Solving the Aircraft Engine Maintenance Scheduling Problem using a Multi-objective Evolutionary Algorithm

Mark P. Kleeman and Gary B. Lamont

Air Force Institute of Technology, Dept of Electrical and Computer Engineering,
Graduate School of Engineering & Management,
Wright-Patterson AFB (Dayton) OH, 45433, USA
Mark.Kleeman@afit.edu, Gary.Lamont@afit.edu
http://www.afit.edu *

Abstract. This paper investigates the use of a multi-objective genetic algorithm, MOEA, to solve the scheduling problem for aircraft engine maintenance. The problem is a combination of a modified job shop problem and a flow shop problem. The goal is to minimize the time needed to return engines to mission capable status and to minimize the associated cost by limiting the number of times an engine has to be taken from the active inventory for maintenance. Our preliminary results show that the chosen MOEA called GENMOP effectively converges toward better scheduling solutions and our innovative chromosome design effectively handles the maintenance prioritization of engines.
Keywords Multi-objective Evolutionary Algorithms, Scheduling Problem, Aircraft Engine Scheduling, Variable-length chromosome

1 Introduction

Scheduling problems are a very common research topic. This is because, for efficiency reasons, our world relies heavily on schedules and deadlines. Aircraft engine maintenance is no exception. The United States Air Force has many planes that it must keep up and running. But with the downsizing that has occurred in recent years, the number of planes that are operational has become more critical. This means that every effort needs to be made to ensure that not only are the engines repaired in an efficient manner, but that their component's scheduled maintenance cycles are in sync so that the engine has fewer trips to the logistics maintenance center.

Section 2 looks briefly at the types of scheduling problems that are researched and describes how each one behaves. Section 3 describes specifically the aircraft scheduling problem. It states why this problem is important, shows the maintenance flow, and gives the particulars as to how a system was implemented

* The views expressed in this article are those of the authors and do not reflect the official policy of the United States Air Force, Department of Defense, or the United States Government.

to solve the problem. Section 4 describes the algorithm used to implement the problem. Section 5 discusses our design of experiments and tells what metrics we used and why. Section 6 show the results of our experiments and looks at them analytically in an effort to discern any trends. Finally, the paper concludes with Section 7 where we summarize our findings and describe the future work we are doing on this problem.

2 Scheduling Problems

Scheduling problems can be formulated in many different ways. They are commonly abstracted into three different classes of problems: open shop, flow shop, and job shop. The definitions for these problems are described in [1]

2.1 Flow Shop

The flow shop scheduling problem consists of m machines and n jobs. All m machines are situated in a defined series. All n jobs have to be processed on each machine. All the jobs must follow the same routing along the series of machines. Once a job is completed on one machine, it is placed into the queue of the next machine in the series. Normally, jobs are removed from the queue on a *first in first out* (FIFO) basis, but this can be modified to fit the needs of the problem, such as higher priority jobs could be bumped to the front of the queue.

2.2 Job Shop

For the job shop problem, unlike the flow shop problem, each job has its own route to follow. With job shop problems, they can be modelled by having jobs visit any machine at most one time, or they can be created to allow for multiple visits to machines [1].

3 Aircraft Scheduling Problem

The U.S. Air Force has many aircraft in its arsenal. Its fleet is aging and it appears that plans are for the aircraft to be around for a while longer. But many of these engines are beyond their design life. In fact, 97% of all F100 engines and 84% of all F110 engines will be past their design life by the year 2010 [2]. One essential thing that the Air Force relies upon is a dependable fleet. If the Air Force does not have reliable aircraft, then they must procure redundant systems in order to ensure the success of any mission with a high degree of certainty. To achieve this type of dependability, the old "fix it when it breaks" mentality or the on condition maintenance (OCM) practice had to be revamped in order to overcome much of the uncertainty that is inherent in this type of repair mentality.

So now, aircraft engines are brought into the shop for two reasons: unscheduled maintenance and routine maintenance. Unscheduled maintenance occurs

when an engine part breaks and requires repair or replacement. Routine maintenance occurs on predefined intervals, when the engine component is still operational, but history shows that the mean time to repair (MTTR) for that component is almost up. For dependability purposes, it is better to refurbish a component before a failure occurs instead of only repairing broken components. This type of mindset is typical of the reliability-centered maintenance (RCM) philosophy. A properly implemented RCM philosophy "systematically analyzes aircraft components/systems to identify and implement the best preventive maintenance policies" [3]. Scheduled maintenance is only part of the equation when it comes to RCM. Another goal of RCM is to reduce the amount of scheduled maintenance. While this is somewhat limited to the MTTR of specific components, it can also be influenced by the components used in the assembly of an engine. For example, it would be better to have two components that have a MTTR around 500 than it would be to have MTTR values of 500 and 1000 for the same two components.

3.1 Maintenance Flow

With RCM as our focal point, our problem is set-up in the following manner. When an engine comes into a logistics workcenter for repair, it is first logged into the system. Aircraft engines are commonly divided into smaller subcomponents which can be worked on individually and then recombined. For this problem, we divided the engine into five logical subcomponents: fan, core, turbine, augmenter, and nozzle. We assumed that the maintenance shop had one specific work area for each of the components. This is an example of the job shop problem, but with a twist. After all maintenance is completed on an engine, each engine component's mean time to repair (MTTR) is compared with other components on the engine. If there is a large disparity among the MTTRs then a component swap may be initiated with another engine in an effort to ensure the MTTRs of the components of a particular engine are similar. This is done so that the engine can have more time on wing (TOW) and less time in the shop.

Once the swaps are done, the engine is reassembled and tested as a whole to ensure functionality. This represents a small flow shop problem in that each engine has to have maintenance done first, followed by swapping and then testing. So the problem at hand is actually a hybrid of two scheduling problems.

Figure 1 shows an example of the flow for two engines. As you can see the problem has a certain flow to it, and the component repair is more of a job shop problem that is embedded into the flow.

3.2 Program Specifics

Static Scheduling The program was written as a static scheduling problem, where the program receives an initial set of inputs and it outputs results based on them. A dynamic scheduling problem allows the user to add additional items to the schedule while the program is running. For our problem, it is reasonable to assume that the logistics center is always alerted ahead of time as to what

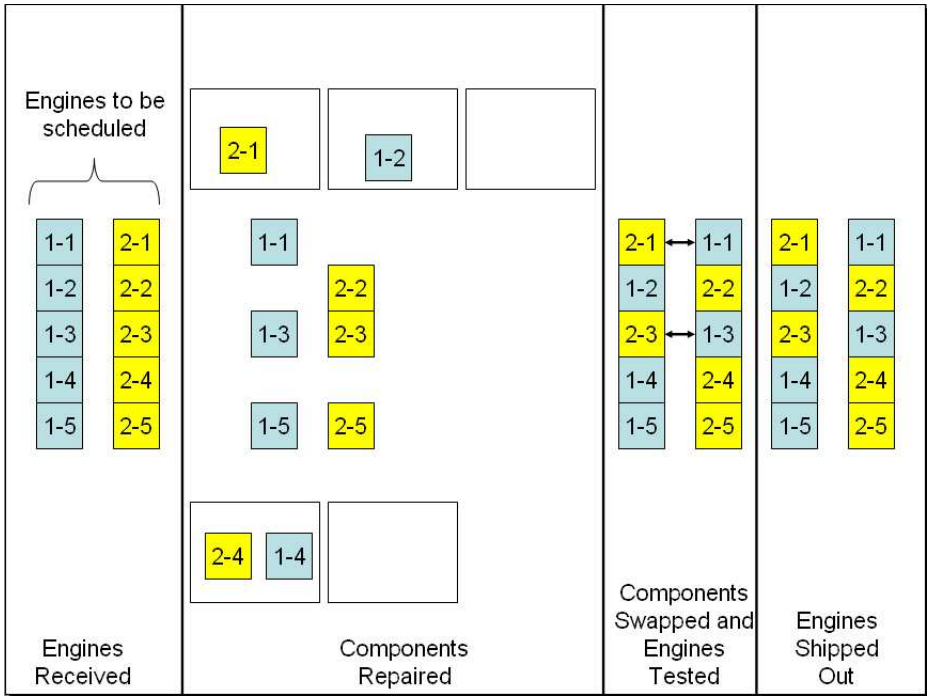

Fig. 1. Example of maintenance flow for two engines

engines are coming and when. Since the shop is alerted a priori of the items to be scheduled, we felt that static scheduling was more appropriate for our problem domain.

Input file Our input file is made up of all the information that may be useful for our scheduling problem. Some of the information is not used currently, but it was included for completeness and for future research. The first line of the file lists the n number of engines that need to be scheduled for repair. This is followed by n lines, where each line details the particulars of a specific engine. First the engine part number is listed. Then the estimated arrival time of the engine is listed. This is followed by the due date and weight. The due date is self explanatory, while the weight is a priority system for the program. A weight of one signifies a top priority, while a weight of three is a low priority job. The next ten numbers are divided into five sets of two, where each set represents one of the five components of the engine. The first number in a set is the time that is estimated for the component to be fixed. A zero in this spot means that the component is good and does not need repair work. The second number of each set is the MTTR.

Chromosome Representation The chromosome representation for this problem is rather unique. The size of the chromosome varies based upon the number of engines that are to be scheduled. Bagchi [4] lists two basic encoding approaches:

1. Direct approach
2. Indirect approach

He then lists nine distinct representations that have been used for job-shop scheduling problems. The chromosome representation used in this research is a direct approach that is based on the "Job-Based GA representation" [4, 5].

The chromosome representation used can be effectively divided into two part:

1. Precedence order for the engines requiring repair
2. Component swaps

The first portion of the chromosome uses the job-based GA representation. Each allele in the chromosome represents an engine. The first allele listed has precedence over all other engines for component repairs. So if it has three components that need repair, they will be scheduled first in those respective repair locations. It follows that the last engine listed will have its components repaired last.

The second portion of the chromosome determines the precedence of the component swaps, the number of swaps, the components to be swapped, and the engines that the components are to come from. The most difficult part of this problem determining how many swaps to allow. By having a lot of swaps, you increase the problem complexity and if you increased the number of swaps beyond a reasonable limit, the problem efficiency is greatly reduced. For example, say I have three engines being repaired. Suppose that it would take four swaps in order to get the best answer. If you allow for 20 swaps, the search space is increased drastically by the additional 16 swaps that are unnecessary. Conversely, if you only allow for 4 swaps and you need 10, then there is no way that you can ever achieve the best answer because you do not allow for enough swaps. We concluded that as the number of engines is increased, the number of swaps would increase as well. So our chromosome was developed in order to accommodate the varying number of swaps based on the number of engines that are to be repaired. But we did not want to force the program to make swaps when none were needed. For example, suppose three engines go in for complete overhauls one week. When the repairs are done, none of the engines need to swap because all of the MTTR values are similar. By doing unneeded swaps, the makespan goes up needlessly. Therefore, the number of swaps can be zero or go as high as the total number of engines being repaired. Section 6 contains an analysis regarding the maximum amount of swaps and the average number of swaps recorded for individuals along the known Pareto front.

Figure 2 show a representation of a chromosome for four engines. Note that the first four alleles are for the engine scheduling precedence, while the last 12 alleles are for the swaps. There are four possible swaps. Each swap is represented by three alleles, the two engines that will be swapping components and the actual

component to be swapped. Note that for this example, there will be only three swaps because one of the swaps has a zero for the component, signifying no swap is to occur.

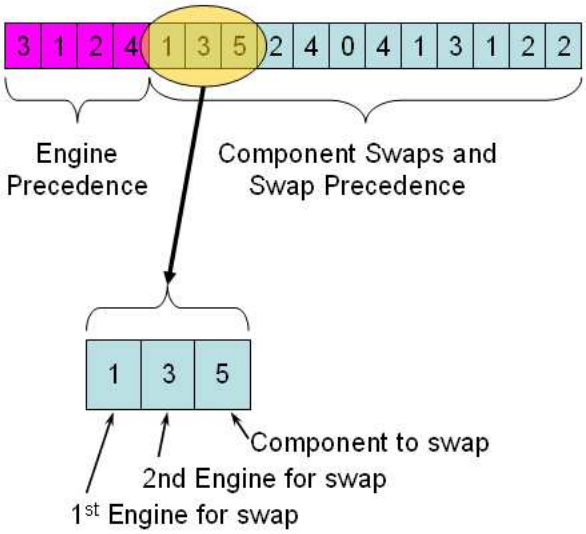

Fig. 2. Example of chromosome representation for four engines

The formula for determining the chromosome length is found in the following equation:

$$L_{Chromo} = E + 3E = 4E \tag{1}$$

where L_{Chromo} is the length of the chromosome, and E symbolizes the number of engines to be scheduled. As you can see the chromosome length is four times the number of engines.

Fitness Functions used Another important aspect of the program are the fitness functions. As stated earlier, the goal is to develop a program that limits the time an engine is in the shop for repairs and to limit the number of times the engine has to go into the shop for scheduled maintenance. These are competing objectives because the first objective is to get the engines out as quickly as possible and the other objective is to match component MTTR values, which requires more time in the shop. It was determined that the best fitness functions to use for this problem were the makespan and the aggregate swap count.

The makespan, C_{max} is equivalent to the completion time of the last job to leave the system. For our problem this is not easily defined. First, every engine component that requires maintenance, is scheduled based on two things:

1. Its precedence defined in the chromosome
2. The arrival time of the engine

If an engine has not arrived in the shop yet and it has the highest precedence, all other components must wait until the work is done on the component with the highest precedence. By doing this, it allows the researcher to be more flexible in having higher priority items getting attention before lower priority items. Each component has its own work area, so components can be worked on simultaneously. Not all repairs take the same amount of time to complete, so finishing times may vary depending on component and on the specific component problem. Equation 2 is the mathematical description of how the repair time for each engine is calculate.

$$\forall e \in E, R_e =_{c \in C}^{max} t_{CompStart} \sum_{i=1}^{P_e} t_R(i) \tag{2}$$

where e is an engine, E is the set of engines, R_e is the repair time for the engine, c is one of the five components of the engine, C is the set of components, $t_{CompStart}$ is the time when maintenance begins for the component with the highest precedence, P_e is the precedence of engine e, and $t_R(i)$ it the estimated repair time for engine i. Basically, the engine repair time is the completion time of the final component.

Next, component swaps are calculated based on two things:

1. The last (maximum) completion time for repair for either of the two engines involved in the swap
2. The precedence of the swap

Once these two items are determined, the swap occurs and the new completion time for both engine components is adjusted. If multiple swaps occur for a single engine, the precedence, as defined in the chromosome, determines which swaps occur first. Also note that different components have different swap times. This is because, in reality, not all component swaps take the same amount of time to perform. Equation 3 shows how the swap times are calculated

$$S_e =_{\forall e \in S}^{max} R_e + t_s(c) \tag{3}$$

where S_e is the engine swap completion time, e is an engine, S is the set of two engines that are involved in the swap, R_e is the repair time for engine e, and $t_s(c)$ is the swap time for component c, the component to be swapped. Since not all engines will participate in swaps, we can conclude that $S_e \geq R_e$.

Finally, each engine is tested to ensure it is working properly before it is shipped back to the owner. Testing occurs after all repairs and swaps are done on an engine. Equation 4 describes how this is done.

$$T_e = S_e + t_{test} \tag{4}$$

where T_e is the engine testing completion time, S_e is the swap completion time, and t_{test} is the time it takes to accomplish engine testing.

Finally, the makespan is calculated according to Equation 5.

$$C_{max} = \forall e \in Engines^{max} T_e \tag{5}$$

where C_{max} is the makespan for a chromosome representation, e is an engine, and T_e is the engine testing completion time. So the test completion time of the last engine to be tested is the makespan. This value is the fitness value for the first fitness function. So the lower the makespan value, the better the fitness for the chromosome.

The second fitness function is the aggregate swap count. The aggregate swap count is calculated after the makespan is calculated. The program compares the MTTR for all the components of an engine. If the MTTR is outside the provided tolerance levels for MTTR, then the aggregate swap count is increased. This is done for each engine and the summation of the results is the aggregate swap count.

$$ASC = \sum_{i=1}^{e} \sum_{k=1}^{c} tol_{ik_{MTTR}} \tag{6}$$

where the ASC is the aggregate swap count, e is the total number of engines, c is the total number of components (for this problem, $C = 5$), and $tol_{ik_{MTTR}}$ represents the total number of components of engine i that are outside the tolerance limits of the component k on engine i. These values are calculated in such a way so that they are counted only once. The total number of components whose MTTR are out of tolerance with others on the same engine are then summed together with all the other engines to get the aggregate swap count.

4 GENMOP Software Design

The problem was integrated into the General Multi-objective Parallel Genetic Algorithm (GENMOP). GENMOP was originally designed as a real-valued, parallel MOEA. This section briefly describes MOEAs and then discusses in more detail GENMOP operations.

4.1 Multi-Objective Evolutionary Algorithms (MOEAs)

Evolutionary algorithms (EA) include genetic algorithms, evolution strategies (ES), and evolutionary programming (EP). EAs consist of a class of algorithms that use the concepts of genetics which enable them to explore the search space. In an evolutionary algorithm, there is a collection of individuals, each one is known as a chromosome. A group of chromosomes are created and compared to

one another. This group is known as a population. Chromosomes consist of alleles that can be encoded into a variety of datatypes: binary, integer, real-valued, etc. These allele values are altered by EA operators such as mutation and recombination. Mutation works by inserting new genetic material into the population by modifying allele values. Recombination is accomplished by exchanging allele values between two or more individuals of the population. There are many varieties of genetic operators each with a different set of parameters that may be modified given a particular EA type [6]. After the chromosomes are modified, a selection process occurs where a new population is determined for the next generation. There many are ways to select one chromosome over another [7], but the main objective of the selection process is to steer the EA toward the solution. Regardless of the selection process used, all EAs have a fitness function that they assign to a chromosome. The selection process compares the fitness functions of the chromosomes in order to determine the new population.

A multi-objective EA (MOEA) differs from an EA in that there is more than one fitness function for each chromosome. This often creates a situation where there is a vector of answers that can be considered optimal. To determine which answer is best, the researcher must either weight the fitness values or must pick one point out of the group via an inspection process. The book by Coello Coello [8] explains many of the important aspects of MOEAs.

4.2 GENMOP

GENMOP is an implicit building block MOEA that attempts to find good solutions with a balance of exploration and exploitation. It is a Pareto-based algorithm that utilizes real values for crossover and mutation operators. The MOEA is an extension of the single objective GENOCOP algorithm [9, 10]. GENMOP was initially used to optimize flow rate, well spacing, concentration of injected electron donors, and injection schedule for bioremediation research [11, 12]. In this research, it was used to maximize percholate destruction in contaminated groundwater and minimize the cost of treatment. The algorithm was later used to optimize quantum cascade laser parameters in an attempt to find viable solutions for a laser that operates in the terahertz frequency range [13, 14].

The algorithm has four crossover methods and three mutation methods that are used. Each operator is chosen based upon an adaptive probability distribution, where the operators that produce the most fit individuals have their selection probability increased.

The algorithm flow is similar to most MOEAs. First, the input is read from a file. For our problem, a list of engines is read in, with each engine listing its arrival time, due time, priority (weight), and MTTR for each of its components. Next, the initial population is created and each chromosome is evaluated. The population is then ranked based on the Pareto-ranking of the individuals. Then a mating pool is created and only the most fit individuals are chosen to be in the mating pool. Crossover and mutation are performed on the members of the mating pool. The children created are then evaluated and saved. These children are then combined with the rest of the population ($\mu + \lambda$). The population is

then put into Pareto-rank order. The program then checks to see if the program has run through its allotted amount of generations. If it has, the program exits. If it has not, the program creates another mating pool and goes through the process again.

5 Design of Experiments and Testing

This section describes the computational system that executes GENMOP and then discusses our experimental method.

5.1 System Configuration

The systems used for testing were Linux Beowulf clusters. Each node runs dual Pentium III, 1 GHz chips. They are all running in a homogenous environment with Linux 7.3 as the operating system. While our current experimentation is done serially, future work will look into effective parallel processing for this problem.

5.2 Experimental Approach

The goals of this research were simple:

1. Develop an effective way to model the real-world aircraft engine scheduling problem
2. Implement the model using GENMOP
3. Determine if the number of swaps encoded in the chromosome is adequate
4. Determine if the data shows trends that may require further study

The first goal was difficult because we had to define some way of quantifying how costs are affected by engine swaps. Several different fitness functions were studied, but most of them did not capture what we were looking for. The aggregate swap count, it turns out, captures everything needed for this problem. Since the goal is decreasing costs by limiting the number of times an engine is in the shop, counting the out of tolerance variance in the MTTRs of each engine appears to be the most logical choice. This and other design decisions have been discussed earlier in the document.

The model implementation was another difficult task. Since the algorithm is a real-valued MOEA, and our problem is not a real-valued problem, some changes had to be made to the algorithm in order to accommodate our problem. GENMOP is discussed at greater length in Section 4.

As mentioned in section 3.2, our chromosome has encoded into it a limited number of swaps. These swaps are critical in limiting the amount of time an engine must be serviced over its lifetime. We had no initial idea of what a good number of swaps would be for each problem size, so we developed a baseline to test and see if we need to increase or decrease the number of swaps allowed per chromosome representation.

The last goal is to determine if the data shows some trends that can be analyzed further. For example, is it better to make large populations with a smaller number of generations or is it better to have smaller populations but more generations? With larger population sizes, we expect to get more exploration of the search space, while with more generations, we expect more exploitation of the good fitness values.

For these initial data results we looked at two instances: the five engine and ten engine scheduling problem. While the five engine problem may be a little small to be of practical use in the real world, the ten engine one is more in line with reality. Table 1 lists the testing parameters that we used.

Table 1. Testing Parameters

Number of engines scheduled	Generation Size	Population size	Number of engines scheduled	Generation Size	Population size
5	10	10	10	10	10
5	10	100	10	10	100
5	10	1000	10	10	1000
5	25	25	10	25	25
5	100	10	10	100	10
5	100	100	10	100	100
5	100	500	10	500	100
5	100	1000	10	100	1000
5	1000	1000	10	1000	1000

The main metrics used are the average number of component swaps used for nondominated members and the Overall Nondominated Vector Generation (ONVG) metric [8, 15]. The average number of component swaps is used to determine if our maximum number of swaps is a limiting factor or if we are increasing our search space with too many swaps. This metric is different from the aggregate swap count fitness function. The metric pertains to the number of swaps utilized by the chromosome representation and the fitness function takes into account how many times an engine would have to be serviced because of varying MTTRs on the components. Our target is to have over half of the swaps utilized by the nondominated members, while at the same time have less than the maximum used.

The ONVG metric is one that we use with caution. We are using this metric to see which runs create the most number of nondominated points. But at the same time, we are also keeping an eye on the values of the these nondominated members. We must do this in order to avoid the situation where one instance generates a lot of nondominated points and another generates a fraction of the first instance, but has many points that dominate the first. This can give us contrary results. This is due to the fact that the ONVG is more of a diversity measure and less of a convergence measure.

Each iteration of the experiment was run 30 times in order for the central limit theorem to apply. This gives our data an approximately normal distribution [16].

6 Analysis

Tables 2 and 3 list the results from our five engine and ten engine experiments.

Table 2. Testing Results for 5 Engines

Number of engines	Generation size	Population Size	Avg/Std Dev of Component Swaps	Avg/Std Dev of ONVG	Avg/Std Dev Makespan
5	10	10	2.04 / 0.93	2.63 / 1.73	971.5 / 39.4
5	10	100	3.43 / 0.93	4.3 / 1.53	941.5 / 78.2
5	10	1000	3.61 / 1.10	4.37 / 1.69	921.0 / 88.3
5	25	25	2.28 / 1.04	3.93 / 1.78	932.8 / 71.5
5	100	10	1.57 / 0.85	4.1 / 2.43	944.6 / 77.22
5	100	100	2.80 / 1.16	4.67 / 2.38	904.3 / 79.3
5	100	500	3.12 / 0.80	4.0 / 1.82	885.9 / 66.6
5	100	1000	3.54 / 0.92	4.4 / 2.46	889.5 / 67.5
5	500	1000	3.36 / 1.03	7.63 / 5.80	918.0 / 76.1
5	1000	1000	2.94 / 0.75	9.3 / 6.43	886.5 / 68.74

Table 3. Testing Results for 10 Engines

Number of engines	Generation size	Population Size	Avg/Std Dev of Component Swaps	Avg/Std Dev of ONVG	Avg/Std Dev Makespan
10	10	10	3.51 / 2.57	4.7 / 3.13	2165.5 / 83.1
10	10	100	5.6 / 2.72	4.33 / 2.20	2148.1 / 50.2
10	10	1000	7.32 / 1.74	3.8 / 1.16	2144.9 / 56.4
10	25	25	3.55 / 2.00	3.77 / 1.63	2142.2 / 43.5
10	100	10	3.72 / 1.79	6.13 / 4.51	2139.0 / 29.1
10	100	100	4.01 / 1.97	4.57 / 2.37	2119.1 / 32.7
10	100	500	4.03 / 2.30	5.29 / 4.11	2128.8 / 36.0
10	100	1000	4.45 / 1.98	4.33 / 1.97	2133.8 / 53.5
10	1000	1000	4.16 / 2.08	15.43 / 21.73	2105.5 / 4.53

First, let's look at the average number of swaps. This value was derived from the swaps done by the members on the Pareto front. The analysis of the average number of swaps shows that the number of swaps for the five engine problem is about right. Of the five swaps allocated, 7 out of 10 instances utilized more

than 50% of them. But with respect to the ten engine problem, only one of nine surpassed the threshold. This indicates that our baseline number of swaps is adequate for the 5 engine problem, but for 10 engines it may be limiting the efficiency of our system. This means our linear swap scaling factor of 1:1 was too much. A more appropriate method would be to scale by a factor of one-half. Better still, would be to implement a variable scaling factor that increases the number of swaps in a chromosome when a certain threshold is reached, or decreases the number when a declining threshold is reached. This would make our chromosome lengths vary while the program is running.

Looking at the average ONVG, there really is no trend that one can decipher between the various instances, with the exception of the 10 engine, 1000 generation instance. That instance has a very high mean with an extremely high standard deviation. This is because of several runs that approached 70 members on the Pareto front. Most of these members had the same fitness values, but had different schedules.

As for the makespan, the trend is as one would expect, the average makespan of the members tends decrease as the number of generations and/or number of population members is increased. It is also interesting to note that the standard deviation tends to creep higher as the number of generations is increased. This can be expected to happen in a multi-objective problem. As population members spread out along the Pareto front, the more diverse the population becomes and therefore the standard deviation increases.

Figure 3 shows a comparison of the Pareto front members of the instances at the two ends of the spectrum. One instance was generated using 1000 population members and 1000 generations, while the other used just 10 population members and 10 generations. Note that there are many duplicate members on the Pareto front for both instances. Each instance has more than 30 nondominated points, but no instance has more than five glyphs on the graph. This illustrates that many different schedules have the same fitness values.

7 Conclusion

Our research confirms that an aircraft engine maintenance scheduling problem can be effectively solved with an MOEA and in particular our GENMOP algorithm. Our chromosome representations permits the handling of swaps effectively and thus GENMOP tends to converge toward an acceptable Pareto Front. We also found that the standard deviation of the makespan tends to increase as the number of generations increases.

With a validated model and MOEA there are other avenues that can be investigated. The problem domain can be integrated with other MOEA algorithms for performance comparisons. Higher engine numbers should be addressed. The current instances at higher population rates and generations could be studied in an effort to determine where the efficiency of the algorithm decreases, with respect to obtaining new members of the Pareto front. Moreover, changing the chromosome representation may improve search effectiveness. Currently the last

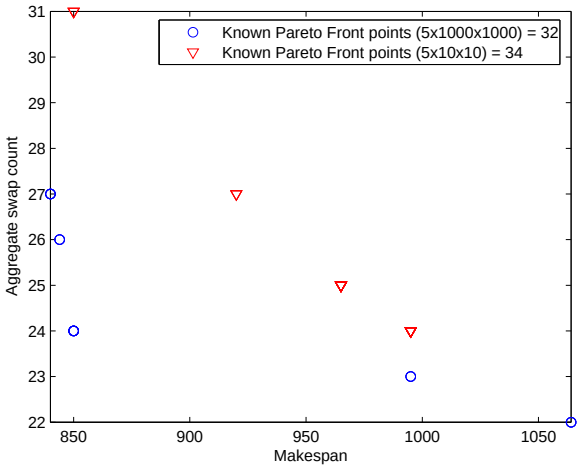

Fig. 3. Comparison of the Pareto front members for the 5 engine instance

3/4 of our chromosome is for the engine swaps. We would like to reserve five alleles for the swaps and let each one represent a component. Then the values could be implemented for the alleles in a linked list. We are currently trying to determine if the whole chromosome should be a linked list, or if multiple data types in the chromosome are better. The big stumbling block for this is the operator pool. We would have to generate special operators that can handle linked lists of data structures instead of real numbers. Additionally, we need to analyze the utility of the four crossover and three mutation operators. Determining which ones are effective and which ones are not in order to improve the algorithm either by limiting the operator pool to a fewer number of operators, or modifying less effective operators in order to generate better results. Currently, better heuristic crossover algorithms are being developed for this type of problem.

Overall, the generation of "good" aircraft maintenance schedules was a success. We were able to effectively create a scheduling system that takes into account much of the real world problem characteristics and variables as well as to model it in an MOEA with promising results. The future work on this problem can give us further insight into the algorithm domain and make this an effective tool that the Air Force can use in its logistic centers.

The authors wish to express their gratitude to Dr. Gregg Gunsch and Brian Beachkofski of the Air Force Research Laboratory, who first brought this problem to our attention.

References

1. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, Englewood Clifs, New Jersey (1995)
2. Dues, T.: Quality Engine Development and Sustainment. In: Proceedings of the 2001 Defense Manufacturing Conference, Oklahoma City Air Logistics Center, Defense Manufacturing Conferences (2001)
3. Donald, E.M.: Reliability Centered Maintenance. Flight Safety Information Special Issue **32** (2003) 107–122
4. Bagchi, T.P.: Multiobjective Scheduling by Genetic Algorithms. Kluwer, Boston, MA (1999)
5. Holsapple, C.W., Jacob, V.S., Pakath, R., Zaveri, J.S.: A Genetics-based Hybrid Scheduler for Generating Static Schedules in Flexible Manufacturing Contexts. IEEE Transactions on Systems, Man and Cybernetics **23** (1993) 953–972
6. Bäck, T.A.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York - Oxford (1996)
7. Bäck, T., Fogel, D.B., Michalewicz, Z., eds.: Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing, Bristol (2000) Contains excerpts from the Handbook of Evolutionary Computation.
8. Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, New York (2002) ISBN 0-3064-6762-3.
9. Michalewicz, Z., Janikow, C.Z.: Genocop: a genetic algorithm for numerical optimization problems with linear constraints. Commun. ACM **39** (1996) 223–240
10. Michalewicz, Z.: Evolutionary computation techniques for nonlinear programming problems. International Transactions in Operational Research **1** (1994) 175
11. Knarr, M.R.: Optimizing an In Situ Bioremediation Technology to Manage Perchlorate-Contaminated Groundwater. Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH (2003)
12. Knarr, M.R., Goltz, M.N., Lamont, G.B., Huang, J.: *In Situ* Bioremediation of Perchlorate-Contaminated Groundwater using a Multi-Objective Parallel Evolutionary Algorithm. In: Congress on Evolutionary Computation (CEC'2003). Volume 1., Piscataway, New Jersey, IEEE Service Center (2003) 1604–1611
13. Keller, T.A.: Optimization of a Quantum Cascade Laser Operating in the Terahertz Frequency Range Using a Multiobjective Evolutionary Algorithm. Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH (2004)
14. Keller, T.A., Lamont, G.B.: Optimization of a Quantum Cascade Laser Operating in the Terahertz Frequency Range Using a Multiobjective Evolutionary Algorithm. In: 17th International Conference on Multiple Criteria Decision Making (MCDM 2004). Volume 1. (2004)
15. Schott, J.R.: Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts (1995)
16. Milton, J.S., Arnold, J.C.: Introduction to Probability and Statistics: Principles and Applications for Engineering and Computer Science. Third edition edn. McGraw Hill (2002)