

**Pattern Recognition and Decision-Making in
Railroad Maintenance**

**A THESIS
SUBMITTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY
OF MINNESOTA
BY**

Todd Wittman

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCES**

August 1999

Acknowledgments

I would like to thank the Minnesota Center for Industrial Mathematics (MCIM) and Loram Maintenance of Way, Inc. for sponsoring this work. Specifically, I would like to thank Dr. Fadil Santosa of the MCIM for guiding me through the mathematical part of this internship and Kenneth Iverson of Loram for guiding me through the industrial part.

Dr. Santosa was my adviser for this program and his advice is reflected throughout this entire paper. His extensive knowledge of mathematical modeling and optimization was an invaluable asset to me. He was always helpful, informative, and patient.

Kenneth Iverson and the rest of the R&D group at Loram were very supportive and friendly. They explained the problem to me clearly, acted as a sounding board for ideas, helped me with the C++ programming, and even provided me with my own office at their headquarters.

Finally, I'd like to thank my friends here at the University of Minnesota for helping me in my early stages as a mathematician.

Contents

| | | |
|-----|----------------------------------------------|----|
| 1 | Introduction | 1 |
| 2 | Identifying the initial rail shape | 4 |
| 3 | Generating the metal removal curve | 10 |
| 4 | Selecting the grinding patterns | 13 |
| 4.1 | Pattern selection problem | 13 |
| 4.2 | Enumerative search | 16 |
| 4.3 | Greedy algorithm | 18 |
| 4.4 | Genetic algorithm | 19 |
| 4.5 | Results | 29 |
| 6 | Conclusions and Further Research | 31 |
| 7 | Bibliography | 32 |

List of Figures

| | | |
|---|---------------------------------------------|----|
| 1 | Flowchart of Decision-Making Process . . . | 3 |
| 2 | Effect of Deformations on Moments | 7 |
| 3 | Trial of Moments Algorithm | 9 |
| 4 | MR Curve Calculations | 11 |
| 5 | Evaluating Grinding with the MR Curve . . . | 12 |
| 6 | Crossover Operator | 21 |
| 7 | Comparison of Algorithms | 30 |

1 Introduction

To ensure the safe operation of freight railroads and a comfortable ride aboard commuter lines, the tracks must be maintained with a smooth, rounded rail head. Even small deformations in the rail head resulting from excessive weight and use are capable of derailing trains. The world's leading railroad maintenance organization, Loram Maintenance of Way, Inc., has developed the C21 Rail Grinder, a multiple car train outfitted with a number of large grinding stones capable of cutting iron. Each of these stones has three axes of motion and can be set from the cab of the train to form grinding patterns. A laser imaging system on the front of the C21 obtains a cross-sectional view of the rail and displays the image on the computer inside the cab. The goal of the C21's operator is to determine the proper grinding patterns and train speeds that will mold the rail head into an ideal shape.

Loram currently owns several C21 Rail Grinders operating on four continents. At the time of this writing, the pattern selection is made by the C21 operator based entirely on his/her knowledge and experience. Loram's engineers determined that the operator was selecting from only a few different patterns and making several passes on each section of track at low speeds to slowly grind the rail into a smooth shape. The Research and Development department was given the task of developing computer algorithms that would efficiently determine the appropriate combination of grinding patterns and track speeds.

A process for the C21 was developed that would be executed on the left and right tracks independently. After the image of the actual rail head is obtained, the first step is to identify the rail shape from among a predetermined set of representative rail profiles. This can be done very quickly by calculating

several geometric invariants, or moments, of the rail profile. Second, the desired extent and locations of grinding need to be determined, which can be expressed with a metal removal (MR) curve. To accomplish this, the actual rail profile is superimposed with the ideal profile and the difference between the two profiles is calculated using cubic spline interpolation. Third, the grinding patterns and track speeds are selected using previously obtained data on the representative rail profiles. Several algorithms were tested for this purpose, including enumerative searches, greedy algorithms, and genetic algorithms.

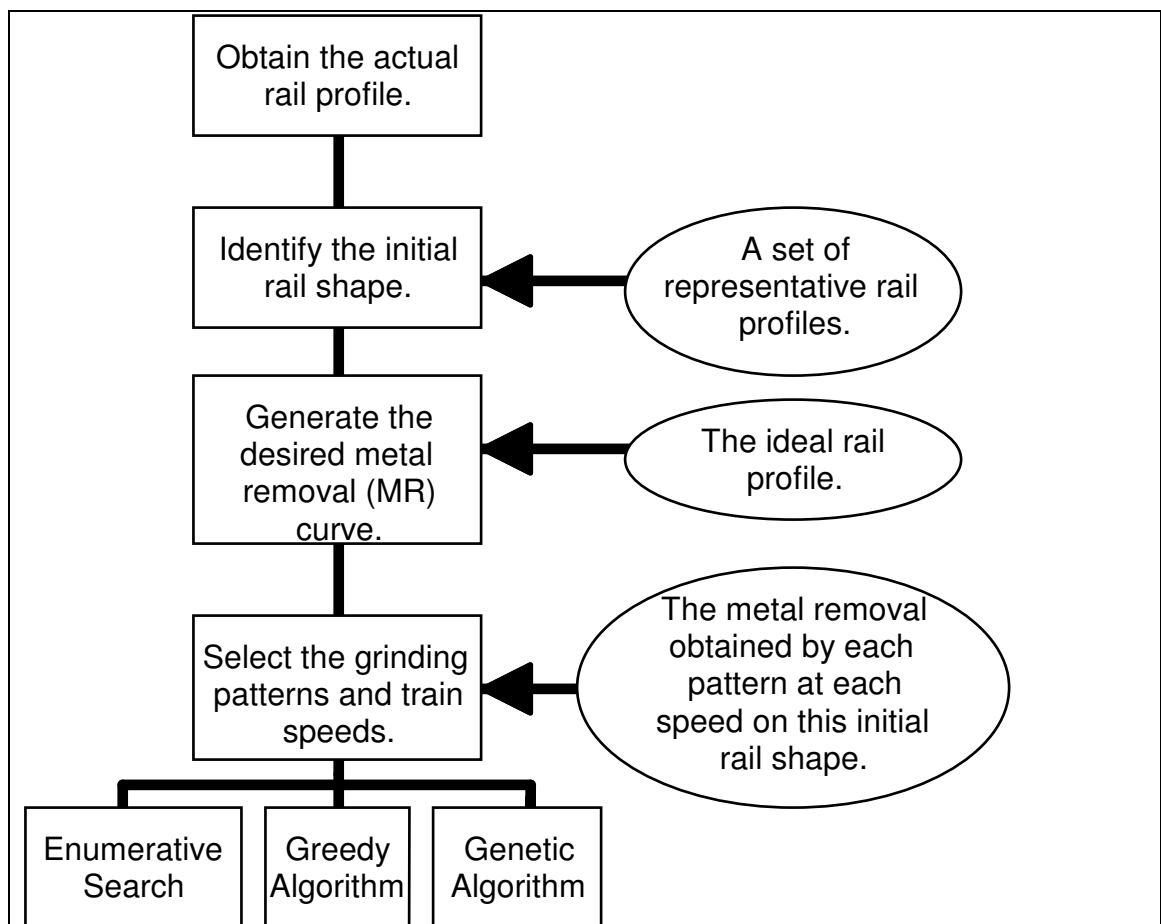


Figure 1: Flowchart of Decision-Making Process

The squares indicate steps of the process and the ovals indicate information that needs to be provided for that step.

2 Identifying the initial rail shape

A rail head may develop dents and bulges that require attention for safe passage on the tracks. These deformations vary in severity based on how often the tracks are used, the weight put upon the track, the thickness of the rail head, the metal used to make the rails, and weather conditions. The location of these deformations on the rail head depends mainly on the curvature of the track, i.e. straight-away, mild right turn, sharp right turn, etc. It is important to classify the given rail profile to facilitate the maintenance procedure. The C21 performs repeated passes on a section of track until that section matches the desired shape, where a section can be anywhere from a few dozen feet to several miles in length. This step identifies when a different type of rail head is encountered and thus signals that a new set of grinding patterns needs to be chosen. This step can also identify when a section of track is in very good shape and does not require any maintenance. Conversely, some extremely worn types of rails should be identified immediately so that the engineer can be alerted that the entire section of track needs to be replaced.

Loram maintains a database of representative rail profiles that are commonly seen in the field. Each representative is stored as (x,y) data points representing the outline of roughly the top inch of the rail. For rail identification, only the top 0.9 inches below the highest point of the rail head needs to be considered. A standard approach to image recognition is to identify geometric features of the object. This can be done by calculating various geometric invariants called moments of the image. In two dimensions, a geometric moment of order p+q of an image is defined by

$$M_x^p y^q = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q F(x, y) dx dy$$

where $F(x,y)$ is a function chosen to contain useful information about the image [1].

The rail shapes are not aligned with respect to a common origin, so translations should be taken into account. However, the motions are rigid, so scaling and rotations will not complicate the task. One way to align the rail heads is to first calculate the center of mass of the rail head and treat this point as the origin. The center of mass (M_x, M_y) for a rail head Ω is given by

$$M_x = \frac{\iint_{\Omega} x dx dy}{\iint_{\Omega} dx dy} \quad M_y = \frac{\iint_{\Omega} y dx dy}{\iint_{\Omega} dx dy}$$

However, to make computations slightly easier, integration was done with respect to arclength, s , rather than with respect to the area. Here the arc represents the top boundary of the rail head. Note that this modification does not change the nature of the approach, even though (M_x, M_y) loses its physical significance:

$$M_x = \frac{\int x ds}{\int ds} \quad M_y = \frac{\int y ds}{\int ds}$$

Note that both of these moments are of order one. The calculation of the moments of higher order will involve subtracting M_x and M_y to translate the image to make (M_x, M_y) the origin. In this manner, the higher moments of different rail profiles can be compared directly. To single out features of the rail head, three binary functions were used:

$$T(x, y) = \begin{cases} 1 & \text{if } y > My \\ 0 & \text{otherwise} \end{cases}$$

$$L(x, y) = \begin{cases} 1 & \text{if } x < Mx \\ 0 & \text{otherwise} \end{cases}$$

$$R(x, y) = \begin{cases} 1 & \text{if } x > Mx \\ 0 & \text{otherwise} \end{cases}$$

The function $T(x,y)$ focuses the calculation on the top portion of the rail profile above the center of mass. The function $L(x,y)$ focuses on the region to the left of the center of mass while $R(x,y)$ looks at the region to right. These three functions were used to calculate the five additional moments described in the table below.

| Name | Exact Formula | Numerical Approximation $ds_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$ |
|------|------------------------------|----------------------------------------------------------------------------------|
| s | $\int ds$ | $\sum_i ds_i$ |
| Mx | $\int x ds / s$ | $\sum_i \frac{1}{2} (x_{i+1} + x_i) ds_i / s$ |
| My | $\int y ds / s$ | $\sum_i \frac{1}{2} (y_{i+1} + y_i) ds_i / s$ |
| MxT | $\int (x - Mx) T(x, y) ds$ | $\sum_i \frac{1}{2} (x_{i+1} + x_i - 2Mx) T(x_i, y_i) ds_i$ |
| MxxL | $\int (x - Mx)^2 L(x, y) ds$ | $\sum_i \left[\frac{1}{2} (x_{i+1} + x_i - 2Mx) \right]^2 L(x_i, y_i) ds_i$ |
| MxxR | $\int (x - Mx)^2 R(x, y) ds$ | $\sum_i \left[\frac{1}{2} (x_{i+1} + x_i - 2Mx) \right]^2 R(x_i, y_i) ds_i$ |
| MyT | $\int (y - My) T(x, y) ds$ | $\sum_i \frac{1}{2} (y_{i+1} + y_i - 2My) T(x_i, y_i) ds_i$ |

| | | |
|-----------|------------------------------|------------------------------------------------------------------------------|
| M_{yyT} | $\int (y - My)^2 T(x, y) ds$ | $\sum_i \left[\frac{1}{2} (y_{i+1} + y_i - 2My) \right]^2 T(x_i, y_i) ds_i$ |
|-----------|------------------------------|------------------------------------------------------------------------------|

These moments were chosen to pick out certain features of the rail profile. For example, a rail head with a bulge on the left side of the rail will have a larger M_{xxL} value than a rail profile that does not have a bulge.

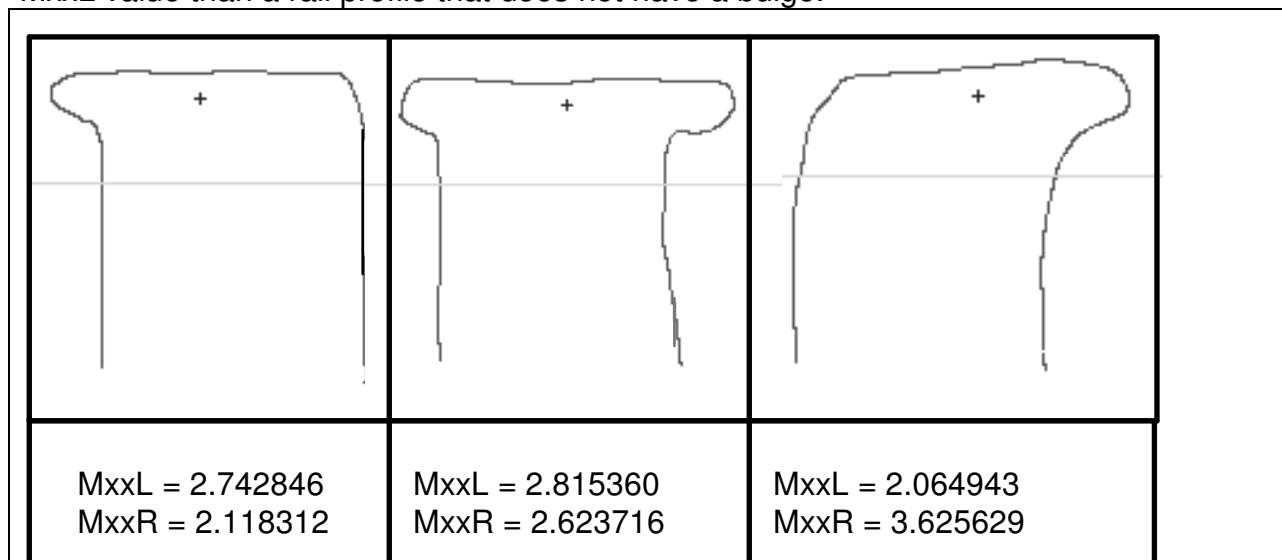


Figure 2: Effect of Deformations on Moments

The moment M_{xxL} is larger for bulges on the left while M_{xxR} detects bulges on the right.

The values of all eight moments are calculated and stored for each of the representative rail profiles. Once an actual rail profile is obtained on the C21, the highest y-value is determined and all data points with y-values less than 0.9 inches below the maximum are ignored. The center of mass (M_x, M_y) is first calculated for the actual profile so that the higher moments can be aligned properly. The remaining six values are calculated and compared to the corresponding six values for each of the representatives in the L2-norm. Note that all the moments of the actual profile can be calculated in two passes through the array of data points. Since the moments of the representative profiles are

calculated off-line, if there are N data points in the actual profile and K representative profiles, then this algorithm runs in linear $O(N+K)$ time.

This algorithm correctly identified the representative class for every test profile supplied by Loram. It is believed that these moments will still suffice if the set of representatives is expanded. If not, it is fairly elementary to expand this algorithm to calculate and recognize moments of higher order.

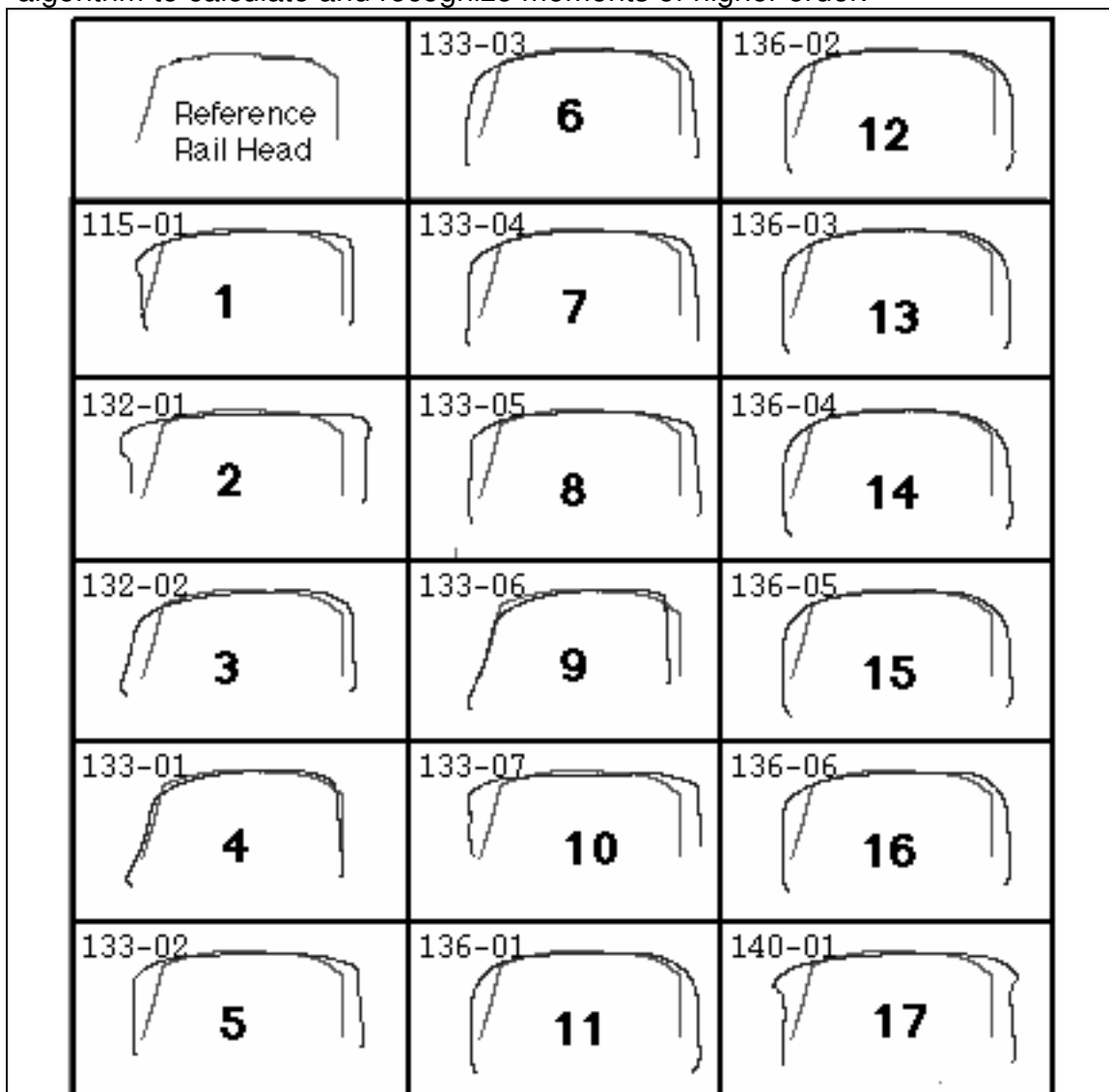


Figure 3: Trial of Moments Algorithm

The rail head at top left was aligned and compared with 17 representative rail shapes using the moments algorithm. The algorithm picked out #4 as the best match.

3 Generating the metal removal curve

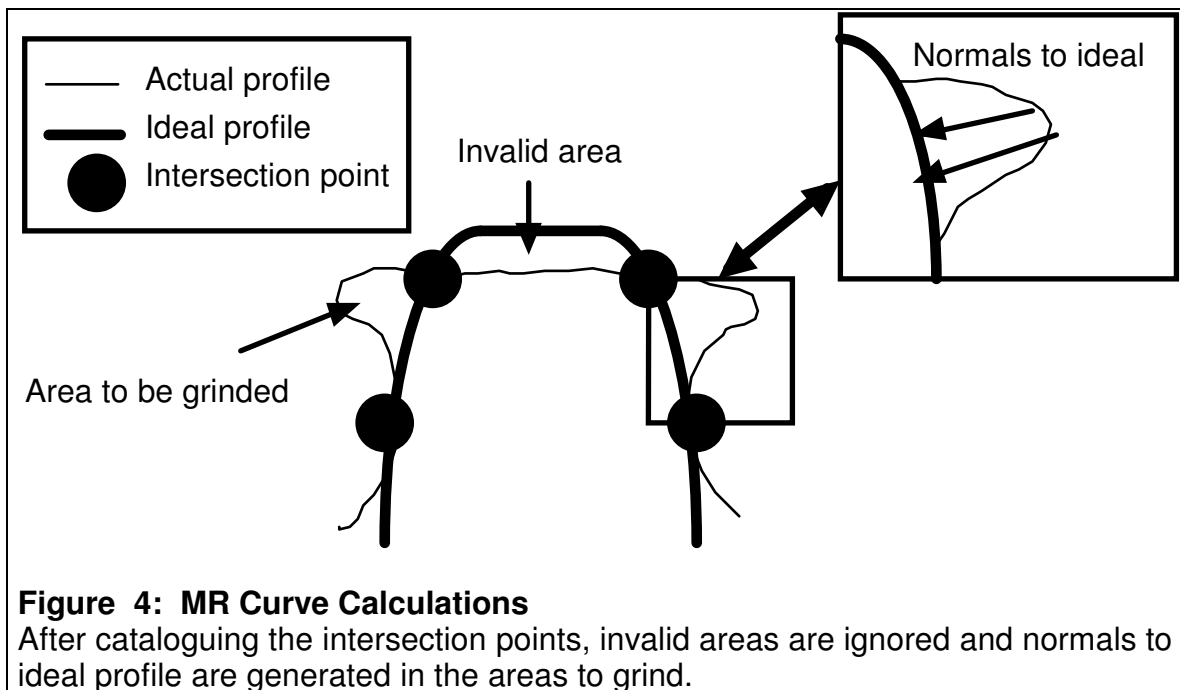
By comparing an actual rail profile with an ideal profile, it is possible to generate a graph displaying the amount of grinding that needs to be performed to obtain the ideal profile. This metal removal (MR) curve guides the C21 engineer in the grinding process. The procedure for generating the MR curve was developed by Thomas Chase and is described in detail in his technical report for Loram [2].

The first step in the procedure is to obtain a mathematical representation of the image. A natural parametric cubic spline is fit to the data points composing the image. This involves solving two cubic equations at each data point so that the spline forms a smooth curve. Second, this spline is aligned with the cubic spline of the ideal profile. The alignment is made by lining up the points on the spline where the tangent curve forms a ten degree angle with the horizon. Third, a pass is made through the set of data points to determine all points of intersection of the actual profile with the ideal profile. Fourth, the area in each region between the ideal and actual profiles is calculated. This is done using the area corollary to Green's theorem:

$$Area_R = \frac{1}{2} \oint_C (x dy - y dx)$$

where C is a counterclockwise oriented curve around the region R . Fifth, it is determined which regions require grinding. If the ideal profile lies above the actual profile in a certain region, then this region is invalid since the C21 cannot add metal to the rail head. If the ideal lies below the actual profile, then the area calculations are used in the last step to determine head loss. The sixth and final step is to generate the MR curve by proceeding along the x-coordinates of the

actual profile and determining the distance between the profiles along a normal to the ideal profile.



This same procedure can be used to determine the effect of running a grinding pattern at a certain speed on a rail head. The rail profile that results is fed into the algorithm as the ideal and the profile prior to grinding is used instead of the actual profile. This generates a MR curve showing the extent and location of metal removal that the pattern produces. These MR curves are used to decide which patterns and speeds to use on a rail head.

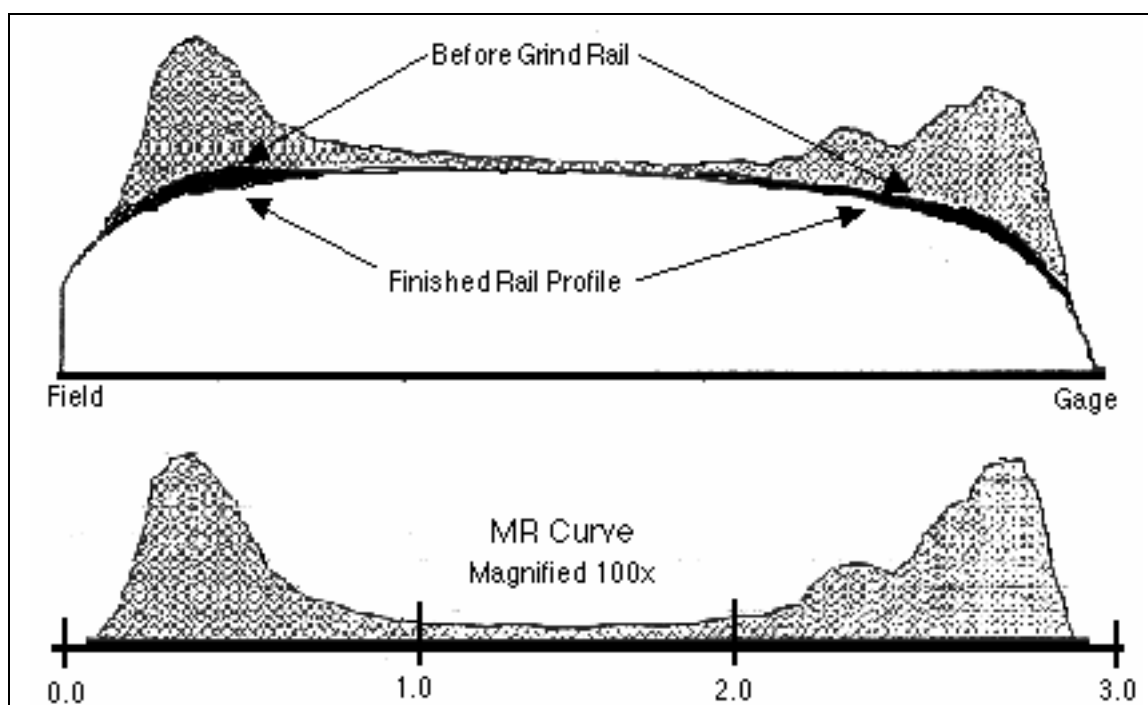


Figure 5: Evaluating Grinding with the MR Curve

This is the metal removed by the C21 Rail Grinder running grinding pattern B8 at 5 mph on a 130 low rail.

4 Selecting the grinding patterns

4.1 Pattern selection problem

There are three major factors that affect the grinding process. First, the grinding pattern selected determines how the grinding stones are set on the rail head. Second, the speed of the C21 affects the extent of grinding since higher speeds give the stones less time to grind the rail head. Third, the initial rail shape affects where the stones contact the rail. Loram is currently developing a three-dimensional database of MR curves that reflects the effect of these three factors on the grinding process. Each cell of this database contains a MR curve that shows the metal removal that results when the C21 runs at a certain speed and executes a chosen grinding pattern on a certain initial rail shape. The procedure described in the last section gives the engineer a target MR curve to match. Hence, determining the patterns and speeds to run corresponds to choosing the cells in the database whose MR curves add up in the pointwise sense to the desired MR curve. Although it remains to be tested in the field, a grinding procedure that theoretically produces the desired MR curve should result in an ideal rail profile in reality.

By identifying the initial rail shape as in Section 2, this narrows the search space from a three dimensions to only two. That is, the initial identification places the search on a certain level of the database. Hereafter, we shall refer to the choice of a grinding pattern and speed simply as a pattern. The pattern selection problem (PSP) consists of finding the combination of patterns that sums to a MR curve as close as possible to the desired MR curve.

The error for PSP will be measured by computing the distance between the desired curve and the summed curve at a discrete number of x-coordinates and taking the L2-norm of the distances. Define the pattern choice by:

$$A_{ij} = \begin{cases} 1 & \text{if pattern } i \text{ at speed } s_j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

Note it is reasonable for A_{ij} to be binary rather than an arbitrary value since running the same grinding pattern at the same speed twice is roughly equivalent to running the grinding pattern at a lower speed. Let $P_{ij}(x)$ be the MR curve that results from running pattern i at speed s_j . Finally, let $f(x)$ be the desired MR curve, or the target. Then the error function E can be expressed as

$$E = \left\| f(x) - \sum_i \sum_j A_{ij} P_{ij}(x) \right\|_2$$

The algorithms should be designed to pick out as few patterns as possible so that the C21 can avoid backtracking over a section of rail many times. To ensure this, a penalty μ_1 can be added to the error function that seeks to minimize the number of patterns selected. In addition, the algorithm should try to select higher speeds so that the grinding process will move as quickly as possible. Another penalty μ_2 can be added that tries to maximize the speeds selected. So the modified error function becomes

$$E = \left\| f(x) - \sum_i \sum_j A_{ij} P_{ij}(x) \right\|_2 + \mu_1 \sum_i \sum_j A_{ij} + \mu_2 \sum_i \sum_j A_{ij} / s_j$$

The values of the constants μ_1 and μ_2 need to be set by the engineer depending on the value he/she places on minimizing the number of passes and maximizing speeds, respectively. Since the values are subjective and do not affect the basic algorithm design, we will assume $\mu_1 = \mu_2 = 0$ hereafter.

Unfortunately, choosing the appropriate combination of patterns is a NP-complete problem. That is, PSP is hard in the sense that if there exists a polynomial-time solution to this problem, then famous problems such as the

Traveling Salesman Problem and the Hamiltonian Cycle Problem are also polynomial-time solvable. NP-completeness is most easily seen by reduction from a known with a known NP-complete, the Subset Sum Problem (SSP):

Subset Sum Problem (SSP)

Given a set S of n integers $\{x_1, x_2, \dots, x_n\}$ and a target integer t , determine the subset of S whose elements come closest to t in sum. That is, find the subset $\{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$ that minimizes

$$\left| t - \sum_{i=1}^k x_{j_i} \right|.$$

Note that it is fairly straightforward to show that solving SSP reduces to solving PSP. Instead of selecting from integers, the problem selects from among MR curves. The summation can be extended from one dimension to two dimensions by using a pointwise summation at M equally spaced interval points. Each integer i in SSP can be translated to a horizontal line at $y = i/M$ in PSP. In this manner, minimizing the L2-norm of the difference of curves accomplishes the same result as minimizing the absolute value of the difference of integers. So if there exists a polynomial-time solution to PSP, then there exists a polynomial-time solution to SSP and hence to all NP-complete problems. Similarly to SSP, the grinding problem PSP can be restated as:

Pattern Selection Problem (PSP)

Given a set S of n curves $\{x_1, x_2, \dots, x_n\}$ and a target curve t , determine the subset of S whose elements come closest to t in sum. That is, find the subset $\{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$ that minimizes

$$\left\| t - \sum_{i=1}^k x_{j_i} \right\|_2 \text{ where the L2-norm is evaluated at } M \text{ equally-spaced}$$

interval points.

The exact solution to the SSP is a modified merge sort that runs in exponential $O(2^n)$ time. However, there exist several approximation algorithms that find a subset that approaches optimal [3]. Similarly, there exist several approximation algorithms for PSP. Since the Loram engineers only require a solution within a specified tolerance, a global optimization scheme is not necessary. Three approaches were coded and tested to solve this combinatorial optimization problem: enumerative search, greedy algorithm, and genetic algorithm.

4.2 Enumerative search

A very inefficient approach to PSP would be to test every possible combination of patterns and select the combination that comes closest to minimizing the L2-norm. Assuming there are n patterns to choose from and each pattern is run at most once, an exhaustive search of the entire search space would take $O(2^n)$ time. At the time of this writing, the Loram database will contain 27 grinding patterns and at least 4 train speeds. This gives $n=4*27=108$ patterns to choose from. Since this search needs to be performed on-line, an exponential running time is unreasonable.

Another type of enumerative search is to test each pattern's MR curve against the target MR curve. This algorithm selects the one pattern that best matches the target by minimizing the error E . The null pattern or empty set is included as a possible choice, since the rail may not require any grinding. Since this algorithm makes one pass through the list of patterns, this is an $O(n)$ algorithm. This enumerative search for one pattern (ES1) cannot be expected to produce a result with tolerable error. Generally, the C21 has to make several passes on a section of rail. ES1 will make a good selection only when the rails

are in very good shape and require only a smoothing grind. The engineers at Loram indicated that this happens infrequently as railroads tend to hire maintenance services only when the tracks are in a state of great disrepair. However, the grinding schedule may be very tight and the C21 may have time to make only one pass on each section. In this case, the C21 does as much grinding as possible on the track with only one pattern. When one-pass grinding is necessary, ES1 is the best algorithm.

It is possible to do an enumerative search that looks at combinations of more than one pattern and still runs in polynomial time. An enumerative search through all combinations of up to three patterns (ES123) was developed. This algorithm tests approximately $C(n,3)$ combinations, so it runs in $O(n^3)$ time. ES123 performed very well on the test data, since most rails encountered in the field can be ground to the ideal profile within three passes if the appropriate patterns are selected. However, the cubic running time is very costly for on-line performance.

4.3 Greedy Algorithm

Greedy algorithms are common approximation schemes for many NP-complete problems, including the Traveling Salesman Problem, the Set-Covering Problem, and the Bin Packing Problem [3]. At each step, the greedy algorithm (GREEDY) makes a local optimization choice. Let D be the desired MR curve that the algorithm is trying to match. GREEDY scans through the list of patterns and chooses the one pattern P_1 that minimizes the error. Next the algorithm sets $D - P_1$ as the target MR curve and chooses the pattern P_2 that minimizes the error. Next the target is $D - P_1 - P_2$, the algorithm selects the local minimizer P_3 , and so on. The algorithm stops when the summed MR curve is within a

specified error tolerance of the desired MR curve. The algorithm may also stop when the null pattern is chosen, indicating that another grinding pass will increase the error. Note that this scheme is equivalent to successively running the algorithm ES1.

Although GREEDY locally reduces the error at each step, this does not guarantee that it will produce a global optimum. To improve the end result, GREEDY can be run several times with a different starting pattern each time. On test data, GREEDY often produced a better pattern combination starting with the second or third best pattern produced in the first step. The greedy algorithm was coded to first select the top three choices from ES1 and then use these choices to generate three pattern combinations.

The algorithm is relatively fast. It is safe to assume that there is a constant upper bound U on the number of grinding passes the C21 will make. So GREEDY scans the list of n patterns at most U times, which means it performs $O(U*n) = O(n)$ work.

4.4 Genetic algorithm

The genetic algorithm (GA) is an optimization scheme modeled after the processes of Darwinian evolution and natural selection. GAs are designed to maximize a fitness function $f(x)$ over a search space X . Each solution $x \in X$ is thought of as representing an organism. The GA uses computational methods similar to the biological processes of breeding, mutation, and selection to "evolve" fitter organisms [4]. The steps of the canonical GA are as follows:

Encode the search space.

Initialize a population.

Repeat

Crossover.

Mutation.
Selection.
Until a *Stopping Condition* is met.

Encoding

To encode the search space, there needs to be an integer N and a bijection $g: X \rightarrow \{0,1\}^N$, where $\{0,1\}^N$ represents the set of binary vectors of length N . Each possible solution $x \in X$ is represented uniquely by a bit vector. This bit vector can be thought of as a DNA string expressing the genetic makeup of organism x , where each bit represents a gene, a 1 indicates the organism possesses that trait, and 0 indicates it does not. These genetic codes are manipulated by the GA in a manner resembling natural processes.

Since it is fairly straightforward to devise an encoding of a search space, the GA has numerous applications. Many problems seem to have a natural encoding, such as using the standard binary coding to maximize a function over the real numbers. However, it is believed that the Gray coding is more efficient for real number encoding, so some thought should be given to the construction of the bijection g [5]. If the encoding is done carefully, complicated structures can be represented as bit vectors, such as when GAs are used to evolve more efficient neural networks [6]. Rather than restricting solutions to binary vectors, it is also possible to encode into any set of integers. This complicates the evolutionary operators, but it is sometimes desirable, as in the case of solving the TSP [7]. This paper will only discuss the standard $\{0,1\}$ encoding.

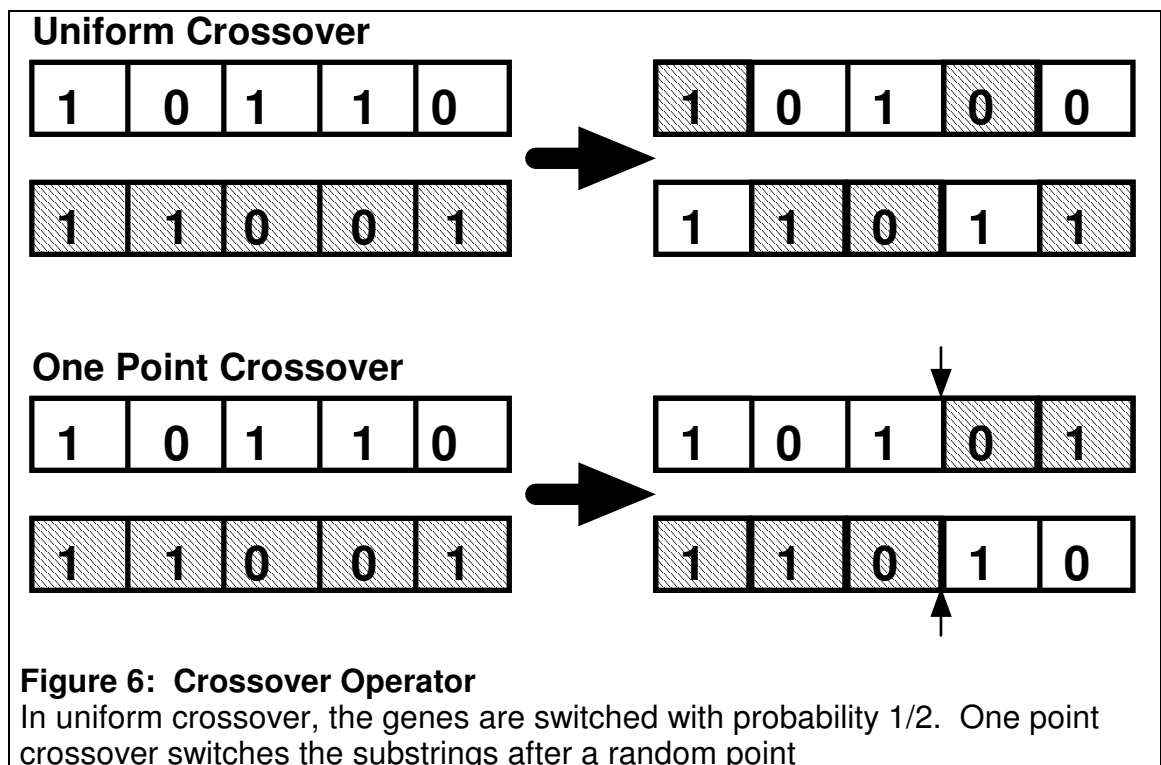
Initialization

The gene pool is initialized by randomly generating K bit vectors of length N . A set of K organisms is referred to as a population and a population at a fixed time is called a generation. The population size K will be maintained throughout the algorithm. In general, a population size $K=2N$ is recommended to provide

sufficient diversity [8]. However, since K individuals need to be evaluated at every step, memory and space requirements may force K to be smaller. The fitness of a generation will be defined as the largest fitness value $f(x)$ of all organisms in the population.

Crossover

The crossover or recombination operator models the process of breeding to combine aspects of different organisms to produce new organisms. Mating pairs are chosen at random and new offspring are created by each pair. The most basic crossover scheme is one point crossover. A position in the bit vector is chosen at random and the genes swap substrings after this pivot point. This scheme can be generalized to n point crossover by selecting n pivot points. This scheme is often preferred since it most closely models how DNA strings combine [9].



Another crossover scheme is uniform crossover. For each gene, the genes of the parents are swapped with probability 1/2. On average, a child will receive 1/2 of its genes from each parent. As in n point crossover, each mating pair produces 2 offspring. Uniform crossover has been shown to be superior in most cases to one point and two point crossover [10].

More exotic crossover schemes have been developed. For example, the organisms within a population can be placed into a caste and only organisms within a caste may breed. One paper suggested assigning a gender to each organism and each mating pair must consist of organisms of opposite sexes. Also, it is not necessary to choose the mating pairs at random. However, it has not been proven that these modifications improve the GA's performance. The basic role of the crossover operator is to explore new regions of the search space [11]. As long as the crossover scheme satisfies this diversification role, the details of the scheme are unimportant.

Mutation

Since the crossover operator combines traits of the parent population, crossover alone cannot introduce new traits into a population. For example, if each of the K organisms in the initial population has a 0 in the first position, the crossover operator cannot explore regions with a 1 in the first position. The mutation operator examines each gene of each of the K organisms and switches the bit with probability p_m . That is, a 0 becomes a 1 or 1 becomes 0. It has been shown experimentally that a constant p_m should be about 0.01, or more generally $p_m \approx \frac{1.76}{K\sqrt{N}}$. The mutation probability p_m need not be a constant; it can be a probability density such as a Gaussian or normal. It has been shown that a Cauchy density function is the best choice. The optimal setting of p_m is a function that is initially 0.5 and decays quickly to 0.01 as the algorithm

converges. However, this setting requires some measure of the convergence rate, which may be unknown for some problems [12].

Although mutation plays an important role, it introduces only minor changes in the overall algorithm. Define the Hamming distance $H(x,y)$ between two binary vectors x and y as the number of bits in which the two vectors differ. The probability that an organism x mutates into a new organism y is $p_m^{H(x,y)}(1-p_m)^{N-H(x,y)}$. Since the average value of $H(x,y)$ is $N/2$, for values $p_m=0.01$ and $N=20$, the average probability of mutating into a new organism is 9.044×10^{-21} .

One way to increase the effect of mutation is to direct the mutation through Lamarckian evolution. The local neighborhood of each organism is searched by evaluating the fitness of every organism within a fixed Hamming distance L of the original organism. The original organism is replaced by the fittest individual in its local neighborhood. This local search for every organism is computationally expensive, but it has been shown that for some cases Lamarckian evolution improves performance [13].

Another way to introduce new traits into the gene pool is to simply restart the algorithm. By re-initializing the population, the algorithm can quickly jump to new areas of the search space. The final solution reported will be the fittest organism produced in each of the runs of the GA. One author suggests restarting the algorithm after a fixed number of iterations or when the average fitness of a population has not improved for a fixed number of generations [14].

Selection

Since the crossover and mutation operators do not evaluate the fitness of the organisms, crossover and mutation alone form an undirected random search. The selection operator directs the search through the concept of "survival of the

fittest." The most common type of selection is fitness proportional selection. In every generation, the fitness proportion $P(x_i)$ of each organism x_i is calculated relative to the average fitness of the population:

$$P(x_i) = \frac{f(x_i)}{\sum_{i=1}^K f(x_i)} .$$

$P(x_i)$ represents the probability that individual x_i will survive into the next generation. Figuratively, this operation is similar to dropping balls into bins in a pachenko game, where the fitter organisms have larger bins. Other schemes include picking the fittest individuals (truncation selection), randomly choosing a subset of individuals and choosing the best for survival (tournament selection), and keeping the two best in each family after a mating pair produces offspring (elitist recombination). Although all truncation schemes are similar in performance, experimental results suggest truncation selection converges fastest [15].

It should be noted that the selection scheme can select survivors from different subsets. In general, the crossover operator has μ parents generate λ offspring, where $\lambda \geq \mu$. The (μ, λ) selection scheme selects the K survivors for the next generation from the λ offspring. A $(\mu + \lambda)$ selection scheme chooses the K survivors from the set consisting of the μ parents and the λ offspring.

In fitness proportional selection, the fittest organism in a generation is the most likely to survive into the next generation, but it is not guaranteed to survive. If the fittest organism does not survive, the overall fitness of the new population may be lower than the that of the previous generation. To correct this, an elitist selection scheme guarantees that the fittest individual in each generation survives. Since one survivor has already been chosen, the selection scheme

may choose only $K-1$ survivors. Since crossover combines the traits of the various organisms, it may happen that the fittest organism will dominate a population. That is, as the algorithm progresses, the entire population will begin to resemble the fittest organism. Some authors suggest modifying the elitist strategy so that the fittest organism is stored, but does not participate in the breeding process [4].

Stopping Condition

In general, the algorithm stops when the overall fitness of a population is above some threshold. That is, the fittest organism in the population is close enough to the global optimum. However, in many problems, the fitness of the global optimum is unknown and the threshold cannot be set accurately. If the threshold is unreasonable, the algorithm could theoretically have an infinite running time. It is more common to end the GA after a fixed number of iterations, iterationMAX . So the stopping condition can be written in pseudocode as:

$$f(t) < f_{MAX} - \varepsilon \quad \text{OR} \quad \# \text{iterations} \geq \text{iterationMAX}$$

where $f(t)$ is the overall fitness of generation t , f_{MAX} is the fitness of the global optimum, and ε is the threshold.

Performance

The error function for the GA can be defined as the difference between the overall fitness and the fitness of the optimal. That is:

$$\text{Err}(t) = f_{MAX} - f(t)$$

where $\text{Err}(t)$ is the error of generation t , f_{MAX} is the fitness of the global maximum, and $f(t)$ is the fitness of the strongest organism in generation t . Under fitness proportional selection without elitism, the fittest organism in a generation is not guaranteed survival. The error from one generation to the next may

increase, so the scheme is unstable and hence not convergent. However, it has been shown through Markov chain analysis that the canonical elitist GA will converge to the global optimum [16], [17].

Although the GA is convergent, the average running time or convergent rate of the algorithm is unknown. This is because the nature of the fitness function can drastically alter the performance of the algorithm. Difficulty for the GA arises from "the size of the search space and irregularity of the function" [18]. The best performance measure is John Holland's $O(K^3)$ estimate of the number of regions of the search space that can be effectively explored [19]. Despite this lack of knowledge about the GA's running time, experimental results indicate the GA performs as well as or better than other stochastic optimization algorithms [5]. For so-called "long path problems" in which the path to the global optimum is so long that following the path is intractable, GAs converge much faster than hill-climbing methods [20]. For particularly noisy landscapes, such as those found in set-covering and other NP-complete problems, GAs have been found to greatly outperform hill-climbing and simulated annealing methods [21].

To optimize the performance of the GA, there are many parameters to set and design choices to make. First, one has to choose the most efficient encoding scheme that will yield the smoothest fitness landscape. Then the specific crossover, mutation, and selection schemes need to be chosen. Using an elitist strategy will guarantee global convergence, but it may hinder the algorithm's ability to explore different regions of the search space. A restart schedule should be planned to re-initialize stagnant populations. Parameters to toggle include the population size K , the mutation probability p_m , the number of parents μ that breed, the number of offspring λ , the threshold ϵ , and the maximum number of iterations. One author suggests optimizing the control

parameters by running a genetic algorithm on different GAs [22]. Unfortunately, the so-called "No Free Lunch Theorems" prove that a specific GA that "performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems [23]. That is, if a GA performs better than random search on a problem, then it will perform worse than random search on other problems. This difficulty has led some authors to believe that efficient GA design is "a holism not unlike that of Zen" [19].

Application to PSP

The encoding for the Pattern Selection Problem is just a sequential list of the A_{ij} 's. That is, a 1 for a gene indicates that a specific pattern should be chosen and 0 means the pattern should not. Since the GA maximizes the fitness, but PSP wishes to minimize the L2 error, the fitness function was the reciprocal of the error:

$$f = \frac{1}{E} = \frac{1}{\left\| I(x) - \sum_i \sum_j A_{ij} P_{ij}(x) \right\|_2}$$

where $I(x)$ is the graph of the ideal rail profile.

The GA chosen for the Pattern Selection Problem involved uniform crossover, constant mutation with $p_m=0.01$, and elitist (μ, λ) fitness proportional selection with $\mu = \lambda = N$, the number of patterns. Since memory allocation was a factor, the population size was chosen to be a relatively small $K=10$. On test data, the error E seemed to level off within thirty generations, so the algorithm was set to restart every 30 iterations. The threshold ε was set to 0.01, but this threshold was too low for the test data and the GA never ended within this error tolerance. The maximum number of iterations could be set arbitrarily high, but for most test runs the restart threshold of 30 iterations

sufficed as iterationMAX. Trial runs indicate that the GA performed better than GREEDY but worse than ES123. Also, GA was slower than GREEDY but faster than ES123 provided iterationMAX was not set too high.

4.5 Results

The test data available was relatively small with only $N=26$ patterns available (27 including the null pattern). Several random MR curves were generated and used as target curves for the four algorithms. The error was the L2-norm of the difference between the combination chosen and the target curve. Speed judged based on the number of error evaluations the algorithm made, that is, how many times a combination was compared to the ideal curve. The algorithms ranked from fastest to slowest and also from least accurate to most accurate are: ES1, GREEDY, GA, ES123. The enumerative search ES1 always made $N=27$ comparisons. GREEDY chose anywhere from 0-5 patterns, giving an average of 58 comparisons. The GA had iterationMAX=30 for the trial runs, so it made $30 \cdot K = 300$ comparisons. The slowest algorithm, ES123, had to evaluate all 1, 2, and 3 pattern combinations, so it always made 3655 comparisons.

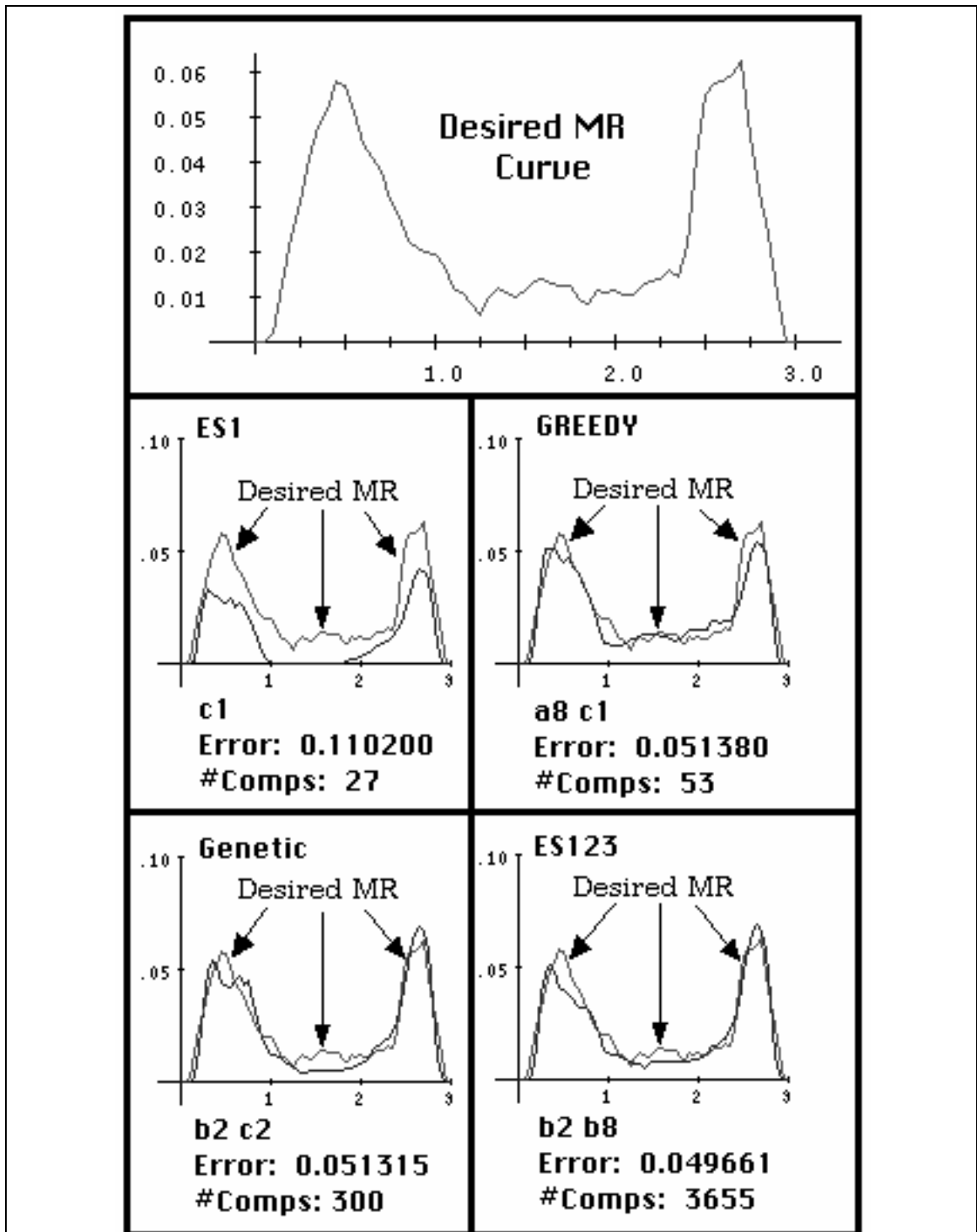


Figure 7: Comparison of Algorithms

Each of the four algorithms develops a pattern combination that best matches the desired MR curve at top. The error is the L2-Norm of the difference and "#Comps" is the number of error evaluations.

6 Conclusions and Further Research

An efficient algorithm for identifying the initial rail shape was developed and coded for the Loram C++ class library. On all test rail profiles, the algorithm accurately identified the representative shape. Four algorithms for choosing a pattern combination, given a target MR curve, were implemented. Test data indicate that the genetic algorithm may be the most appropriate algorithm for field use.

Areas for further research include:

- Acquiring MR data for three-dimensional database.
- Development of rail identification algorithm:
 - refinement of moments,
 - implementing a curvature histogram,
 - signaling severely damaged rail heads,
 - extending to recognize arbitrary shapes.
- Exploring other optimization methods:
 - hill-climbing,
 - simulated annealing,
 - integer programming.
- Refinement of the genetic algorithm:
 - exploring alternative crossover, mutation, and selection schemes,
 - adjusting control parameters,
 - hybridization with other stochastic methods.

Bibliography

- [1] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, San Diego, 1999.
- [2] Thomas Chase. *Rail Head Loss Algorithms*. Technical Report for Loram Maintenance of Way, Inc. June 1994.
- [3] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, 1995.
- [4] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms and Engineering Design*. John Wiley & Sons, Inc., New York, 1997.
- [5] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, 1992.
- [6] Geoffrey Miller, Peter Todd, and Shailesh Hegde. Designing Neural Networks using Genetic Algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 379-384, 1989.
- [7] John Dzubera and Darrell Whitley. Advanced Correlation Analysis of Operators for the Traveling Salesman Problem. *Lecture Notes in Computer Science, Vol. 866: Parallel Problem Solving from Nature -- PPSNIII*, pages 68-77, 1994.
- [8] Dirk Thierens and David Goldberg. Convergence Models of Genetic Algorithm Selection Schemes. *Lecture Notes in Computer Science, Vol. 866: Parallel Problem Solving from Nature -- PPSNIII*, pages 119-129, 1994.
- [9] David Goldberg. Zen and the Art of Genetic Algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 80-85, 1989.
- [10] Gilbert Syswerda. Uniform Crossover in Genetic Algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2-9, 1989.
- [11] Xiaofeng Qi and Francesco Palmieri. Theoretical Analysis of Evolutionary Algorithms With an Infinite Population Size in Continuous Space: Analysis of the Diversification Role of Crossover. *IEEE Transaction on Neural Networks*, 5(1): 120-129, January 1994.
- [12] Jurgen Hesser and Reinhard Manner. Towards an Optimal Mutation Probability for Genetic Algorithms. *Lecture Notes in Computer Science, Vol. 496: Parallel Problem Solving from Nature -- PPSNI*, pages 23-32, 1990.

- [13] Darrell Whitley, V. Gordon, and Keith Mathias. Lamarckian Evolution, The Baldwin Effect, and Function Optimization. *Lecture Notes in Computer Science, Vol. 866: Parallel Problem Solving from Nature -- PPSNIII*, pages 6-15, 1994.
- [14] Alex Fukunaga. Restart Scheduling for Genetic Algorithms. *Lecture Notes in Computer Science, Vol. 1498: Parallel Problem Solving from Nature -- PPSNV*, pages 357-366, 1998.
- [15] Dick Thierens and David Goldberg. Convergence Models of Genetic Algorithm Selection Schemes. *Lecture Notes in Computer Science, Vol. 866: Parallel Problem Solving from Nature -- PPSNIII*, pages 119-129, 1994.
- [16] Gunter Rudolph. Convergence Analysis of Canonical Genetic Algorithms. *IEEE Transactions on Neural Networks*, 5(1): 96-101, January 1994.
- [17] Alexandru Agapie. Genetic Algorithms: Minimal Conditions for Convergence. *Lecture Notes in Computer Science, Vol. 1363: Artificial Evolution*, pages 183-193, 1998.
- [18] Evelyne Lutton and Jacques Levy Vehel. Hölder Functions and Deception of Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 2(2): 56-71, July 1998.
- [19] David Goldberg. Sizing Populations for Serial and Parallel Genetic Algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 70-79, 1989.
- [20] Jeffrey Horn, David Goldberg, and Kalyanmoy Deb. Long Path Problems. *Lecture Notes in Computer Science, Vol. 866: Parallel Problem Solving from Nature -- PPSNIII*, pages 149-158, 1994.
- [21] Dave Corne and Peter Ross. Some Combinatorial Landscapes on which a Genetic Algorithm Outperforms Other Stochastic Iterative Methods. *Lecture Notes in Computer Science, Vol. 993: Evolutionary Computing*, pages 1-13, 1995.
- [22] John Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1): 122-128, January/February 1986.
- [23] David Wolpert and William Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1): 67-82, April 1997.