Research in Intelligent Manufacturing Systems at the University of Pittsburgh

Alice E. Smith and Bryan A. Norman
Department of Industrial Engineering
University of Pittsburgh
1048 Benedum Hall
Pittsburgh, PA 15261 USA
aesmith@engrng.pitt.edu or banorman@engrng.pitt.edu

Abstract

New techniques from computational intelligence have attracted the interest of manufacturing researchers in academia, government and industry. These techniques include artificial neural networks, tabu search and evolutionary computing. While these computational techniques inspired by nature have shown promise in many manufacturing applications such as robotics, machine vision, process control, process planning and scheduling, there is little in the literature on their practical use. Moving these techniques from simulated data sets, toy problems or laboratory settings to real industrial applications is a large and uncertain step. This paper focuses on the technology transfer issues and solutions when using computational intelligence for off-line control of manufacturing processes and scheduling of production environments.

1. INTRODUCTION

The field of computational intelligence embraces several techniques that have been inspired by nature but are mathematical. These techniques are artificial neural networks, tabu search and evolutionary algorithms. Often these techniques are considered part of artificial intelligence, however the name artificial intelligence is more properly given to techniques that try to capture and emulate biological intelligence, such as expert systems and thinking computers. Computational intelligence is still an immature field. Neural networks got their start in the 1940's but were not commonly researched until the 1980's. Evolutionary algorithms were founded independently in the United States and in Germany in the 1960's, however they, too, only became popular research topics in the 1980's. Tabu search had its foundation in the fields of artificial intelligence and operations research and became popular in the late 1980's. When considering manufacturing uses of computational intelligence, the timeline is even more recent. Neural networks began in the field of cognitive neuroscience, and has been dominated by that field along with computer science and electrical engineering. Evolutionary algorithms were founded by computer scientists, mathematicians and electrical engineers, and these fields still produce most of the research in the area. Only tabu search arose from the operations research field, and it is perhaps better accepted than the other two techniques.

However, there is a wealth of difficult and real problems in manufacturing that could benefit from computational intelligence techniques. These problems often involve modeling and optimization of complex systems. This paper will discuss each of these three techniques - neural networks, evolutionary algorithms and tabu search - in turn and how they might be used in manufacturing. The kind of problems these techniques are best suited for will be defined, and competing techniques will be compared and contrasted. Example applications from the authors' research will be shared.

2. ARTIFICIAL NEURAL NETWORKS

Artificial neural networks are computing mechanisms roughly modeled after the biological brain. Neural networks depend on an organized group of simple elements, called neurons. Neurons are uni-directional computing elements that receive multiple inputs, sum them, then produce a single output through a non-linear transfer function *f* (see figure 1). Neurons exist in parallel (a layer) and in series. Weighted connections exist between neurons to move the output of a neuron to other neurons. A neural network can be quite small and simple, but is

more likely to be large (many neurons in multiple layers) and complex. The biological human brain has about 10^{14} weighted connections (synapses), so even a very large artificial neural network is a poor substitute for any living brain.

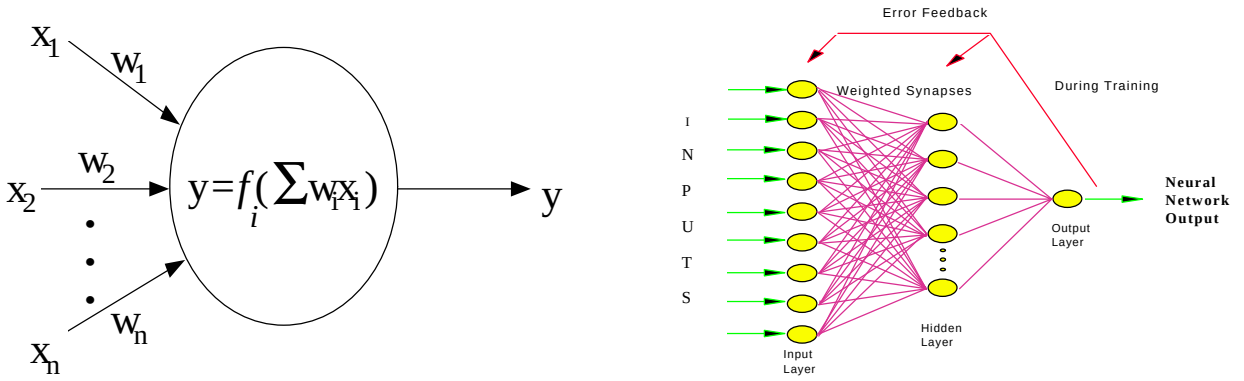

Figure 1. Typical neuron and structure of a neural network.

Neural networks usually begin with their weights in a random state, which is "untrained." The network weights are iteratively evolved to a fixed (trained) state by repeated data input and error calculation. This data set is called the training set and generally consists of input values and one or more target output values. Output errors are reduced by using a training algorithm to change weights. Therefore, the neural network is an empirical or data-driven model, and the process of training can be viewed as a statistical one, where the final weight set is statistic of the training set [43]. This statistical analogy, while not true to the biological inspirations, is most useful when considering manufacturing applications. Neural networks can be used for problems where observational data (inputs and outputs) exist and for which a simpler statistical model or a theoretic relationship is not appropriate. In many real manufacturing problems, theory and analytic relationships drawn from first principles are not adequate to explain the many complexities, interactions and imperfections. Similarly, many problems do not adhere to well known statistical techniques such as linear regression or clustering algorithms. For these problems, a neural network may provide the only reasonable mathematical solution technique. In fact, a neural network should be considered as a technique of last resort where other techniques have failed. This is because neural networks have some serious drawbacks. They are empirical models and depend heavily on data. They are opaque models; that is, the final neural network does not yield much, if any, information about the relationship modeled. A regression model gives a straightforward function with coefficients, however a neural network is a mass of weights and trying to distill them to something understandable is difficult, if not impossible. A final significant drawback is that the development of neural networks is still largely an art. An experienced neural network designer must work interactively and iteratively to build, train and validate an appropriate neural network for a given problem. There are a large array of alternative network architectures, training algorithms and parameters to choose from, with little general advice on how to do so.

However, a neural network can be a successful technique for manufacturing applications. These include real-time control systems, diagnostic systems, machine vision systems, robotic and AGV control systems, and others. The authors have specialized in using neural network models for manufacturing process control and optimization. These are usually static models which use the raw material characteristics, the product design specifications, the ambient conditions, operator information and actions, the machine settings and so on to predict outcomes of the process. Model dynamics can be added by using feedback during the process. If such models mimic the manufacturing process with fidelity, they can be used for a variety of purposes. They can be used to interactively select manufacturing conditions and machine settings that produce the best outcome. They can be used to select design features that improve manufacturability. They can be used to estimate proper settings for new products without trial-and-error testing. They can be used to identify the most important variables of a process. They can be used with an optimization algorithm to directly identify optimal settings or conditions.

The authors have used neural networks to model the following processes when working with industrial partners: injection molding [39], plastic pipe extrusion [40], ceramic casting [9,19,26], metal furniture assembly [44, 45], wave soldering [8,9], and abrasive flow machining [20,21]. These are very diverse processes but they share important common elements. First, theoretic or analytic models were not adequate for the processes. Second, they exhibited non-linear behavior with variable interactions. Third, observational data was available. Fourth, the companies desired to improve control of the processes by systematic selection of the controllable variable settings. However, each process required a somewhat different approach depending on the company's objectives, the available data, the kind and number of process variables, and the repeatability of the process.

2.1. Case Study on Abrasive Flow Machining (AFM)

A neural network based process controller for abrasive flow machining of engine air intake manifolds was developed for a consortium including an AFM manufacturer and a U.S. automotive manufacturer. The first objective of this research is to improve the functional performance of U.S. automotive engines, hence generate the economic benefits of reduction in fuel consumption [41]. The second objective is to enable predictive process control of the AFM process, with an understanding of the relationship between the AFM media to the specified air flow rate of the engine manifolds. This understanding may be useful in controlling and optimizing the AFM process for parts similar to the manifold.

AFM is the removal of material by a viscous, abrasive laden semi-solid grinding media flowing under pressure, through or across a workpiece. The process is abrasive only where the media flow is restricted. Generally, the media is extruded through or over the workpiece with motion usually in both directions. The velocity of the extruded media is dependent upon the principal parameters of viscosity, pressure, passage size, geometry and length [47,48]. Places that have the greatest restriction will produce the largest grinding forces. Four main types of abrasives are used in AFM. These are aluminum oxide, silicon carbide, boron carbide and diamonds. The AFM process acts in a manner similar to grinding or lapping where the extruded abrasive media gently hones edges and surfaces. It is particularly useful when applied to workpieces containing passageways that are considered to be inaccessible with conventional deburring and polishing tools [3,10,12,25,34,36,37,38,46,47,48].

Four major tasks were undertaken to develop the model: (1) identification of the key process variables, (2) data collection, (3) preliminary neural network development, and (4) model validation.

Process Variables and Data Set

The first step was to determine which process variables were critical to the AFM process and should be included as process input parameters to the neural network. Table 1 summarizes these process variables. The development of the predictive model was an attempt to capture the behavior of both the independent and interaction effects of these variables so that it can accurately predict the flow of the orifice fluid (*viz*., air) through the manifold. The main categories of process variables are:

- Incoming part - weight, surface finish, air flow, throttle body diameter
- AFM machine setting - pressure, volume of media extruded, number of passes, media flow rate
- Media condition - grit, freshness, media temperature, part temperature
- Ambient conditions - temperature, humidity

There is variability among the incoming parts due to the limitations of the sand casting process. The total volume of media extruded and the pressure for the AFM processing is preset by the operator depending on his judgment of when the manifold reaches airflow specifications. These two machine settings determine the number of passes needed and the media flow rate for the process. The ambient conditions impact the condition of the media. The primary variables of media condition are extremely important. The media starts new with an amount of grit and no impurities. Over time, impurities enter into the media from the metal being AFM'ed and the grit becomes less abrasive. This has a profound impact on the AFM process, however measurement of media condition during processing is impossible. Another change in media condition occurs daily. The behavior of the media depends partially on its temperature. At the beginning of a day, the media is cold, however after repeated processing, it

becomes heated. The sequence of production and the number of parts machined prior to the current part (table 1) are very crude approximations to this heating effect. Time of production was divided into five periods beginning in the morning (period 1) and ending with the workday (period 5). Each part was assigned to one of these periods depending on its time of production. The number of parts machined prior to the current part is simply a counter for another measure of the changing characteristics of the media during a workday.

Table 1. Process Variables.

Process Variables	Definition
Incoming weight	The shipped weight of the manifold
Throttle body diameter	The diameter of the throttle body orifice
Surface finish	The surface roughness inside the throttle body orifice
Runner airflow	The airflow rate through each paired runner
Variability of runner airflow	The variability of airflow rate among the 6 pairs of runners
Ambient temperature	The temperature in the plant
Ambient humidity	The humidity in the plant
Production sequence	The sequence of the production during the day
Number of parts prior to current part	The number of parts machined prior to the current part in a work day
Media condition	The condition of the media (cutting ability, contamination level)
Hydraulic pressure	The extrusion pressure of the media
Volume of media extruded	The volume of media extruded through the part
Media flow rate	The rate of media flow through the part
Number of passes	The number of cycles of the AFM machine piston
Media temperature	The temperature of the media
Part temperature	The temperature of the part

The primary specification is average outgoing (i.e., after AFM processing) air flow through the manifold. It is this specification that the model described here targets.

Production data on the variables described in table 1 (except the ambient conditions and the media condition) was collected by the company's technicians. Fifty-eight observations were collected. For the processing variables (i.e., pressure, volume of media extruded, media flow rate and media temperature) which were collected on a per pass basis during the AFM process, additional statistics such as: the range, the median, the average, the gradient and the standard deviation were derived to represent some of the dynamics during the AFM process. A first order stepwise regression model was developed using the entire data set to identify the statistical significant variables for the neural networks. Regression results indicated that these variables could explain 87.00% of the variance of the outgoing average airflow. Input variables to the networks include the following: incoming average air flow, median of media flow rate, range of media temperature, range of part temperature, standard deviation of volume of media extruded, average pressure and the number of parts machined prior to the current part. The outcome variable studied was outgoing average airflow.

Cascade-Correlation Neural Network Results

The neural network architecture for predicting the outgoing average airflow of engine manifolds was created using the cascade-correlation learning algorithm, available in the software package of NeuralWorks. The training parameters and the maximum number of epochs were selected through experimentation and examination of preliminary networks. A cascade-correlation learning algorithm was used because it learns very quickly and the network determines its own size and topology [11]. This algorithm starts off with no hidden neurons, with only direct connections from the input units to the output units. Then the hidden neurons are added one at a time, and the purpose of each new hidden neuron is to predict the current remaining output error in the network. Unlike the traditional backpropagation learning algorithm, the hidden neurons are allowed to have connections from the pre-existing hidden neurons along with the connections from the input units. The final network architecture had 7

inputs, one hidden layer with 10 neurons, and a single output. The learning rate was fixed at 0.10 and the unipolar sigmoid transfer function was used.

A leave-one-out cross-validation approach, using the entire data set for model evaluation, was used [42]. Despite the computational effort, this approach was chosen in favor of a grouped cross-validation approach because it provides a better estimate of the performance of the final network especially when the amount of data is scarce. This approach required building fifty-eight validation networks with each trained on fifty-seven observations and tested on the left-out observation. Each validation network has a different left-out data point and combining all the left-out data points for all the validation networks exhausts the entire data set. All of the validation networks were built using parameters identical to those of the final network described above with an upper bound of 10 hidden neurons.

Figure 2 shows the leave-one-out cross-validation networks and their predictions on each left-out data point against the actual observed outgoing average airflow. The final network was able to predict the outgoing average airflow with a mean absolute error of 0.0972 (0.0569%) and a root mean square error of 0.1358 (0.0794%). The R-squared (coefficient of determination) value of the final network, which was estimated by the validation networks, is 0.8741 which means that the network can explain about 87 percent of the variance in the outgoing average air flow.

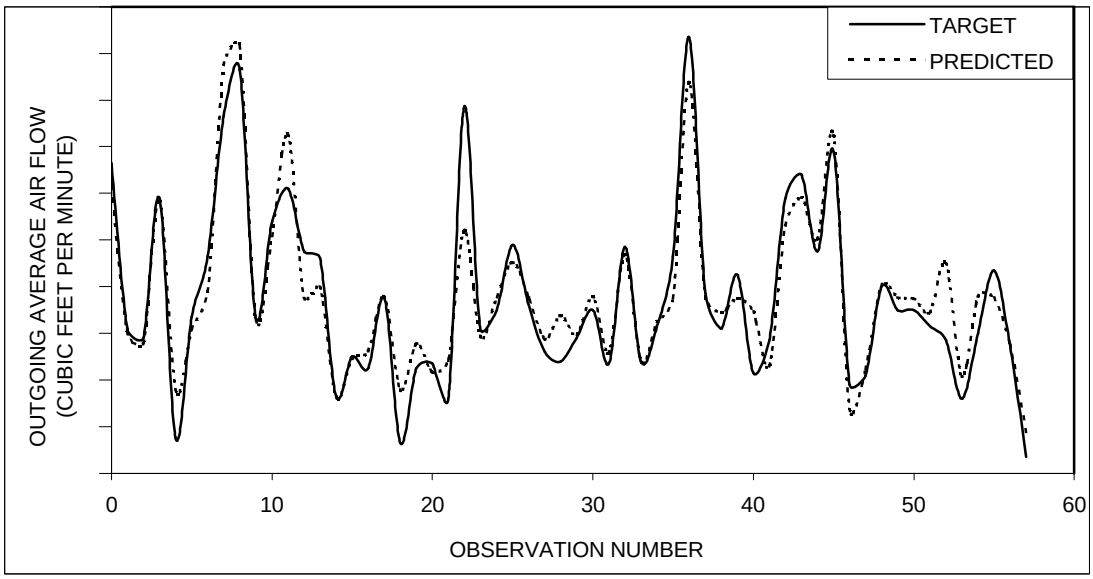

Figure 2. Performance on leave-one-out cross-validation: Predicted against target outgoing average airflow.

2.2 Final Remarks on Neural Networks

Neural networks do have a long learning curve for developers to become proficient. There are commercial neural network packages for development such as NeuralWorks by NeuralWare [28] and Brainmaker by California Scientific Software [4]. Coding a neural network with a procedural language such as C or Pascal is not difficult. However, to successful build and validate a neural network model requires much data, time and expertise. There is little in the way of definitive guidelines or rules, as there is in statistical model building and validating. At least for the near-term, using neural networks in manufacturing applications requires the assistance of an experienced person and some specialized software development. Neural networks can also be implemented in hardware, i.e. chip, form for real-time applications. This also takes specialized expertise and development software. However, once a neural network is built and validated, it requires no special knowledge or expertise on the part of the user to successfully use it for prediction, classification and decision making.

3. EVOLUTIONARY COMPUTATION

The field of evolutionary computation (EC) includes the following subjects: genetic algorithms, evolutionary strategies, genetic programming and classifier systems. All are meta-heuristics inspired by the process of biological evolution where an initially random population evolves iteratively to a superior, or optimized, state. Solutions are selected for recombination based on their objective value function, where a better value yields a higher *fitness*. The recombination, called *crossover*, usually involves two selected solutions (*parents*) that are combined to form one or more new solutions (*children*). The children solutions are randomly perturbed slightly (*mutation*) to move the search to new regions. Evolutionary computation is generally used for optimization, whether it is of a continuous function, a combinatorial problem, or finding optimal rules to explain data. The advantages of evolutionary computing over more traditional approaches such as mathematical programming are that a *population* (small group) of superior solutions are obtained, no assumptions about form or derivatives are made, the iterative nature is usually diminishing in improvements so the computational time needed is flexible, EC is very flexible and can accommodate almost any problem, and EC is easy to code and to understand. Moreover, EC is a global technique; that is, it is resistant to becoming trapped in local optima. Disadvantages of evolutionary computing are that it is stochastic and may return different solutions depending on random number seed, it cannot guarantee convergence or optimality except under very restrictive conditions, and it may not be computationally efficient compared to other problem-specific methods.

The advantages of not specifying a functional form that is differentiable, continuous and so on is a tremendous advantage in real world problems. It is also advantageous to use a global technique since it is usually unknown whether a surface is convex, a condition required for gradient methods to converge to the global optimum. The ease of coding and the flexible computing time are added inducements for the use of EC on real manufacturing problems in optimization. To use EC, the solution space must be encoded as a series of bits, floating point numbers, or as a permutation. The traditional genetic algorithm uses a bit encoding, however this is not required, or even desirable in many instances. Solutions must be able to be compared on a numeric basis using an objective function, which translates to *fitness* in EC. The better the objective function, the more likely the solution will survive in later iterations and also produce children solutions. Poorly evaluated solutions tend to die off immediately. To summarize, the important steps to using EC are an encoding, crossover and mutation algorithms and a method of calculating the objective function value of a solution. There are problem-specific parameters that must be set, such as population size, probability of mutation and termination criteria, but EC is robust to a wide variety of these settings.

3.1 Scheduling of Parallel Machine Tools Case Study

A genetic algorithm was developed for scheduling operations on Parallel Machine Tools (PMTs). A key difference between Parallel Machine Tools (PMTs) and conventional CNC machines is that the former contain multiple spindles and can hold multiple workpieces concurrently. As a result, a PMT can process more than one workpiece at a time and/or perform more than one operation at a time.

To properly describe PMTs it is necessary to define some terms. We retain the terminology introduced in [23]. A *part machining location* (PML) refers to a valid workholding location. The main spindle and subspindle(s) always represent valid PMLs. A *machining unit* (MU) refers to a tool holding device, which may hold a single tool or a turret containing multiple tools. Relative motions between the tool on the MU and the workpiece held in the PML accomplish the machining. Conventional machines have only one MU and one PML. PMTs have $PML_{max} \geq 1$ PMLs and $MU_{max} \geq 1$ MUs. For a more detailed discussion of the structure of PMTs, see [23].

The scheduling problem involves determining an operation sequence that minimizes the processing time for a given job. A job consists of one or more workpieces, each of which must have a number of operations performed on it. We are given the times required for each operation, the mode of each operation (defined below), and the precedence relations associated with each operation. The goal is to determine the sequence of operations that will minimize the overall time required to process the set of jobs. There are four problem characteristics that complicate

the scheduling of operations on a PMT: precedence constraints between operations, mode restrictions, assignment of operations to PMLs, and the assignment of tools to MUs. These characteristics are now described in more detail.

- Precedence constraints arise due to the part geometry, tolerancing considerations, and manufacturing practice.
- The operations performed on a PMT may be classified into three *modes* depending on the motion of the workpiece at a PML and the motion of the MUs that are machining the workpiece. The three modes are defined as in [23]: turning - when the workpiece is rotating and the MU is stationary; milling - when the part is stationary and the MU is in motion, as in drilling or milling; contouring - when both the part and the tool are in motion, as in contour-milling. Operations that require different modes cannot be performed concurrently at the same PML due to technological limitations.
- Because a PMT has multiple PMLs, we must determine the set of operations that will be completed at each PML. In this research we assume that the operations have already been assigned to the PMLs.
- The assignment of tools to MUs also has an effect on processing times. Only operations using tools on different MUs can be performed in parallel. This problem is complicated by the fact that not every PML may be accessible to every MU. Tool assignment is a difficult problem and will be explored in future research.

Consider the example problem data given in Table 2. An optimal operation sequence is shown in Figure 3 and has a makespan of 45.

Table 2. PMT example problem data.

Operation	$P_{i,j}$	Time	PML	Mode	MU
1		10	1	1	2
2		7	1	3	1
3	1	9	1	2	3
4	1	6	1	2	1
5	4	8	1	3	2
6	4	6	2	2	1
7	5	3	2	1	3
8	5	8	2	1	2
9	7,8	3	2	2	2
10	5	10	2	2	3

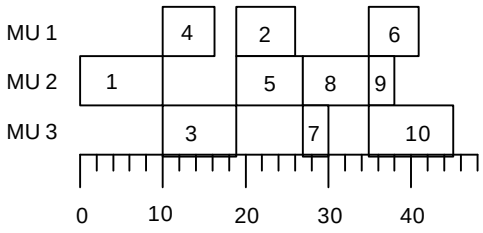

Figure 3. Optimal solution to example PMT problem.

We make four problem assumptions: (1) the MUs are continuously available, (2) there is no preemption of operations, (3) tools have been assigned to MUs and there is no duplicate tooling, and (4) the objective is to minimize the makespan (i.e., complete the job as soon as possible). The makespan objective is reasonable for the PMT scheduling problem as it occurs entirely within a single machine. We assume that inventory and due dates are considered at a higher decision making level when order-release is determined. Our objective in determining a process plan for the PMT is to get the workpieces contained within an order through as quickly as possible. Makespan is an appropriate measure for this objective.

A genetic algorithm, referred to as algorithm RKGA, is proposed that utilizes the random keys encoding introduced in [2] and applied to general scheduling problems in [32]. The random keys representation encodes a solution with *random* numbers. These values are used as sort *keys* to decode the solution. The chromosome is interpreted in the fitness evaluation routine in a way that avoids feasibility problems. The precedence constraints between operations are enforced by maintaining a sorted list of random keys that only contains the schedulable operations. A schedule builder is used to map chromosomes of random keys to schedules of operations. It begins by using the sorted random keys sequence to determine the order for placing operations into the schedule. The schedule is constructed moving forward in time and making sure that no precedence, MU usage, or mode constraints are violated. Using this schedule builder, the RKGA explores a subset of the semi-active schedules. A schedule is *semi-active* if it cannot be improved by sliding an operation to the left in the Gantt Chart. For a regular

measure (e.g. makespan), an optimal schedule exists within the set of semi-active schedules (based on a proof in [1] with only minor modifications).

To construct the schedules we employ a *left-shift* operator. The number of *left-shifts* investigated is controlled dynamically by the RKGA. Each operation now has two genes - one containing the random key and one containing a *delay factor* with a real value between 0 and 1. The initial operation sequence is based on the sorted random key values. The schedule builder moves forward in time inserting operations. However, if scheduling operation *i* next results in a change of mode and creates idle time on MU_i, the remaining schedulable operations are examined to see if any of them have the same mode as the current mode. Any operations that satisfy these criteria are placed in the set δ. We then search δ to find any $j \in \delta$ where $ES_j + t_j \leq delay\ factor_j * t_j + ES_i$ (ES_i represents the earliest time that operation *i* can start and t_j represents the processing time of operation *j*). If there exists an operation, j^*, that satisfies this criteria, then j^* is scheduled next instead of operation *i*. Thus *left-shifts* attempt to maximize the number of operations that can be performed in parallel. It can be shown that there exists a set in the chromosome space with nonzero measure that maps, using the schedule builder with *delay factors*, to an optimal schedule. Utilizing *left-shifts* leads to an improvement in both solution quality and rate of convergence for the RKGA. The *left-shift* concept could be modified to insure that the RKGA investigates only the set of active schedules, but it is very computationally intensive to explore all global left-shifts to verify that a given schedule is an active schedule.

Reconsider the example problem data given in Table 2. The random key values given in part a) of Figure 4 result in the schedule shown in Figure 3. Using the delay factor concept the random keys shown in part b) of Figure 4 would also result in the optimal schedule. In this case, after the first two operations are scheduled the partial schedule contains operations 1 and 4. Now, applying straight random keys results in scheduling operation 2 next. This results in a schedule that requires 41 time units at PML 1 and at least 59 time units overall. However, using the delay factors in conjunction with the random keys results in the optimal sequence shown in Figure 3. The RKGA could be run using only the straight random keys but the delay factor concept improves the performance of the RKGA.

a. Random Keys	(.71, .81, .46, .32, .56, .72, .23, .46, .28, .91)
b. Random Keys	(.71, .44, .46, .32, .56, .72, .23, .46, .28, .91)
Delay Factors	(.16, .21, .51, .08, .61, .41, .77, .18, .27, .39)

Figure 4. Sample random keys and delay factors for PMT example problem data.

Extensive computational testing was conducted using data sets that were randomly generated to represent the types of parts machined on PMTs. An investigation of the types of parts that are manufactured using PMT's has been conducted by [49] and [50]. Using this information we randomly generated 648 different problem instances. The problems varied in the problem characteristics described below. We examined each combination of these problem characteristics and generated three data sets for each combination.

- Problem sizes of 50, 75, and 100 operations were tested.
- Problems with 3, 4, and 5 MUs were tested.
- Operations were assigned modes in a manner that reflects the operation to mode distribution found in practice \cite{yiphoi94}. Typically, there are more turning operations early in the precedence network and more milling and contouring later in the precedence network. Three mode distributions were used.
- Two different precedence densities resulting in both shallow and deep precedence trees were tested.
- Problems with an outtree structure and a more general precedence structure where each operation could have up to four immediate predecessors were examined.
- Two different distributions of operations to PMLs were tested.

The RKGA was compared with other heuristics including three dispatching rule heuristics, LDB from [24] and PRIO1 and PRIO2 from [31], and an alternative genetic algorithm, YDGA from [51], and the lower bounds developed in [31]. It is also possible to model the scheduling problem using math programming [31] but

experimental results indicate that problem instances with 25 operations require 40 hours of computation time to solve using CPLEX on a Sun Sparc 20 and problem instances with 50 operations could not be solved in 200 hours.

The following parameter values were used based on preliminary testing. See [29] for a complete discussion of these parameters.

- Population size = 312.
- Maximum run length = 150 generations.
- Clones = 15. This clones or copies the 15 best solutions from generation *i* directly into generation *i+1*. (This is often referred to as an elitest strategy.)
- Immigrants = 36. Immigration is a type of mutation where randomly generated chromosomes are introduced into the population each generation. This introduces new genetic material into the population when the immigrants are chosen as parents in the crossover procedure.
- Mutation chromosomes = 54. This is a more conventional form of mutation. Since the random keys and delay factors are real values we mutate them by perturbing the current value by a random variate uniformly distributed between -0.5 and 0.5. Each gene is mutated with a probability of 0.50.
- Crossover probability = 0.7. Uniform crossover is utilized and the probability of selecting the allele value from parent 1 is 0.7.
- Tournament selection with $t = 2$. To select each parent for the crossover procedure two solutions are randomly chosen from the current population and the best of the two is retained as the parent.

Each problem was solved 5 times using a different initial random number seed resulting in 5 replications for each problem. From the results in Table 3 it is clear that the RKGA provides the best average solution results. In addition, the RKGA's performance was very stable across the replications. Across all of the problems the average deviation from the lower bound was 3.9%. The maximum deviation in solution quality from the best replication to the worst for each set of problem characteristics was less than 2%. The data in Table 3 also indicate that even if we consider the worst replication the RKGA still performs significantly better than the other heuristics with respect to solution quality. Overall, the worst replication provided a solution that was on average 5.3% better than PRIO2, 9.8% better than LDB, 17.6% better than PRIO1, and 4.6% better than the best replication of the YDGA. Table 3 shows the effect of changing the number of MUs. As the number of MUs increased from three to five the deviation from the lower bound increased for all methods tested. This is because the lower bounds become less tight as the number of MUs increases and the problem becomes more difficult to solve because it is more difficult to optimize the number of operations being performed in parallel. The data in Table 3 show that the performance of the RKGA relative to the other methods improved as the number of MUs increased. For a more detailed analysis of the how the problem characteristics affected the performance of the different algorithms see [31]. The average computation times for the 50, 75, and 100 operation problems are about 55, 85, and 125 seconds respectively on a Sun Sparc 20.

Table 3. Average percent deviation from the lower bound for each heuristic method.

		Operations								
		50			75			100		
		MUs			MUs			MUs		
		3	4	5	3	4	5	3	4	5
	PRIO1	14.3	24.3	31.8	11.6	22.1	31.6	12.2	22.4	29.3
	PRIO2	5.4	12.5	19.5	3.7	10.5	18.0	4.1	10.4	15.7
	LDB	8.3	17.5	24.0	5.9	13.4	21.1	5.8	12.7	18.6
	Min	6.4	16.8	18.8	11.9	20.7	29.6	15.4	29.1	39.6
YDGA	Avg	9.5	19.7	22.3	13.9	25.0	35.2	18.2	31.9	43.6
	Max	11.9	22.2	25.7	15.5	29.0	41.1	21.3	34.5	46.6
	Min	0.8	4.0	7.5	0.2	1.4	4.7	0.0	0.4	2.9
RKGA	Avg	1.0	5.3	10.3	0.6	3.2	7.1	0.3	2.1	5.3

Max	1.5	7.3	14.2	1.3	5.5	9.7	6.6	9.9	10.0

3.2 Job Shop Scheduling Case Study

A similar RKGA approach was used for job shop scheduling in [33]. The delay factor concept was used to identify a search space that comprised a subset of the semi-active schedules. The method was tested on the benchmark problems of [22] and this approach was demonstrated to be quite effective compared to other genetic algorithm approaches that have been developed for the job shop scheduling problem.

4. TABU SEARCH

Glover (14, 15) provides an overview of tabu search and [16] discusses applications to a number of operation research problems. Tabu search is a neighborhood search procedure that begins with an initial seed solution and improves the solution through a series of moves. Consider a single machine scheduling problem, if the neighborhood is defined as all adjacent pair-wise interchanges of jobs in the current seed solution, then for a given seed sequence of *1-2-3-4* some of its neighbors would be *1-2-4-3*, *2-1-4-3*, *1-3-2-4* among others. The new sequences would change if the moves and hence the neighborhood of moves are defined differently. It is possible to define different neighborhoods based on different definitions of a move. In every iteration of the tabu search procedure, the entire neighborhood or only part of the neighborhood if that is very large, is explored and the solutions are ranked based on their quality. Unless the move required to do is forbidden or *tabu*, the best neighboring solution becomes the new seed solution for the next move. A term *tabu list length* is defined which represents the number of iterations for which a move remains tabu. The *tabu list length* is often determined empirically but a value of *5* to *15* has been found to work well. The list of tabu moves and their tabu status are updated at the end of each iteration of the tabu search procedure.

The tabu search is not a greedy search procedure and hence does not necessarily follow a path with monotonic improvement. This enables the tabu search to escape from local optima and try to reach the global optimum. If the seed solution is a local optimum, this implies that the new seed solution will have a worse objective function value. Since the procedure cannot differentiate between a global optimum and a local optimum, the procedure has to be terminated by some other means - either by setting a limit on the maximum number of iterations or on the maximum number of iterations without improvement in the best solution found. There numerous modifications that can be made to the tabu search procedure, see [14,15,16].

4. Tabu Search Scheduling Application

The tabu search application considered is flowshop scheduling with finite buffer capacity. In a flowshop with m machines, each job consists of m operations that must be completed on the m machines and each job follows the same machine routing sequence. In a flowshop with unlimited buffers between the machines, the earliest start time for operation k of job i (performed on machine k) is denoted S_{ik} and

$$S_{ik} = max\{C_{i,k-1}, C_{j,k}\}$$

where $C_{i,k-1}$ denotes the completion time of job i on machine $k-1$ and j is the job preceding i on machine k.

The problem of limited buffers is commonly seen in the manufacturing and processing industries. A buffer is usually a storage space in front of a machine or some container that holds intermediate work-in-process (WIP) inventory. Both space and financial constraints might restrict the number of available buffers. For example, in chemical and processing industries large reactor vessels are used for intermediate storage. The number of vessels that can be used is limited due to the space available. Additionally, the buffer unit cost for WIP inventory can get high when dealing with costly and specialized products. In this case the number of buffer units may be restricted due to financial reasons. Having finite buffers serves to reduce material handling and work-in-process inventory, both of these are very important especially in manufacturing settings and form a major part of the production costs.

When there is a finite buffer between machines k and $k+1$, denoted by b_k, then the start time expression becomes

$$S_{ik} = \max\{C_{i,k-1}, C_{j,k}, S_{i-b_k,k+1}\}$$

The last term $S_{i-b_k,k+1}$ represents a blocking term. Job i-b_k is the job that is b_k jobs ahead of job i in the schedule. Job i-b_k must have begun processing on machine $k+1$ for a unit of buffer to be free to store job i. If there is insufficient buffer space between machines k and $k+1$ then jobs on machine k may be blocked. This blocking may increase the completion time of one or more jobs in the schedule. Consider a 5 machine flowshop with 4 buffers allocated as shown in Figure 4.

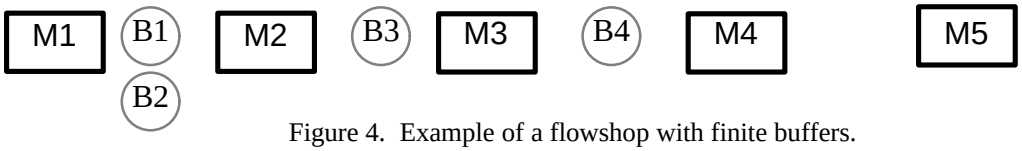

Figure 4. Example of a flowshop with finite buffers.

A job completed on machine M1 can wait in buffers B1 or B2 until machine M2 is available and M1 is available to work on the next job. But consider machines M4 and M5 where there are no buffers between the machines. A job completed on M4 has to wait on M4 if M5 is not available because it is processing another job. In this instance M4 is blocked and cannot operate. This can potentially lead to blocking on machine M3 that in turn can lead to blocking on M2 and perhaps ultimately to blocking on M1. This likelihood of blocking is reduced by providing storage for the jobs between the machines. If there are infinite buffers between the machines, no blocking takes place. However, as previously noted, in general there are not infinite buffers between the machines. Thus, it is very important to allocate the buffer storage that is available in a manner that minimizes the potential for blocking and improves production.

The following assumptions are also made. (1) There is only one type of each machine and each machine can process only one job at a time. (2) There is a known set of jobs to be processed and the processing time required at each of the machines is known. (3) The machines are continuously available, i.e. breakdowns are not considered. (4) Setup times, cleanout times and transport times, if they exist, are included in the processing times. (5) Each job requires exactly one unit of buffer for storage. (6) The maximum number of buffers that can be allotted is fixed. (7) The objective is to determine a job sequence and a buffer allocation to minimize the makespan or completion time of the last job in the sequence on the last machine. (8) The solution space is restricted to the set of schedules where the job sequence is the same for all the machines. Schedules of this type are referred to as *permutation schedules*.

The problem of scheduling flowshops with unlimited buffers is NP-hard [13]. For this reason a number of heuristics have been proposed for the problem including both constructive heuristics and the application of metaheuristics such as tabu search, simulated annealing and genetic algorithms (see [1, 5, 35]). However, the flowshop scheduling problem considering both sequencing and buffer allocation problems has been investigated by only a few authors. Karabati and Kouvelis [17,18] discuss the interaction between the buffer allocation and the sequencing in a cyclic flowshop structure. The authors discuss a method of getting lower bounds using Lagrangian relaxation and also propose a branch and bound solution procedure. However this procedure can solve only small problems in a reasonable amount of time. For larger problems, they propose a heuristic method that uses a greedy buffer allocation heuristic and an insertion heuristic to sequence the jobs in order to minimize the cycle time. Chu et. al. [7] propose a heuristic for the sequencing problem with the objective of minimizing the makespan. An improvement procedure is also provided for their algorithm by introducing a minimal bound on the makespan. However, very large problems cannot be solved using this heuristic due to the computational effort involved.

4.2 Tabu Search Heuristic

Details on the specifics of the proposed tabu search procedure are now provided. There are two types of moves – sequence and buffer moves. Sequence moves are made by removing one job at a time from the current sequence and computing the makespan that results from inserting it into the remaining $n-1$ positions. Repeat this procedure for all n jobs and pick the sequence leading to the maximum reduction in the makespan. A buffer move is made by removing one unit of buffer from its current location and placing it in the location that leads to the greatest reduction in the makespan. This procedure is repeated for all the locations where buffers are allocated and the buffer allocation leading to the maximum reduction in the makespan is selected. It is tabu for a job A to

immediately succeed job *B* if job *A* has been removed from a position immediately after job *B* in a recent move. Similarly, it is tabu for a job *A* to immediately precede job *C* if job *A* has been removed from a position immediately before job *C* in a recent move. If a unit of buffer has been removed from the buffer immediately after machine *k*, then a unit of buffer cannot be placed into that buffer location (the one after machine *k*) as long as that buffer location is tabu. Both moves use a simple aspiration criterion that permits a tabu move if it leads to a new globally best solution. The tabu list length is dynamic but distributed uniformly between 5 and 10. The number of moves with no improvement in the objective function that can be made before stopping the search is 200. Preliminary testing on the tabu length and the number of no-improvement steps concluded that these numbers performed well. The procedure also utilizes a simple backtracking structure that dynamically stores the five best local optima that have been encountered during the search and revisits these solutions to commence the search from these nodes. The number of backtracks; i.e. the number of times that the procedure can go back to the same node is restricted to two. The starting solution for the tabu search is obtained by applying a greedy sequencing heuristic (see [6]) based on the insertion procedure of [27]. The tabu search stops when all the promising solutions have been exhausted, i.e. it has backtracked to all five best local optima without finding any improved solutions.

4.3 Computational Results

Computational testing was conducted on a large suite of test problems. Since there is no existing test suite for the flowshop problem with finite buffers, randomly generated problems were tested. The processing times were drawn from a uniform distribution ranging from 1 to 100. The number of jobs used in the test problems was 10, 20, 50 and 100, the number of machines was 5, 10 and 20 and the number of buffers to be allocated was 3, 5, 7 and 10. For each combination of number of jobs, machines, and buffers to allocate, 5 sets of processing times were generated. The result is 48 problem combinations and 5 problem sets for each combination resulting in a total of 240 problem instances. Due to the size and complexity of these problems, it is not possible to determine provably optimal solutions for these problems. Tight lower bounds cannot be obtained for these problems due to the blocking caused by the finite buffers. Therefore, the results for each heuristic are compared with the best known solution found for each problem instance.

The test problem results are shown in Table 4. The tabu search results are compared to the heuristic of Karabati and Kouvelis [18] denoted KK and two heuristics introduced in [6] denoted GH (greedy heuristic) and SM (single move heuristic). N represents the number of jobs and m the number of machines. Each cell represents the average percent deviation from the best known solution for the different problem sizes averaged across the four different buffer sizes. The tabu search found the best solution in all problem instances and gave the best overall results in terms of solution quality. Statistical analysis using a t-test indicated that the difference between the Tabu Search and the Greedy and Single Move heuristics over all four job sizes was statistically significant with a p-value < .0001. The performance of the heuristics did vary depending on the characteristics of the problem. However, for each different combination of numbers of jobs, machines, and buffers the Tabu Search consistently gave the best performance. Statistical analysis indicated that the Tabu Search was significantly better than the other heuristics for each combination of problem parameters with a p-value < .01.

Table 4. Flowshop scheduling test results.

	KK	GH	SM	TS
N=10, m=5	1.134	1.006	1.013	1.000
N=10, m=10	1.136	1.017	1.024	1.000
N=10, m=20	1.088	1.017	1.018	1.000
N=20, m=5	1.103	1.012	1.005	1.000
N=20, m=10	1.119	1.028	1.023	1.000
N=20, m=20	1.119	1.029	1.035	1.000
N=50, m=5	1.101	1.009	1.012	1.000
N=50, m=10	1.143	1.040	1.043	1.000
N=50, m=20	1.130	1.038	1.044	1.000

N=100, m=5	1.085	1.011	1.009	1.000
N=100, m=10	1.131	1.038	1.036	1.000
N=100, m=20	1.128	1.046	1.042	1.000

The computation times (on a Sun Sparc 20) for each method had the following characteristics for the largest problems: the KK heuristic took up to 30 seconds, Greedy Sequencing took up to 60 seconds, the Single Move Heuristic took up to 40 seconds, and the Tabu Search heuristic took up to 500 seconds. Thus, there was a trade-off between solution quality and computation time. However, the tabu search required less than 9 minutes for the largest problems, which makes it a practical solution methodology for most industrial settings. The CPU time increases as the number of jobs and machines increases. This is because of the increase in the number of sequencing moves and the time taken to evaluate each sequencing move. The CPU time is not very sensitive to the buffer size because the number of buffer moves to be made is very small compared to the number of sequencing moves made.

The tabu search consistently outperforms the other methods and finds the best solution in all problem instances. The tabu search requires more computation time but can evaluate problems with 100 jobs and 20 machines in less than 9 minutes. In addition, the tabu search can be extended to include variable buffer costs and the related problem of minimizing both the cost of buffers and the production cost.

Tabu search has also been applied to a similar flowshop problem where the buffers have already been allocated, there are sequence dependent setup times and the objective is to minimize the makespan. Norman [30] presents a tabu search algorithm that is shown to be quite effective relative to other methods that could be applied to the problem.

References:

[1] Baker, K., *Elements of Sequencing and Scheduling*, Amos Tuck School of Business Administration, Dartmouth College, Hanover, NH, 1995.

[2] Bean, J. C., "Genetics and random keys for sequencing and optimization," *ORSA Journal on Computing*, 1994, vol. 6, 154-160.

[3] Benedict, G. F., *Non-Traditional Manufacturing Processes*, New York, NY: Marcel Dekker Inc., 1987, pp. 53-65.

[4] *Brainmaker User's Guide* and software, California Scientific Software, Grass Valley, CA, various years.

[5] Chakravarthy, K., *Flowshop Scheduling With Work-In-Process Inventory Constraints*, unpublished Master's Thesis, University of Pittsburgh, 1998.

[6] Chakravarthy, K. and B. A. Norman, "Job Sequencing and Buffer Allocation in a Flowshop with Limited Buffers", Technical Report, Dept. of Industrial Engineering, University of Pittsburgh, 98-5, 1998, submitted to *European Journal of Operational Research*.

[7] Chu C., Proth J-M., Sethi S., "Heuristic procedures for minimizing makespan and the number of required pallets," *European Journal of Operations Research*, 1995, vol. 85, 491-502.

[8] Coit, David W., Jay Billa, Darren Leonard, Alice E. Smith, William Clark and Amro El-Jaroudi, "Wave soldering process control modeling using a neural network approach," *Intelligent Engineering Systems Through Artificial Neural Networks, Volume 4* (C. H. Dagli, B. R. Fernandez, J. Ghosh and R. T. S. Kumara, editors), ASME Press, 1994, 999-1004.

[9] Coit, David W., Bonnie Turner Jackson and Alice E. Smith, "Static neural network process models: Considerations and case studies," *International Journal of Production Research*, vol. 36, no. 11, 1998, 2953-2967.

[10] Davies, P. J. and Fletcher, A. J., "The assessment of the rheological characteristics of various polyborosiloxane/grit mixtures as utilized in the abrasive flow machining process," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 209, ,1995, pp. 409-418.

[11] Fahlman, S. E. and Lebiere, C., "The cascade-correlation learning architecture," *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), San Mateo, CA: Morgan Kaufmann, 1990, pp. 524-532.
[12] Fletcher, A. J., Hull, J. B., Mackie, J. and Trengrove, S. A., "Computer modelling of the abrasive flow machining process," *Proceedings of the International Conference on Surface Engineering: Current Trends and Future Prospects*, Toronto, Ontario, Canada (Abington, Cambridge: Welding Institute, 1990, pp. 592-601.
[13] Garey, M. R., D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, 1976, vol. 1, 117-129.
[14] Glover, F., Tabu Search: Part I, *ORSA Journal on Computing*, 1989, vol. 1, 190-206.
[15] Glover, F., Tabu Search: Part II, *ORSA Journal on Computing*, 1990, vol. 2, 4-32.
[16] Glover, F. And M. Laguna, Tabu Search, *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves (Ed.), Blackwell Scientific Publishing, 1993.
[17] Karabati, S. and P. Kouvelis, "Cyclic scheduling in flowlines: Modeling observations, effective heuristics and a cycle time minimization procedure," *Naval Research Logistics*, 1996, vol. 43, 211-231.
[18] Karabati, S. and P. Kouvelis, "The interface of buffer design and cyclic scheduling decisions in deterministic flow lines," *Annals of Operations Research*, 1994, vol. 50, 295-317.
[19] Lam, Sarah, Kimberly Petri and Alice E. Smith, "A hierarchical system of neural networks and fuzzy logic: prediction and optimization for ceramic casting," under review for *IIE Transactions.*
[20] Lam, Sarah S. Y. and Smith, Alice E., "Process monitoring of abrasive flow machining using a neural network predictive model," *6th Industrial Engineering Research Conference Proceedings*, Miami Beach, FL, May 1997, pp. 477-482.
[21] Lam, Sarah S. Y. and Smith, Alice E., "Process control of abrasive flow machining using a static neural network model," *Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE '98*), St. Louis, MO, November 1998, ASME Press, New York, in print.
[22] Lawrence, S., "Supplement to Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques," GSIA Technical Report, Carnegie Mellon University, Pittsburgh, PA, 1984.
[23] Levin, J. and D. Dutta, "Computer-aided process planning for parallel machines," *Journal of Manufacturing Systems*, vol. 11, 1992, 79-92.
[24] Levin, J., D. Dutta, J. C. Bean, "PMPS: A prototype CAPP system for parallel machining," Technical Report, University of Michigan, 1993, 48103-2117.
[25] *Machining Data Handbook*, 3rd Edition, vol. 2, compiled by the Technical Staff of the Machinability Data Center, Cincinnati, Ohio: Metcut Research Associates Inc., 1980.
[26] Martinez, Sergio, Alice E. Smith and Bopaya Bidanda, "Reducing waste in casting with a predictive neural model," *Journal of Intelligent Manufacturing*, vol. 5, 1994, 277-286.
[27] Nawaz, M., E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, 1983, vol. 11, 91-95.
[28] *NeuralWorks Reference Guide* and Explorer software, NeuralWare, Pittsburgh, PA, various years.
[29] Norman, Bryan A., *Scheduling Using the Random Keys Genetic Algorithm*, Unpublished PhD. Dissertation, Univ. of Michigan, Ann Arbor, 1995.
[30] Norman, B. A., "Scheduling flowshops with finite buffers and sequence dependent setup times," Technical Report 96-8, Department of Industrial Engineering, University of Pittsburgh, 1996.
[31] Norman, Bryan A. and James C. Bean, "Scheduling operations on parallel machine tools," Technical Report 95-09, Department of Industrial Engineering, University of Pittsburgh, 15261, 1995. *IIE Transactions*, to appear.
[32] Norman, Bryan A. and James C. Bean, "Random keys genetic algorithm for complex scheduling problems," Technical Report 96-7, Department of Industrial Engineering, University of Pittsburgh, 15261, 1996. *Naval Research Logistics,* to appear.
[33] Norman, B. and J. Bean, "Random keys genetic algorithm for job shop scheduling," *Engineering Design and Automation*, vol. 3, 1996, 145-156.
[34] Petri, K. L., Billo, R. E. and Bidanda, B., "A neural network process model for abrasive flow machining operations," *Journal of Manufacturing Systems*, vol. 17, 1998, 52-64.
[35] Pinedo, M., *Scheduling Theory, Algorithms, and Systems*, Prentice Hall, 1995.

[36] Rhoades, L. J., "Abrasive flow machining: A case study," *Journal of Materials Processing Technology*, vol. 28, 1991, 107-116.

[37] Rhoades, L. J., "Abrasive flow machining," *Manufacturing Engineering*, vol. 101, 1988, 75-78.

[38] Rhoades, L. J., "Abrasive flow machining", *American Society of Mechanical Engineers, Production Engineering Division*, New York, NY: ASME, vol. 34, 1988, pp. 149-162.

[39] Smith, Alice E., "Predicting product quality with backpropagation: A thermoplastic injection moulding case study," *International Journal of Advanced Manufacturing Technology*, vol. 8, no. 4, 1993, 252-257.

[40] Smith, Alice E. and Hulya Yazici, "An intelligent composite system for plastic extrusion process control," *Journal of Engineering Applications of Artificial Intelligence*, vol. 5, no. 6, 1992, 519-526.

[41] Smith, S. C., "Four-door party animals," *Car and Driver*, vol. 42, 1997, 46-56.

[42] Twomey, J. M. and Smith, A. E., "Bias and variance of validation methods for function approximation neural networks under conditions of sparse data," *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 28, 1998, 417-430.

[43] White, Halbert, "Learning in artificial neural networks: a statistical perspective," *Neural Computation*, vol. 1, 1989, 425-464.

[44] Wilhelm, Mark, Alice E. Smith and Bopaya Bidanda, "Integrating an expert system and a neural network for process planning," *Engineering Design and Automation*, vol. 1, no. 4, 1995, 259-269.

[45] Wilhelm, Mark, Alice E. Smith and Bopaya Bidanda, "Process Planning Using an Integrated Neural Network and Expert System Approach," in *Hybrid Intelligent System Applications* (J. Liebowitz, editor), Cognizant Communications/ISIS, Elmsford, NY, 1996, pp. 3-23.

[46] Williams, R. E. and Rajurkar, K. P., "Metal removal and surface finish characteristics in abrasive flow machining," *Mechanics of Deburring and Surface Finishing Processes* (R. J. Stango and P. R. Fitzpatrick, eds.), New York, NY: ASME, vol. 38, 1989, pp. 93-106.

[47] Williams, R. E. and Rajurkar, K. P., "Stochastic modeling and analysis of abrasive flow machining," *ASME Journal of Engineering for Industry*, vol. 114, 1992, 74-81.

[48] Williams, R. E., Rajurkar, K. P. and Kozak, J., "metal removal distribution and flow characteristics in abrasive flow machining," *Transactions of NAMRI/SME*, Vol. XX, 1992, 145-150.

[49] Yip-Hoi, D., 1994, Personal communication.

[50] Yip-Hoi, D. and D. Dutta, "Issues in computer-aided process planning for parallel machines," *Advances in Design Automation*, ASME, DE vol. 65 part 1, 1993, 153-161.

[51] Yip-Hoi, D. and D. Dutta, "A genetic algorithm application for sequencing operations in process planning for parallel machining," *IIE Transactions*, vol. 28, 1996, 55-68.