

# **Using Genetic Algorithms as a Controller for Hot Metal Temperature in Blast Furnace Processes**

By Danny Lai

May 22, 2000

Advanced Undergraduate Project (6.199)

Supervised by Dr. Amar Gupta

## **Abstract**

The hot metal temperature (HMT) of a blast furnace is an important indicator of the furnace's internal state. Because the blast furnace system is non-linear, neural networks have been employed to approximate these systems. Artificial neural nets have been developed to approximate these dynamics and predict future HMT hours into the future, with limited accuracy. This paper focuses on inversion, focusing on the development of an HMT controller. The controller will, given a desired HMT and desired future time, determine the appropriate input values now in order to achieve the goals. Genetic algorithms will be used to derive the controller.

Implementation of the genetic algorithm required creating a genome representation, appropriate genome operators, a fitness function, and termination criteria. In each genome, functions were approximated using multiple data points. Roughly 13 data points were used, reflecting a HMT spread of 30 degrees above and below a set operating point. Input variables were limited to the 10-14 that showed highest consistent correlation with HMT for the given operating point. A basic genetic algorithm was employed, using constant mutation and crossover probabilities and a terminate-upon-generation criterion. Preliminary results showed that after a fixed number of generations, the algorithm did not reliably terminate with acceptable solutions. This phenomenon was likely due to the large genome strings (over 500 bits per genome) and loose evolution protocol. It is suggested that a more powerful platform be used in the future.

# TABLE OF CONTENTS

<b>INTRODUCTION</b> .....	<b>3</b>
<b>BACKGROUND</b> .....	<b>4</b>
GENETIC ALGORITHMS OVERVIEW .....	4
GENETIC ALGORITHMS DESCRIPTION.....	5
<i>Genome Representation</i> .....	6
<i>Initialization</i> .....	6
<i>Crossover</i> .....	7
<i>Mutation</i> .....	13
<i>Objective (fitness) function and generation transition specification</i> .....	14
<i>Termination conditions</i> .....	18
PREVIOUS WORK.....	18
<i>Neural Networks</i> .....	18
<i>Genetic Algorithms</i> .....	21
NNRUN DEVELOPMENTS.....	22
<b>PROCEDURE</b> .....	<b>23</b>
PROGRAM METHODOLOGY .....	23
PROGRAM SPECIFICATIONS.....	24
REPRESENTING THE GENOMES.....	25
<i>Genome representation</i> .....	25
<i>Genome operators</i> .....	28
FORMULATING THE FITNESS FUNCTION.....	30
<b>RESULTS</b> .....	<b>30</b>
GENETIC ALGORITHM SIMULATION.....	30
COMPARISON TO OTHER IMPLEMENTATION APPROACHES .....	32
<i>Inversion</i> .....	32
<i>Decision Trees</i> .....	33
<b>CONCLUSION</b> .....	<b>34</b>
IMPROVEMENTS/FUTURE WORK.....	35
EXTERNAL ISSUES .....	36
<b>REFERENCES</b> .....	<b>37</b>

## INTRODUCTION

The hot metal temperature (HMT) of a blast furnace is an important indicator of the furnace's internal state. The HMT must be maintained at appropriate levels to facilitate production of high quality pig iron. Hence, predicting and controlling the HMT would lead to a more efficient productive blast furnace system. Control theory has experienced a great deal of advancement over the last two decades, focusing mainly on systems with nonlinear dynamics. These nonlinear control methods work well if the system is linearizable; however, many linear approximations to nonlinear systems yield inconclusive models that do not provide a means to control the system. Current research employs neural networks to control such systems.

The approach of neural networks provides a more straightforward way to handle problems involving several inter-dependent variables. Neural networks are inherently parallel machines that have the ability of modeling complex mathematical functions between inputs and outputs even when the form of the function is unknown. They have the capability of "learning" by example. The neural network, instead of having to be programmed with data, learns the general relationships and patterns in the data by training on a set of examples (Winston, 1993). Since neural networks are capable of modeling non-linear relationships embodied in the data, these models have been and continue to be quite useful in modeling the complex relationships between various chemicals and parameters in the blast furnace. In fact, previous research at MIT has produced an artificial neural net, NNRUN, which predicts HMT based on normalized, lagged input values (Sarma, 1999).

The goal of this paper is to extend the research by investigating the applications of genetic algorithms in producing an HMT controller. The controller would be a black box application that can propose changes to make in each of the input variables now in order to achieve a desired HMT at a desired future time. Each controller would be specific to an operating temperature. The controller would approximate delta functions using several data points, and relies on the NNRUN program for verification.

## **BACKGROUND**

### **GENETIC ALGORITHMS OVERVIEW**

Genetic algorithms are collections of modeling, search, and optimization methods inspired by evolutionary biology. In evolutionary biology, species evolve via inherited variation, either through mutation, recombination, or some other process. By natural selection, the fittest survive and reproduce, thereby transmitting their genetic material to future populations. A genetic algorithm simulates the evolutionary process. Possible solutions are encoded as genomes (usually bit-vectors), and during each generation, the genomes are assigned a “fitness” based on a pre-determined fitness function. The genomes that score higher fitness survive to the next generation. In addition, these fitter genomes can be mutated, and can produce offspring (through recombination of bit segments) for the next generation. In the end, genomes with superior “fitness” evolve as the optimal solution to the problem.

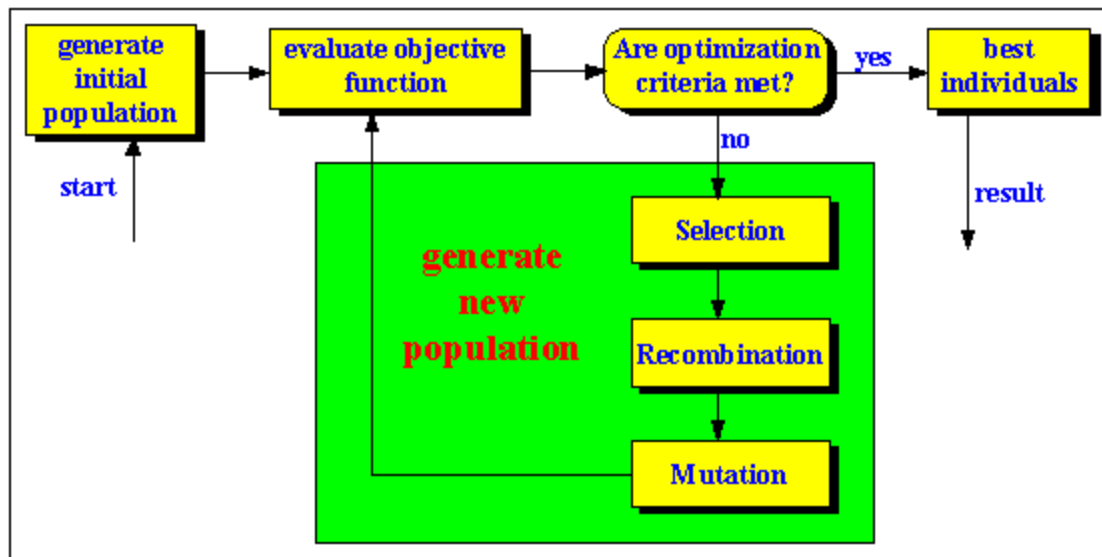
Genetic algorithms are especially powerful as optimization mechanisms in complex search spaces that could be discontinuous, multi-modal, or highly nonlinear. Unlike many classical optimization techniques, genetic algorithms do not rely on computing local derivatives to guide the search process. The search is probabilistic, not deterministic. Genetic algorithms can control a dynamic system without any prior knowledge about the system. They work directly with string of characters representing the parameter set, not the parameters themselves. In addition, genetic algorithms consider a population of points, not just a single point. Genetic algorithms have been used in conjunction with fuzzy logic controllers (FLCs) to design minimal spanning rule sets for implementing controllers (Varsek, Urbancic, and Filipic, 1993). They have also been used with limited success for controlling non-deterministic systems, or systems that can be only partially modeled.

Genetic algorithms work better than traditional symbolic AI systems because unlike the traditional systems, genetic algorithms are transferable. In traditional systems, the

architecture was designed for a specific problem. If the problem were to change, the system would produce a sub-optimal answer. Genetic algorithms can be applied to a wide variety of problem and search spaces. The concepts and basic operations are the same. Only the genome representation and fitness functions need to be changed.

## GENETIC ALGORITHMS DESCRIPTION

A brief overview of how a genetic algorithm works is described below:



First, a number of individuals (the population) are randomly initialized. The objective function is then evaluated for these individuals, producing the first generation of genomes. If the optimization criteria are not met, the creation of a new generation starts. Individuals are selected according to their fitness for the production of offspring. Parents are recombined (crossover) to produce offspring. All offspring will be mutated with a certain probability. The fitness of the offspring is then computed. The offspring are inserted into the population replacing the parents, producing a new generation. This cycle is performed until the optimization criteria are reached, or until a pre-set maximum number of generations have been generated.

Such a single population evolutionary algorithm is powerful and performs well on a broad class of problems. However, better results can be obtained by introducing many populations, called sub-populations. Every sub-population evolves in isolation for a few

generations (like the single population evolutionary algorithm) before one or more individuals are exchanged between the sub-populations. The multi-population evolutionary algorithm models the evolution of a species in a way more similar to nature than the single population evolutionary algorithm (Goldberg, 1989).

### **Genome Representation**

Genomes should represent data structures can capture the form of a solution, whether the solution is a real number or a list of characters. The representation should be minimal but completely expressive, and if possible, should be designed so that infeasible solutions to the problem cannot be represented. There is great flexibility in choosing the genome representation, but when designing the representation, one should consider how it would affect the behavior of mutators during each generation. For instance, a representation that mixes integers with floating point numbers will complicate genome operations, as the program must make sure that during crossover or mutation, floating point numbers are not inserted into slots where an integer is expected.

Candidate solutions are usually represented as strings of fixed length, called chromosomes (may be used interchangeably with genomes), coded with a binary character set. Several parameters can be coded into one long string. The string length determines the resolution. Generally, it is better for encoded bit-string lengths to be kept as short as possible, since the size of the search space increases exponentially with string size.

### **Initialization**

Each genome has three primary operators: initialization, crossover, and mutation. The initialization operator simply initializes the population with the pre-specified genetic information from which all solutions will evolve. The extent to which the initialization operator is randomized depends on the developer's desire for a wide search space, keeping in mind that wide search spaces will take more time if the algorithm is not executed in parallel (Goldberg, 1989). The more randomized the initialization, the wider the search space.

## Crossover

Crossover is a systematic information exchange between two strings, implemented using probabilistic decisions. In a crossover process, two newly reproduced strings are chosen from the mating pool and arranged to exchange their corresponding portions of binary strings at a randomly selected partitioning position along them. Crossover combines better qualities among the preferred good strings, and can be implemented in a variety of ways, depending on the set of crossover points.

### *Real-valued recombination*

Discrete recombination performs an exchange of variable values between the individuals. Consider the following two individuals with 3 variables each (3 dimensions), which will also be used to illustrate the other types of recombination:

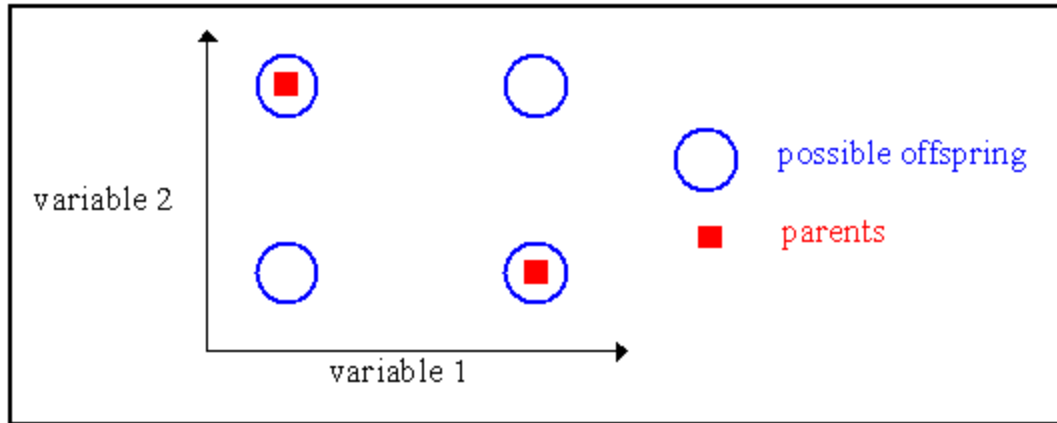
Individual 1	12	25	5
Individual 2	123	4	34

For each variable the parent who contributes its variable to the offspring is chosen randomly with equal probability. In the below table, Sample 1 gets variables 1 and 2 from Individual 2, and variable 3 from Individual 1.

Sample 1	2	2	1
Sample 2	1	2	1

After recombination the new individuals are created:

Offspring 1	123	4	5
Offspring 2	12	4	5



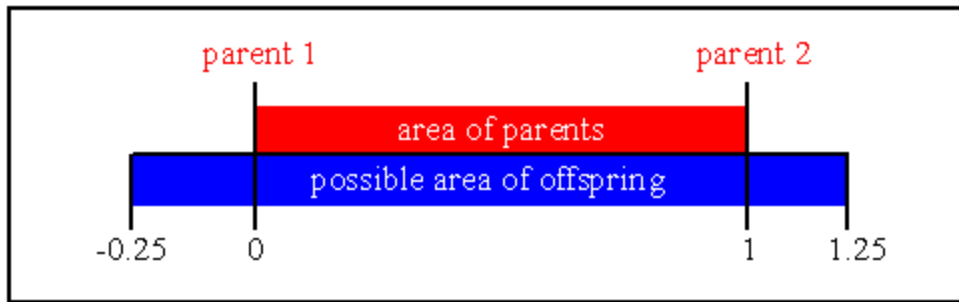
(Harmut, 1991)

Discrete recombination generates corners of the hypercube defined by the parents. The above figure shows the geometric effect of discrete recombination. Discrete recombination can be used with any kind of variables (binary, real or symbols).

Intermediate recombination is a method only applicable to real variables. Here, the variable values of the offspring are chosen somewhere around and between the variable values of the parents. Offspring are produced according to the rule:

$$\text{Offspring} = \text{Parent}_1 + \alpha(\text{Parent}_2 - \text{Parent}_1)$$

The variable  $\alpha$  is a scaling factor chosen uniformly at random over an interval  $[-d, 1 + d]$ . In standard intermediate recombination  $d = 0$ . For extended intermediate recombination,  $d > 0$ . A good choice is  $d = 0.25$ . Each variable in the offspring is the result of combining the variables according to the above expression with a new Alpha chosen for each variable. On the next page is a picture of the area of the variable range of the offspring defined by the variables of the parents:



(Harmut, 1991)

Consider again the following two individuals with 3 variables each:

Individual 1	12	25	5
Individual 2	123	4	34

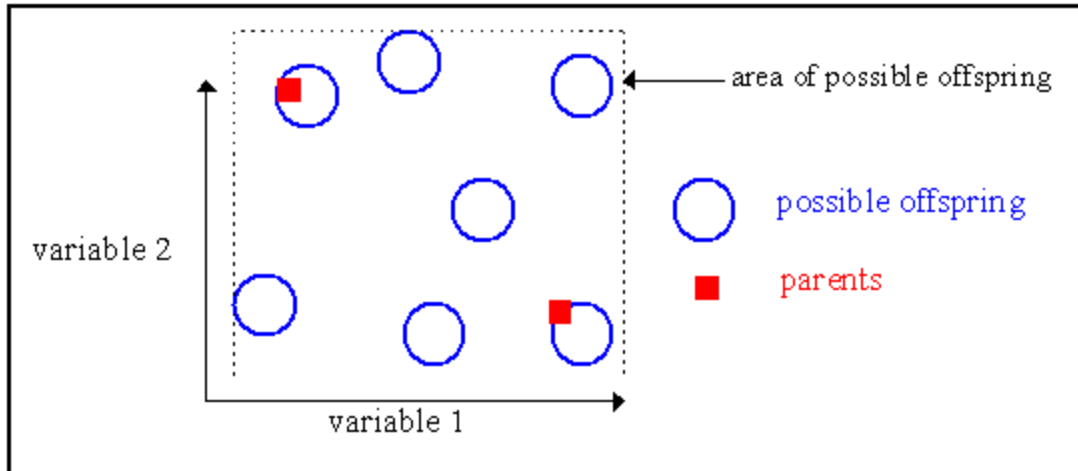
The chosen  $\alpha$  for this example are:

Sample 1	0.5	1.1	-0.1
Sample 2	0.1	0.8	0.5

The new individuals are calculated as:

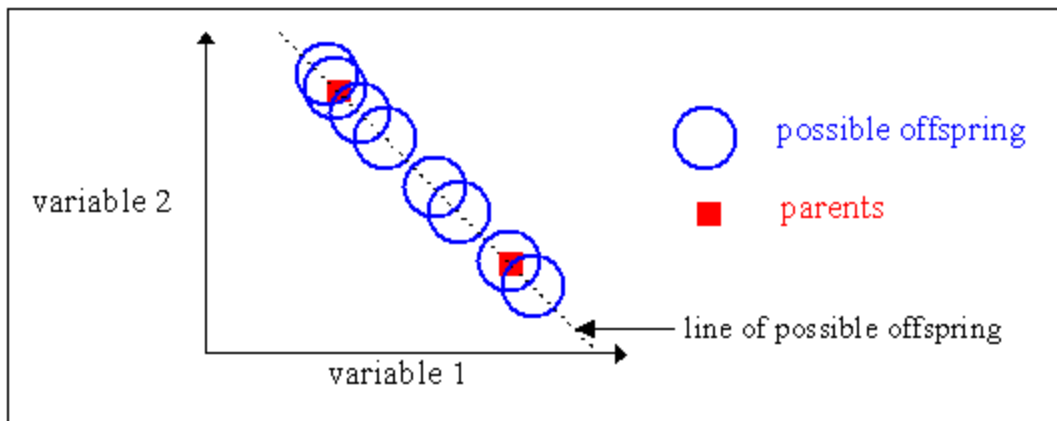
Offspring 1	67.5	1.9	2.1
Individual 2	23.1	8.2	19.5

Intermediate recombination is capable of producing any point within a hypercube slightly larger than that defined by the parents, thus allowing for greater diversity. The following figure shows the possible area of offspring after intermediate recombination.



(Harmut, 1991)

Line recombination is similar to intermediate recombination, except that only one value of  $\alpha$  is used for all variables. Line recombination can generate any point on the line defined by the parents. The following figure shows the possible positions of the offspring after line recombination.



(Harmut, 1991)

*Binary-valued recombination*

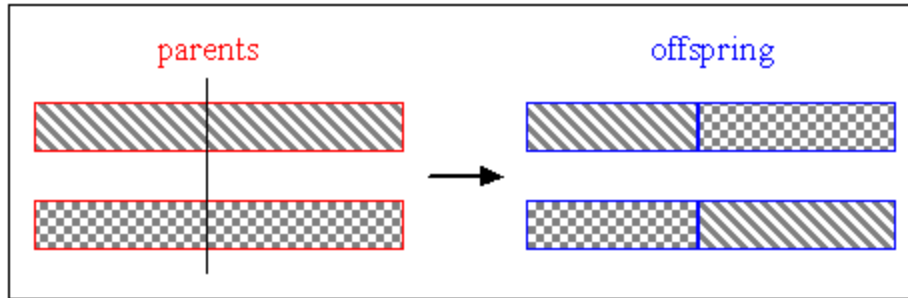
In single-point crossover one crossover position – chosen from the set of variables in the chromosome – is selected uniformly at random, and the variables are exchanged between the individuals about this point, producing two new offspring. Consider the following two individuals with 11 binary variables each:

Individual 1	0	1	1	1	0	0	1	1	0	1	0
Individual 2	1	0	1	0	1	1	0	0	1	0	1

With a crossover position being the fifth position, the new individuals are:

Offspring 1	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	1	0	0	1	0	1
Offspring 2	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	0	1	1	0	1	0

The process is generalized in the figure below:



(Harmut, 1991)

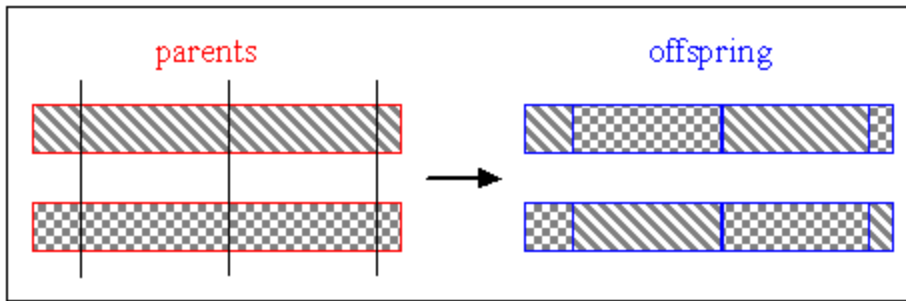
For multi-point crossover, multiple crossover positions are chosen at random with no duplicates, and then sorted in ascending order. Then, the variables between successive crossover points are exchanged between the two parents to produce two new offspring. The section between the first variable and the first crossover point is not exchanged between individuals. To illustrate, consider the following two individuals with 11 binary variables each:

Individual 1	<b>0</b>	<b>1</b>	1	1	0	0	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	0
Individual 2	<b>1</b>	<b>0</b>	1	0	1	1	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	1

Consider a three-point crossover, with crossover positions at the second, sixth, and tenth positions. The new individuals are:

Offspring 1	<b>0</b>	<b>1</b>	1	0	1	1	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	1
Offspring 2	<b>1</b>	<b>0</b>	1	1	0	0	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	0

The process is generalized in the figure below:



(Harmut, 1991)

The idea behind multi-point, and indeed many of the variations on the crossover operator, is that parts of the chromosome representation that contribute most to the performance of a particular individual may not necessarily be contained in adjacent sub-strings. Further, the disruptive nature of multi-point crossover appears to encourage the exploration of a wider search space, rather than favoring the convergence to highly fit individuals early in the search. This globalization helps prevent the search from getting stuck at local maxima, thus making it more robust.

Uniform crossover generalizes the single and multi-point crossover schemes. This scheme makes every locus (position between any two adjacent variables) a potential crossover point. A crossover mask, whose length is the same as the individual structure, is created at random. The parity of the bits in the mask indicates which parent will supply the offspring with which bits. Consider the following two individuals with 11 binary variables each:

Individual 1	<b>0</b>	1	<b>1</b>	1	<b>0</b>	0	<b>1</b>	1	<b>0</b>	1	<b>0</b>
Individual 2	<b>1</b>	0	<b>1</b>	0	<b>1</b>	1	<b>0</b>	0	<b>1</b>	0	<b>1</b>

For each variable, the parent who contributes its variable to the offspring is chosen randomly with equal probability, according to the mask:

Mask	0	1	1	0	0	0	1	1	0	1	0
------	---	---	---	---	---	---	---	---	---	---	---

The first offspring is produced by taking the bit from the first parent if the corresponding mask bit is 1, or from the second parent if the corresponding mask bit is 0. The second offspring is usually created using the inverse of the mask. Thus, the new offspring are:

Individual 1	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
Individual 2	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

Uniform crossover, like multi-point crossover, has been claimed to reduce the bias associated with the length of the binary representation used and the particular coding for a given parameter set. This helps to overcome the bias in single-point crossover towards short sub-strings without requiring precise understanding of the significance of the individual bits in the individual's representation. Spears and De Jong have demonstrated how uniform crossover may be parameterized by applying a probability to the bit swapping. This extra parameter can be used to control the amount of disruption during recombination without introducing a bias towards the length of the representation used. The algorithm of uniform crossover is identical to that of discrete recombination.

Shuffle crossover is related to uniform crossover. A single crossover position (as in single-point crossover) is selected. But before the variables are exchanged, they are randomly shuffled in both parents. After recombination, the variables in the offspring are unshuffled. This removes positional bias as the variables are randomly reassigned each time crossover is performed. Shuffle crossover is rarely used, primarily because it may be computationally expensive.

### **Mutation**

After recombination, a genome may undergo mutation. Mutators introduce new genetic material into the population. The mutation operator prevents irreversible loss of certain patterns by introducing small random changes into chromosomes (Wall, 1996). That is, it will prevent the GA from being trapped at local maxima. However, mutation may suddenly spoil the opportunity of the current appropriate generation, so this process usually occurs with a small probability. It is a common practice to set the mutation rate to  $1/n$ , where  $n$  is the number of variables (dimensions) in the genome. Sheble and

Britting showed that dynamic crossover and mutation probability rates provide faster convergence in GA when compared to constant probability rates. Implementation of mutators depends heavily on the genome representation, because they do different things, depending on the implementation. For example, a common mutator for a binary string genome flips each bit in the string with a given probability. For a genome represented by a tree, on the other hand, a mutator would swap sub-trees with a given probability (Wall, 1996).

### **Objective (fitness) function and generation transition specification**

The fitness function is used to determine which genomes are better solutions to the problem. Based on the fitness function, each genome in a generation is assigned a “fitness” score. The genomes that score higher fitness survive to the next generation (and can reproduce or mutate in the process). Fitness values can be normalized, scaled, shared, or left unchanged.

One must also specify the reproduction algorithm: how new genomes will replace genomes from the previous generation. The population at each generation should remain constant, to avoid an exponentially increasing search breadth. Sometimes, the entire population is replaced in each cycle. Other times, only a subset of the population is replaced. A generation gap value determines the percentage of a population that is replaced. In addition, a certain number of best individuals can be guaranteed to stay in the population; this principle is called elitism (Varsek, 1993).

What follows are some common selection schemes. In evaluating these schemes, some of the following metrics were used:

- **Selective pressure** - probability of the best individual being selected compared to the average probability of selection of all individuals
- **Bias** - absolute difference between an individual's normalized fitness and its expected probability of reproduction
- **Spread** - range of possible values for the number of offspring of an individual
- **Loss of diversity** - proportion of individuals of a population that is not selected during the selection phase

- **Selection intensity** - expected average fitness value of the population after applying a selection method to the normalized Gaussian distribution
- **Selection variance** - expected variance of the fitness distribution of the population after applying a selection method to the normalized Gaussian distribution

*Roulette-wheel selection*

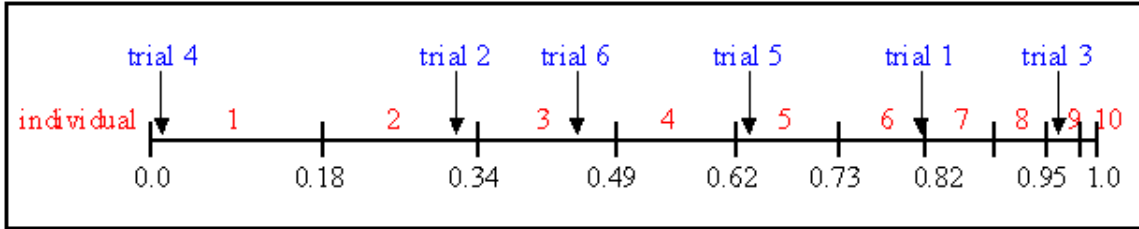
The simplest selection scheme is roulette-wheel selection, also called stochastic sampling with replacement. This is a stochastic algorithm and involves the following technique:

The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness. A random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained (called mating population). This technique is analogous to a roulette wheel with each slice proportional in size to the fitness (Goldberg, 1989).

The table below shows the selection probability for 11 individuals, using linear ranking together with the fitness value. Individual 1 is the fittest individual and occupies the largest interval, whereas individual 10 as the second least fit individual has the smallest interval on the line. Individual 11, the least fit interval, has a fitness value of 0 and gets no chance for reproduction.

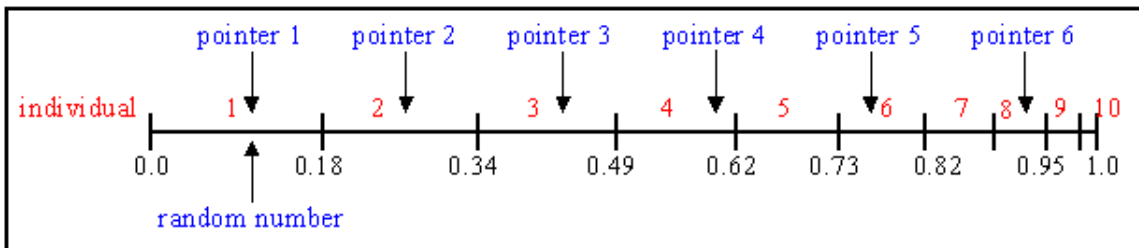
Number of individual	1	2	3	4	5	6	7	8	9	10	11
Fitness value	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2	0.0
Selection probability	0.18	0.16	0.15	0.13	0.11	0.09	0.07	0.06	0.03	0.02	0.00

For selecting the mating population, the appropriate number of uniformly distributed random numbers (uniform distributed between 0.0 and 1.0) is independently generated. As an example, six random numbers were generated: 0.81, 0.32, 0.96, 0.01, 0.65, and 0.42. The following figure shows the selection process of the individuals for the above example.



### *Stochastic Universal Sampling*

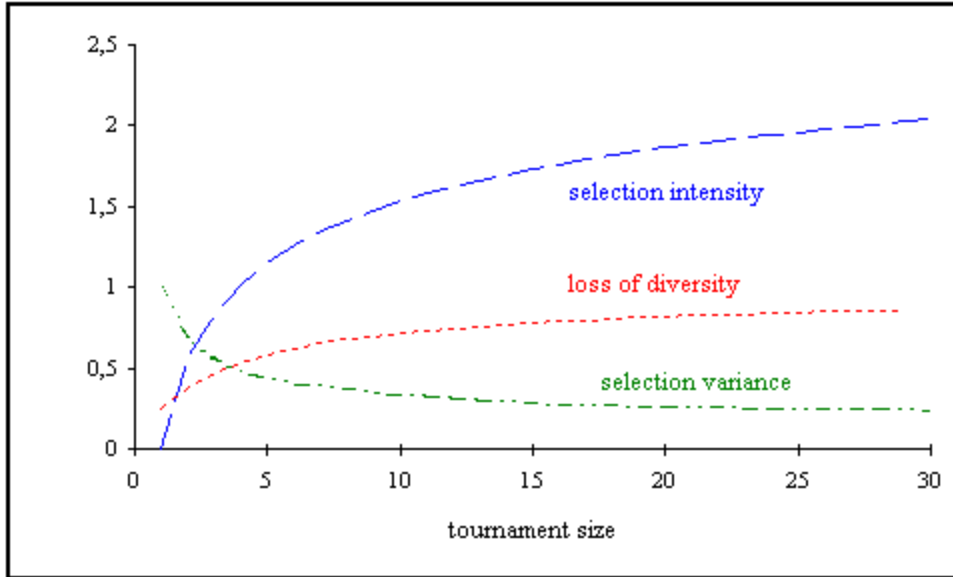
Stochastic universal sampling provides zero bias and minimum spread. The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness exactly as in roulette-wheel selection. Here, equally spaced pointers are placed over the line as many as there are individuals to be selected. Refer back to the example used in the roulette-wheel selection. In stochastic universal sampling, the pointers are distributed uniformly, not randomly:



Stochastic universal sampling ensures a selection of offspring which is closer to what is deserved than roulette wheel selection (Goldberg, 1989).

### *Tournament Selection*

In tournament selection a number (the “tournament size”) of individuals is chosen randomly from the population, and the best individual from this group is selected as a parent. If the tournament size is four, then four parents will be chosen. The following figure shows how selection intensity, selection variance, and loss of diversity are affected by the tournament size.



(Hartmut, 1991)

### *Rank-based Fitness*

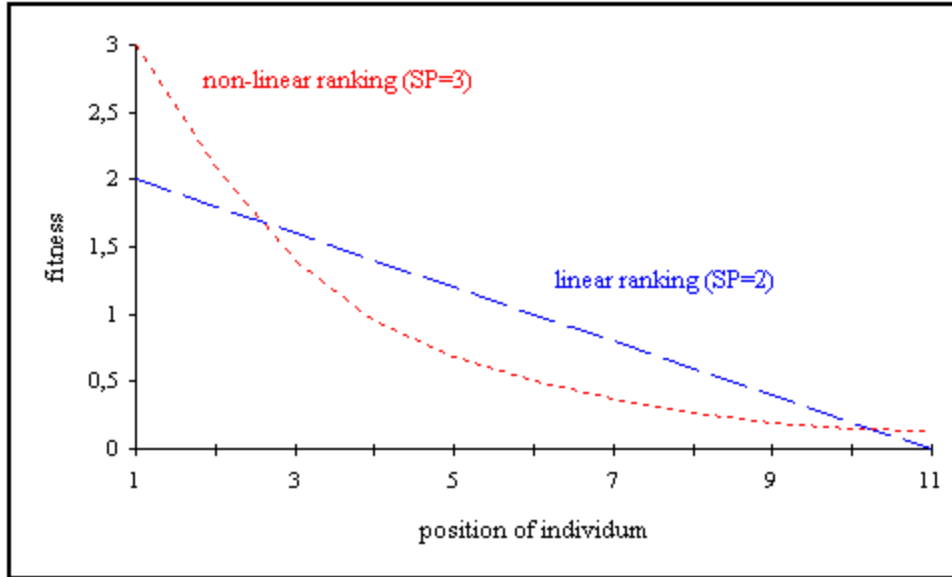
Rank-based fitness assignment overcomes the scaling problems of the proportional fitness assignment. The reproductive range is limited, so that no individuals generate an excessive number of offspring. Ranking introduces a uniform scaling across the population and provides a simple and effective way of controlling selective pressure (Hartmut, 1991).

As an example, let  $N_{ind}$  be the number of individuals in the population,  $x_{pos}$  be the position of individual  $x$  in this population (where the least fit individual has position 1, and the fittest individual has position  $N_{ind}$ ), and  $SP$  be the selective pressure. The fitness value for individual  $x$  is calculated as:

Linear ranking:

$$\text{Fitness}(x) = 2 - SP + 2 \cdot (SP - 1) \cdot (x_{pos} - 1) / (N_{ind} - 1)$$

Linear ranking allows values of selective pressure in [1.0, 2.0].



(Hartmut, 1991)

The probability of each individual being selected for mating is its fitness, normalized by the total fitness of the population.

### Termination conditions

Genetic algorithms will typically run forever, until an appropriate termination condition is reached. The termination condition usually takes one of two forms. The “terminate-upon-generation” condition stops the program after a fixed number of generations have been simulated, at which point the best existing solution is returned. The “terminate-upon-convergence” condition terminates the program if the best existing solution scores a fitness level that is above a pre-set threshold level. These two termination strategies can be combined as well.

### PREVIOUS WORK

#### Neural Networks

Much work has been done involving neural networks to predict state variables of a blast furnace. Abhay Bulsary, Henrik Saxen, and Bjorn Saxen observed promising results when using multi-layered feed-forward artificial neural network (ANN) models to predict the silicon content of hot metal from a blast furnace (1995). Time-dependencies (time lags) between each of the various inputs used and the output (silicon content) were found

to be significant. For this reason, each input was lagged from the output by a set amount. The amount is determined by the correlation between the input variable and the output. Ideally, each input should be lagged so that the correlation between the input variable and the output is maximized. The input variables used included: blast pressure (as 15 minute averages, with time lags of 30 minutes and one hour), blast volume (as one hour averages, with a time lag of five hours), calculated heat loss (as one hour averages, with time lags of one hour and seven hours), oil injection (one hour averages lagged by five hours), and the silicon content of the previous two taps. Feed-forward networks with hidden layers were tried. The neural network's weights were updated using a non-linear variant of the traditional back-propagation learning algorithm. The work showed that the feed-forward ANNs had produced considerably better results than standard linear time-series prediction. Since silicon content is a fairly reliable indicator of HMT, the success here in predicting silicon content using ANNs provides an indication that ANNs may also be useful in the prediction of HMT (Bulsari, Saxen, and Saxen, 1992).

Feed-forward neural networks were used again by Bulsari and Saxen to classify the state of a blast furnace based on the measurement of blast furnace temperatures. Internal blast furnace temperature provides information concerning the internal gas distribution of the blast furnace. However, the relationship between blast furnace temperature and gas distribution within the furnace is highly complex and non-linear, and, traditionally, had only been inferred by people who have had extensive experience operating the blast furnace. Thus, a neural network seemed an appropriate solution. Based on the measurements of the horizontal temperature probes, the network classified the state of the gas distribution in the blast furnace into one of 6 categories. Bulsari and Saxen constructed a feed-forward network using back-propagation in order to train the network. One major result was that larger networks, with more hidden nodes and hidden layers, had greater success than smaller networks. As the number of hidden nodes decreased, the accuracy of the network also decreased. The worst model, according to Bulsari and Saxen, was the linear regression model. Thus, since more hidden nodes provided better results, Bulsari and Saxen claimed that the larger networks were more capable of

capturing the complex relationships between the variables in the system (Bulsari and Saxen, 1995).

Himanshu Singh, Nallamal Venkata and Bramha Deo tried four different artificial neural network models in order to predict the silicon content of hot metal based on the following set of variables: coke rate, hot blast temperature, slag rate, top pressure, slag basicity and blast kinetic energy (1996). The learning algorithm used was standard back propagation. The networks consisted of three layers (input, hidden and output), and the number of hidden nodes varied from 6 to 11. Other models tried included using a dynamic learning rate model, functional link model, and a fuzzy neural network. The best results were obtained from the fuzzy neural network, with the performance of the back-propagating model providing the next best results. The results of Singh's paper clearly show that the use of ANNs increased the predictive power of silicon content when compared conventional models (Singh, Sridhar, and Deo, 1996).

Osamu Lida, Yuichi Ushijima and Toshiro Sawada used neural networks in order to implement a module of a blast furnace control system. The blast furnace control system diagnoses the operating condition of the blast furnace from a large amount of data and gives advice to the operator about the necessary changes to make in order to stabilize the state of the furnace, or to reach some particular condition within the furnace. In particular, neural networks were used to explain the relationship between gas flow distribution and three critical parameters: the charging pattern permeability in the furnace, the HMT, and the hot metal silicon content. The type of neural network used was self-organized feature mapping. In this neural network model, neurons are located in a 2-dimensional arrangement and all neurons have dimensional connectivity weights whose number is equal to the number of input variables in the data. The weights of these connections are repeatedly updated through learning. This work showed the ability of neural networks to capture non-linearities in blast furnace data to provide reasonable classification of the current conditions in the furnace (Lida, Ushijima, and Toshiro, 1992).

## **Genetic Algorithms**

Unfortunately, there has been very limited work using genetic algorithms to control internal blast furnace processes. However, there has been extensive research into general controller applications using genetic algorithms, and many of them have provided the methodology and tools which are portable to the steel blast furnace.

One of the most successful applications of GA's was in the benchmark problem of inverted pendulum control. This problem was studied by Chin and Qi as well as by Varsek, Urbancic, and Filipic. The system to be controlled consists of a cart and a rigid pole hinged at the top of the cart. The cart can move along one dimension, and the pole can swing in the vertical plane determined by the cart. Essentially, the goal is to provide a controller that can balance the pole. A genetic algorithm was used to implement the controller, because the system is nonlinear and unstable, and more than one system variable is to be controlled by a single input variable. The initial population size was 50 genomes, and the algorithm employed a terminate-upon-generation strategy, running for a maximum of 100 generations. The system employed uniform crossover with probability 0.8 and a mutation probability of 0.1, and a fitness function of minimum time-weighted integral of squared errors. The genetic algorithm optimized the rule base controller for the inverted pendulum system, and demonstrated that the GA-based controller had better performance than traditional differential systems analytical methods. (Chin and Qi, 1998; Varsek, Urbancic, and Filipic, 1993)

Badran and Al-Duwaish applied genetic algorithms to create an optimal output feedback controller to improve the damping of power system oscillations when subject to a disturbance. The GA was used to search for the optimal values of the feedback gains to minimize the given performance index and ensure the stability of the closed loop system.

The GA used a randomized initial population, a fixed population size of 30, and crossover and mutation probabilities of 0.7 and 0.01, respectively. Simulation results showed that the damping of the AC/DC system is improved by the output feedback obtained using the proposed GA design approach. This work showed that GAs offer greater flexibility in the choice of different objective functions and different output

feedback structures where closed form analytical solutions are not available. These are all properties of the blast furnace system, so the results imply that the blast furnace system may be a good candidate for a GA-implemented controller. (Badran and Al-Duwaish, 1999)

Lian, Marzuki, and Rubiyah tuned a neuro-fuzzy controller using genetic algorithms to control liquid levels in a coupled-tank plant system. The coupled-tank plant system is particularly interesting, because it has many of the same characteristics as a blast furnace system. The plant dynamics involve higher-order transfer functions, are highly nonlinear, and exhibit time delays associated with certain parameters. The genetic algorithm was highly customized to span a wide search space. The initial GA population was randomized, and exponentially decreasing crossover and mutation probability rates were used. Two-point crossover was applied in exchanging the gene information.

For each generation, 200 chromosomes were evaluated. Lian, Marzuki, and Rubiyah found that the GA-based fuzzy-logic controller performed better than conventional FLCs in terms of step response, load disturbance, and changes in plant dynamics. The results also showed promise for implementation for real-time control. (Lian, Marzuki, and Rubiyah, 1998)

## **NNRUN DEVELOPMENTS**

This research is an extension to an ongoing effort to monitor, predict, and control the hot metal temperature (HMT) of a steel blast furnace. Ravindra Sarma investigated the ability of feed-forward neural networks to predict HMT. His project produced NNRUN, a program that trains and tests artificial neural nets based on steel plant data designed for execution entirely in MATLAB. From the black-box perspective, the program takes as input selected variables and outputs a HMT prediction one, two, four, or eight hours into the future. Within the black box, the data is normalized, lagged, and interpolated as necessary. (Sarma, 1999)

To supplement Sarma's work, Brian Purville used time-delayed neural networks to produce the artificial neural network (2000). The data that he had to work with was raw

data from Steel Corp. Purville formatted and normalized the input values, and performed individual derivative analysis to select a subset to represent in the neural net. In training the neural net, optimizations such as variable time delays, moving window averages, and incorporation of last-known HMT reduced the mean-squared error of the predictions. His results are summarized in the following table:

<b>Prediction Horizon (hours)</b>	<b>Average Error</b>	<b>Mean Squared Error</b>	<b>Normalized Mean Square Error</b>
1	0.0651	0.01534	0.3824
2	0.0875	0.02480	0.5903
3	0.0965	0.03066	0.7507
4	0.0969	0.03401	0.8285

(Purville, 2000)

## **PROCEDURE**

### **PROGRAM METHODOLOGY**

Deriving an effective controller for the steel blast furnace requires a minimal spanning set of fuzzy rules. It has been established that the blast furnace system is a nonlinear system characterized by higher-order transfer functions and multiple independent time delays. These complexities make a controller modeled after standard decision trees and logic tables difficult to implement. Decision trees and tables grow exponentially with the number of interdependent input variables, and are an inflexible representation of human decision making. Fuzzy controllers allow for a nonlinear output mapping of a number of inputs using fuzzy linguistic variables and fuzzy rules. Previous research has shown that genetic algorithms can simplify and optimize the rule set. Genetic algorithms process candidate solutions independently, making it highly adaptable to parallel processing.

Genetic algorithms require only information concerning the quality of the solution produced by each parameter set; this differs from many optimization methods which require derivative information or in some cases, complete knowledge of the problem structure and parameters. Thus, genetic algorithms are efficient and easy to implement.

However, the disadvantage to using genetic algorithms is that when one uses them for controlling a system, he does not learn as much about the system's processes. Genetic algorithms produce rule sets for the controller, but these rule sets are not particularly useful to the operator, since they were derived using bit manipulation instead of disciplinary inferences.

The nature of the blast furnace system is such that the mapping of input to output values is still not very well understood. When controllers are to be built without having an accurate mathematical model of the system to be controlled, two challenges arise. First, one must establish the structure of the controller. This problem has been researched extensively, and many strategies have been developed, including qualitative model-based approach and machine learning. Second, the numerical values of the controller parameters must be set. This task assumes some knowledge of mapping input parameters to output values. For this problem, the mapping is not well understood, so a broad search is necessary. Genetic algorithms are especially suited to these searches, because they can be processed in parallel. Thus, genetic algorithms should be used to learn controller structure from scratch and to tune controller parameters.

A genetic algorithm was used to create a controller for the HMT. Given current conditions (specified by the current HMT, which is referred to as the “operating point,” and current values of the input variables), the solution should determine what changes need to be made to each variable in order to achieve a desired HMT at a desired time (some hours into the future). To avoid overly complex genomes, the problem space has been separated out. Each run of the genetic algorithm will produce a solution that is suitable for one operating point and one desired time frame. Because the ranges for both operating point (roughly 1400 degrees Celsius to 1500 degrees Celsius) and desired time frame (one to eight hours) are small, this fragmentation will not be tedious.

## **PROGRAM SPECIFICATIONS**

MATLAB was chosen to be the developing environment for the HMT controller. Extensive genetic algorithm libraries exist for the C++ environment. Matthew Wall

created a comprehensive C++ genetic library for MIT's Mechanical Engineering Department, and SGI has is developing genetic algorithm applications within the MLC++ group for their MineSet product, an optimizer based on genetic algorithms. However, MATLAB was chosen for this project because of compatibility issues. NNRUN is an essential component of this project. Because NNRUN was written for execution in MATLAB, it was decided that the controller should also be written in MATLAB, so code portability would be trivial. Simpler genetic algorithms can be found in MATLAB as well; one of them will be used as the foundation for this project. This program is based on generic genetic algorithm code in the MathWorks Library.

## **REPRESENTING THE GENOMES**

### **Genome representation**

Ideally, the genome would be a set of functions, with each function specifying exactly how much to change a variable based on the HMT gap and future time. However, there is no effective way to encode functions as bit streams. Thus, delta functions will be approximated using several data points.

Candidate solutions take the form of an array of  $r$  rows and  $c$  columns. Each column refers to a specific change in the HMT. Changes to the HMT will be expressed as an absolute change. Because the HMT range spans only 100 degrees – a mere 7% of its standard operating point – expressing changes as a percentage will not be precise. Each row refers to an input variable. Array element  $A[n,m]$  means that for the HMT to change by  $m\%$ , variable  $n$  must change according to the following formula (recall that all values are normalized between 0 and 1):

Let  $n'$  be the new value of variable  $n$ :

If  $A[n,m]$  is positive:  $n' = n + (1-n)*A[n,m]$

If  $A[n,m]$  is negative:  $n' = n*(1-A[n,m])$

Otherwise,  $n' = n$

To minimize computation and improve efficiency, not all of the proposed variables will be represented in the array (that is, it will be presumed that they will remain constant). We will use the variables that have the most impact on the HMT. Previous work on neural nets has yielded data on the derivative of the HMT with respect to the different variables for a given operating point (1450 degrees Celsius). The results are shown on the next page:

<b>VARIABLE NAME</b>	<b>CORRELATION COEFF. WITH HMT</b>
RAFT temperature	-0.3409
Group 2 heat flux	-0.3134
Group 1 heat flux	-0.3065
Steam	0.2491
Actual Coal injection (kg/thm)	-0.2117
Hydrogen (from top gas analysis)	0.1860
Central working index	0.1466
Total quartz	0.1240
Differential pressure	-0.1199
Total limestone	-0.1132
Wind volume	-0.1092
Coke moisture from bin #9	-0.1052
Top gas % hydrogen	0.1010
Total dolo	0.0898
Coke moisture from bin #7	-0.0788
Hot blast temperature (deg. C)	-0.0759
% carbon dioxide	-0.0629
Top pressure (kg/sq. meters)	0.0627
Ore/Coke ratio	-0.0625
BGVOL	-0.0619
Hot blast pressure	-0.0606
Cold blast pressure	-0.0594
Charge time for 10 semi-charges	0.0584
Total Ore Charged	-0.0538
% Sinter in Burden	0.0529
Total sinter	-0.0529
Total nut coke	-0.0437
% Oxygen Enrichment	-0.0393
Total coke	0.0310
Carbon oxide (from top gas analysis)	-0.0235
Permeability Index, K	0.0148
% carbon oxide	0.0070
Total Ti-ore	0.0065
Theoretical carbon rate	0.0009

(Sarma, 1999)

Many of the measured inputs are lagged as necessary. The correlation coefficients represent a relationship between input and output values for only a single operating point; it is not representative of the blast furnace system in general. The 11 chosen variables to include in the representation were taken from those used in the ANN. They were: total coke, carbon oxide (from top gas analysis), hydrogen (from top gas analysis), steam (tons/hr), group 1 heat flux, group 2 heat flux, actual coal injection, % oxygen enrichment, ore/coke ratio, hot blast temperature (degrees Celsius), and charge time for 10 semi charges.

It was found that genome operators – especially crossover operators – were more easily implemented if the genomes were represented as long bit stream. Sub-strings can be more easily spliced and recombined using vector operations; swapping rows and columns is more complicated. Thus, the candidate solutions were converted to the bit stream genome representation using the following scheme: Each array element value will be encoded using four bits (accuracy of up to three significant digits). The values are then concatenated into the bit stream in row-order format (the array is traversed from left to right, top to bottom).

### **Genome operators**

Genomes were initialized to have randomized delta functions. If extensive derivative analysis between the various inputs and HMT had been done for various operating points, the extra knowledge could allow initialization values which are closer to ideal solutions. Currently, such analysis exists for only a couple of operating points, which does not allow for general use of derivative analysis to initialize genomes.

The mutation probability for each input field was set at a high value, because it is believed that there are several optimal controllers. A higher mutation probability increases the search space and search time, but also increases the diversity of the population and prevents the algorithm from termination with a large cluster near a local maximum.

In deciding the crossover operators and selection schemes, the genome representation must be analyzed for possible optimizations. Because the encoded values were stored in row-order format, contiguous “entries” (a four-bit segment) in the genome corresponded to the same variable. In other words, the first 11 “entries” corresponded to variable 1, representing the response to  $\Delta\text{HMT}_1$ ,  $\Delta\text{HMT}_2$ , . . .  $\Delta\text{HMT}_{13}$  (where  $\Delta\text{HMT}_1 = -30$  degrees,  $\Delta\text{HMT}_2 = -25$  degrees, . . .  $\Delta\text{HMT}_{13} = +30$  degrees). The next 11 “entries” corresponded to variable 2 with the same pattern, etc. Thus, it follows that the chunks of 11 entries are somewhat correlated to each other, since they represent the responses of a single variable to a steadily increasing temperature gradient. Also, because the temperature steps are small (5 degrees), contiguous “entries” should not differ drastically in value. These observations will be useful in determining crossover and selection schemes.

The crossover scheme should be chosen to minimize disruption between the 11-entry chunks in the bit stream. A single-point crossover scheme is not useful, because there are several such chunks, and having only one crossover point will not produce enough diversity within the populations. On the other extreme, a uniform crossover scheme is not appropriate, because the randomly generated bit mask will destroy the intrinsic pattern discovered in the genome. Thus, a multi-point crossover scheme was chosen. Higher probabilities were assigned at the boundaries between two variables. This way, the 11-entry chunks are more likely to move as a group (which more accurately represents moving a “function”). Non-boundary crossover points were assigned non-zero probabilities to increase the search space increase population diversity. If it is found that the disruption caused by these non-zero probabilities merely increase the search space without finding other relative maxima, subsequent runs of the algorithm can employ a zero probability.

The selection criteria will be stochastic universal sampling. Because the genome bit streams are large, and because the algorithm will be running on single processor systems, a sensitive issue is time to convergence. Stochastic universal sampling is more fair than roulette-based selection scheme, since it uses fixed pointers instead of randomly

generated ones. Rank-based selection will not work well for this program, because it is assumed that several optimal controllers exist. If many of the genomes have high fitness, they should remain in the population with high probability. Rank-based selection will punish potentially good genomes if there are lots of others who have high fitness.

The program will first execute a “terminate-upon-generation” strategy in order to fine-tune the mutation probabilities and crossover algorithms. Once the algorithm has passed stability tests, the strategy will be switched to “terminate-upon-convergence,” ideally producing a family of optimal controllers.

### **FORMULATING THE FITNESS FUNCTION**

For a genome, each column will be tested individually. We utilize the black-box NNRUN for the fitness test. The current HMT and proposed new values for the input variables will be fed into NNRUN, and the output will be compared to the desired HMT.

For each column of the genome:

Mean-squared-error =  $(\text{HMT}_P - \text{HMT}_D)^2$ , where

$\text{HMT}_P$  = NNRUN predicted HMT based on the column data

$\text{HMT}_D$  = pre-specified desired HMT

The fitness function for each genome is the negative of the sum of the mean-squared-error of each of its columns (recall what the goal is to maximize the fitness function).

## **RESULTS**

### **GENETIC ALGORITHM SIMULATION**

The genetic algorithm was implemented using a basic template from the MATLAB MathWorks Library, and algorithmic techniques from Goldberg. Unfortunately, the results are inconclusive, owing to stability issues preventing the algorithm from terminating at an appropriate value. The mutation and crossover probabilities were lowered to reduce the variance of the population, but produced the same result. Apparently, the tabulated data for the default operating point was not accurate, as the

initial population scored very low fitnesses. Subsequent evolutions produced genomes that did not seem to be improving in fitness. From discussions with Sarma, it was found that indeed the printed derivative results were outdated. Thus, it is possible that the algorithm did not converge because of incorrectly initialized populations due to incorrect correlation data.

Isolating the policies that are preventing convergence proved to be very difficult, because the bit stream representation is not easily readable and decipherable anyone except the developer. From the fitness evaluation of the initialized population, it seems as if the initialization process can be improved. That is, derivative tests should be run again on the inputs to get more accurate delta values. Mutation and crossover probabilities did not seem to significantly affect the system behavior. An explanation for this behavior is that the bit vector genomes span a wide space (over  $2^{500}$  possibilities). The initial population represents an infinitesimal portion of this space. Mutating or mating these genomes would not grow the search space quickly enough to cover a significant portion of this realm, at least not in the number of generations specified in the algorithm. The algorithm would have to run for many more generations. In another implementation, the populations could be divided into sub-populations (with different initialized genomes), with each running on separate machines to simulate parallelism.

One of the reasons why the algorithm did not converge may be the fact that the crossover and selection schemes were implemented from scratch. The developer was not familiar with MATLAB prior to this project. Thus, the implemented algorithm may have been much less efficient than one that utilized all of MATLAB's capabilities. During research, it was found that comprehensive genetic algorithm libraries exist in a variety of developing environments, including MATLAB. It is assumed that these libraries contain optimized implementations for a variety of genetic algorithm schemes. However, many of them are not for public domain, and the creators could not be reached in a reasonable amount of time.

As mentioned earlier, genetic algorithms are more powerful when they can be run in parallel. The implementations were run on single processor machines, which in a way compromised the one distinct advantage of genetic algorithms. If multi-processor machines are not available, the technique of genetic algorithms may not be appropriate.

## **COMPARISON TO OTHER IMPLEMENTATION APPROACHES**

The technique of using genetic algorithms was not the only strategy considered by the group. Neural net inversion and decision trees are other powerful AI techniques that could be applied to this problem as well. The results of these efforts are described below:

### **Inversion**

Ashima Dua studied the technique of inversion of neural networks to control HMT in steel blast furnaces (2000). Dua similarly utilized the NNRUN network, but inverted the propagation mechanism. That is, instead of running the feed-forward network from multiple inputs to a single output (the predicted HMT), she wanted to derive the inputs from the output. The inversion problem was formulated as a non-linear programming problem, and the MATLAB optimization toolbox was used to solve this problem (Dua, 2000).

The input/output black box representation of Dua's program is very similar to the one described in this paper. A desired temperature at a desired time, along with the current input values, is entered, and a list of suggested input values is returned. However, Dua's program limited the solution space to only the set of input values that are closest to the current input values. This is a good insight, because when the controller is actually used in the blast furnaces, making small changes to the input values will be much easier than making large changes, especially if all of the changes need to be made within a short period of time. However, NNRUN is not a perfect predictor, and as Dua's results showed, the mapping of inputs to output using the neural network can have considerable error. Therefore, more solutions should have been considered. Dua should modify her program so that all solution vectors that score above a certain level are returned as possible solutions. It is interesting to note that if this suggestion were implemented, the inversion technique would be very similar to the one presented in this paper.

## Decision Trees

Mutaz Qubbaj investigated the technique of using rule-based decision trees to provide a more operator-friendly controller to control HMT temperature (2000). The controller was developed using commercial decision tree software from RuleQuest. Cubist was used to find correlations between input variables and HMT, and See5 was used to create the actual decision tree. Qubbaj also mentioned that the decision tree structure allows the controller to be updated from previous predictions, thus simulating machine learning. However, details for implementation were not provided.

Qubbaj implemented a wide-spanning decision tree, carrying information about all of the input variables at each intermediate node. This scheme facilitates the derivative analysis used to guess which direction to branch at each level (what is referred to as "boosting" in Qubbaj's paper). In addition, constraint sets were included in the rule set. Qubbaj noted that some input values were more difficult or costly to change. Thus, decisions made at each node were cost minimizing as well as HMT optimizing (Qubbaj, 2000).

The primary advantage of the decision tree-based controller over the genetic algorithm-based controller is comprehensibility and information transfer. Genetic algorithms produce controllers using probabilistic rules. Thus, while they produce solutions that are optimized according to the specified fitness function, they cannot explain how the solution was derived. In other words, the algorithm does not simulate learning; each run is independent of previous runs. The decision tree technique is more methodical, and the process involves reasoning and learning instead of probabilistic evolution. Decision trees are expert systems. The correlation between input, intermediate, and output variables are encoded in rules, and the proposed solution can be justified by walking down the decision path leading to the solution. In creating a controller using decision trees, the developer learns much about the problem. This information is more easily transferred to the plant operators through the produced rule set.

However, the complex interactions between the many input variables in a steel blast furnace cannot be simplified into a condensed set of descriptive rules. A rule set that

captures all of these correlations would number over a thousand rules. Such a complicated rule set is not necessary, considering that the controller will be used only by plant operators to maintain blast furnace HMT. A simple, unjustified solution would be more efficient and effective than a solution accompanied by thousands of detailed rules.

Of course, this analysis does not mean that implementing the HMT controller using decision trees is a bad idea. Decision trees are powerful tools, and the produced rule sets can provide insight into cost-effective ways to change the state of the furnace. In fact, there have been several controller implementations that used both decision trees and genetic algorithms. The decision trees were initially used to create the rule set, and a genetic algorithm was run to prune down the rule set into a minimal spanning list. This strategy could be used for the HMT controller as well.

## **CONCLUSION**

With the development of the ANN to predict steel blast furnace HMT temperatures hours into the future with reasonable accuracy, the next logical step would be to use this information to control future HMT values. Several AI techniques were considered to produce the controller. The group focused on neural net inversion, decision trees, and genetic algorithms. Implementation using decision trees yielded the most information about the process. However, the complex set of rules involved may make it unsuitable for an HMT controller, which requires solutions to be simple and quickly generated. For the HMT controller, both the neural net inversion and genetic algorithm techniques show similar paradigms. Both rely extensively on the ANN to test results, and both have a similar black box abstraction. The only difference is that Dua's inversion program restricted the solution space only to those whose delta functions are small. This modification increases the likelihood of producing sub-optimal solutions, but also cuts down considerably on the program's execution time. The genetic algorithm approach allows for a more exhaustive, parallelized search for multiple optimal controllers. However, it has been shown in this paper that this approach increases the complexity of the solutions (because they have to be encoded and decoded), increases the program's execution time, and may not converge with poor initialization or limited resources.

This paper focused on the issues associated with using genetic algorithms to develop the HMT controller. Although the presented research and design indicate that a genetic algorithm is a good candidate for implementing a controller for steel blast furnaces, the actual implementation proved to be rather complex, and requires a more powerful set of genetic algorithm tools. Genetic algorithms are essentially aggressive parallel searches for quickly searching a large sample space for one or more optimal solutions. The design of the genome and algorithm is important for verification and interpretation, but the implementation is important for algorithm termination. For this paper, the design issues were thoroughly discussed and specified. However, implementation was less methodical. As a result, inefficient code limited machine parallelism slowed the program's search speed, and consequently led to failed convergence.

#### **IMPROVEMENTS/FUTURE WORK**

Genetic algorithm-based implementation of controllers for steel blast furnaces should be further investigated. When implementing future versions, the following should be considered:

Multi-processor machines should be used whenever possible to maximize parallelism and minimize running time. If only single processor machines are available, the genetic algorithm needs to be modified accordingly to increase the likelihood of convergence. One proposed solution is to run the algorithm on several sub-populations for a smaller number of generations. These sub-populations may be saved, and the algorithm may be resumed at a future time. After a few runs, genomes from one sub-population should migrate into others. This scheme can simulate parallel processing on a single processor machine, because it allows the system to stop and restart after only a few generations.

If code optimizations do not lead to reliable convergence, then the granularity of the genomes should be reduced. Currently, only four bits are used to specify a normalized value between 0 and 1. This parameter can be reduced, but only as a last resort. Fewer

data points can be used to approximate functions. If needed, the HMT step can be increased to preserve the same HMT range.

### **EXTERNAL ISSUES**

When successfully produced, the genetic algorithm-based controller can be integrated to the blast furnace system. However, applying the HMT controller at the steel plant requires some other considerations as well.

1. The current HMT controller assumes that all the parameter values can be changed instantaneously when, in reality, it will take some time for the engineer to change each one of the parameter values. The HMT controller should be re-calibrated so that it takes this additional time into account.
2. The current HMT controller also assumes that all parameter values can be changed at exactly the same time. Certainly, this is not the case, as the most inputs can be changed only individually. Because changing individual input values affects the entire system, the order in which the variables are changed may affect the output (or, in the very least, the “desired values” of the inputs that have not yet been changed).
3. Currently, the design of the genomes allows for only a single operating point. This constraint means that the function would work well only for a certain HMT (or a small range of HMT values around the operating point). So, for different operating points (or operating ranges), the genetic algorithm multiple would have to be run multiple times, with different starting values of HMT. This is not a big concern, because the HMT range is only 100 degrees, so there are a limited set of ranges that need to be considered. However, for other applications with a larger range of operating points, a modification to the genome-encoding scheme may be needed.

## REFERENCES

- Badran, S.M. and Al-Duwaish, H.N. "Optimal output feedback controller based on genetic algorithms." Electric Power Systems Research 50 (1999).
- Bulsari, Abhay and Saxen, Henrik. "Classification of blast furnace probe temperatures using neural networks." Steel Research. Vol. 66. 1995.
- Bulsari, Abhay and Saxen, Henrik and Saxen Bjorn. "Time-series prediction of silicon in pig iron using neural networks". International Conference on Engineering Applications of Neural Networks (EANN '92).
- Chiang, Chih-Kuan, and Chung, Hung-Yuan, and Lin, Jin-Jye. "A Self-Learning Fuzzy Logic Controller Using Genetic Algorithms with Reinforcements." IEEE Transactions on Fuzzy Systems. Vol. 5 (August 1997), No. 3.
- Chin, T.C. and Qi, X.M. "Genetic algorithms for learning the rule base of fuzzy logic controller". Fuzzy Sets and Systems 97, 1998.
- Dua, Ashima. "Inversion of Neural Networks: A Solution to the Problems Encountered by TISCO" (MIT, 2000).
- Goldberg, D.E. Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
- Lian, Seng Teo, and Marzuki, Khalid, and Rubiyah, Yusof. "Tuning of a neuro-fuzzy controller by genetic algorithms with an application to a coupled-tank liquid-level control system." Engineering Applications of Artificial Intelligence 11 (1998).
- Lida, Osamu and Ushijima, Yuichi and Toshiro, Sawada. "Application of AI techniques to blast furnace operations." Iron and Steel Engineer, October 1992.
- Pohlheim, Hartmut. "GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB" (1991).  
[http://www.systemtechnik.tuilmnau.de/~pohlheim/GA\\_Toolbox/index.html](http://www.systemtechnik.tuilmnau.de/~pohlheim/GA_Toolbox/index.html).
- Purville, Brian. "Data Mining Using Time-Delay Neural-Networks Applied to Steel Production" (MIT, 2000).
- Qubbaj, Mu'taz M. "Using Decision Trees to Create a Control Mechanism for Regulation of the Hot Metal Temperature of the "G" Blast Furnace at Steel Corp" (MIT, 2000).

Sarma, Ravindra K. "Using Neural Networks to Predict the Hot Metal Temperature of the "G" Blast Furnace at Tata Steel" (MIT, 1999).

Singh, Himanshu and Sridhar, Nallamali and Deo, Brahma. "Artificial neural nets for prediction of silicon content of blast furnace hot metal." Steel Research, vol. 67 (1996). No. 12.

Varsek, Alen, and Urbancic, Tanja, and Filipic, Bodgan. "Genetic Algorithms in Controller Design and Tuning". IEEE Transactions on Systems, Man, and Cybernetics, vol. 13 (September/October 1993) No. 5.

Wall, Matthew. "GA lib: A C++ Library of Genetic Algorithm Components." (MIT, 1996).

Winston, Patrick Henry. Artificial Intelligence. Addison-Wesley Publishing Company, New York. 1993.