Dispatching Mineral Deposits by Rail using Genetic Algorithms

V. Salim
Department of Information Management and Marketing
University of Western Austalia

X. Cai
Department of Systems Engineering and Engineering Management
Chinese University of Hong Kong

Abstract The particular problem of mineral dispatching examined in this paper involves the transportation of mineral ore using a rail system. An optimal strategy for transporting the cargo should consider two important criteria: first, none of the trains conflicts with any other en route; second, the costs of each schedule related to delay and stopping should be minimised. The latter two imply that it is desirable for a train to reach its destination in the shortest time possible. The problem at hand is particularly well suited to a genetic algorithmic formulation as it is an NP-hard problem and, hence, it is impossible in practice to use a constructive algorithm to obtain a solution. Various GA specifications are presented for an example extracted from a real system and the results obtained are compared against each other.

1. INTRODUCTION

The efficient transportation of mineral ore by rail requires a timetable, or schedule, dictating arrival and departure times of the trains at given junctions along a railway system. Timetables should feature the shortest possible transit times for all trains in order to minimise costs. This characteristic is distinct from trains carrying passengers, where the arrival and departure times are fixed. In this case, only the departure time of each train is given. Certain salient constraints must also be observed: the trains must arrive at their given destinations, and no collisions must arise from the time schedule.

The model of the railway system being considered is based on the Mt Newman mines railway system in Western Australia. It comprises a single track long haul railway line with passing loops where trains may enter and wait to avoid a head-on collision, or overtake other trains. In the present model, overtaking is not considered. An optimal schedule to transport mineral ore by rail should minimise economic, environmental and social costs. A genetic algorithm (GA) taking into account the economic and environmental costs was presented in Salim and Cai (1995). One of the aims of this paper is to test the algorithm described therein against some other GA specifications to obtain the best performing specification.

2. PROBLEM STATEMENT[1]

The specific task of the algorithm is to generate arrival and departure times of every train in the system at every passing loop on the railway line. Hence, the algorithm dictates how long a train waits in a passing loop for another to pass, ensuring that no collisions or infeasibilities arise. At the same time, a cost function for the schedule which factors costs associated with delays and stopping is to be minimised. The problem is cast into a form easily analysed by a GA through using a binary $p \times q$ matrix, the decision matrix, where p is the number of trains in the system and q is the number of passing loops. Every element in the decision matrix is a decision variable such that:

$$x_{ik} = \begin{cases} 0 \text{ if train i is required to stop at passing loop k and} \\ 1 \text{ if train i is not required to stop at passing loop k} \end{cases}$$

The cost function of a decision matrix m depends on γ_{ik}, the cost of stopping train *i* at loop *k*, μ_{ik}, the cost of train *i* being delayed per unit time and ω_{ik}, the time that train *i* stops at loop *k*. Therefore,

$$C(m) = \sum_{i=1}^{p} \sum_{k=1}^{q} \gamma_{ik} + \mu_i \omega_{ik} \qquad (1)$$

is the cost function to be minimised. The advantage with this simple formulation is that γ_{ik} and μ_{ik} inherently contain information about train priorities as well as load. The main constraints to be satisfied, which are self-evident, are listed below:

2.1 Constraints

No collisions should occur.
No more than one train can occupy a passing loop.
One train must stop when two trains are at a loop.

[1] For a more detailed exposition of the problem, see Salim and Cai (1995)

3. ANALYSIS AND DESCRIPTION OF ALGORITHMS

For simplicity, and because the Mt Newman system is just a single line, we first examined a single railway line assumed to run north-south. The algorithm derived previously to solve the scheduling problem in this case given in Salim and Cai (1995) will be summarised. Then, improvements are made to the algorithm.

3.1 Algorithm One

The algorithm is initialised by a population of decision matrices whose 0,1 elements are randomly generated. Three situations arise from this formulaion.

1. It will be likely that there will be possible crashes in the resultant randomly determined schedules.

2. The decision matrix may direct a train to stop when it is not necessary (that is, when there is no impending conflict with another train).

3. The binary elements in the matrix contains no information about how long a train is required to stop and wait at a passing loop if the decision is 0.

In order to "educate" the GA about point 1, a penalty function is added to the cost function (equation (1)) which penalises the number of infeasibilities in a decision matrix, *m*. A combined objective function is defined as

$$C(m) = \sum_{i=1}^{p}\sum_{k=1}^{q}(\gamma_{ik} + \mu.\omega_{ik}) + h(x) \qquad (2)$$

where *h(x)* is the penalty function with *x* being the number of infeasibilities in the schedule *m*. A method is required to detect crashes and to count them. The possible combinations of infeasibilities are now considered between passing loops *l* and *l*+1.

3.1.1 Counting Infeasibilities

Clear Crashes

Let a northbound train *k* depart from passing loop *l* at time τ^{d}_{kl}. Let a southbound train *n* depart from loop *l*+1 at time $\tau^{d}_{n(l+1)}$. The time of arrival of the trains at the next passing loop can be easily calculated given the cruising speeds of the trains and the distance between the two given passing loops. Then let $\tau^{a}_{k(l+1)}$ and τ^{a}_{nl} denote the respective arrival times. If

$$\tau^{d}_{n(l+1)} \le \tau^{a}_{k(l+1)} \textbf{ and } \tau^{d}_{kl} \le \tau^{a}_{nl}$$

then we have a clear crash. Examples of these conditions are given in figures 1and 2, where it should be noted that $td(n)(l)=\tau^{d}_{nl}$ and $ta(n)(l)=\tau^{a}_{nl}$.

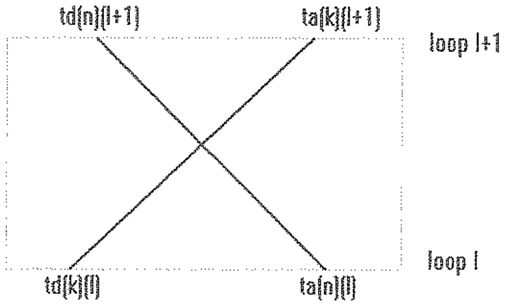

Figure 1. A clear crash between loops *l* and *l*+1.

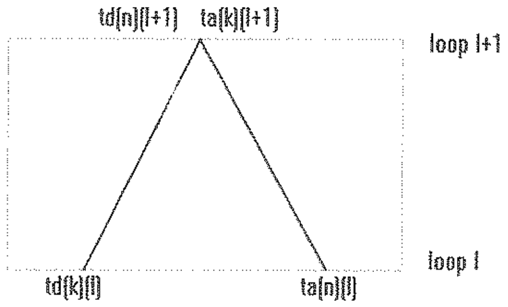

Figure 2. A clear crash at loop *l*+1

Trains occupying the same loop

Two trains are not permitted to be waiting in the same passing loop at the same time. Detection of this is in the same way as above. Assuming that trains k and n both have positive waiting times, if any of the following inequalities holds, then we have an infeasibility.

1. $\tau^{a}_{nl} = \tau^{a}_{kl}$ **and** $\tau^{d}_{nl} = \tau^{d}_{kl}$, where k and n are heading in opposite directions;

2. $\tau^{a}_{kl} < \tau^{a}_{nl} < \tau^{d}_{kl}$ **and** $\tau^{d}_{kl} < \tau^{d}_{nl}$, where k and n are both heading in the same direction;

3. $\tau^{a}_{nl} < \tau^{a}_{kl} < \tau^{d}_{nl}$ **and** $\tau^{d}_{kl} > \tau^{d}_{nl}$, where k and n are both heading in the same direction.

An example of trains occupying the same loop is given in the figure below:

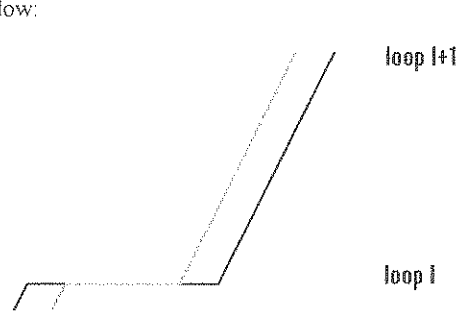

Figure 3. The two trains are occupying the loop *l* at the same time.

Coalescing trains

This last infeasibily occurs only with trains heading in the same direction. This case is detected if the arrival and departure times are sufficiently close for two trains heading in the same direction.

3.1.2 Fitness function

The fitness function of each individual m is calculated as

$$f(m) = C_{\max} - C(m) \qquad (3)$$

where $C_{\max} = \max_j\{C(j)\}$ is the highest cost in the population.

Points 2 and 3 are closely related as a train needs to stop and wait for some period of time in loop k only when there will be a clear crash with another train between loops k and $k+1$, or between loops k and $k-1$. When an element in the decision matrix is 0 and there is no impending conflict, then the algorithm changes the decision to 1 so that the train does not stop. Clearly, if the train does not stop, then waiting time need not be calculated. In the reverse case where the decision is 1 in the face of a collision between two trains, two factors must be determined: which train stops, and for how long?

Consider a northbound train i that arrives at loop k at time τ_{ik} and a southbound train j that arrives at loop $k+1$ at time τ_{jk+1}. Let the time taken to travel the track section between loop k and $k+1$ be T_k. If both trains continue without stopping, then the result will be an infeasible intersection. Suppose train i stops while train j passes, then clearly the minimum waiting time is given by equation (4) below:

$$\omega_i = \tau_{jk+1} + \left(T_k - \tau_{ik}\right) + \delta(i,j) \qquad (4)$$

Similarly, if train j stops while train i continues, then the delay is given by equation (5):

$$\omega_j = \tau_{ik} + \left(T_k - \tau_{jk+1}\right) + \delta(i,j) \qquad (5)$$

$\delta(i,j)$ is a safety tolerance for small factors which may not have been accounted for. A train is stopped according to the criterion in the following section.

3.1.3 Conflict Resolving Criterion

If $\gamma_{j(k+1)} + \mu_j\omega_{jk+1} \le \gamma_{ik} + \mu_i\omega_{ik}$, then let train i pass and train j stop and wait at loop $k+1$. Otherwise, let train j pass and train i stop and wait at loop k.
After the fitness values for every matrix in the random population has been determined, in order to obtain another generation of improved solutions, the genetic operators reproduction, crossover and mutation are used.

3.1.4 Reproduction

The process used here is generational reproduction, with the probability that an individual is chosen given by

$$P_{si} = \frac{C_{\max} - C(i)}{\sum_j (C_{\max} - C(j))} \qquad (6)$$

Note that the individuals with fitnesses equal to $C_{\max}$ are deleted from the new population with probability 1. This results in a gradual shrinking of the gap between the fittest and least fit in the population.

3.1.5 Crossover

During this recombination step, two point crossover is used. The properties of two point crossover, when performed on matrices, is different from when it is used on strings. Two point crossover on matrices would be a natural extension to one point crossover on strings. Say cutpoints (x_1, y_1) are chosen. Then (x_1,y_1) is the lower right hand corner element of the submatrix in a matrix m_1 that is to be interchanged with a different matrix m_2 to produce two new matrices.

3.1.6 Mutation

We carry out mutation as the random alteration of an element in the decision matrix from 0 to 1, or vice versa. The elements corresponding to starting and destination loops are not mutated.

Algorithm One can now be stated as follows.

Algorithm One

Step Zero -- Initialization

Generate n=popsize binary matrix solutions randomly. These solutions form the initial population POP[0]. Let k=0.

Step One -- Determine waiting time

Find the arrival and departure times of each train at every loop by using stopping and starting matrix schedules, through calculating the required waiting times as specified in equations (4) and (5) of Section 3.1.2, and using the Conflict Resolving Criterion. Update POP[k].

Step Two -- Determine costs

For every solution $i=1,..,n$, evaluate the cost C(i) of POP[k].

Step Three -- Reproduction

Select n individuals from POP[k] according to probability P_{si}. This forms the population POP[k+1].

Step Four -- Crossover and Mutation

Choose p parents, where p is the nearest integer value to $n/2$, from POP[k+1]. For each pa rent perform crossover with probability P_c. Update POP[k+1] by replacing the parent chromosomes by their offspring. Then mutate.

Step Five -- Convergence Check

Terminate if the var(POP[k+1])<ε, where ε is some small number.

Otherwise, set k:=k+1 and return to Step One.

The effectiveness of Algorithm One above was tested and, using an example with 12 passing loops, 4 northbound trains and 5 southbound trains, a good feasible result for the minimum cost was $C(m^*)$=108.4. This is a dramatic improvement to the minimum of $C(m^*)$=662.5 obtained by just running the local optimality criterion without the genetic operators. The same control problem was also solved by Cai and Goh (1994), and the solution obtained therein was 142. Further improvements made on the performance of the algorithm through modifications are discussed below.

3.2 Improving Algorithm One

3.2.1 Algorithm Two

Improvements in the convergence rate of the algorithm were achieved by using an elitism selection scheme. Furthermore, instead of using the Conflict Resolving Criterion, the algorithm was altered so that either train may stop and enter a passing loop to avoid a conflict with equal probability. Experimentation showed that a quadratic, moderate penalty function, $h(x) = 20x^2$, where x is the number of crashes, yields the lowest minima This moderate penalty function works well since the more severe the penalty, the lower the resulting P_{si}'s in the population and the higher the number of iterations required for the reproduction step. The abovementioned modifications resulted in the algorithm finding better solutions due to an increased diversity in the population at each iteration. However, the moderate penalty function could give rise to solutions which are low in cost but infeasible. Therefore, two simple restrictions were imposed:

1. If the variance of the population was small and the minimum is infeasible, then increase mutation rates;

2. Modify the convergence check step in the following way:
   Terminate the algorithm if var(POP[k+1])<ε and if the minimum in the population is feasible.

The algorithm was run 20 times. The best solution obtained on a particular run was $C(m^*)$=86.3 (see figure 4), which is presumed to be the global minimum.

However, there is a problem with replicability in this algorithm. Repetitions of the same algorithm seldom converge to the presumed global minimum. In addition, the algorithm often prematurely converges to the local feasible minimum, $C(m_l)$=122.4 (see figure 5).

The train graph for this minimum has a small number of stops but long waiting times, and this feature seems to propagate well through the population. The restrictions placed on the algorithm also implies that the GA has very limited information to exploit if the variance is small but the minimum is still infeasible. Algorithm Three is devised to overcome the problems in Algorithm Two.

3.2.2 Algorithm Three

Step Zero - Step Two --

Perform these steps as in Algorithm One.

Step Three -- Select and Cross.

Find all the duplicate strings in POP[k].

For every string with a duplicate in the population, choose parents with probability P_{si} given in equation (7).

Perform crossover with probability P_c. If crossover is performed, replace the duplicate string(s) with the newly formed chromosomes.

If no duplicate strings exist in the population, then select two strings with probability P_{si} and perform crossover with probability P_c. Start replacement from the least fit strings in the population with the newly formed pair of chromosomes.

Update POP[k].

Step Four -- Mutation

Perform mutation as in Algorithm One.

Step Five -- Convergence Check

Rank the individuals in ascending order of cost. If k = 0, then set min:=C(1). If k > 0, then if C(1) < min, set k:=0 and min:=C(1). If C(1) = min, then set k:=k+1. Terminate when k = N, where N is the upper bound on the number of iterations for which the best string of the population has not changed.

The only differences between Algorithms Two and Three are that the above algorithm imposes the condition that every string in the population is unique at any one step, and the entire population is not replaced at every iteration. Table 1 shows that this method has countered the problem of premature convergence. Of 10 runs, the algorithm did not converge to the presumed global minimum only once. A higher number of iterations is required as a criterion for convergence as fewer recombination processes occur at each iteration.

4. CONCLUSION

The specification of a GA is found to affect the performance of a GA to a great extent, although the underlying coding of the problem into a form solvable by GA remains identical. Based on the numerical comparison of 3 algorithms, a GA which performs very well has been formulated, which arrives consistently at (or close to) the global minimum. Algorithm Three does not compromise computing time. For the test problem, with 12 passing loops, 9 trains and a population size of 100, 10,000 iterations can be achieved on modest computing resources[2] in approximately one and a half hours. Further research will focus on applying the GA to the case of a more general network.

[2] The programs were written in Turbo Pascal for Windows on a 486 computer.

Repeat number	Minimum at convergence	Iterations at convergence
1	86.3	12396
2	95.3	10270
3	86.3	15992
4	86.3	13459
5	86.3	16199
6	86.3	11475
7	86.3	18269
8	86.3	10129
9	86.3	10003
10	86.3	11989

Table 1. Results from Algorithm Three. Population size=100, N=10 000, P_c=0.7, P_m=0.05.

Acknowledgements

The authors would like to thank Michael McAleer and John Taplin for helpful comments and suggestions.

References

Cai, X. and Goh, C.J. (1994), A Fast Heuristic for the Train Scheduling Problem, *Computers and Operations Research: An International Journal*, **21**, pp. 499-510.

Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading.

Michalewicz, Z. (1992), *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York.

Salim, V. and Cai, X. (1995), A Genetic Algorithm for Railway Scheduling with Environmental Considerations, forthcoming in *Environmetrics*.

Sysweda,G. (1991), A Study of Generational and Steady State Genetic Algorithms, in G.J.E. Rawlins (ed.), *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, California.

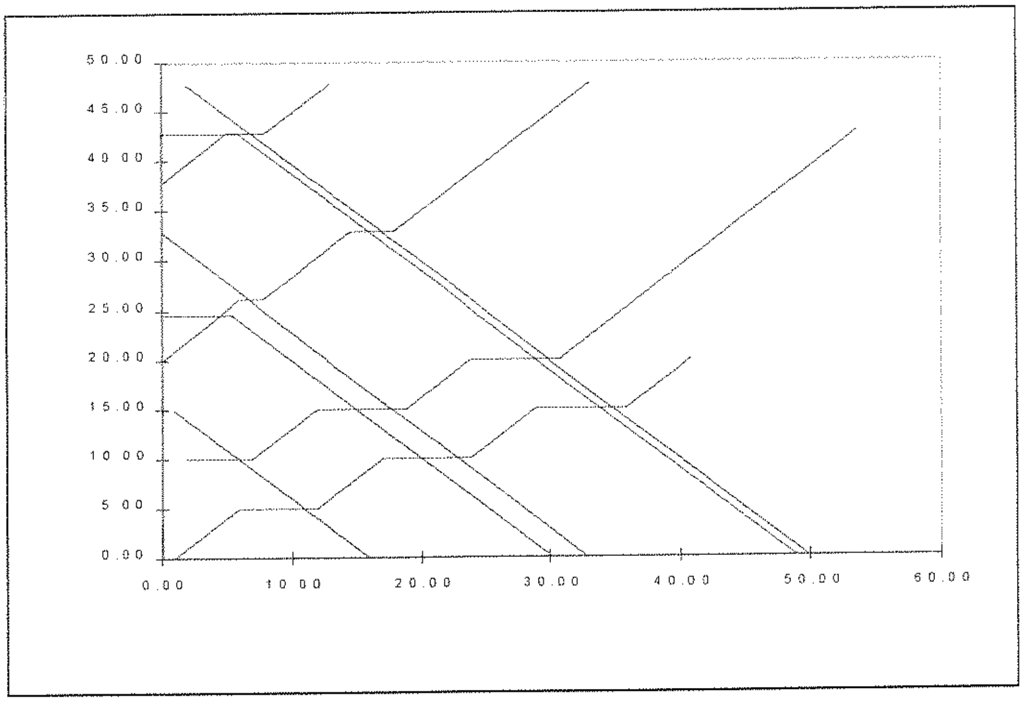

Figure 4. Distance (vertical axis) and time (horizontal axis) graph for the generated schedule with cost 86.3.

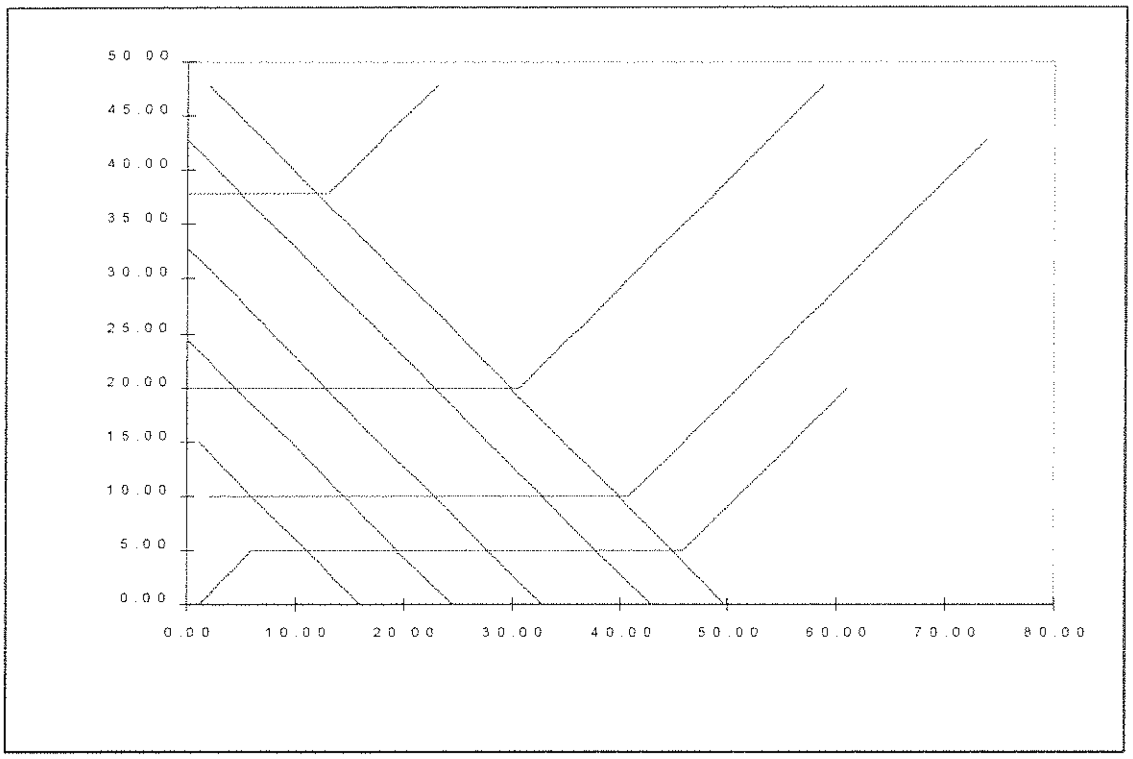

Figure 5. Distance (vertical axis) and time (horizontal axis) graph for the generated schedule with cost 122.4.