F-ratio	**Problem 6** Delta: 3294.27 Time: 68 min.	**Problem 8** Delta: 883.62 Time: 22 min.

4.1.2 Large Problems

The results of the Simulated Annealing search are in Table 4.2. The problems were run on 486 33Mhz personal computers. The Simulated Annealing, as described briefly in Section 2.2, looks at one solution at a time. After generating an initial solution, the software transfers tasks from one station to another and swaps one task with another until a stopping criteria is met. This method, like the exhaustive search, does not fill up the stations if another task could fit. Instead, it might start a new station without requiring that station to be full.

4.2 GENETIC ALGORITHM RESULTS

4.2.1 Small Problems

The best solution obtained by the GA search was higher or not as good as the exhaustive because of the GA's representation. The GA search keeps track of solutions by feasible orders which are determined by the precedence diagram. The order determines how the tasks are assigned to stations. The tasks are assigned to a station until the next task does not fit, thus filling each station as much as possible. At this point, a new station is created. For this reason, the GA search does not represent the entire solution space. But it does appear to represent most of it. The best solutions obtained with the GA search are close to the optimal obtained by the exhaustive search. For Problem 3, which took the longest to perform the exhaustive search, the best solution the GA search obtained matched the optimal solution. Problem 3 took 1260 minutes with a restricted search. The stations in Problem 3 were required to be at least 50% full to be considered a feasible

station. The times for Problems 1, 2, and 4 are 109, 0.72, and 75 minutes, respectively. For Problems 1, 2, and 4, the best solutions obtained by the GA search were 127%, 113%, and 124% of the optimal solutions, respectively.

It was not predicted that filling a station as much as possible would exclude good, especially optimal, sections of the solution space. If a station has room for all the possible tasks which could follow it, then there seems to be no reason to start a new station. Yet this is the case on three of the four small problems. The results of Problems 1, 2, 3, and 4 are in Tables 4.3, 4.4, 4.5, and 4.6, respectively and a summary of the results in Table 4.7.

4.2.2 Large Problems

The small problems yielded knowledge about which parameters to use on the big problems. Originally, the number of generations was going to be fixed, but each problem required a different number of generations to get the best solution enough times to make it useful in predicting parameters for the larger problems. Problem 2 was quite useless. Here, higher generations, populations, and mutation rates gave better results; but the best solution was too easy to obtain to be useful. Using four generations and a population of 110 on all mutation rates, the longest run time was 93.19 seconds. In Problem 1, which required longer run times to obtain the best solution, it appears that higher generations yield better solutions. Also, the higher populations and mutation rates look slightly better. Problem 3 seems to show a mutation rate of 30% may be optimal. The results from a 10% mutation rate and a population of 110 were surprisingly poor. Problem 4 is

Table 4.3: Delta Data for Problem 1

Mut 10	Gen 250	Gen 500	Gen 1000	Gen 1500
Pop 50	102	102	76	76
	102	102	102	76
	88	97	102	102
Pop 70	76	88	76	102
	102	76	88	102
	97	76	76	76
Pop 90	102	102	76	76
	108	88	76	76
	102	102	76	76
Pop 110	76	76	78	76
	97	102	76	76
	102	76	76	76

Mut 20	Gen 250	Gen 500	Gen 1000	Gen 1500
Pop 50	102	94	78	76
	102	94	88	102
	76	94	76	76
Pop 70	76	76	88	94
	90	76	76	76
	88	88	76	76
Pop 90	76	102	76	76
	102	76	76	76
	76	102	76	76
Pop 110	78	76	76	76
	97	97	76	76
	76	94	76	76

Mut 30	Gen 250	Gen 500	Gen 1000	Gen 1500
Pop 50	76	88	76	102
	102	76	102	76
	102	102	76	76
Pop 70	78	76	76	76
	102	76	76	76
	102	76	76	76
Pop 90	90	76	76	76
	76	102	76	76
	94	102	76	76
Pop 110	102	76	76	76
	88	76	102	76
	97	97	76	76

Mut 40	Gen 250	Gen 500	Gen 1000	Gen 1500
Pop 50	105	76	76	76
	102	102	76	76
	102	76	76	76
Pop 70	88	76	102	76
	88	102	76	76
	102	102	76	76
Pop 90	76	76	97	76
	102	94	76	76
	94	76	76	76
Pop 110	76	76	76	76
	102	76	76	102
	97	76	76	76

Table 4.4: Delta Data for Problem 2

Mut 10	Gen 1	Gen 2	Gen 3	Gen 4
	234	186	186	186
Pop 50	186	186	186	186
	186	186	186	186
	234	186	186	186
Pop 70	186	234	186	186
	186	186	186	186
	186	186	186	186
Pop 90	186	186	186	186
	186	186	186	186
	186	186	186	186
Pop 110	186	186	186	186
	186	186	186	186

Mut 20	Gen 1	Gen 2	Gen 3	Gen 4
	186	186	186	186
Pop 50	234	186	186	186
	186	186	186	234
	186	186	186	186
Pop 70	186	186	186	186
	186	186	186	186
	186	186	186	186
Pop 90	186	186	186	186
	186	186	186	186
	186	186	186	186
Pop 110	186	186	186	186
	186	186	186	186

Mut 30	Gen 1	Gen 2	Gen 3	Gen 4
	186	186	186	186
Pop 50	234	186	186	186
	186	234	186	186
	186	186	186	186
Pop 70	186	186	186	186
	186	186	186	186
	186	186	186	186
Pop 90	186	186	186	186
	186	186	186	186
	186	186	186	186
Pop 110	186	186	186	186
	186	186	186	186

Mut 40	Gen 1	Gen 2	Gen 3	Gen 4
	234	186	186	186
Pop 50	186	186	186	186
	186	186	186	186
	186	186	186	186
Pop 70	186	186	186	186
	186	186	186	186
	186	186	186	186
Pop 90	186	186	186	186
	186	186	186	186
	186	186	186	186
Pop 110	186	186	186	186
	186	186	186	186

Table 4.5: Delta Data for Problem 3

Mut 10	Gen 20	Gen 30	Gen 40	Gen 50
	161.7	161.7	167.4	161.7
Pop 50	185.7	167.4	161.7	161.7
	161.7	172	172	172
	186.3	172	161.7	161.7
Pop 70	186.3	167.4	161.7	161.7
	176.6	161.7	167.4	161.7
	172	161.7	161.7	161.7
Pop 90	161.7	161.7	161.7	161.7
	183.4	167.4	178.3	161.7
	167.4	167.4	167.4	161.7
Pop 110	167.4	167.4	167.4	167.4
	167.4	172	167.4	167.4

Mut 20	Gen 20	Gen 30	Gen 40	Gen 50
	167.4	177.7	161.7	161.7
Pop 50	181.1	183.4	177.7	175.4
	167.4	167.4	167.4	167.4
	181.1	167.4	167.4	161.7
Pop 70	173.1	167.4	161.7	161.7
	167.4	167.4	161.7	167.4
	178.3	161.7	161.7	161.7
Pop 90	172	167.4	167.4	161.7
	173.1	167.4	167.4	161.7
	178.3	161.7	161.7	161.7
Pop 110	177.7	161.7	167.4	161.7
	161.7	161.7	161.7	161.7

Mut 30	Gen 20	Gen 30	Gen 40	Gen 50
	173.1	167.4	186.3	161.7
Pop 50	184	167.4	161.7	161.7
	167.4	172	161.7	167.4
	189.7	172	161.7	161.7
Pop 70	167.4	161.7	161.7	161.7
	172	161.7	161.7	161.7
	161.7	172	161.7	167.4
Pop 90	177.7	167.4	161.7	161.7
	167.4	167.4	167.4	167.4
	167.4	161.7	161.7	167.4
Pop 110	167.4	167.4	161.7	161.7
	161.7	167.4	161.7	161.7

Mut 40	Gen 20	Gen 30	Gen 40	Gen 50
	177.7	167.4	172	161.7
Pop 50	178.9	161.7	167.4	167.4
	167.4	186.3	178.9	178.9
	181.1	167.4	161.7	161.7
Pop 70	185.7	167.4	167.4	167.4
	182.3	161.7	161.7	167.4
	167.4	172	161.7	172
Pop 90	167.4	167.4	161.7	161.7
	176.6	167.4	183.4	167.4
	167.4	161.7	161.7	167.4
Pop 110	161.7	167.4	161.7	161.7
	167.4	161.7	167.4	167.4

Table 4.6: Delta Data for Problem 4

Mut 10	Gen 5	Gen 10	Gen 15	Gen 20
	327.5	314	314	314
Pop 50	314	314	327.5	314
	343	339	314	314
	327.5	314	339	314
Pop 70	327.5	327.5	314	314
	335.5	314	322	327.5
	335.5	327.5	322	314
Pop 90	322	327.5	314	314
	335.5	335.5	314	314
	322	327.5	314	314
Pop 110	322	314	322	314
	322	314	327.5	327.5

Mut 20	Gen 5	Gen 10	Gen 15	Gen 20
	314	327.5	314	327.5
Pop 50	322	327.5	327.5	314
	343.5	327.5	322	327.5
	335.5	340.5	327.5	314
Pop 70	335.5	335.5	327.5	314
	340.5	314	314	314
	335.5	314	314	314
Pop 90	327.5	322	314	314
	314	314	322	314
	335.5	322	314	314
Pop 110	314	322	314	322
	327.5	314	314	322

Mut 30	Gen 5	Gen 10	Gen 15	Gen 20
	339.5	348.5	322	314
Pop 50	340.5	322	322	327.5
	327.5	335.5	314	314
	343.5	314	327.5	314
Pop 70	327.5	343.5	327.5	314
	339.5	327.5	314	314
	327.5	327.5	314	314
Pop 90	335.5	335.5	327.5	314
	322	327.5	322	314
	327.5	327.5	314	314
Pop 110	327.5	314	314	314
	335.5	314	322	314

Mut 40	Gen 5	Gen 10	Gen 15	Gen 20
	348.5	314	335.5	314
Pop 50	340.5	340.5	314	327.5
	335.5	322	335.5	327.5
	343.5	327.5	322	327.5
Pop 70	314	327.5	327.5	322
	327.5	335.5	314	314
	340.5	314	314	314
Pop 90	314	322	314	314
	335.5	335.5	327.5	314
	322	314	322	314
Pop 110	314	314	314	314
	314	314	314	314

Table 4.7: Summary of GA Search for Small Problems

	Low WEST ratio	High WEST ratio
Low F-ratio	**Problem 4** Delta: 314.0 Time: 27 seconds*	**Problem 2** Delta: 186.0 Time: 46 seconds*
High F-ratio	**Problem 3** Delta: 161.7 Time: 47 seconds*	**Problem 1** Delta: 76.0 Time: 1190 seconds*

* Mutation 30%, Population 90, Generations 3rd attempted for each problem

consistent with the others, higher generations, populations, and slightly higher mutation rates yield the best results.

A mutation rate of 30% seems to be the best, especially when considering Problem 3. Both the higher generations and populations simply cause the GA search to last longer and to search more of the solution space; thus it seems logical that these two parameters would yield better solutions as they increase. Therefore, the four large problems will use a mutation rate of 30%, a population size of 90 and 110, and generations numbering 2000, 3000, and 4000. The results of this experimentation appear in Table 4.8. Table 4.9 holds a summary of the best results.

Using the correct number of generations is crucial to finding a good solution. Therefore, we will look at the delta value as the number of generations increases. First, Problem 1, since it required the highest generations amoung the small problems, may help to find out the most appropriate number of generations for the large problems. The GA search was run on Problem 1 by increasing the number of generations in steps of five each time. As shown in Figure 4.1, the optimal solution is reached at 55 generations. It appears that the populations level off fairly quickly. Thus, the number of generations started at 100, and proceed by increments of 100 until they reach 1000. Then the number of generations is set at 2,000, 3,000, and 4,000 to experiment with higher values. The results for Problems 5, 6, 7, and 8 appear in Figures 4.2, 4.3, 4.4, and 4.5, respectively. Looking at Figures 4.2, 4.3, 4.4, and 4.5, delta appears fairly level after 200 generations. Therefore, it appears that the GA search approaches good solutions quickly, but running the program for 3000 generations yields slightly better solutions.

Table 4.8: Delta Data for Large Problems

Problem 5

	Gen 2000	Gen 3000	Gen 4000
Mut 30			
	4225.9	3738.2	4084.8
Pop 90	4091.0	3997.7	4048.4
	4173.7	4264.7	4055.6
	4282.5	4233.5	4033.2
Pop 110	4307.2	3850.6	3898.7
	4090.6	3956.3	4101.2

Problem 6

	Gen 2000	Gen 3000	Gen 4000
Mut 30			
	3611.4	3538.0	3756.0
Pop 90	3647.7	3841.6	3610.6
	3724.6	3683.5	3795.9
	3652.7	3561.1	3915.2
Pop 110	3745.9	3545.2	3705.8
	3568.6	3650.8	3543.9

Problem 7

	Gen 2000	Gen 3000	Gen 4000
Mut 30			
	1272.0	1290.9	1214.8
Pop 90	1256.3	1276.0	1272.0
	1244.2	1260.2	1253.8
	1247.6	1280.6	1244.2
Pop 110	1283.9	1277.8	1286.4
	1260.2	1260.0	1221.1

Problem 8

	Gen 2000	Gen 3000	Gen 4000
Mut 30			
	828.7	815.9	852.5
Pop 90	871.6	878.1	823.7
	996.9	999.9	840.5
	860.7	903.4	866.5
Pop 110	802.4	936.2	909.7
	959.6	824.3	809.7

Table 4.9: Summary of GA Data for Large Problems

	Low WEST ratio	High WEST ratio
Low F-ratio	**Problem 5** Delta: 3658.58 Time: 3.71 hrs*	**Problem 7** Delta: 1168.49 Time: 2.29 hrs*
High F-ratio	**Problem 6** Delta: 3399.51 Time: 3.56 hrs*	**Problem 8** Delta: 765.94 Time: 2.43 hrs*

* Mutation 30%, Population 90, Generations 3000

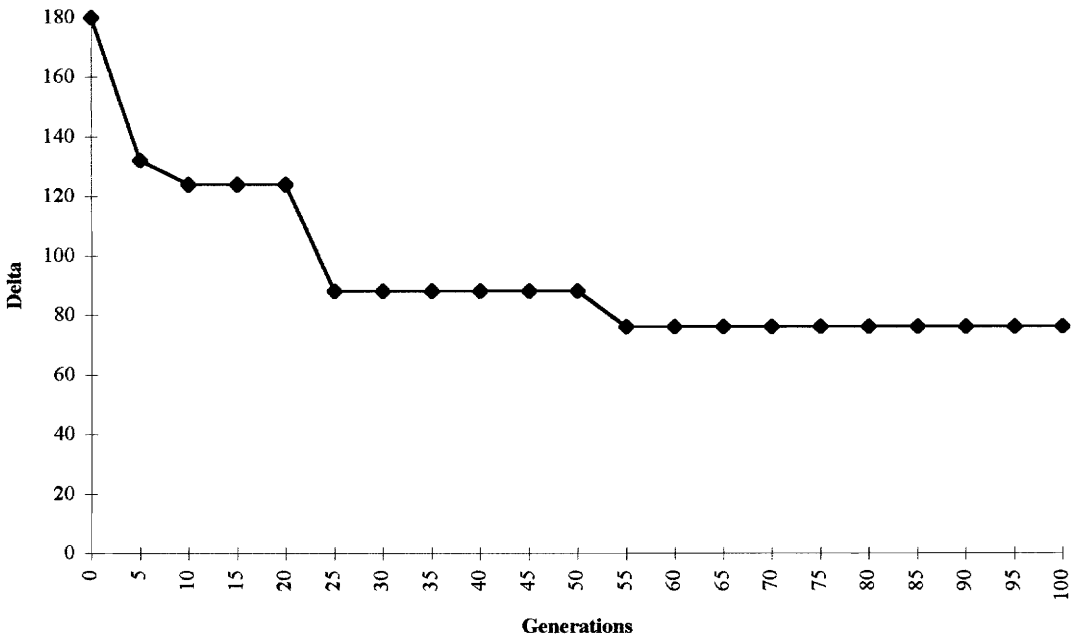


Figure 4.1: Problem 1 Delta vs. Generations

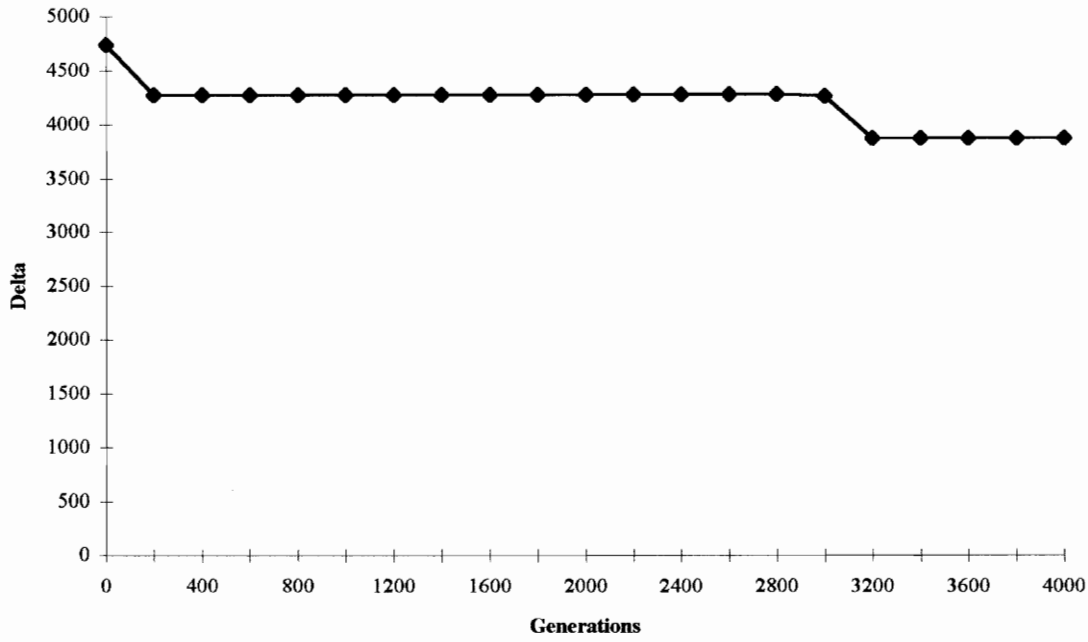


Figure 4.2: Problem 5 Delta vs. Generations

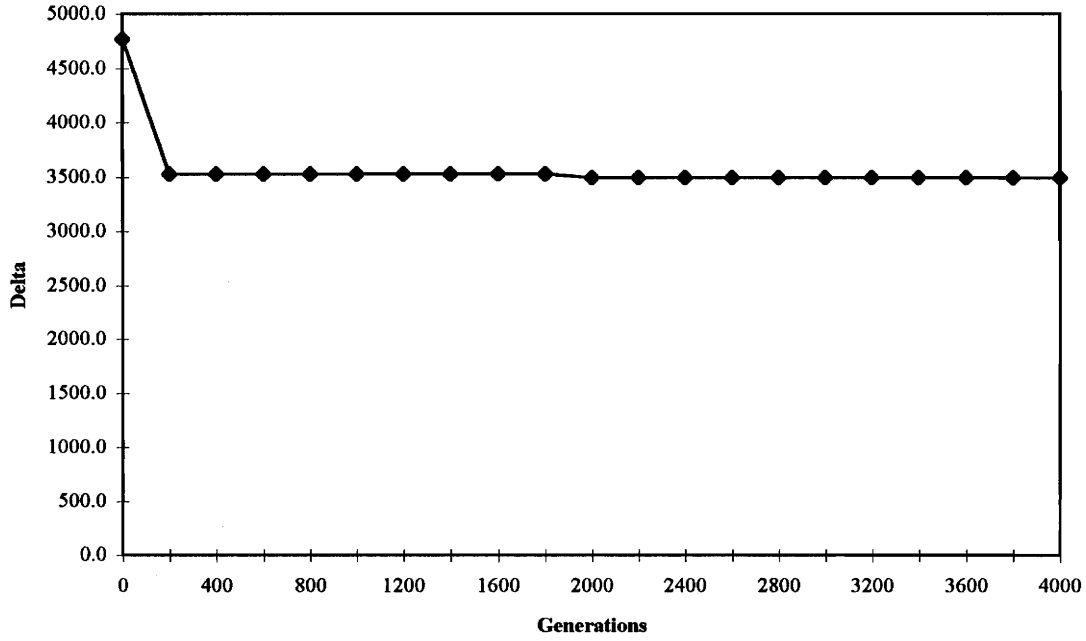


Figure 4.3: Problem 6 Delta vs. Generations

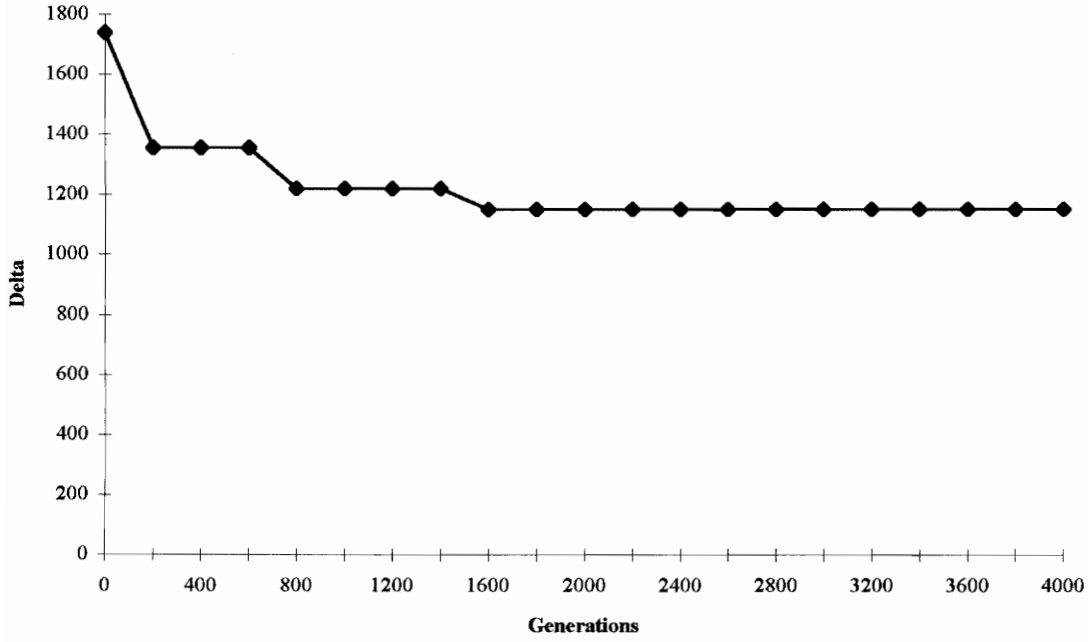


Figure 4.4: Problem 7 Delta vs. Generations

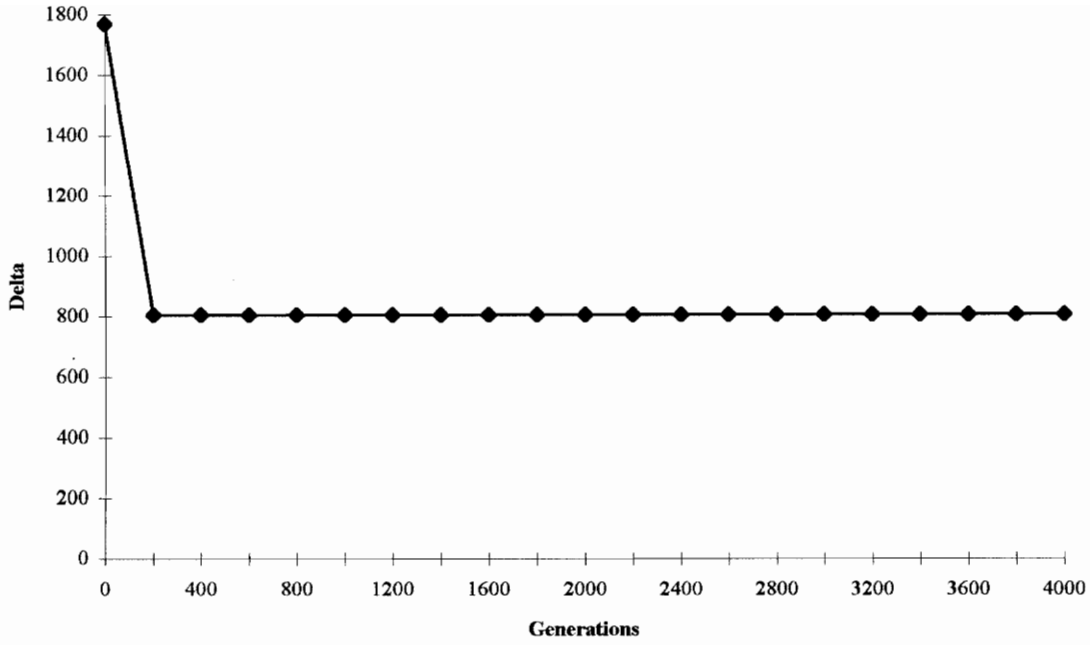


Figure 4.5: Problem 8 Delta vs. Generations

4.3 SUMMARY OF RESULTS

Table 4.10 provides a summary of the best results from the exhaustive search for the small problems, the SA search for the large problems, and the GA search for both problems. The times are also given. For the small problems, a typical time for a mutation rate of 30%, a population size of 90, and the 3rd generation value used for each problem. For the large problems a typical time for a mutation rate of 30%, a population size of 90, and 3000 generations.

Table 4.10: Summary of Results from Small and Large Problems

	Exhaustive Search	Genetic Algorithm
Problem 1	Delta: 60.0 Time: 109 min.	Delta: 76.0 Time: 20 min.
Problem 2	Delta: 164.0 Time: 1 min.	Delta: 186.0 Time: 1 min.
Problem 3	Delta: 161.7 Time: 1260 min.	Delta: 161.7 Time: 1 min.
Problem 4	Delta: 254.0 Time: 75 min.	Delta: 314.0 Time: 1/2 min.

	Simulated Annealing	Genetic Algorithm
Problem 5	Delta: 3531.46 Time: 90 min.	Delta: 3658.58 Time: 223 min.
Problem 6	Delta: 3294.27 Time: 68 min.	Delta: 3399.51 Time: 214 min.
Problem 7	Delta: 1189.75 Time: 41 min.	Delta: 1168.49 Time: 137 min.
Problem 8	Delta: 883.62 Time: 22 min.	Delta: 765.94 Time: 146 min.

CHAPTER 5: CONCLUSIONS

5.1 CONCLUSIONS

The objective of this thesis was to determine whether the Genetic Algorithm could be used to solve the mixed-model assembly line balancing problem and, if it could, to discover when to use it rather than other methods and what parameters would work best if applied to a general mixed-model assembly line balancing problem.

5.1.1 Assessment of the GA to Mixed-Model Line Balancing

In most other applications of the GA, the problem is not as constrained as the mixed-model assembly line balancing problem. The precedence diagram caused the representation used here to differ from the norm. Nevertheless, the GA worked well in this application. It was not able to achieve the optimal solution for all of the small problems because of the method used to fill stations, this is discussed in Section 4.2.1. However, the GA search was able to provide solutions near or better than those of the SA search.

5.1.2 Guidelines for When to Use the GA

For small problems, the exhaustive search should be used. Small problems are defined here as problems with less than 25 work elements. Also, it is suggested that the SA search or the GA search be used to determine parameters, such as minimum percent full and maximum number of stations, in order to shorten the exhaustive search. When

determining these parameters, it is important to keep the parameters loose enough to get feasible solutions. On large problems, the exhaustive search may be very slow even if one of the other two methods is used to approximate parameters. Using the results of a SA search of Problem 5, the parameters for the exhaustive search can be tightened; but, after running that problem for two days, not a single feasible solution was found.

If the problem is larger than 25 work elements, then the GA or the SA search should be used. According to the results of the two methods, it appears the GA works best when the WEST ratio is high. For Problems 7 and 8, the GA search performed slightly better than the SA search. When solving Problem 5, the SA search averaged 576.25 solutions/minute while the GA search averaged 2,261.72 solutions/minute. The SA search transfers or swaps work elements to form new solutions. If the transfer or swap generates a better solution, the SA search discards the old solution. The GA search keeps the best solution it has so far, but also keeps the mediocre or even bad solutions, at least until the next generation, when the bad solutions might be replaced. The GA search is much more cumbersome than the SA search, and takes longer to get good solutions. For this reason, the SA search or a low number of generations with the GA search should be used for quick solutions.

5.1.3 Applying the GA to the General Case

As described in Section 4.1.2, the best parameters to use for the GA search appear to be a mutation rate of 30% and a population size of 90. The number of generations depends on the problem size, but 3,000 generations will most likely give a good solution. These parameters yielded best solutions for the large problems. Edwards[23] has two methods of solving the mixed-model assembly line balancing problem and gives several

suggestions on using the SA search for the general case. It appears that the GA search is not as sensitive to the problem characteristics.

5.2 FUTURE RESEARCH

This research demonstrated that the GA could not only be applied to the mixed-model assembly line balancing problem, but that the GA can obtain competitive solution to other methods. Nevertheless, there are several ways that this research could be continued. The first would be to change the way the feasible orders are represented in the computer code. Currently, the feasible orders are stored in arrays, but by using pointers the program may run more quickly and be able to accommodate problems with over 128 work elements. Possibly more experimentation could be performed, to find out more about the solution space. For example, what would happen if the mutation rate were raised to 70%? Another area of further research would be to find a way to represent the solution such that it can obtain the optimal solutions for the small problems. It may be possible to randomly start a new station even though it is not be full. This would require some method of recording where this occurred so that it can be represented when it is displayed. Currently, the work elements are put in stations only when their fitness is determined and when the solution is printed. Possibly a dummy station could be inserted to force the start of a new station. And lastly, the search could be modified to include the model sequencing. One other improvement would be a stopping criteria which could make the processing time of the GA search much closer to the processing time of the SA search. These suggestions may slightly improve the GA search, but the methods and parameters used in the current software perform well.

REFERENCES

1. Askin, Ronald G. and Standridge, Charles R., 1993, *Modeling and Analysis of Manufacturing Systems*, John Wiley & Sons, Inc., pg. 9.

2. Merchant, Eugene, 1977, The Inexorable Push for Automated Production, *Production Engineering*, Jan., p45-46.

3. Stalk, Jr., George, 1988, Time-The Next Source of Competitive Advantage, *Harvard Business Review*, July-August, p41.

4. Samitt, Mark D. and Barry, Al, 1993, Mixed Model Operations: Solving the Manufacturing Puzzle, *Industrial Engineering*, August, p46-50.

5. Salveson, M. E., 1955, The Assembly Line Balancing Problem, *The Journal of Industrial Engineering*, Vol. 6, No. 3.

6. Thomopoulos, Nick T., 1967, Line Balancing-Sequencing for Mixed-Model Assembly, *Management Science*, Vol. 14, No. 2, October, pB-59-B75.

7. Thomopoulos, Nick T., 1970, Mixed Model Line Balancing with Smoothed Station Assignments, *Management Science*, Vol. 16, No. 9, May, p593-603.

8. Macaskill, J. L. C., 1972, Production-Line Balances for Mixed-Model Lines, *Management Science*, Vol. 19, No. 4, December, p423-434.

9. Roberts, S., and Villa, C., 1970, A Multiproduct Assembly Line-Balancing Problem, *AIIE Transactions*, Vol. 2, No. 4.

10. Prenting, T. O., and Battaglia, R. M., 1964, The Precedence Diagram: A Tool for Analysis in Assembly Line Balancing, *Journal of Industrial Engineering*, Vol. 15, No. 4.

11. Dar-El, E. M., 1973, MALB--A Heuristic Technique for Balancing Large Single-Model Assembly Lines, *AIIE Transactions*, Vol. 5, No. 4, p343-356.

12. Dar-El, E. M., and Cother, R. F., 1975, Assembly Line Sequencing for Model Mix, *International Journal for Production Research*, Vol. 13, No. 5, p463-477.

13. Dar-El, E. M., and Curry, S., 1977, Optimal mixed-model sequencing for Balanced Assembly Lines, *Omega*, Vol. 5, p333-341.

14. Dar-El, E. M., and Naidivi, A., 1981, A Mixed-Model Sequencing Application, *International Journal for Production Research*, Vol. 19, No. 1.

15. Kilbridge, M. D. and Wester, L., A Heuristic Method of Assembly Line Balancing, *Journal of Industrial Engineering*, Vol. 12, No. 4.

16. Wester, L. and Kilbridge, M. D., 1962, Heuristic Line Balancing: A Case, *Journal of Industrial Engineering*, Vol. 13, No. 3.

17. Johnson, R. V., 1983, A Branch and Bound Algorithm for Assembly Line Balancing Problems with Formulation Irregularities, *Management Science*, Vol. 29, No. 11.

18. Chakravarty, A. K., and Shtub, A., 1985, Balancing Mixed Model Lines with In-process Inventories, *Management Science*, Vol. 31, No. 9.

19. Arcus, Albert L., 1966, *COMSOAL: A Computer Method of Sequencing Operations for Assembly Lines*, The International Journal of Production Research, Vol. 4, No. 4.

20. Schofield, Norman A., 1979, *Assembly Line Balancing and the Application of Computer Techniques*, Computers and Industrial Engineering, Vol. 3, p53-69.

21. Smith, Pieter R., 1990, *A Computerized Search Methodology for the Design of Mixed Model Assembly Systems*, Master's Thesis, Virginia Polytechnic Institute and State University.

22. Pantouvanos, John P., 1992, *A Computerized Methodology for Balancing and Sequencing Mixed Model Stochastic Assembly Lines*, Master's Thesis, Virginia Polytechnic Institute and State University.

23. Edwards, Sherry L., 1993, *The Application of Simulated Annealing to the Mixed Model, Deterministic Assembly Line Balancing Problem*, Master's Thesis, Virginia Polytechnic Institute and State University.

24. Davidor, Yuval, 1991, *Genetic Algorithms and Robotics*, World Scientific Publishing Co., Singapore, p21.

25. Holland, John, 1992, *Adaptation in Natural and Artificial Systems*. Cambridge, Mass.: The MIT Press First edition 1975 The University of Michigan

26. Goldberg, David E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley Publishing Company, Inc.

27. Davis, Lawrence, 1991, *Handbook of Genetic Algorithms*. New York, N.Y.: Published by Van Nostrand Reinhold.

28. Leu, Y., Matheson, L.A., Rees, L.P., 1994, *Assembly Line Balancing Using Genetic Algorithms with Heuristic-Generated Initial Populations and Multiple Evaluation Criteria*. Decision Sciences, Vol. 25, No. 4.

29. Grefenstette, J. J., 1986, *Optimization of Control Parameters for Genetic Algorithms*. IEEE Transactions on System, Man, and Cybernetics, Vol. Smc-16, No. 1.

APPENDIX A: PROBLEM 1 INPUT FILE

```
414 414 19 3
120
60
40
.5
0
1
.4
.8
1.2
0
.2
.4
.4
0
0
.2
.2
.2
.2
0
0
.4
.5
.6
0
.5
.5
.4
.3
.2
0
0
.2
.3
.3
.3
.1
```

.3
.5
.1
0
.1
.2
.2
.2
.7
1
1.5
0
.1
0
.5
.5
0
.3
.5
.3
.4
.3
0
0
0
0
0
0
1
3
2
6
4
24
192
1024
1024
2304
1028
4096
8192
73728

APPENDIX B: RESULTS FROM EXHAUSTIVE SEARCH AND SIMULATED ANNEALING SEARCH

Small Problems solved with Exhaustive Search

Problem 1

Station 1: 2, 3, 4, 5, 11
Station 2: 1, 8, 13, 14, 16, 17
Station 3: 7, 9, 10, 12, 19
Station 4: 6, 15, 18

Delta: 60.0
Time: 109 minutes

Problem 2

Station 1: 1, 2, 3, 4, 8
Station 2: 5, 6, 7, 9, 10, 12
Station 3: 11, 13, 14, 15
Station 4: 16, 17, 18, 19

Delta: 164.0
Time: 1 minute

Problem 3

Station 1: 2, 4
Station 2: 1, 5, 8
Station 3: 7, 11
Station 4: 3, 6, 10, 13, 16, 17
Station 5: 12, 14, 19
Station 6: 9, 18
Station 7: 15

Delta: 161.7
Time: 1260 minutes

Problem 4

Station 1: 2, 6
Station 2: 5, 7, 10
Station 3: 1, 12
Station 4: 3, 9, 11
Station 5: 4, 8, 13, 14
Station 6: 15
Station 7: 16, 17
Station 8: 18, 19

Delta: 254.0
Time: 75 minutes

Large Problems solved with Simulated Annealing Search

Problem 5

Station 1: 5
Station 2: 3, 7
Station 3: 1, 6, 8
Station 4: 4, 10, 12
Station 5: 14, 16
Station 6: 9, 13
Station 7: 17
Station 8: 20, 22, 23
Station 9: 2
Station 10: 18, 24
Station 11: 25, 27
Station 12: 21, 26, 28
Station 13: 11, 29
Station 14: 30
Station 15: 32
Station 16: 31, 34
Station 17: 15, 33
Station 18: 35, 36
Station 19: 37, 38
Station 20: 19, 39
Station 21: 40, 41
Station 22: 42
Station 23: 43, 44
Station 24: 45
Station 25: 46
Station 26: 47, 48
Station 27: 49, 50

Delta: 3531.46
Time: 90 minutes

Problem 6

Station 1: 3, 7
Station 2: 2
Station 3: 5
Station 4: 8, 12, 20, 23
Station 5: 25
Station 6: 11, 29
Station 7: 4, 10
Station 8: 16
Station 9: 1, 22
Station 10: 15, 33
Station 11: 6, 14, 37
Station 12: 18, 24
Station 13: 21, 40
Station 14: 28, 42
Station 15: 19, 26
Station 16: 30
Station 17: 32
Station 18: 9, 34
Station 19: 13, 31
Station 20: 35, 36
Station 21: 17
Station 22: 38, 39
Station 23: 27, 43
Station 24: 41, 45
Station 25: 44, 47
Station 26: 46
Station 27: 48, 49, 50

Delta: 3294.27
Time: 68 minutes

Problem 7

Station 1: 3, 4, 5, 8, 12
Station 2: 9, 10, 13, 16, 20
Station 3: 1, 2, 7, 11, 14, 18
Station 4: 21, 22, 23, 24, 25
Station 5: 15, 17, 26, 28, 29
Station 6: 6, 27, 30, 31, 33, 35
Station 7: 19, 32, 37, 39
Station 8: 34, 36, 38, 40, 41
Station 9: 42, 43, 44, 45
Station 10: 46, 47, 48, 49, 50

Delta: 1189.75
Time: 41 minutes

Problem 8

Station 1: 3, 4, 5, 9
Station 2: 8, 10, 12, 13, 16
Station 3: 2, 7, 14, 18, 20, 21, 23, 28
Station 4: 1, 25, 29, 30
Station 5: 6, 11, 22, 24, 33
Station 6: 15, 26, 32, 37
Station 7: 31, 34, 36, 40
Station 8: 17, 27, 35, 38, 42
Station 9: 19, 43, 45, 47, 48
Station 10: 39, 41, 44, 46, 49, 50

Delta: 883.62
Time: 22 minutes

APPENDIX C: RESULTS FROM GENETIC ALGORITHM SEARCH

Problem 1

Station 1: 2, 3, 4, 5, 11
Station 2: 1, 8, 10, 13, 14, 16, 17
Station 3: 7, 9, 12, 19
Station 4: 6, 15, 18

Delta: 76.0
Time: 1190 seconds*

Problem 2

Station 1: 2, 5, 6, 7, 10
Station 2: 1, 3, 4, 8, 9, 12
Station 3: 11, 13, 14, 15, 16
Station 4: 17, 18, 19

Delta: 186.0
Time: 46 seconds*

Problem 3

Station 1: 2, 4
Station 2: 1, 5, 8
Station 3: 7, 11, 13
Station 4: 3, 6, 10, 16, 17
Station 5: 12, 14, 19
Station 6: 9, 18
Station 7: 15

Delta: 161.7
Time: 47 seconds*

Problem 4

Station 1: 2, 6
Station 2: 5, 7, 10
Station 3: 1, 12
Station 4: 3, 9, 11
Station 5: 4, 8, 13, 14
Station 6: 15
Station 7: 16, 17, 18
Station 8: 19

Delta: 314
Time: 27 seconds*

*Mutation 30%, Population 90, Generations 3rd attempted for each problem

Problem 5

Station 1: 5
Station 2: 2
Station 3: 4, 10
Station 4: 3, 7
Station 5: 1, 6, 8
Station 6: 9, 13
Station 7: 12, 17
Station 8: 14, 16
Station 9: 20, 22, 23
Station 10: 18, 24
Station 11: 25
Station 12: 11, 26
Station 13: 21, 27, 28, 29
Station 14: 30
Station 15: 32
Station 16: 31, 34
Station 17: 15, 33
Station 18: 35, 36
Station 19: 37, 38
Station 20: 19, 39
Station 21: 40, 41
Station 22: 42
Station 23: 43, 44
Station 24: 45
Station 25: 46
Station 26: 47, 48, 49
Station 27: 50

Delta: 3658.58
Time: 3.71 hrs*

Problem 6

Station 1: 5
Station 2: 4, 7
Station 3: 2
Station 4: 3, 6
Station 5: 1, 8
Station 6: 9, 10, 12
Station 7: 13, 20
Station 8: 16
Station 9: 22, 23
Station 10: 25
Station 11: 24
Station 12: 11, 26
Station 13: 15, 29
Station 14: 17
Station 15: 19, 33
Station 16: 14, 18, 37
Station 17: 21, 40
Station 18: 28, 42
Station 19: 30
Station 20: 32
Station 21: 31, 34
Station 22: 35, 36
Station 23: 27, 43
Station 24: 38, 39
Station 25: 41, 45
Station 26: 44, 47
Station 27: 46
Station 28: 48, 49, 50

Delta: 3399.51
Time: 3.56 hrs*

Problem 7

Station 1: 1, 2, 5
Station 2: 3, 4, 6, 7, 8, 11, 12
Station 3: 9, 10, 13, 16, 20
Station 4: 14, 18, 22, 23, 24, 25
Station 5: 17, 21, 26, 27, 28, 29
Station 6: 15, 30, 31, 33, 35
Station 7: 19, 32, 37, 39
Station 8: 34, 36, 38, 40, 41
Station 9: 42, 43, 44, 45
Station 10: 46, 47, 48, 49, 50

Delta: 1168.49
Time: 2.29 hrs*

Problem 8

Station 1: 1, 2, 5
Station 2: 3, 4, 6, 7, 10, 12, 20
Station 3: 8, 9, 11, 14, 23, 25
Station 4: 13, 15, 18, 21, 29, 33
Station 5: 16, 17, 27, 28, 37
Station 6: 19, 22, 30, 40
Station 7: 24, 32, 42
Station 8: 26, 31, 34, 35, 36
Station 9: 38, 43, 45, 47, 48
Station 10: 39, 41, 44, 46, 49, 50

Delta: 765.94
Time: 2.43 hrs*

*Mutation 30%, Population 90, Generations 3,000

APPENDIX D: COMPUTER CODE

The computer code for the GA search was written using Borland Turbo C++ Version 3.0.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#include <string.h>

#define MODSIZE 30
#define ELEMSIZE 128
#define NUMINTSS 4
#define MAXSTNSS 50
#define NumGens 3000
#define MUTRATE 30
/*if MUTRATE is 30 the mutation rate is 30%*/
#define POPSIZE 90
#define MAXPOP 182

void OneGen();
float rannd();
void Mutate(int inv);
void Xover(int ch1,int ch2);
void Sort_Check_Twins ();
void GetData ();
void Rand_Inv (int inv);
float Eval_Fitness (int pid);
int Pick_Parent();
void SaveResults();

float  fitness[MAXPOP],
       TTc,                    /*ideal cycle time*/
       TTcUL,                  /*upper cycle time same as TTc here*/
       ModelTe[MODSIZE],       /*Total time to build one unit of a model*/
       Ts[ELEMSIZE][MODSIZE],  /*Station Time [station][model]*/
                               /*resulting from given balance.*/
       TTsjk[ELEMSIZE][MODSIZE], /*Ts[j][k] (above) times Q[k] */
       TTsk[MODSIZE],          /*ModelTe[k] times Q[k] divided by NumStations */
       Te [ELEMSIZE][MODSIZE], /*Element time [element i][model k]*/
       TTe[ELEMSIZE],          /*Total element time [element i] to produce */
                               /*entire model mix.*/
       TTs [ELEMSIZE];         /*Total station time*/
```

```
int     order[ELEMSIZE][MAXPOP],/*feasible order for an individual
                                        this is the individual or chromosome*/
        NUMINTS=4,      /*4 if >96 elements, 3 if >64 elements,
                         2 if >32 elements, else 1 used with NUMINTSS
                            which establishes array size*/
        MAXSTNS=75,          /*User inputs the expected max # of stations*/
        NumElems,           /*Number of Elements*/
        NumMods,            /*Number of Models*/
        Q[MODSIZE],          /*Quantity of [model] to make per shift*/
        DataFlag,         /*Flag to determine if things are in memory*/
        sum1,sum2;

long int    prec[ELEMSIZE][NUMINTSS]; /*precedence diagram*/


void Calculations ()
{
        int     k, elem, SumQ;
        SumQ=0;
        for (k=0; k<NumElems; k++) TTe[k] = 0;
        for (k=0; k<NumMods; k++)
                {
                ModelTe[k] = 0;
                SumQ+=Q[k];
                }
        for (elem=0; elem<NumElems; elem++)
                {
                for (k=0; k<NumMods; k++)
                        {
                        TTe[elem]+= Te[elem][k]*Q[k];
                        ModelTe[k] += Te[elem][k];
                        }
                }
}

void GetData()
{
        FILE    *fptr;
        int     h,j,k,l;
        char filename[50];
        clrscr();
        printf("\n\n\n\tEnter data file name:  ");
        gets(filename);

        if ((fptr = fopen(filename, "rt")) == NULL)
                {
                printf("\n\tThe file does not exist");
                printf("\n\tHit any key to continue ..");
                getch();
                }
```

```
        else
        {
        fscanf(fptr,"%f %f %d %d",&TTc, &TTcUL, &NumElems, &NumMods);
        for (h=0; h<NumMods; h++) fscanf(fptr, "%d", &Q[h]);
        for (j=0; j<NumElems; j++) {
                for (k=0; k<NumMods; k++) {
                        fscanf (fptr, "%f", &Te[j][k]); }}
        for (l=0; l<NumElems;l++)
         {
         if (NumElems>32) fscanf (fptr, "%ld %ld",&prec[l][0],&prec[l][1]);
         else fscanf (fptr, "%ld", &prec[l][0]);
         }
        fclose(fptr);
        if (NumElems>96) NUMINTS=4;
        else if (NumElems>64) NUMINTS=3;
        else if (NumElems>32) NUMINTS=2;
        else NUMINTS=1;
        Calculations ();
        sum1=(NumElems+1)*NumElems/2;
        }

}

void Rand_Inv (int inv)  /*Fuction to generated a random solution*/
{
        long int assn[NUMINTSS], /*elements assigned*/
                        okay;
        int x,y,z,i,rn;
        int assnble[ELEMSIZE];

        for (i=0; i<NUMINTSS; i++) assn[i]=(long)0;

        for (x=0; x<NumElems; x++)
                {
                for (z=0; z<NumElems; z++) assnble[z]=0;
                z=0;
                for (y=0; y<NumElems; y++)
                        {
                        okay=(long)0;
                        for (i=0; i<NUMINTS; i++) okay|=prec[y][i]&~assn[i];
                        okay |= assn[y/32] & (long)1 << (y%32);
                        if (okay==0)
                                {
                                assnble[z]=y+1;
                                z++;
                                }
                        }
                rn = rannd()*z;
                order[x][inv]=assnble[rn];/*this is the chromosome*/
                assn[(assnble[rn]-1)/32] |= (long)1 << ((assnble[rn]-1)%32);
                }
```

```
        fitness[inv] = Eval_Fitness (inv);
}

float Eval_Fitness (int pid)
{
        float delta, TTss[ELEMSIZE];
        long int solution[ELEMSIZE][NUMINTSS];
        int s=0,dum,nmnt,x,y,elem,i,j,k,b,l;

        for (x=0; x<ELEMSIZE; x++)
                {
                TTss[x]=0;
                for (y=0; y<NUMINTS; y++) solution[x][y]=(long)0;
                }

        for (x=0; x<NumElems; x++)/*assign elements to stations*/
                {
                dum=order[x][pid]-1;
                nmnt=dum/32;
                if ((TTss[s]+TTe[dum]) > TTc)/*if current st. is full start new*/
                        {
                        s+=1;
                        TTss[s]=TTe[dum];
                        solution[s][nmnt] ^= ((long)1 << (dum-32*nmnt));
                        }
                else /*else not a new station*/
                        {
                        TTss[s]+=TTe[dum];
                        solution[s][nmnt] ^= ((long)1 << (dum-32*nmnt));
                        }
                }

/*calculate delta*/
        s++;
        for (j=0; j<s; j++)
                for (b=0; b<NumMods; b++)
                        Ts[j][b] = 0;
        for (j=0; j<s; j++)
                {
                for (l = 0; l < NumElems; l++)
                        {
                        if ((solution[j][l/32] >> l%32) &(long)1) {
                                for (b=0; b<NumMods; b++) Ts[j][b]+=Te[l][b];}
                        }
                }
        delta=0;
        for (j=0; j<s; j++)
        {
                for (k=0; k<NumMods; k++)
                {
```

```
                        TTsk[k] = Q[k] * ModelTe[k]/(s);
                        TTsjk[j][k] = Ts[j][k]*Q[k];
                        delta+= fabs(TTsk[k] - TTsjk[j][k]);
                }
        }
        return (delta);
}

void Sort_Check_Twins ()
{
        int temp[ELEMSIZE][1], id, fit, x,y,z,test,bbb,n;
        float low[1];

        for (x=0; x<(2*POPSIZE); x++) /*checks for twins ie similar delta*/
                for (y=x+1; y<(2*POPSIZE); y++)
                  if ((fitness[x]-.01)<fitness[y])
                        if (fitness[y]<(fitness[x]+.01))
                                Rand_Inv (y);

        for (x=0; x<((2*POPSIZE)-1); x++) /*sort */
                {
                low[0]=25000;
                for (y=x; y<(2*POPSIZE); y++) /*find low fitness*/
                  {
                  if (fitness[y]<low[0])
                        {
                        low[0]=fitness[y];
                        id=y;
                        }
                  }

                        for (z=0; z<NumElems; z++)  /*swap orders       */
                                {
                                temp[z][0]=order[z][x];
                                order[z][x]=order[z][id];
                                order[z][id]=temp[z][0];
                                }

                        fitness[id]=fitness[x];  /*swap fitness*/
                        fitness[x]=low[0];
                }
}

void Xover(int ch1,int ch2)
{
        int par1id=0,par2id=0,pos1=0,pos2=0,x,y,z,assnd,
                child1order[ELEMSIZE],child2order[ELEMSIZE];

        while (par1id==par2id)
                {
                par1id=Pick_Parent();
```

```
                par2id=Pick_Parent();
                }

        while (pos1==pos2)
                {
                pos1=rannd()*(NumElems-2);
                pos2=rannd()*(NumElems-2);
                }

        if (pos1>pos2)/*want pos1 to be low position*/
                x=pos1,  pos1=pos2,   pos2=x;

        for (x=0; x<NumElems; x++)
                child1order[x]=0,child2order[x]=0;

        for (x=0; x<pos1+1; x++)        /*set first section*/
                child1order[x]=order[x][par1id],
                child2order[x]=order[x][par2id];

        for (x=pos2+1; x<NumElems; x++) /*set last section*/
                child1order[x]=order[x][par1id],
                child2order[x]=order[x][par2id];

/*assign middle section child1*/
        y=0;
        for (x=pos1+1; x<pos2+1; x++)
                {
                while (child1order[x]==0)
                        {
                        assnd=0;
                        for (z=0; z<NumElems; z++)
                                if (order[y][par2id]==child1order[z]) assnd=1;
                        if (assnd==1) y++;
                        else child1order[x]=order[y][par2id],y++;
                        }
                }

/*assign middle section child2*/
        y=0;
        for (x=pos1+1; x<pos2+1; x++)
                {
                while (child2order[x]==0)
                        {
                        assnd=0;
                        for (z=0; z<NumElems; z++)
                                if (order[y][par1id]==child2order[z]) assnd=1;
                        if (assnd==1) y++;
                        else
                                {
                                child2order[x]=order[y][par1id];
                                y++;
```

```
                        }
                    }
                }

        for (x=0; x<NumElems; x++)   /*assigns child order to new orders*/
                {
                order[x][ch1]=child1order[x];
                order[x][ch2]=child2order[x];
                }

        fitness[ch1] = Eval_Fitness (ch1);
        fitness[ch2] = Eval_Fitness (ch2);
}

void Mutate(int inv)
{
        int assnble[ELEMSIZE],i,rn,x,y,z,parent,
                parorder[ELEMSIZE],
                childorder[ELEMSIZE];
        long int assn[NUMINTSS], okay;

        for (i=0; i<NUMINTSS; i++) assn[i]=(long)0;

        parent=Pick_Parent();

        for (x=0; x<NumElems; x++) childorder[x]=order[x][parent];

        rn = rannd()*NumElems-2;/*random mutation point*/

        for (x=rn+1; x<NumElems; x++)  childorder[x]=0;
        /*sets order after random point = 0*/

        for (x=0; x<rn+1; x++)
                assn[(childorder[x]-1)/32] |= (long)1 << ((childorder[x]-1)%32);
        /*sets assn = to the tasks still in the order*/

        for (x=rn+1; x<NumElems; x++)
                {
                for (z=0; z<NumElems; z++) assnble[z]=0;
                z=0;

                for (y=0; y<NumElems; y++)
                  {
                  okay=(long)0;
                  for (i=0; i<NUMINTS; i++) okay |= prec[y][i] &~ assn[i];
                  okay |= assn[y/32] & (long)1 << (y%32);
                  if (okay==0)
                  {
                  assnble[z]=y+1;
                  z++;
                  }
```

```
                }
                rn = rannd()*z;
                childorder[x] = assnble[rn];
                assn[(assnble[rn]-1)/32] |= (long)1 << ((assnble[rn]-1)%32);
                }

        for (x=0; x<NumElems; x++) order[x][inv]=childorder[x];

        sum2=0;
        for (x=0; x<NumElems; x++) sum2+=order[x][inv];
        if (sum1!=sum2) Rand_Inv (inv);

        fitness[inv] = Eval_Fitness (inv);
}

int Pick_Parent()/*scaled selection of a parent*/
{
        int x;
        double rn;
        float a=0,b=0,min,avg,max,fmult=2.0,sum=0,high,newfit[POPSIZE];

        min=fitness[0];
        max=fitness[POPSIZE];

        for (x=0; x<POPSIZE; x++)
                {
                if (fitness[x]>max) max=fitness[x];
                if (fitness[x]<min) min=fitness[x];
                sum+=fitness[x];
                }

        avg=sum/POPSIZE;
        a=(fmult-0.5)*avg/(max-min);
        b=avg*(max-fmult*avg)/(max-min);

        high=0;
        for (x=0; x<POPSIZE; x++)
                if ((a*fitness[x]+b)>high) high=a*fitness[x]+b+.01;

        sum=0;
        for (x=0; x<POPSIZE; x++)
                {
                newfit[x]=high-(a*fitness[x]+b);
                if ((a*fitness[x]+b)<0) newfit[x]=0.01;
                sum+=newfit[x];
                }
        rn=rannd()*sum;

        sum=0;
        x=0;
        while (rn>=sum)
```

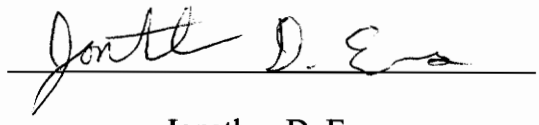
```
                {
                sum+=newfit[x];
                x++;
                }
        x--;
        return x;
}

float rannd()
{
        float rd_v;
        int t;
        t = random(10000);
        rd_v = 1.0*t/10000;
        return(rd_v);
}

void SaveResults ()
{
        FILE *fptr;

        float TTss[ELEMSIZE];
        long int solution[ELEMSIZE][NUMINTSS];
        int s=0,dum,nmnt,x,y,z,elem,i,j,k,b,l,mins;
        char *num;

        if ((fptr = fopen ("output.txt", "wt")) == NULL)
                printf ("Trouble opening outfile");
        else
        {
        fprintf (fptr,"Number of Generation is %d",NumGens);
        fprintf (fptr,"\nMutation rate is %d%%",MUTRATE);
        fprintf (fptr,"\nPopulation size is %d",POPSIZE);

/****************print out top 10***************/
/*********************************************/
        for (z=0; z<10; z++)
        {
        s=0;
        for (x=0; x<ELEMSIZE; x++)
                {
                TTss[x]=0;
                for (y=0; y<NUMINTS; y++) solution[x][y]=(long)0;
                }
        for (x=0; x<NumElems; x++)/*assign elements to stations*/
                {
                dum=order[x][z]-1;
                nmnt=dum/32;
                if ((TTss[s]+TTe[dum]) > TTc)/*if current st. is full start new*/
                        {
                        s+=1;
```

```
                        TTss[s]=TTe[dum];
                        solution[s][nmnt] ^= ((long)1 << (dum-32*nmnt));
                        }
                else /*else not a new station*/
                        {
                        TTss[s]+=TTe[dum];
                        solution[s][nmnt] ^= ((long)1 << (dum-32*nmnt));
                        }
                }
        s+=1;
        for (j=0; j<s; j++)
        {
        y=0;
        fprintf (fptr,"\nStation: %d-->  ", j+1);
        for (i=0; i<NUMINTS; i++)
                {
                for (x=0; x<32; x++)
                {
                if ((solution[j][i] >> x) & (long)1)
                        {
                        if (y) fprintf (fptr,", %d", x+1+32*i);
                        else fprintf (fptr," %d", x+1+32*i);
                        y = 1;
                        }
                }
                }
        fprintf (fptr,"   ");
        }
        fprintf (fptr,"\nDelta is:  %7.3f",fitness[z]);
        }
        }
        fclose(fptr);
}

void OneGen()
        {
        int x,rn;
        x=POPSIZE;

        while (x<(2*POPSIZE))
                {
                rn=rannd()*100;
                if (rn<MUTRATE)
                        {
                        Mutate(x);
                        x++;
                        }
                else
                        {
                        Xover(x,x+1);
                        x+=2;
```

```
                        }
                }
        Sort_Check_Twins();
        }

void main (void)
{
        int i,x;
        float start, end;
        randomize();
        GetData();
        clock_t ticks;

                if ((ticks=clock())!=(clock_t)-1) start=ticks/CLK_TCK;
                else printf("Error with the clock() function/n");

                for (i=0; i<POPSIZE; i++) Rand_Inv (i);
                for (i=0; i<NumGens; i++) OneGen();

                if ((ticks=clock())!=(clock_t)-1) end=ticks/CLK_TCK;
                else printf("Error with the clock() function/n");

                printf("\nGens %d,Mut %d,POP %d,Best fitness is %.1f,",
                        NumGens,MUTRATE,POPSIZE,fitness[0]);
                printf(" %.2f hours",(end-start)/3600);

        SaveResults();
        printf("\nHit any key to continue");
        getch();
}
```

VITA

Jonathan D. Evans was born on March 04, 1971, to James D. and Judy D. Evans, of Rustburg, Virginia. Jonathan graduated from Rustburg High School in 1989. In 1994, he obtained a Bachelor of Science degree in Industrial and Systems Engineering from Virginia Polytechnic Institute and State University. As an undergraduate, Jonathan was in the cooperative education program at Babcock and Wilcox's Navy Nuclear Fuel Division and was selected as a member of Alpha Pi Mu, the Industrial Engineering Honor Society.



Jonathan D. Evans