Searching for Multiobjective Preventive Maintenance Schedules: Combining Preferences with Evolutionary Algorithms

Gang Quan* Garrison W. Greenwood† Donglin Liu‡ Sharon Hu ‡

Abstract

Heavy industry maintenance facilities at aircraft service centers or railroad yards must contend with scheduling preventive maintenance tasks to ensure critical equipment remains available. The workforce that performs these tasks are often high-paid, which means the task scheduling should minimize worker idle time. Idle time can always be minimized by reducing the workforce. However, all preventive maintenance tasks should be completed as quickly as possible to make equipment available. This means the completion time should be also minimized. Unfortunately, a small workforce cannot complete many maintenance tasks per hour. Hence, there is a tradeoff: should the workforce be small to reduce idle time or should it be large so more maintenance can be performed each hour? A cost effective schedule should strike some balance between a minimum schedule and a minimum size workforce.

This paper uses evolutionary algorithms to solve this multiobjective problem. However, rather than conducting a conventional dominance-based Pareto search, we introduce a form of utility theory to find Pareto optimal solutions. The advantage of this method is the user can target specific subsets of the Pareto front by merely ranking a small set of initial solutions. A large example problem is used to demonstrate our method.

Keywords: Evolutionary computations, Scheduling, Utility theory, Preventive Maintenance, Multi-objective optimization.

*The author is with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208, USA. gquan@cse.sc.edu

†The author is with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207, USA. greenwd@ee.pdx.edu

‡The authors are with the Department of Computer Science and Engineering, University of Notre Dame. Notre Dame, IN 46556, USA. {dliu, shu}@cse.nd.edu

1 Introduction

All systems eventually fail. It therefore becomes imperative that any system downtime resulting from these failures be kept to an absolute minimum if a company is to remain competitive. However, just reducing the repair time is not sufficient because that policy is strictly reactive—i.e., actions only take place after the failure happens. An effective strategy should also take steps to reduce the probability of failures from occurring in the first place.

Preventive maintenance (PM) is the name given to those tasks which are performed to keep a fully operational system running. PM tasks do involve maintenance activities, but they are not performed because something is not working. Rather, they are performed before a failure is likely to occur. A good example is the automobile manufacturer's recommendation to change the engine oil every 3,000 miles. The oil is changed not because something has failed; it is changed to ensure the engine is adequately lubricated to reduce wear and tear. The exact nature of the PM tasks depends on the system under consideration, but typical tasks include replacing worn parts, changing lubricants, running diagnostics and re-calibrating adjustable subsystems. It has been known for some time that a comprehensive PM program can effectively improve system availability [18, 24]. But that is not the only reason PM should be performed. In June 2000, the U.S. Federal Aviation Administration threatened to suspend a portion of Alaska Airlines operations because of several crashes attributed to poor maintenance [30].

PM tasks are labor intensive and the labor pool that performs those tasks is highly skilled. Labor costs can therefore be quite high. Therein lies a dilemma: a small labor force would help control costs, but a small labor force cannot perform many PM tasks per hour—and equipment that isn't available doesn't generate any revenue. A long completion time is not cost effective but neither is having too many idle workers. A proper balance would minimize labor costs while simultaneously finishing all PM tasks in a timely manner. In other words, a tradeoff must be made between the number of workers and a timely completion of all PM tasks.

The PM scheduling problem is NP-hard, which means finding the optimal schedule—i.e., minimum time with a minimum size labor pool—is not easy. This scheduling problem has been extensively studied and the interested reader is directed to the extensive literature review provided in [3]. That publication is also relevant to this work because it demonstrated how an *evolutionary algorithm* (EA) can effectively search for optimal PM schedules. However, in that work the scheduling problem was constructed as a constrained

optimization problem where the number of workers was fixed and the objective was to find the best PM schedule subject to the number of available workers.

In this paper we formulate the problem as a *multiple objective problem* (MOP) for one very important reason: while a maintenance facility may know the nature and number of PM tasks that must be done, in all likelihood it will not know how many maintenance workers to hire. Hiring too few makes it difficult to complete all PM tasks on time, whereas hiring too many results in excessive worker idle time and excessive labor costs. When evaluating the fitness of a PM schedule under the above two conflicted objectives, a nature and commonly adopted strategy is to combine all the objectives using an aggregating formula such as a weighted-sum. The MOP is thus reduced to a single objective optimization problem [6, 14, 22]. However, assigning meaningful weights requires that we know, to some extent, the behaviors of each objective functions, which can be practically expensive if not totally impossible. To overcome the difficulty involved in the aggregating approach, another most popular strategy is to apply the *Pareto-optimal* approach [11, 5]. This approach intends to identify the set of solutions that are not *dominated* by others [2, 26, 21]. Almost all Pareto-optimal searches use dominance to pick better solutions. Note that this approach assumes that all the non-dominated solutions are equally important, which is often not the case. In contrast, our EA searches for Pareto optimal solutions to the PM scheduling MOP, but we use preferences among solutions rather than dominance to guide the search. We will show preferences renders solutions more in line with a manager's expectations later.

The paper is organized as follows. Section 2 formally defines the PM scheduling problem, explains why the problem is of interest, and discusses previous PM scheduling work. Section 3 introduces evolutionary algorithms, but specific details about our evolutionary algorithm are deferred until Section 5. Section 4 first describes the differences between a dominance-based and a preference-based search and then tells how to construct a preference-based search. Experimental results are presented and discussed in Section 6. Finally, Section 7 mentions some future work.

2 PM Problem Description

A comprehensive PM program identifies what maintenance tasks are performed on each item of equipment, what resources are required to do each maintenance task, and how often each task is scheduled. An effective PM program is absolutely essential to keeping equipment available.

The United States Army is the best example of an organization that relies heavily on an effective PM program to keep its equipment operational. All rules, statues and ordinances in the Army are contained in a series of Army Regulations (ARs). AR 750-1 covers the maintenance of supplies and equipment in the United States Army [25]. Army maintenance is founded on the principle that the useful service life of Army equipment is achieved when the item is operated within its intended purpose and parameters and is maintained in accordance with its designed or engineered specifications. AR 750-1 puts the importance of an effective PM program this way:

> *Performance observation is the foundation of the Army maintenance program. Performance observation is the basis of the preventive maintenance checks and services known as PMCS that are required by all equipment technical manuals in the before, during and after operation checks. . . . [The] technical manuals designate the standards for all equipment. This allows leaders the ability to designate the time and location of repair that saves precious manpower and materiel resources. It is also the most effective method of managing a large fleet of equipment when time and labor are limited and distances between support and the supported equipment are great.*

A PM program can only be effective if PM tasks are scheduled at regular intervals. Each task requires certain resources such as expendable supplies (oil, solvents etc.), repair parts, and workers with specific skills. If one assumes the expendable supplies and repair parts are on-hand, then the start time and completion time of a task depends on the availability of the workers. Any system undergoing PM is not available for operation—which means it isn't making any money for the company. It therefore becomes imperative that tasks be started on time and their completion time be minimized.

So exactly how does a manager complete all PM tasks in minimum time? The obvious way to do this is to hire enough workers so that all PM tasks can be started concurrently. Unfortunately, this is not always good management practice. For example, suppose there are ten PM tasks, each requiring five workers. Task 1 only takes one hour while tasks 2-9 take four hours. If all tasks are started concurrently, then in four hours all PM tasks are finished. But after one hour five (highly-paid) workers are idle, which is not cost effective. A better solution would be to schedule tasks 1-9 and defer task 10 for one hour. As soon as task 1 is finished, those workers can be reassigned to task 10. This new schedule does add one hour to the overall completion time, but it eliminates worker idle time thereby saving on labor costs.

The PM scheduling MOP we investigate has two attributes: number of workers and PM task completion time. The objective is to find a PM schedule that minimizes both attributes. A straightforward way of evaluating a potential solution is to form a weighted sum of the attributes, but it is not always easy to assign meaningful weights. We describe a new method based on *incompletely specified multiple attribute utility theory* (ISMAUT) that eliminates the need to precisely specify weights. This method has been shown to efficiently target a subset of Pareto optimal solutions based on a user's preferences [12].

We are now ready to formally define the PM problem. The notation shown below is the same as was used in [3].

N	=	number of tasks
TS	=	total number of skills available in the workforce
S	=	set of skills (say, with TS=2, a = mechanic, b= electrician)
W_a	=	number of workers with skill a
W_b	=	number of workers with skill b
W	=	total number of workers (i.e., $W_a + W_b$)
SPT	=	maximum number of skills required per task

We assume each worker can perform only one skill. A given task may not start until at least one worker in each required skill is available and, once assigned, a worker may not be re-assigned to another task until the current task is completed. For simplicity, all workers begin a task at the same time and are re-assigned as soon as they are finished. For example, suppose a task starts at time 10 and it requires one worker with skill a for 2 hours and one worker with skill b for 4 hours. Then both workers start at time 10 but the worker with skill a can be reassigned at time 12. Of course it is possible to do multiple PM tasks at the same time so long as workers are available.

Makespan (MS) refers to the total amount of time it takes to complete all N tasks—i.e., the schedule length. Clearly makespan is minimized if the required number of workers in each skill category are always available for assignment. But, that optimal situation is unlikely to occur for two reasons: (1) workers are not always available due to unforeseen circumstances such as injury or illness, and (2) it is not cost effective to have highly paid workers sitting around idle.

It is now possible to define the two attributes for a set of PM tasks. One attribute is the makespan (which is to be minimized) and the other attribute is the workforce (which is also to be minimized). Notice that

these two attributes are conflicting because minimizing the workforce increases the makespan.

Problem Definition

Given a set of PM tasks $\{T_1, T_2, \ldots, T_N\}$, find a solution that simultaneously minimizes W and MS.

Before discussing specific details about our method, it is worthwhile to briefly highlight how it differs from previous work. Although others have used EAs to solve PM scheduling problems, multiobjective EA versions for this problem are rare—even though the PM scheduling problem is best expressed as a MOP. Our EA-based approach is designed to solve a PM scheduling MOP, but we use preferences rather than conventional dominance to guide the search. (The difference between these two type of searches is described in Section 4.1.)

Cavory et. al [6] used a genetic algorithm to schedule PM tasks on a single product manufacturing line. The tasks execute cyclically and each task could occur several times during a production run. But the problem was not formulated as a MOP. The manufacturing line had multiple machines and each machine had several PM tasks. However, all PM tasks were done by a single operator, and that operator was always available. The only parameter to optimize was the elapsed time before a PM task first executed. (Subsequent executions take place at fixed time intervals set by the defined cycle time.) Hence, this problem has only a single objective: find the set of first start times that yields the minimum makespan.

Tsai et. al [27] splits PM tasks into two categories: 1P simple preventive maintenance actions (denoted by 1P) and preventive replacement actions (denoted by 2P). 1P actions improve the reliability to some small degree whereas 2P actions improve the reliability to that of a brand new system. Dynamic reliability equations are used to model degraded component behaviors. They used a genetic algorithm to determine which type of PM task should be applied to a degrading component. For example, the individual 1021001221 says a 1P task should be applied to the first component, the second component needs no PM task, a 2P task should be applied to the third component and so on. Each assignment affects the system reliability but it also affects the maintenance cost (1P tasks cost less than 2P tasks). Note the purpose of the genetic algorithm is not to find the best PM task schedule, but rather to choose the type of PM tasks that improves reliability while at the same time minimizes the operational costs. This is a single objective optimization problem.

Of course not all PM scheduling work relies on evolutionary algorithms. Percy and Kobbacy [23], for

example, used a suite of mathematical models and simulations. However, instead of scheduling a defined set of tasks, they tried to find the optimal PM interval duration. Specially designed probability density functions, derived from historical data, were used to represent the lifetime of the system following PM activities. The objective was to find the shortest interval duration that would minimize operating costs per unit time. One real problem with this approach, and other approach which uses mathematical models, is it requires accurate failure data to properly fine-tune the probability density functions. It is also worth noting that merely specifying the PM interval duration only solves half the problem—it doesn't tell which specific PM tasks should be executed during that time interval.

One example of previous work where a genetic algorithm was used to solve a PM scheduling MOP is the work by Marseguerra et. al [20]. They used a genetic algorithm combined with a Monte Carlo simulation, to find an optimal task schedule. However, this schedule was found implicitly because they considered a different trigger event to start executing a PM task. Usually PM tasks take place at regular intervals of some parameter such as time or mileage or units produced by a machine on a manufacturing line. The goal is to find the best start time for each PM task subject to some resource constraints. In the Marseguerra et. al work they wanted to schedule condition-based maintenance tasks. These are PM tasks, but instead of scheduling them at regular intervals, they are not scheduled until the system condition deteriorates below some defined threshold. Their goal was to use an evolutionary algorithm and Monte Carlo simulation to find the best threshold levels. A genetic algorithm evolved a population of individuals, each encoding threshold degradation values, while a Monte Carlo simulation estimated the system mean availability and net profit. The problem was framed as a multi-objective search aimed at simultaneously maximizing profit and availability. Since this was a dominance-based Pareto optimal search, it was not possible to incorporate user preferences as does our method.

3 Evolutionary Algorithm Overview

Evolutionary algorithms (EA) perform searches through high-dimensional, multi-modal solution spaces by emulating biological principles of adaptation found in nature. They have been effectively used to solve a wide variety of NP-hard optimization problems, e.g. [2, 1, 8, 21]. This section provides only a brief overview. A number of excellent books on the theory and design of EAs are available. We recommend the recent book written by Eiben and Smith [7].

Procedure EA
    Randomly generate initial population of solutions
    Evaluate population
    While stopping criteria not met **do**
        Select parents for reproduction
        Apply genetic operators to produce offspring
        Evaluate offspring
    End while
End EA

Figure 1: A Canonical EA for MOPs

EAs follow the neo-Darwinian philosophy which says stochastic processes such as reproduction and selection, acting on species, is responsible for the present the life forms we know. In basic terms natural evolution describes how a population of individuals strives for survival. During reproduction genetic material from each parent creates an offspring. Each individual has an associated fitness that ultimately determines the survival probability. Highly fit individuals have a high probability of surviving to reproduce in future generations.

Figure 1 shows a canonical form of an EA. Each individual in an EA's population is a unique solution to the optimization problem of interest. A population of individuals is therefore a set of possible solutions and the fitness of a solution measures its quality. The initial population is randomly generated. In each subsequent generation (iteration) current solutions (parents) are stochastically altered (via genetic operators) to render new solutions (offspring). Each solution is evaluated to determine its worth and a deterministic selection procedure culls the best solutions and discards the rest[1]. Done properly, the algorithm will quickly abandon regions of the solution space that contain poor solutions and focus on those regions with good quality solutions. The stopping criteria can be either an acceptable solution has been found or a fixed number of generations have been produced.

The quality of a solution is denoted by its *fitness*. That is, highly fit solutions are desirable. The EA uses these fitness values to decide which solutions to keep for the next generation. Of course the definition of fitness depends on the type of optimization problem. For example, in the PM scheduling problem highly fit solutions have both a low makespan and a low number of required workers.

[1]Evolutionary algorithms can use a variety of selection methods [13]. For example, genetic algorithms typically use a stochastic selection method where better quality solutions are selected with higher probability. Others, such as evolution strategies, use strictly deterministic selection where individuals are sorted by quality. The evolutionary algorithm used in this work is patterned after the evolution strategy.

A simple example, although not related to PM scheduling, will help illustrate how an EA works. Suppose we want to maximize a given function $f(\vec{x})$ where $\vec{x} \in \Re^n$ and each component $x_k \in [-5, 5]$ for all $k = 1, 2, \ldots n$. Each individual in our EA would be an $n \times 1$ real vector. We can easily create the initial population of say 100 such vectors by randomly picking a number, using a uniform distribution, between -5 and 5 for every vector component. The offspring would be created by copying each member of the current population and then randomly perturbing those copies. For example, we could mutate each component of a vector by adding to it a small, Normally-distributed random variable. The 100 original parents and the 100 offspring could then be combined into a single large population. The fitness of individual $\vec{x}_k$ is given by $f(\vec{x}_k)$ where higher values denote higher fitness. After sorting the population according to fitness, the top 100 individuals are kept and the rest are discarded. This is one generation. This process repeats until the termination criteria is met. Typical termination criteria are (1) a fixed number of generations have been processed, (2) a sufficiently good enough solution has been found, or (3) the algorithm has converged (i.e., no improvement in fitness has occurred over a specified number of generations).

The next section describes how the basic EA paradigm is modified to search for good solutions to arbitrary MOPs. In Section 5 we provide detailed information about the EA we designed specifically to solve PM scheduling MOPs.

4 Preference Based Searches

4.1 Dominance vs Preference

The definitions and concepts presented in this section are used in later sections for discussion of fitness functions for MOPs. More detailed information can be found in a number of sources [17, 19].

Optimization problems are essentially search problems. Each individual in the population of an EA constitutes a potential solution or *alternative* within the problem space of an optimization problem. *Objectives* define desirable properties of a good alternative and *attributes* are used to determine the degree to which a specific objective is met. An objective is normally formulated as an *objective function* with the attributes as the function's arguments. In the terminology of EAs, attribute levels are used to quantify the fitness of an alternative and the objective function is equivalent to a fitness function. The goal of the EA is to find an alternative which optimizes the fitness.

Suppose we are given a MOP with $\mathcal{X}$ representing the set of all feasible alternatives. Further, let $\mathcal{A}$ represent the set of n attributes. Each alternative $x \in \mathcal{X}$ has an assigned level for each $a_i \in \mathcal{A}$. We let $\mathcal{A}_x$ denote the set of attribute levels associated with the alternative x. Our goal is to represent the fitness of x by defining an appropriate objective function $f : \mathcal{A}_x \rightarrow \Re_+^0$ where $\Re_+^0 \in [0, \infty)$.

Let x and x' be two alternatives from $\mathcal{X}$ with their associated attribute level sets $\mathcal{A}_x = \{a_1, ..., a_n\}$ and $\mathcal{A}_{x'} = \{a'_1, ..., a'_n\}$, respectively. We say x *dominates* x' if $\forall i$ a_i is better than or equal to a'_i and, in addition, there exists at least one a_j such that a_j is strictly better than a'_j. Let $x \succ_D x'$ denote that x dominates x'. The set of non-dominated alternatives lies on a surface in attribute space known as the *Pareto optimal frontier*[2]. In each generation of an EA there exists a set of non-dominated alternatives.

Two important aspects of dominance are worth noting. First, dominance applies only to the *ordinal* characteristics of a_j and a'_j and not to their *cardinal* characteristics[3]. Second, dominance does not require that a_i be compared against a'_j for $i \neq j$.

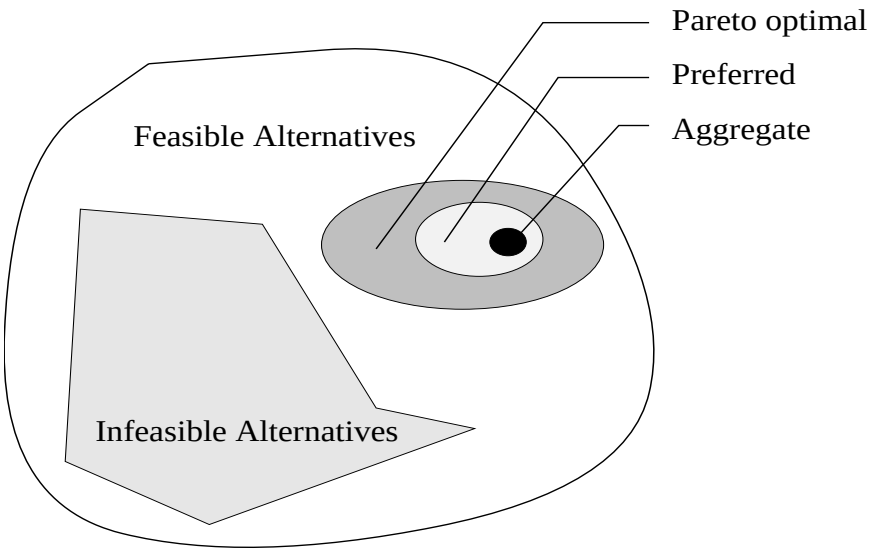


Figure 2: A picture of alternative space. Each set may be the union of a number of disjoint subsets of the same type.

Given two non-dominated alternatives, a decision maker may still prefer one over the other. This concept is expressed with the following two relationships:

R1: $x \succ x'$ (read as " x is preferred-to x' ")

[2]Some authors refer to this as the Pareto optimal set.

[3]In other words, one only needs to know that the attribute level of a_j is greater than that of a'_j. How much greater is of no importance in determining dominance.

R2: $x \sim x'$ (read as " x is indifferent-to x' ")

x and x' are indifferent when $x \not\succ x'$ and $x' \not\succ x$ indicating that there is no clear preference between them. Relationships **R1** and **R2** together establish a *partial order* on $\mathcal{X}$. If relationship **R2** does not exist (*i.e.*, $\forall\ x, x' \in \mathcal{X}$, either $x \succ x'$ or $x' \succ x$), then a *total order* on $\mathcal{X}$ is established. Preference is purely subjective and thus is different from dominance (which is purely objective). In a later section we will show how dominance and preference are related.

Figure 2 depicts alternative space which contains all possible alternatives. Only a subset of the feasible alternatives are Pareto optimal. A subset of the Pareto optimal alternatives are preferred since they coincide with the decision maker's preferences. This preferred subset has a "fuzzy boundary" since the attribute weights are imprecisely specified. (This aspect will be discussed further in the next section.) The smallest subset of the Pareto optimal alternatives is the aggregate subset which can only be identified if the attribute weights are precisely specified. Our objective is to identify alternatives from within the preferred subset.

4.2 Evaluating Alternatives

Evaluating alternatives requires the resolution of conflicting objectives. These conflicts arise because of the physical relationships among objectives as well as resource limitation. For example, reducing the production cost of a system may adversely affect its performance. Therefore, a decision maker must identify a set of measurable attributes for evaluating alternatives, and then apply a consistent set of preferences that quantify how the tradeoffs among the attributes are to be made.

Our problem is thus the one of choosing an appropriate function format so that tradeoffs between alternatives can be represented as a measure of fitness. An intuitive format for this fitness function may be a weighted sum

$$f = \sum_k w_k\ a_k \tag{1}$$

where a_k is the k-th attribute and $w_k > 0$ is its associated weight. (Higher weight values reflect greater importance.) The weights must satisfy $\sum_k w_k = 1$. Unfortunately, this format for a fitness function is a bit naive and has a number of problems. Specifically,

1. Attributes are usually expressed in different units of measure which makes direct summation not possible.

2. There is no pedantic method for specifying the attribute weights.

3. There is no convenient way of capturing any decision maker's preferences between alternatives.

We replace the objective function of Equation (1) with an *imprecise value function*, which does not require direct specification of the attribute weights [28]. In this section, we describe how the imprecise value function is created and applied to evaluate alternatives.

A scaling of the attribute levels may be necessary whenever fitness sharing is used to prevent genetic drift. This scaling process maps each raw attribute level to a convenient subset of $\Re_+^0$ (normally $0 \rightarrow 1$). As before, let $\mathcal{A}_x = \{a_1, a_2, \cdots, a_n\}$ represent the set of n attributes associated with the alternative x. Suppose there exists a set of real-valued functions $\{v_1, v_2, ..., v_n\}$ on $\mathcal{A}$ such that $v_i : a_i \rightarrow [0, 1]$ where $v_i \rightarrow 1$ as the attribute level of a_i improves[4]. The set of real-valued functions are referred to as *attribute value functions*.

These functions should (as the attribute value increases) monotonically increase for a "more-is-better" attribute and monotonically decrease for a "less-is-better" attribute. Let $\tilde{a}_i$ and $\hat{a}_i$ denote the maximum and minimum levels, respectively, of the attribute a_i. In our work we chose the "more-is-better" attribute value function to be

$$v_i(a_i) = \frac{a_i - \hat{a}_i}{\tilde{a}_i - \hat{a}_i} \tag{2}$$

and for the "less-is-better" attribute value function

$$v_i(a_i) = \frac{a_i - \tilde{a}_i}{\hat{a}_i - \tilde{a}_i} \tag{3}$$

Note that $v_i(\cdot)$ is not restricted to a linear form as shown in Equations (2) and (3). Indeed, any arbitrary nonlinear function can be used as long as it satisfies the monotonicity requirements.

An imprecise multi-attribute value function corresponding to the alternative x has the following form:

$$V_x = \sum_k w_k \; v_k(a_k) \tag{4}$$

where a strictly positive w_k is the weight and $v_k(a_k)$ is the attribute value function for attribute a_k. All weights must satisfy

$$\sum_k w_k = 1 \; ; \; w_k > 0 \tag{5}$$

[4]Some authors in this context will refer to v_i as a "utility function".

V_x is imprecise in the sense that each w_k does not have a specific assignment, but is constrained by preferences among attributes. Such constraints can be formulated based upon preferences between distinct alternatives. (These alternatives can be provided by the decision maker or generated by an EA.) For example, let $x, x' \in \mathcal{X}$ be two alternatives with corresponding attribute level sets $\mathcal{A}_x$ and $\mathcal{A}_{x'}$ for which the decision maker has decided that $x \succ x'$. By definition,

$$x \succ x' \implies V_x > V_{x'} \Longrightarrow \sum_{k=1}^{n} w_k[v_k(a_k) - v_k(a'_k)] > 0. \tag{6}$$

Such an expression defines a constraint for the attribute weights. When several alternative pairs are ranked by the decision maker, a series of such constraints are defined. The set of all such constraints confines the w_k's to a subspace $W \subset \Re_+^n$ where $\Re_+^n$ is the n-dimensional space of positive real numbers.

Using the attribute value functions and the constraint subspace W, other configurations created from running an EA may be evaluated. More specifically, by definition

$$V_x - V_{x'} = \sum_{k} w_k[v_k(a_k) - v_k(a'_k)] > 0 \Longrightarrow x \succ x' \tag{7}$$

It follows that alternatives x'' and x can be compared by solving the following linear programming problem:

$$\begin{array}{ll} \text{Minimize (w.r.t. } w_k\text{):} & \sum_k w_k[v_k(a''_k) - v_k(a_k)] \\ \\ \text{Subject to:} & w_k \in W \end{array} \tag{8}$$

Then x'' is preferred to x if Equation (9) is true.

$$z = \min_{w_k} \sum_{k} w_k[v_k(a''_k) \ - \ v_k(a_k)] > 0 \tag{9}$$

However, knowing that $z \leq 0$ is not sufficient to determine preference. We must reverse the terms in Equation (9) as shown below.

$$\overline{z} = \min_{w_k} \sum_{k} w_k[v_k(a_k) \ - \ v_k(a''_k)] > 0 \tag{10}$$

Now, if Equation (9) is false and Equation (10) is true, then $x \succ x''$. If both equations are false, then x and

x'' are pairwise indifferent.

It is important to emphasize that the ranking of the selected alternatives is done merely to obtain the constraint subspace W. W is then used in the series of linear programming problems that must be solved to conduct pairwise comparisons between alternatives. Note that the ranking of the selected alternatives may lead to conflicting constraints for the resulting linear programming problem if there are too many alternatives and the ranking is not done in a consistent manner. In such a case the linear programming instance would be infeasible. To avoid such a problem, the alternatives should be selected carefully so that they can be ranked consistently. In practice the ranking is normally done by experts so this type of problem rarely happens.

Table 1: Feasible Alternative of Example Design Problem

Alternatives	**Attributes**			**Preference**
	$v_1(a_1)$	$v_2(a_2)$	$v_3(a_3)$	
x_1	0.75	1.0	0.4	2
x_2	0.5	0.0	0.8	1
x_3	0.0	1.0	1.0	-
x_4	1.0	0.0	0.0	-

To illustrate some of the above concepts, consider the following example problem. Table 1 shows four alternatives and their corresponding attribute levels. The first two of these were randomly generated and ranked by the decision maker (x_2 is the most preferred). Alternatives x_3 and x_4 were identified by the optimization process (*e.g.*, using an EA). Since $x_2 \succ x_1$, substituting into Equation (6) yields

$$w_1[0.5 - 0.75] \ + \ w_2[0.0 - 1.0] \ + \ w_3[0.8 - 0.4] \ > 0$$

Or,

$$-0.25w_1 \ - \ w_2 \ + \ 0.4w_3 \ > 0$$

This constraint (in conjunction with Equation (5)) completely defines the imprecise value function. Suppose now that we wish to determine if $x_2 \succ x_4$. This preference exists if the following equation is true:

$$\min_{w_k} \sum_k w_k[v_k(a_k^2) \ - \ v_k(a_k^4)] > 0 \tag{11}$$

where a_k^j denotes the k-th attribute for the j-th alternative. Solving the following linear program

Minimize (w.r.t. w_k): $\quad z = \sum_k w_k[v_k(a_k^2) - v_k(a_k^4)]$

Subject to:

$$-0.25w_1 \; - \; w_2 \; + \; 0.4w_3 \; > 0$$
$$\forall k, \; w_k > 0$$
$$\textstyle\sum_k \; w_k = 1$$

will determine the relationship between x_2 and x_4. It is easy to show that for the above example $z > 0$ and so $x_2 \succ x_4$. Similarly, one can verify that x_2 and x_3 are indifferent to each other.

We conclude this section with an important theorem that evinces the link between our precedence relationship ($\succ$) and the dominance relationship ($\succ_D$). Specifically, it proves that preference relationships will preserve existing dominance relationships.

Theorem: $x \succ x' \Rightarrow x' \not\succ_D x$.

Proof: (By Contradiction.) Let $x \succ x'$ and assume $x' \succ_D x$. By Equation (6), $V_x > V_{x'}$ which means there must exist at least one j such that $[v_j(a_j) - v_j(a'_j)] > 0$. But this means a_j is strictly better than a'_j which contradicts the assumption that $x' \succ_D x$. **Q.E.D.**

4.3 Representing Fitness in an EA

Each alternative explicitly specifies the attribute levels which can be mapped to [0,1] on the real number line using Equation (2) or (3) as appropriate. The evaluation of alternatives requires computing their fitness; alternatives with high fitness should, with high probability, survive and reproduce while alternatives with low fitness should die out.

We use the preference ordering discussed in the previous section to assign fitness to each alternative. Alternative x is said to have a higher fitness over alternative x' if $x \succ x'$. Equation (7) can be used to determine these preference relationships. However, this will typically establish only a partial order. What

is really needed is a total order so that the best fit alternatives can be identified and marked for survival. One possible way of getting a total order is to rank all of the alternatives with a technique described by Goldberg (see [11], pg 201). Assign rank 1 to all preferred alternatives and then remove them from further contention. A new set of preferred alternatives can then be found, ranked 2, and so on until all alternatives have been ranked. Survival can be determined using a variety of techniques such as truncation selection. Another method is to conduct a series of tournaments (see Section 5.4). It is important to note that a fitness assignment based upon preference relationships will preserve existing dominance relationships. It is also important to emphasize that the decision maker need only establish preference relationships for a small number of alternatives from the initial population[5].

Our use of imprecise value functions for establishing a survival criteria is what differentiates our approach from conventional Pareto optimal searches. Complete enumeration of the entire Pareto optimal set is not practical. Hence the goal in most other Pareto optimal approaches is to "sketch out" the set by finding a (hopefully) uniformly distributed subset of samples from it. While certainly laudable, this assumes that all Pareto optimal solutions are equally preferable which is often not the case. Using preferences to influence the probability of survival allows the decision maker to drive the search process of the EA. This helps to prevent the EA from conducting a simple blind exploration of the tradeoff space.

Independent of the survival criteria chosen, there is a non-zero probability that genetic drift will occur[6]. This genetic drift cause a loss in population diversity, which leads to premature convergence of the EA. Fitness sharing [9] and equivalence class sharing [16] are two methods which help to avoid this problem, although sharing does increase the computational effort. We were able to maintain population diversity without sharing by using a particular type of tournament ranking (see Section 6 for details).

5 Evolutionary Algorithm Design Details

5.1 Representing PM Tasks

There are N PM tasks indexed by $\{1, 2, \ldots, N\}$. Each task has a predefined set of skills needed. The first thing needed is some way of encoding the order of executing tasks. This can be done with an integer list

[5]Three or four alternatives is sufficient for most problems. Indeed, ranking more than this amount may prove to be difficult.

[6]Critical relationships between mutation rate, population size and selection pressure determine how significant this genetic drift will become [15].

where the left-to-right order indicates the order of executing tasks. An example for $N = 8$ is shown below.

7	2	1	4	6	3	8	5

Each integer list represents a candidate schedule. The makespan depends on the availability of workers with the appropriate skill set, but no task can start until all workers in each needed skill category are assigned. We assume that, once assigned, a worker may not be reassigned until the task is completed. Finally, we assume any required test equipment or special tools are also available before the PM task can start.

The above assumptions are not only reasonable, but they are also common practice. For example, some aircraft maintenance PM tasks cannot be physically accomplished unless two mechanics are present (either because of the complexity of the task or the manual labor involved). Safety concerns or union rules may also dictate a particular number of workers with specified skills be present before a task can begin. For these same reasons a PM task would have to be suspended if one of the workers has to leave before the task is completed. Of course PM tasks cannot start unless all of the necessary tools and equipment are also present, but this does not have to be explicitly considered when developing a PM schedule. (In practice a worker wouldn't be assigned anyway unless this were true.)

Let L be the set of workers available. The cardinality of L varies as PM tasks are executed because all personnel assigned to a task may not be needed for the entire task. For instance, a PM task may require a mechanic for 4 hours but an electrician may be needed for only one hour.

The makespan is computed as follows. The first task (from the left) of the integer list is chosen and workers with the needed skill are taken from L. If L is not empty, all remaining tasks (scanned from left-to-right) are assigned workers *but only to tasks that could start*. In other words, tasks cannot "hoard" resources; tasks may start if and only if all of the workers needed are currently available in L. Whenever a task releases a worker he is returned to L and the integer list is immediately scanned to see if another task can start. This process continues until all PM tasks are completed. Completing the last task determines the makespan. The lower the makespan, the higher the fitness of the solution.

This data structure is incomplete because it does not have any means of encoding the number of workers. We will shortly show how to modify the data structure to do this.

5.2 Operators for Perturbing a Task Schedule

The two reproduction operators used to produce offspring by perturbing a task schedule are called *insertion* and *inversion*. Both of these operators have been used to find solutions to the Traveling Salesman Problem [10]. They are classified as mutation operators because the reproduction method is asexual. The insertion operator randomly chooses a task and inserts it in another randomly chosen location in the integer list. With task 6 selected,

Before:

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

After:

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

The inversion operator randomly chooses two points in the integer list and reverses the order of the tasks between them (including the two end points). With task 2 and 8 being the two selected tasks,

Before:

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

After:

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

5.3 Overall Data Structure

The data structure for task scheduling must now be augmented to encode the number of workers. Assume there are two types of workers: "a" is an electrician and "b" is a mechanic. Let W_a and W_b denote the number of each type of worker. A solution to the PM task scheduling problem must contain not only a permutation of the N tasks, but also W_a and W_b. However, W_a and W_b are not directly stored in the data structure. The data structure now looks like this

7	2	1	4	6	3	8	5	x_a	σ_a	x_b	σ_b

where $x_a, x_b, \sigma_a, \sigma_b \in \Re$. For simplicity only the x_a mutation is shown, but x_b is done in a similar way. x_a

is perturbed by adding a random variable to it and then taking the integer ceiling of that sum to get W_a.

σ_a is a *strategy parameter*. Let $N(0,1)$ denote a zero mean, normally distributed random variable with standard deviation 1.0. Then $\sigma_a \cdot N(0,1)$ is a zero mean, normally distributed random variable with standard deviation σ_a.

The first thing we need to do is to adapt σ_a. This should be done for every individual before mutating x_a. The adaption is done as follows

$$\sigma_a \;=\; \sigma_a \cdot \exp(0.2 \cdot N(0,1)) \tag{12}$$

Then x_a can be mutated as follows

$$x_a \;=\; \begin{cases} x_a^{\min} & \text{if } x_a \leq x_a^{\min} \\ x_a^{\max} & \text{if } x_a \geq x_a^{\max} \\ x_a + \sigma_a \cdot N(0,1) & \text{otherwise} \end{cases} \tag{13}$$

Finally, $W_a = \lceil x_a \rceil$. Some notes:

1. Eq. (12) can make σ_a become arbitrarily small. Hence, it is common to choose a lower bound such as $\sigma_a > 0.01$.

2. The adaptation and mutation methods described above have been extensively analyzed [7]. The methods are not sensitive to the 0.2 coefficient value in Eq. (12) nor to the initial σ_a value. These parameters are user defined. We chose 0.5 as the initial σ_a value.

3. $x_a^{\min}$ and $x_a^{\max}$ are user defined lower and upper bounds that depend on the problem at hand.

4. Each individual in the population has a unique pair of strategy parameters. They are adapted before executing Eq. (13).

The strategy parameter σ is used to help the search process. An example will help to illustrate why it is adapted. Suppose the objective is to find the value of x that minimizes some multi-modal function $f(x)$. A new solution could be produced from the current solution by adding a small random variable $\sigma \cdot N(0,1)$ to the current x value. If the function is highly multi-modal than x should be small or a good solution might be missed. However, if the function is somewhat smooth, a large σ value could be used to keep the

search process moving with little risk of missing a good solution. In most cases (at least in high-dimensional functions), it is difficult to ascertain how smooth the function is, which makes it difficult to choose a good fixed σ value. Moreover, over some ranges of x a small σ is needed whereas over other ranges a large value would be better. The best way to deal with this is to let the σ value adapt as the search proceeds. That is precisely the purpose of Eq. (12). This particular form is called lognormal adaption and it is a widely used [4].

5.4 Tournament Selection

During every generation each of the μ parents produces a single offspring. The parents and offspring are combined into a single population and ranked according to preferences. Only the μ best are retained to be parents in the next generation; the others are discarded. Thus parents and offspring compete for survival.

Recall that every preference evaluation requires solving a linear program. Preferences are determined in a pairwise manner, which means $\mu(\mu - 1)$ linear programs must be solved to evaluate the entire population. This is too computationally expensive. *Tournament selection* is a good compromise between identifying the best fit individuals while not expending too much computational effort to do so.

The idea is to conduct pairwise preference checks only between an individual and a small subset of the current population. This small subset, called a *tournament set*, is randomly chosen. A "win" is recorded every time an individual is preferred over a member of its tournament set. (It is possible for an individual to have zero wins.) Conversely, one could record a "loss" every time a member of the tournament set is preferred over the individual. A new tournament set is randomly chosen for every individual. After all of the tournaments are conducted, the individuals are sorted according to the number of wins (losses), and the μ with the most wins (fewest losses) survive.

Obviously the tournament set size impacts the computation time. Too large a value causes the search to degenerate into a pairwise preference check against the entire population. Conversely, too small a value makes it difficult to differentiate between individuals because many of them will have the same number of wins. We have found a tournament set size of around $0.1\mu - 0.2\mu$ works reasonably well.

6 Experimental Results

In our experiments, the test cases require two worker skill classes: worker "a" (an electrician) and worker "b" (a mechanic). The required workers of each type for each task were randomly picked from 1 to 5. As explained before, no task could start until the requisite workers are available in order not to hoarding expensive maintenance equipment. Once assigned, a worker could not be reassigned to another task until the current task is completed. We set $6 \leq W_a, W_b \leq 20$ throughout the investigation.

The EA has a population size of $\mu = 80$. Each parent produces a single offspring using either insertion, inversion, or changing the number of workers. Insertion and inversion are applied most often (80% probability) because changing the PM task order has the most dramatic effect on MS. The probability of choosing insertion, inversion, change W_a or change W_b as the mutation operator is 0.4, 0.4, 0.1 and 0.1 respectively. The tournament set size is 10. The EA is run for 50 generations and the preferred solutions are output so the decision maker can make a final choice.

Four solutions from the randomly generated initial population are ranked to define the constraint subspace. Three total runs are made: (1) a preference-based search with small makespan preferred (MS-preferred), (2) a preference-based search with low number of workers preferred (WN-preferred), and (3) a dominance-based search. The random number generator seed from the first run is used in all subsequent runs. Hence, the identical random numbers were generated in the same order in each run to ensure the same initial population was used for all three schemes so that the comparisons can be as fair as possible. Four solutions from the initial population are randomly chosen and ranked manually before conducting the first preference-based search; the ranking order is simply reversed before conducting the second preference-based search.

The highest ranked individuals from the current generation survive to become the parents in next generation. For the dominance-based search we used the number of "losses" as the rank so that the solutions dominated by fewer other solutions will have a high rank. This rendered a broad Pareto front without having to use any form of fitness sharing. A similar method was used for the preference-based searches. In these searches the rank of an individual A is inversely proportional to the number of other individuals in its tournament set that are preferred over it. For instance, if many individuals in the tournament set are preferred over A, then the rank of A is low. Individual A has the highest rank when all individuals in its tournament set are indifferent to it.

Before plotting any results both the makespan and the number of workers are normalized—i.e., each attribute is mapped onto the unit interval in a way that makes the optimal values 1.0. The worst/best case makespan (denoted by $\widehat{MS}$ and $\widetilde{MS}$, respectively) are recorded among the alternatives that have been explored before the termination of the search process. The normalized MS value is therefore

$$\frac{\widehat{MS} - MS}{\widehat{MS} - \widetilde{MS}}.$$

Similarly the worst/best case number of workers (denoted by $\widehat{MW}$ and $\widetilde{MW}$, respectively) are obtained the same way. The normalized MS value is therefore

$$\frac{\widehat{MW} - W}{\widehat{MW} - \widetilde{MW}}$$

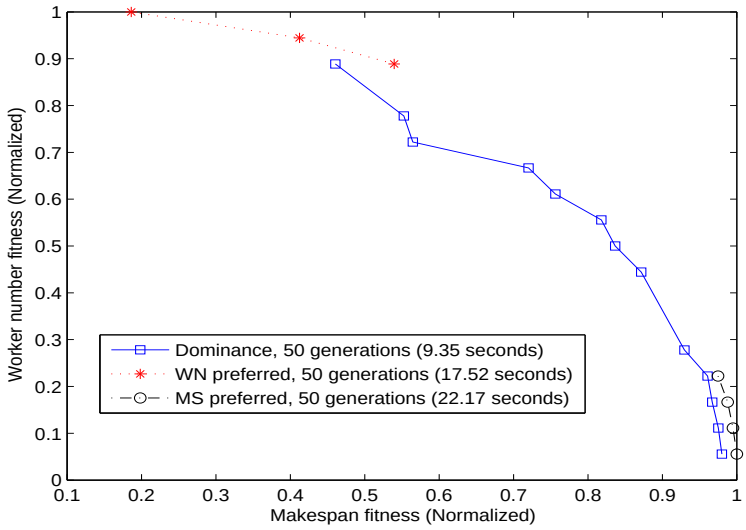


Figure 3: Plots showing the best fit solutions from three types of EA searches for 50 generations.

Figure 3 shows the results for the case when $N = 100$ PM tasks. As shown in the figure, the initial population is widely dispersed, which is essential for a thorough search. Dashed lines are added to depict the Pareto Front location. Notice how the dominance-based search has a broad front whereas the preference-based searches target specific subsets of that front.

The dominance-based search found a broad Pareto front even though no form of fitness sharing was used. Conversely, the two preference-based searches targeted only a narrow region of that Pareto front.

What is particularly noteworthy is the targeted regions on that front were completely different. This clearly demonstrates how ranking a small subset of individuals from the initial population effects the search dynamics. The same four solutions were ranked, but the constraint subspace changed when the ranking was changed. This caused the search to move in an entirely different direction—*even though the same initial population and the same random number generator seed were used.*

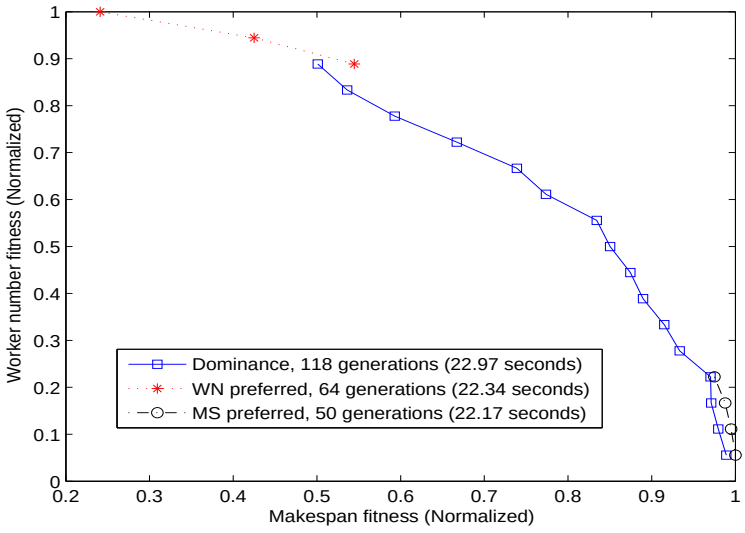
Since one or even two linear program problems need to be solved when comparing two alternatives in the preference-based search, it is interesting to examine how these "extra" computation costs may effect the search effectiveness. Table 2 lists the average computation cost for one generation by different search schemes.

Table 2: Average searching time per generation for different search schemes (in seconds)

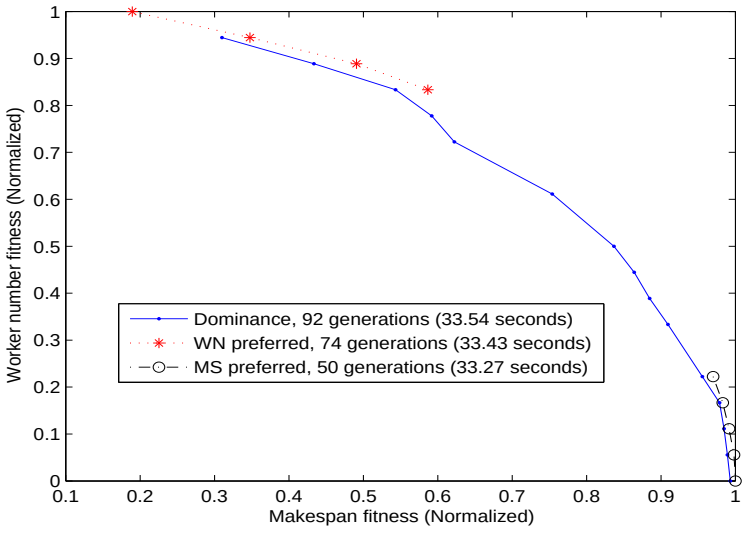
PM task number	Dominance-based search	MS preferred	WN preferred
100	0.2114	0.4223	0.3684
200	0.4859	0.8582	0.5015

From Table 2, the average search time for the MS-preferred search is almost twice as the one by the dominance-based search. However, one should not simply take the difference as the additional cost solely caused by solving the linear programming problems. Since the MS-preferred search scheme is in favor of shorter makespan, it therefore directs the searches toward the solution space potentially containing alternatives with large number of workers. The larger the worker number is, the more costly it becomes to compute the makespan. Note that this is essentially the price that one has to pay in order to search for PM-schedules with relative shorter makespans.

To investigate how effective different search strategies can be under the considerations of computation cost, we conducted an other set of experiments. We used the test case as stated above, but terminated different search schemes at different generations such that they consumed similar computation time. The Pareto-optimal fronts by different EA searches are shown in Figure 6. Note that we intentionally run the dominance-based search for more generations such that the search time it takes is no less than that by either MS-preferred or WN-preferred search. As shown in the figure, with the same computation cost, the solutions found with the two preference-based searches outperform the ones by dominance search from the decision maker's point of view.



(a)



(b)

Figure 4: Plots showing the best fit solutions from three types of EA searches using similar search time. (a) N=100 PM tasks. (b) N=200 PM tasks.

7 Conclusions and Future Work

In this paper, we present a novel evolutionary algorithm to solve the preventive maintenance scheduling problem, which is formulated as a multiple objective problem. Comparing with previous research, our EA searches for Pareto-optimal solutions to the MOP using preferences among solutions rather than dominance to guide the search process. Our experiment results clearly that the preference-based research renders solutions more in line with a manager's expectations.

Several directions are to be pursued along the directions of this research. First, the preference-based search requires the ranking of a small number of individuals (typically taken from the initial population). In large, complex MOPs this initial ranking may inadvertently lead to conflicting constraints. The LP solver cannot produce a valid solution under these circumstances. We need to develop methods that can efficiently detect any conflicting constraints

Second, we assumed a PM task could not start until all of the necessary workers in each skill category were assigned. In practice this policy may not be necessary. Indeed, work often can start so long as at least one of the needed skill workers is available. Additional skilled workers could also be assigned to a PM task already in progress, which can result in an earlier completion time. This means the task duration time should not be fixed, but a function of the number of assigned workers. We intend to do this in the near future and describe here briefly our approach. Let $t_j^a(k)$ denote the time it takes to complete task j using k workers with skill a. (This time will be specified with a yet to be defined nonlinear function.) Then the total time it takes to complete PM task j with n_ℓ workers of skill ℓ is given by

$$T_j \;=\; \max_{\ell \in S}(t_j^\ell(n_\ell)) \tag{14}$$

Third, our further investigation will involve incorporating PM task priorities. PM tasks are conducted at regular time intervals, but there is usually some small "slippage" allowed in the completion time. For instance, a piece of machinery may require a certain PM task be performed after 1000 hours of operation. However, it does not have to be performed precisely at 1000 hours—it may be perfectly acceptable to do this PM task any time between 950 and 1050 hours of operation. The priority for this task may be low at 950 hours, but it will become very high around 1040 hours. To incorporate task priorities will require modification to our currently used reproduction operators.

References

[1] J. Andrews and L. Bartlett “Genetic algorithm optimization of a firewater deluge system”, *Quality and Reliability Engineering International*, Vol.19, 2003, pp. 39-52.

[2] M.A. Abido “A novel multiobjective evolutionary algorithm for environmental economic power dispatch”, *Electric Power Systems Research*, Vol.65, 2003, pp. 71-81.

[3] S. Ahire, G. Greenwood, A. Gupta and M. Terwilliger “Workforce-constrained preventive maintenace scheduling using evolution strategies”, *Decision Sci. J.*, Vol.31, No.4, 2000, pp 833-859.

[4] T. Bäck, U. Hammel and H.-P. Schwefel “Evolutionary computation: comments on the history and current state”, *IEEE Trans. Evolutionary Computation*, Vol.1(1), 1997, pp 3-17.

[5] C.A. Coello “A comprehensive survey of evolutionary-based multiobjective optimization techniques”, *Knowledge and Information Systems*, Vol.1(3), 1999, pp 269-308.

[6] G. Cavory, R. Dupas and G. Goncalves “A genetic approach to the scheduling of preventive maintenance tasks on a single product manufacturing production line”, *Int. J. Prod. Econ.*, Vol.74, 2001, pp 135-146.

[7] A. Eiben and J. Smith “Introduction to Evolutionary Computing”, *Springer-Verlag*, 2003.

[8] C. Elegbede and K. Adjallah “Availability allocation to repairable systems with genetic algorithms: a multi-objective formulation”, *Reliability Engineering & System Safety*, Vol.82, 2003, pp 319-330.

[9] C. Fonseca and P. Fleming “An overview of evolutionary algorithms in multiobjective optimization”, *Evolutionary Computation*, Vol.3, No.1, 1995, pp 1-17.

[10] D. Fogel “Empirical estimation of the computation required to discover approximate solutions to the traveling salesman problem using evolutionary programming”, *Proc. 2nd Annual Conf. on Evol. Programming*, 1993 pp 56-61.

[11] D. Goldberg “Genetic Algorithms in Search, Optimization, and Machine Learning”, Addison-Wesley Pub. Co., 1989.

[12] G. Greenwood, X. Hu and J. D'Ambrosio "Fitness functions for multipleobjective optimization problems: combining preferences with Pareto rankings", *Foundations of Genetic Algorithms*, 1997, pp 437-455.

[13] J. Grefenstette "Proportional selection and sampling algorithms", in *Evolutionary Computation 1: Basic Algorithms and Operators*, T. Bäck, D. Fogel, and T. Michalewicz (Eds.), IOP Publish., 2000, 172-180.

[14] N. Grudinin "Reactive power optimization using successive quadratic programming method", *IEEE Trans. Power Sys*, Vol.13(4), 1998, pp 1219-1225.

[15] I. Harvey "The puzzle of the persistent question marks: a case study of genetic drift", *Proc. of Fifth Int. Conf. on Genetic Algorithms*, 1993, pp 15-22.

[16] J. Horn, N. Nafpliotis and D. Goldberg "A niched Pareto genetic algorithm for multiobjective optimization", *Proc. 1st IEEE Conf. Evol. Comp.*, 1994, pp 82-87.

[17] R. Keeny and H. Raiffa "Decisions with Multiple Objectives: Preferences and Value Tradeoffs", John Wiley & Sons, NY, 1976.

[18] J. McCall "Maintenance policies for stochastically failing equipment: A survey", *Management Sci.*, Vol.11, No.5, 1965, pp 493-524.

[19] K. Murty "Linear Programming", John Wiley & Sons, NY, 1983.

[20] M. Marseguerra, E. Zio and L. Podofillini "Condition-based maintenance optimization by means of genetic algorithms and Monte Carlo simulation", *Reliab. Engr. & Sys. Safety*, Vol.77, 2002, pp 151-166.

[21] M. Marseguerra, E. Zio and L. Podofillini "Multiobjective spare part allocation by means of genetic algorithms and Monte Carlo simulation", *Reliability Engineering & System Safety*, Vol.87, 2005, pp 325-335.

[22] S. Martorell, S. Carlos, A. Sanchez and V. Serradell "Constrained optimization of test intervals using a steady-state genetic algorithm", *Reliability Engineering & System Safety*, Vol.67(3), 2000, pp 215-232.

[23] D. Percy and K. Kobbacy “Determining economical maintenance intervals”, *Int. J. Prod. Econ.*, Vol.67, 2000, pp 87-94.

[24] W. Pierskalla and J. Voelker “A survey of maintenance models: The control and surveillance of deteriorating systems”, *Naval Res. Logistics Quar.*, Vol.23, No.3, 1976, pp 353-388.

[25] Army Regulation 750-1 “Army Materiel Maintenance Policy”, Headquarters, Department of the Army, Washington DC, 18 August 2003.

[26] R. Sarker, K.H. Liang and C. Newton “A new multiobjective evolutionary algorithm”, *European Journal of Operational Research*, Vol.140, 2002, pp 12-23.

[27] Y. Tsai, K. Wang and H. Teng “Optimizing preventive maintenance for mechanical components using genetic algorithms”, *Reliab. Engr. & Sys. Safety*, Vol.74, 2001, pp 89-97.

[28] C. White, A. Sage and S. Dozono “A model of multiattribute decisionmaking and tradeoff weight determination under uncertainty”, *IEEE Trans. Syst., Man & Cybern.* , Vol.SMC-14(2), 1984, pp 223-229.

[29] D. Wienke, C. Lucasius and G. Katema “Multicriteria target vector optimization of analytical procedures using a genetic algorithm”, *Analytica Chimica Acta*, Vol.265, 1992, pp 211-225.

[30] M. Wald “F.A.A. threatens action that could shut airline”, *New York Times*, June 3, 2000, pp A-20.