A Real-World Genetic Algorithms Case Study: Optimising Daily Production in a Gold Mine

Charles Tonkinson, Stephen Greig, Karl van Olden*
Knowledge Engineers, Knowledge Based Engineering
Corporate Place, 23 Fredman Drive, Sandton, Johannesburg, South Africa
Phone: +27-11-8840510, Fax: ++27-11-8840471
Email: charlest@kbe.co.za, stepheng@kbe.co.za
*Mining Engineer, Western Areas Ltd.
P.O. Box 57, Western Areas, 1780
Phone: +27-11-7515961, Fax: +27-11-7515914
Email: kvanolde@wagm.co.za

ABSTRACT: This paper deals with optimising the daily schedule of blasting in a gold mine. The ore-body is to be mined using a modified cut and fill method. Each cut is mined by blasting drifts in a grid pattern throughout the exposed portion of the ore-body (the cut). The sequence in which these drifts are blasted is limited primarily by rules concerning the availability of drifts.
A particular cut takes of the order of 200-400 days to mine, with four blasts each day. The primary objective for this project was to be able to schedule blasting for the mining life of the cut such that a minimum amount of gold - referred to as the pay-limit - is extracted each day. This was successfully achieved using genetic algorithms where a day's sequence of blasting is considered to be a gene and a full sequence of blasting throughout the total cut is considered to be a chromosome.

KEYWORDS: Genetic algorithms, gold mine, daily optimised blast scheduling

INTRODUCTION

This project was concerned with the South Deep Section of Western Areas Ltd.

THE MINING OPERATIONS AT THE SOUTH DEEP SECTION

Mining operations can be divided into two distinct activities:

- Mining of rock outside of the target ore-body in order to gain access to the ore (development)
- Mining of the gold-bearing ore in order to produce the revenue generating product (stoping)

Development does not produce revenue and is kept to a minimum in order to reduce the overhead costs of the operation. Stoping is controlled by the geology of the ore-body. The major geological features of any ore-body are the structural characteristics and the varying concentrations of the target mineral. The concentrations of gold in the ore-body (grade) vary quite considerably between adjacent portions of the deposit. The operation is further complicated by structural disturbances such as faults and sedimentological features. This can result in a situation where it is desirable to blast around an "island" or "pillar" of very low grade ore.

The South Deep ore-body is planned to be mined using a modified cut and fill method. This method exposes horizontal sections of the ore-body approximately 4 m high and with an area of 150 m by 250 m. Each cut is mined using a drift and fill method whereby drifts with dimensions of approximately 4 m wide by 4 m high are blasted in a grid pattern throughout the exposed portion of the ore-body (Figure 1). These production drifts can be mined in any sequence, provided that certain operational rules and physical constraints are followed. Each drift is mined to its limit and is then

backfilled using the waste product from the beneficiation process. The backfill is placed and allowed to settle for a period of 28 days. After this settling period, the ore in the drifts adjacent to the backfilled area is available to be mined.

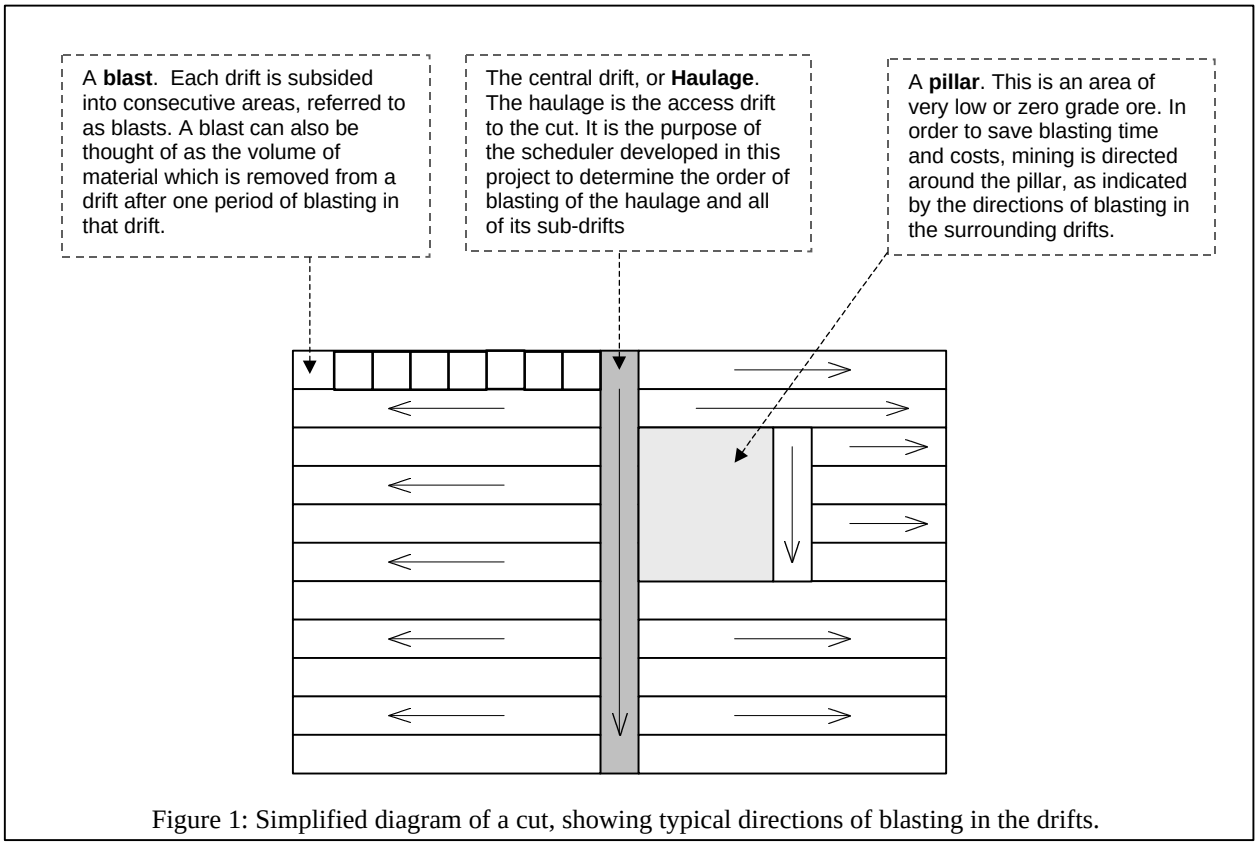

Figure 1: Simplified diagram of a cut, showing typical directions of blasting in the drifts.

The major parameters which govern the effective operation of this mining method are:

- the average grade, and the variation of grade over the cut
- the choice of initial drifts for blasting
- the profitability of the operation on a daily basis
- the maximum extraction of the target mineral - gold in this case - over the life of the operation.

INIITAL SCHEDULING CONSIDERATIONS

Each cut of the ore-body consists of approximately 2000 blasts that are scheduled over a period of between 300 and 500 days. The grade in each blast can vary from 5 grams gold per ton of rock to 30 g/ton and higher. The aim of the scheduling process is to ensure that there is a consistent feed of broken rock at an average grade that is profitable. The *pay-limit* of an ore is the minimum value at which it can be mined and treated without profit or loss, i.e. when revenue obtained from the specific mineral product balances expenditure incurred in mining and treating the ore. It is used to classify ore either as payable or unpayable. The *target-grade* value could be the value of the *pay limit* or could be higher in order to establish a profit margin that meets the objectives of the operation. The scheduling process is aimed at extracting the ore-body consistently at the *target-grade*.

The scheduling process is further complicated by the fact that the initial valuation of the distribution of the grade is an estimation derived from the application of geostatistical techniques to scattered samples within the ore-body. As mining progresses, ongoing sampling increases the density of data points within the ore-body and hence increases the accuracy of and changes to the grade estimation. Another factor to consider in the initial scheduling of the extraction is that the operation is subject to machine breakdowns and other unforeseen events. It is therefore apparent that a scheduling technique will require the flexibility and ease of use necessary to react to the changing nature of the circumstances, in a time period that enables the implementation of new information in a continuous real-time environment.

THE RULES OF THE CUT AND FILL MINING METHOD

The cut and fill method rules, as proposed for the South Deep Section, can be summarised as follows:

1. There are two shifts per day. In each shift, two blasts may be blasted, allowing a total of four blasts per day, or a **blasting-period** of four blasts.

2. Within a drift, only one blast may be blasted per blasting-period. This is to allow time for removal of blasted ore and to prepare the face of the drift for the next blast.

3. A complex air circulation and cooling system is required underground. The number of open faces – i.e. the number of drifts being blasted – must be kept below a certain maximum. This is because as the number of open drifts increases, the following factors also increase:
   3.1. the rate of underground heating due to increased exposed surface area
   3.2. the cost of cooling due to the increased distance over which cooling systems must operate

4. The most important rule has to do with the backfilling and settling of drifts. Once a drift has been fully mined (all its blasts have been blasted), it must be backfilled and the backfill must be allowed to settle (for 28 days) before adjacent drifts can be blasted (Figure 2). Another way of putting this is that parallel drifts may never be blasted in parallel.

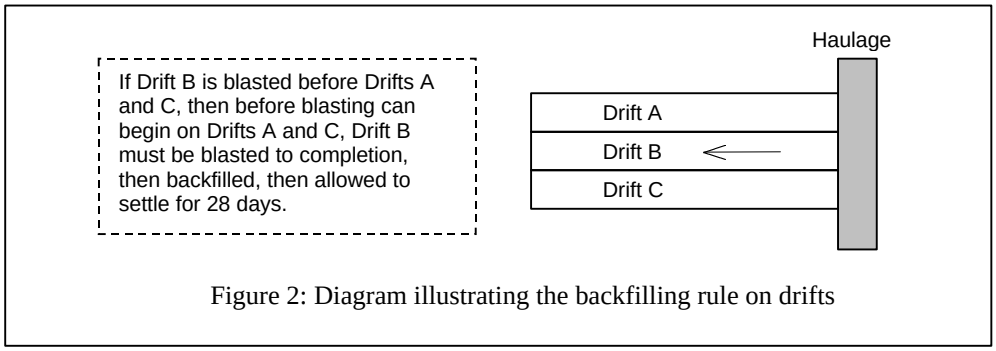

Figure 2: Diagram illustrating the backfilling rule on drifts

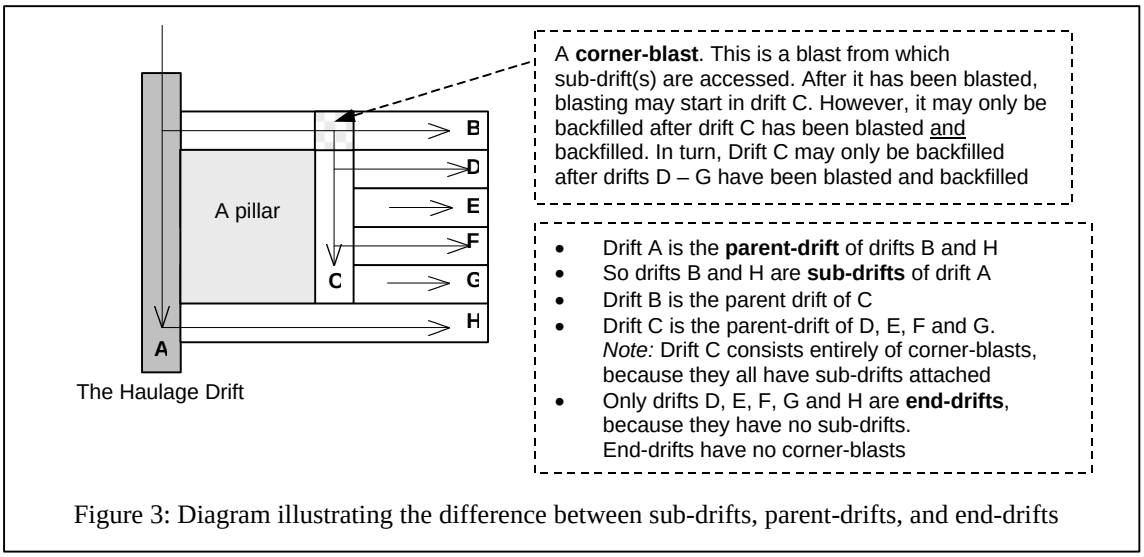

Figure 3: Diagram illustrating the difference between sub-drifts, parent-drifts, and end-drifts

5. The next set of rules arises as a consequence of the existence of pillars (Figure 1) in a cut. In order to understand these rules, the terms parent-drift, sub-drift and end-drift are defined in Figure 3:
   5.1. A parent drift must be blasted up to and including the relevant corner blast (Figure 3) before blasting may commence in sub-drift(s).

5.2. Parent drifts may not be fully backfilled until all their subdrifts are backfilled. For example, the corner blast indicated in Figure 3 may only be backfilled once its' sub-drift (Drift C) is backfilled. Sub-drifts do not need to be settled before backfilling of their parent-drifts continues.

5.3. When a parent-drift becomes fully blasted, and if there are non-corner blasts at the end of the parent-drift, backfilling may immediately commence provided that it stops just before the first corner blast whose sub-drift(s) are not yet backfilled. Sometimes this partial backfilling is a necessity; for example, if Drift B in Figure 3 is blasted to completion before blasting commences in drift D, then according to rule 4, Drift B must be backfilled up to its corner-blast (and the backfill must settle) before blasting can commence in Drift D.

5.4. End-drifts are always blasted last and backfilled first

6. The final rule has to do with the logistical difficulties of removing ore from two blasts "close" to each other in the same blasting-period. Figure 4 illustrates this rule.

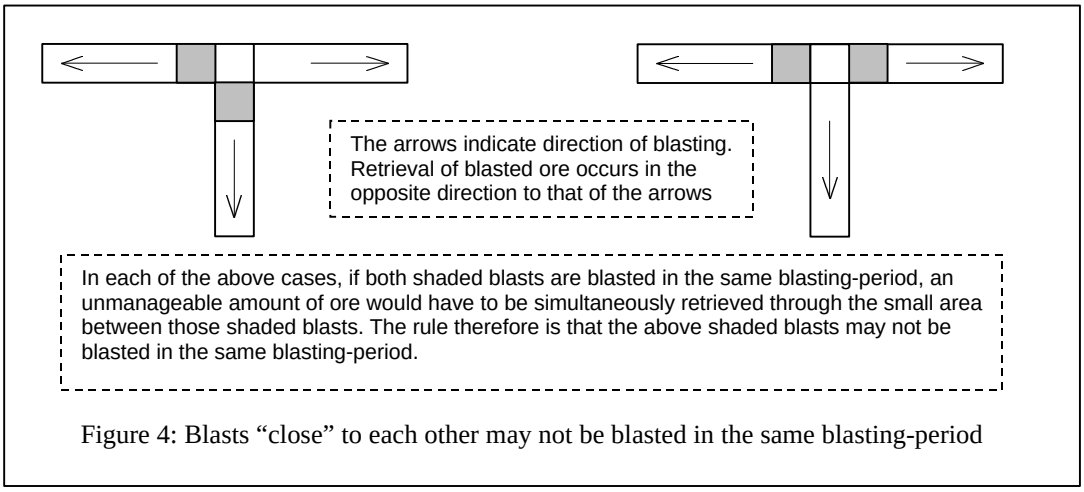

Figure 4: Blasts "close" to each other may not be blasted in the same blasting-period

THE PROJECT SPECIFICATION

The specification of the project by Western Areas Ltd. was as follows:

1. To create a scheduling tool for individual cuts that generates a blasting schedule for the entire cut.
2. The primary objective of the schedule is to minimise the number of days on which the total kg gold extracted is less than the pay-limit.
3. The scheduling tool must allow for the existence of pillars, an example of which is shown in Figure 3.
4. The tool should allow a user to add optional constraints:
   - 4.1. The user should have the ability to give a priority value to individual blasts. When the scheduler is choosing a blast, and if there is more than one available blast, it will choose a blast of the highest possible priority.
   - 4.2. The user should have the ability to prohibit backfilling of certain drifts. The rationale here is that some subdrifts may eventually act as the haulage drift to new cuts, which means they can't be backfilled.
   - 4.3. The user must be able to specify the maximum number of end-drifts (drifts without sub-drifts attached – Figure 3) which may be open at any one time. This is to allow for cooling constraints in the cut.
5. The tool must have a graphical interface which allows:
   - 5.1. Manual pre-scheduling. This facility is particularly important if the tool is eventually to be used on-line, as is the intention. It may be the case that the tool generates a satisfactory schedule which is followed for x days, but on day $x + 1$, a deviation from the schedule occurs due to unforeseen circumstances. In this case, the user must be able to enter into the tool the sequence of blasts performed up to and including day $x + 1$. The tool should then generate a new schedule from day $x + 2$ until the cut is fully mined.
   - 5.2. A user to visually step through a proposed schedule, and see blasts being blasted as well as the progress of backfilling and settling of drifts.
   - 5.3. Loading and graphical representation of data describing cuts of different dimensions, with drifts of varying lengths (i.e. varying numbers of blasts), and any number of pillars within the cuts.
   - 5.4. Graphical displays of statistics such as daily kg's gold plotted against time and the average grade in the cut.

5.5. Setting of the optional constraints mentioned in 4. above.

6. The tool must arrive at an acceptable solution (defined later - a global "best" solution is not essential) within a reasonable time frame – less than half an hour, typically.

SOFTWARE TOOLS USED – GENSYM'S G2®AND C++

G2 is normally described as a real-time expert system. However, it also has a number of features which made it appropriate for this project:

- G2 is an incremental development environment – an extremely advanced real-time translator. This means that the pseudo-compiling of G2 code is hidden from the developer and takes place automatically. This property of G2 together with its rich and descriptive syntax makes it an ideal rapid-application-development environment. G2 was therefore used as a proof-of-concept and prototyping tool for the scheduling application.
- G2 has very powerful native graphical and animation tools which make user-interface and front-end generation comparatively simple activities.
- G2 is object-oriented.
- G2 has full procedural syntax together with all the usual procedural control structures.
- G2 has a C application programmer's interface, which allows delegation of computationally-intensive code to C or C++.

A disadvantage of incremental development environments is that their performance is nowhere near as good as compiled languages such as C++. Therefore, after a proof-of-concept study of using genetic algorithms was successfully completed in G2, the genetic algorithm was migrated from G2 to C++. G2 was then used simply as a front-end to the C++ application.

The final scheduling application therefore consists of a G2 user-interface, which interacts with a C++ genetic algorithm program.

REPRESENTING THE MINING RULES

At any one time during the generation of a schedule for blasting, there will be blasts that are available and blasts that are unavailable. Selecting a blast for blasting must automatically cause a reduction in the domain of available blasts. Conversely, when a blast becomes settled (after backfilling) a possible increase in the domain of available blasts follows. The rules of how propagation through the domain of available blasts must occur are partially documented below.

BLASTING RULES : EFFECTS AND DEPENDENCIES

When a blast is blasted, it has *Effects* on the availability for blasting of the blasts surrounding it. A blast which causes an effect is the **G**overning-blast, whereas the blast that is on the receiving end of the Effect is the **D**ependent-blast. There are two types of effects:

1. Blasting of a governing-blast causes the dependent-blast only to become (possibly) available at the end of the current blasting-period – denoted as an **EndOfBP** Effect..
2. Blasting of a governing-blast causes the dependent-blast only to become (possibly) available when the governing-blast has become backfilled and settled – denoted as a **Settled** effect.

Conversely, in order for a blast to be blasted, a varying number of *Dependencies* have to be fulfilled. A blast may have three types of dependencies:

1. The dependent-blast may have to wait to the end of a blasting-period if a particular governing-blast was blasted first. (EndOfBP)
2. The dependent-blast may have to wait until a particular governing-blast is backfilled & settled before it can be blasted. (Settled)
3. With the exception of the very first blast in the haulage of a cut, every blast is dependent on having the blast "behind" it blasted first.

An Effect for one blast will always be a Dependency for another. In order for a blast to be available for blasting, all its Dependencies must have been deactivated.

Some examples of Effects are given in Figure 5 below.

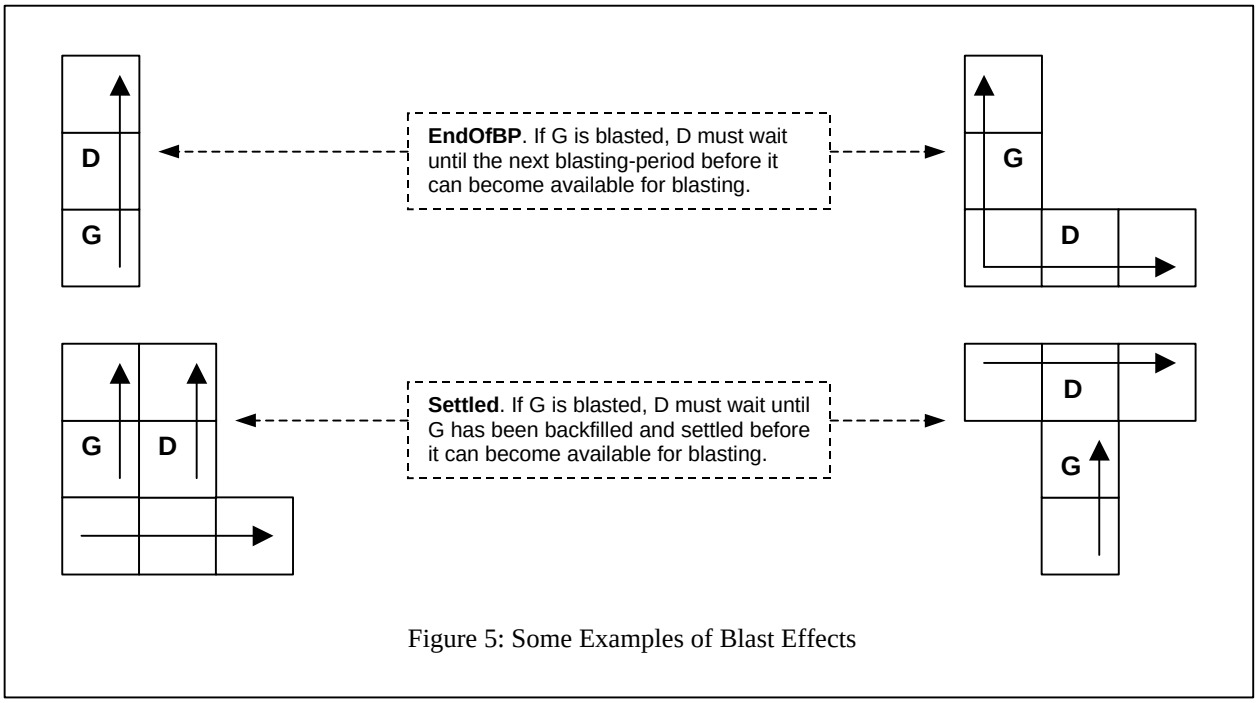

Figure 5: Some Examples of Blast Effects

BACKFILLING RULES

In order for any blast B to be backfilled, the following conditions must be fulfilled:
1. The drift of B must be allowed to be backfilled.
2. If B is not the last blast in a drift, then the next blast in the drift must already have been backfilled.
3. If B is a corner-blast (it has subdrifts), then its subdrifts must already have been backfilled.

THE GENETIC ALGORITHM

GENES AND CHROMOSOMES

In this application, a gene was conceptualised as a day's blasting. This consists of a sequence of up to four blasts. A chromosome consists of as many genes as are required to complete mining of the ore body in a cut.

An extremely important factor here is that it is always the earlier genes in the chromosome that have the largest effect on the cost function for the chromosome. This is because the starting drifts are chosen in the initial genes. This eliminates the option of blasting adjacent drifts by well over a month (the settling of a drift takes 28 days alone). Choosing a different combination of starting drifts dramatically changes the propagation of available and unavailable blasts, which will then result in a significantly different schedule.

THE COST FUNCTION

Figure 6 below illustrates the four bands into which the amount of gold extracted in one blasting-period may fall.

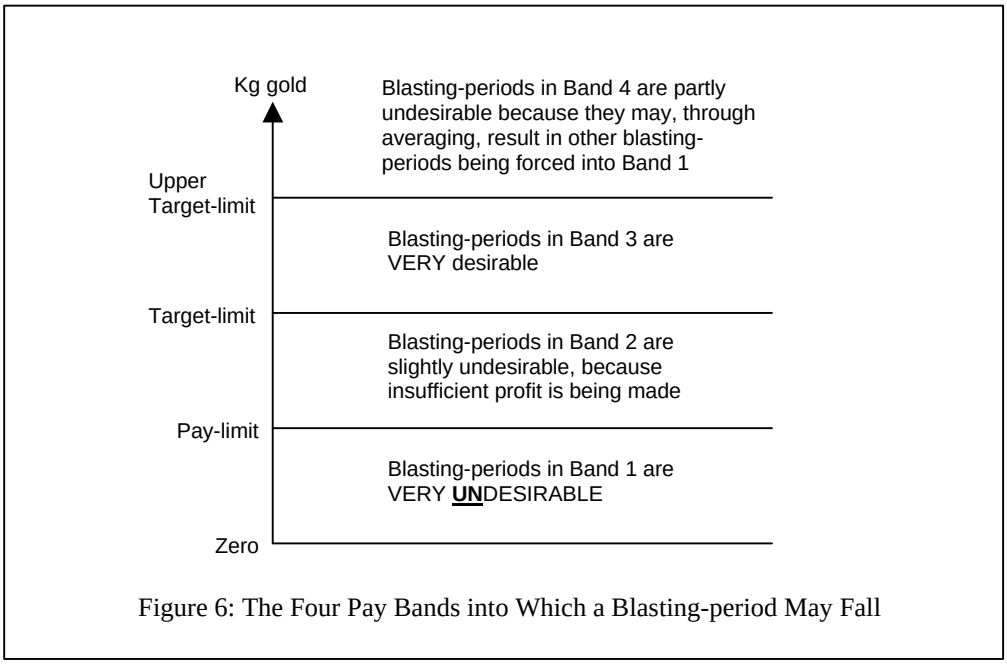

Figure 6: The Four Pay Bands into Which a Blasting-period May Fall

The ideal schedule is one in which all blasting-periods fall into Band 3. Realistically, a good schedule is one in which the number of blasting-periods in Band 1 has been minimised. The cost function therefore imposes a heavy penalty for blast-scenarios falling into Band 1, a small penalty for blast-scenarios falling into Band 2, a large reward for occurrences in Band 3, and a slight reward for occurrences in Band 4.

In addition, penalties have a proportionately smaller weighting if bad blasting-periods occur later in the chromosome. The philosophy is that if there have to be bad days, rather have them later than earlier. In other words, it is important that blasting-periods of good grade (better than Band 1) occur early in the chromosome, and blasting-periods of bad grade are pushed to the end of the chromosome.

RANDOM CHROMOSOMES

The first generation of chromosomes are generated randomly. Genes are built up by randomly selecting blasts from the domain of available blasts. Subsequent generations are produced by crossovers and mutations of the previous generations' chromosomes.

In order to demonstrate examples of mutation in, and crossover of chromosomes, we will use the simplified cut displayed in Figure 7.

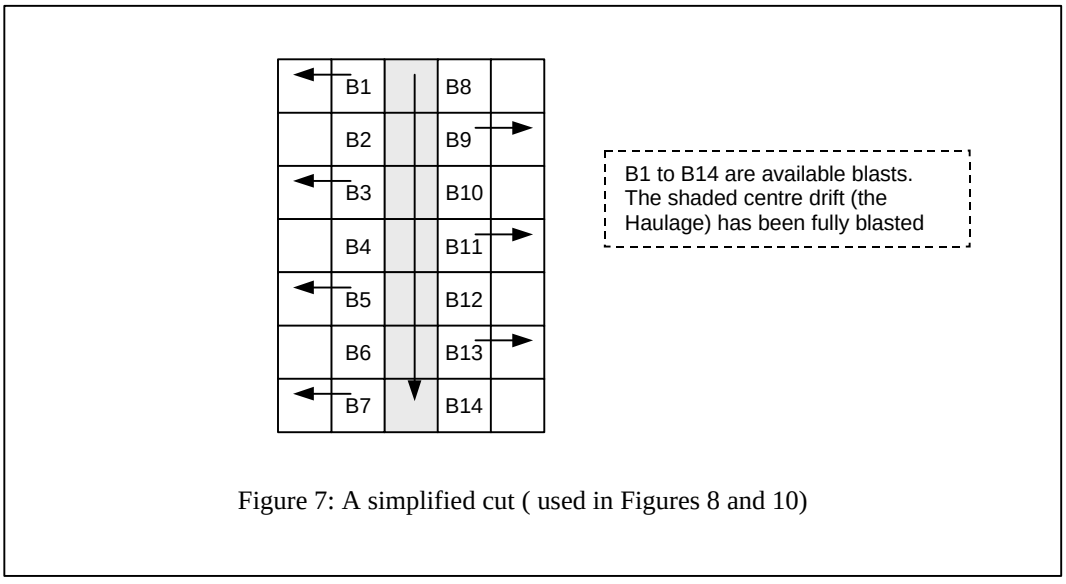

Figure 7: A simplified cut (used in Figures 8 and 10)

MUTANT CHROMOSOMES

The following algorithm is used in generating a mutant chromosome from a Parent:

1. Use an exponential distribution to choose a gene to mutate (this ensures that genes closer to the beginning of the chromosome get mutated more often)
2. Randomly choose a blast in that gene. Assume it is the i'th blast in the chromosome.
3. All blasts in the mutant with a chromosome position of x < i are clones of the respective blasts in the parent.
4. Replace the i'th blast of the parent with some other blast from the current domain of available blasts (the mutation)
5. Attempt to make all blasts in the mutant with a chromosome position of x > i clones of the corresponding blasts in the parent. Where this is not possible (due to the x'th blast in the parent not being part of the mutant's available domain), randomly choose a blast from the current domain of available blasts of the mutant.

Figure 8 shows an example of a gene being mutated, and the consequence of this on a subsequent gene.

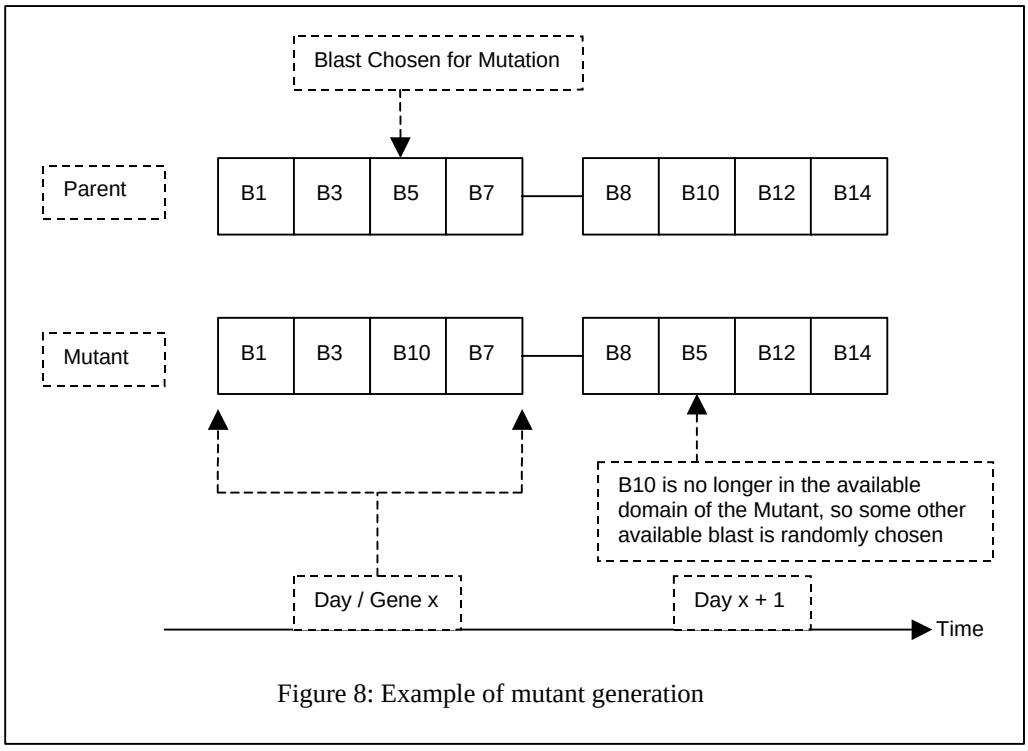

Figure 8: Example of mutant generation

THE CROSSOVER TECHNIQUE

A single crossover generates a child chromosome by "mating" two parent chromosomes. The idea is that the child chromosome receives a mixture of genes from each parent. In this application, "pure" crossover is not possible as mixing genes from two parents results in a different available blast domain for the child compared to that in either of the two parents. This is illustrated in the following crossover algorithm:

Take two chromosome parents, Parent A and Parent B. The i'th blast in the child chromosome is determined as shown in Figure 9:

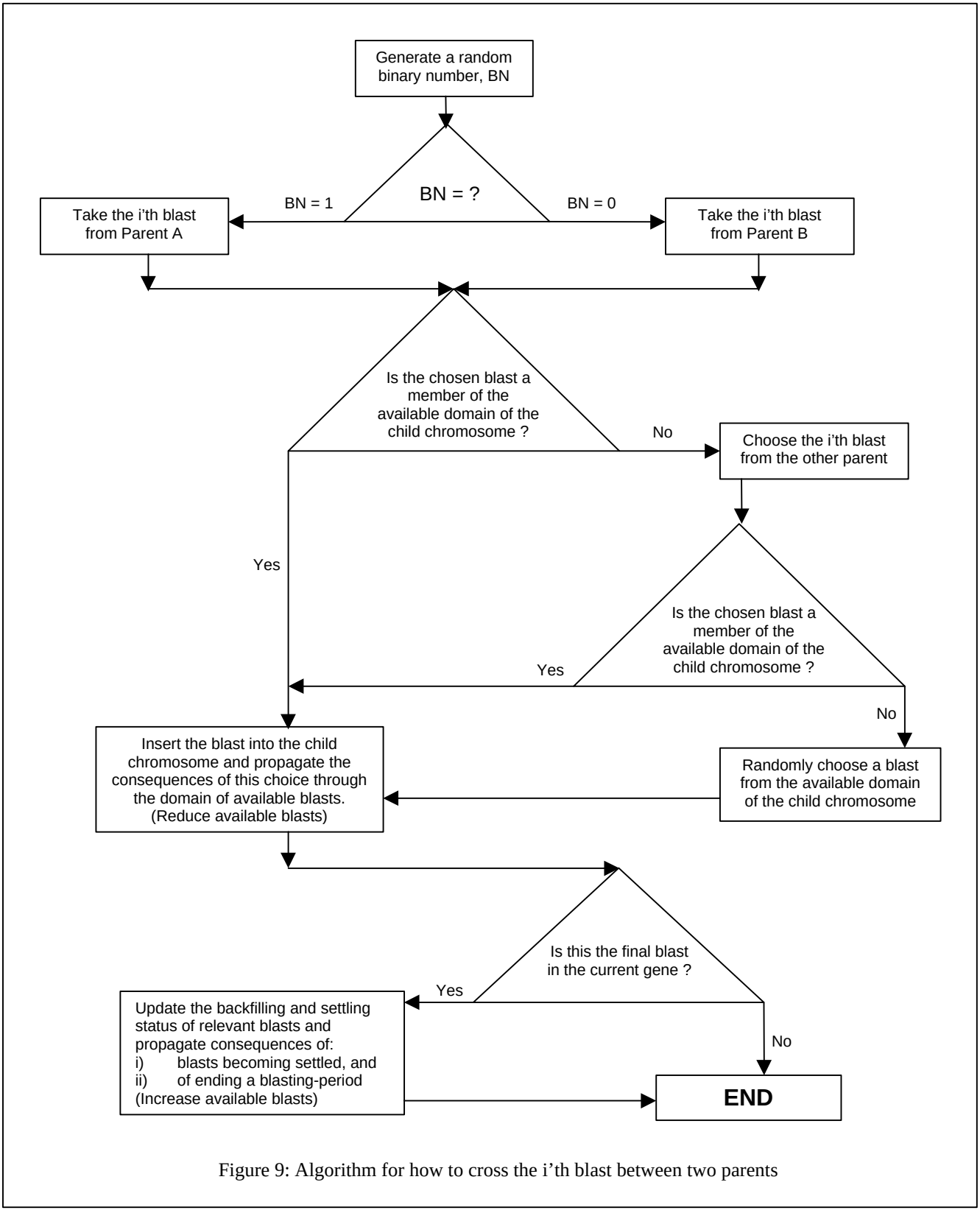

Figure 9: Algorithm for how to cross the i'th blast between two parents

Figure 10 below illustrates an example of crossing two genes.

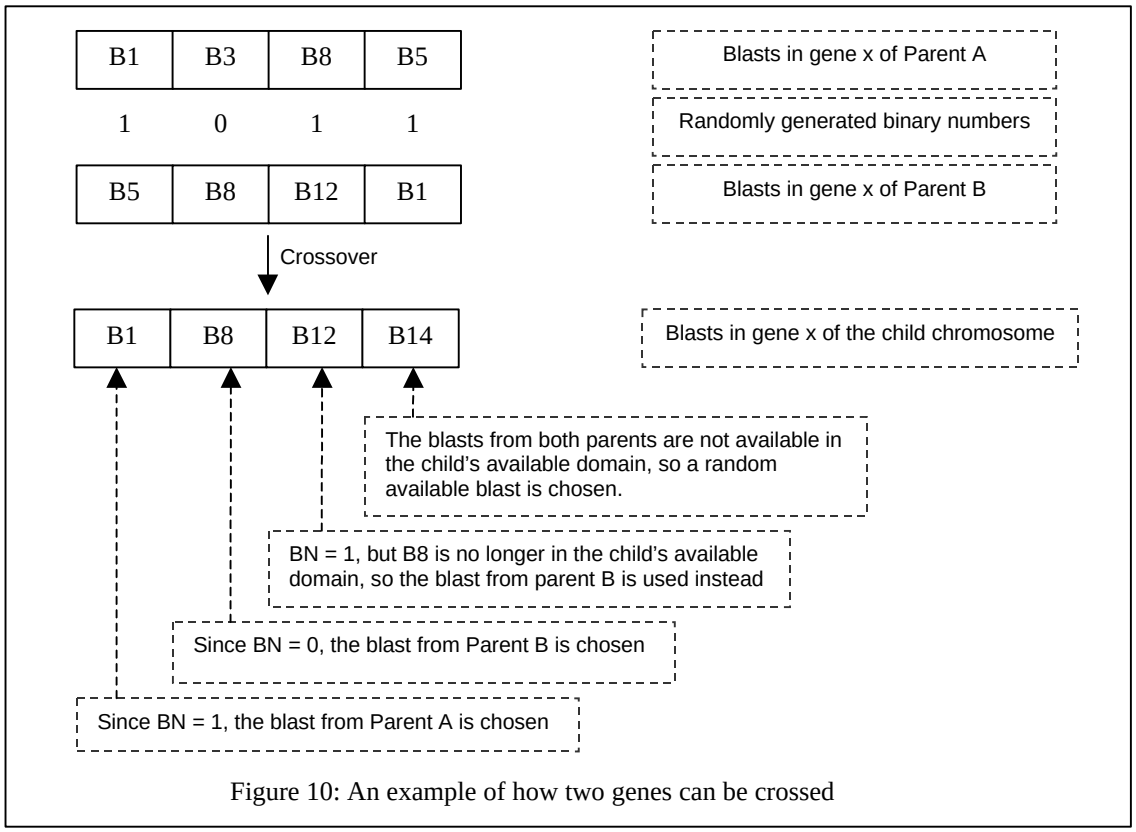

Figure 10: An example of how two genes can be crossed

GENERATING POPULATIONS OF CHROMOSOMES

All populations (or generations) of chromosomes generated subsequent to the first population have the following generalised creation methodology:

- A few chromosomes are cloned from the best chromosomes of the previous generation.
- Many chromosomes from the previous population are mated (crossed) to produce new (child) chromosomes in the current population. Each chromosome in the old population is given a mating factor (an integer number). The mating factor is roughly proportional to the value of the cost function for that chromosome. So, "good" chromosomes mate many times, while "bad" ones may not mate at all. However, at least a few "bad" chromosomes are chosen for mating just in case they have a few good genes.
- Many chromosomes from the old population are mutated to create mutants in the new population. A mutating factor is given to each chromosome in the old population in the same way that mating factors are given. Therefore, "good" chromosomes are mutated more than bad ones. However, at least a few "bad" chromosomes are always mutated.
- If there is stagnation in improvement of chromosomes over several generations, the percentage of mutants is gradually increased until some favourable new gene is introduced.
- A population typically contains between 80 and 200 chromosomes.

RESULTS

Figure 11 shows scheduling results for an ore body cut with the following specifications:

Table 1: Specifications of data and pay-limit bands used in genetic algorithm study

Specification	**g/ton (grams gold per ton of ore)**
Total blasts in the cut	~ 900
Average grade over all blasts in the cut	6.5
Pay-limit per blasting-period, expressed as the average grade obtained during a blasting-period (a gene)	4.5
Target-limit	6.5
Upper target-limit	10.0

The purpose of the optimisation scheduler is therefore to:

1. Minimise the number of genes with an average grade < 4.5 g/ton (i.e in band 1)
2. Maximise the number of genes with 6.5 g/ton ≤ average gene grade < 10 g/ton (i.e in band 3)

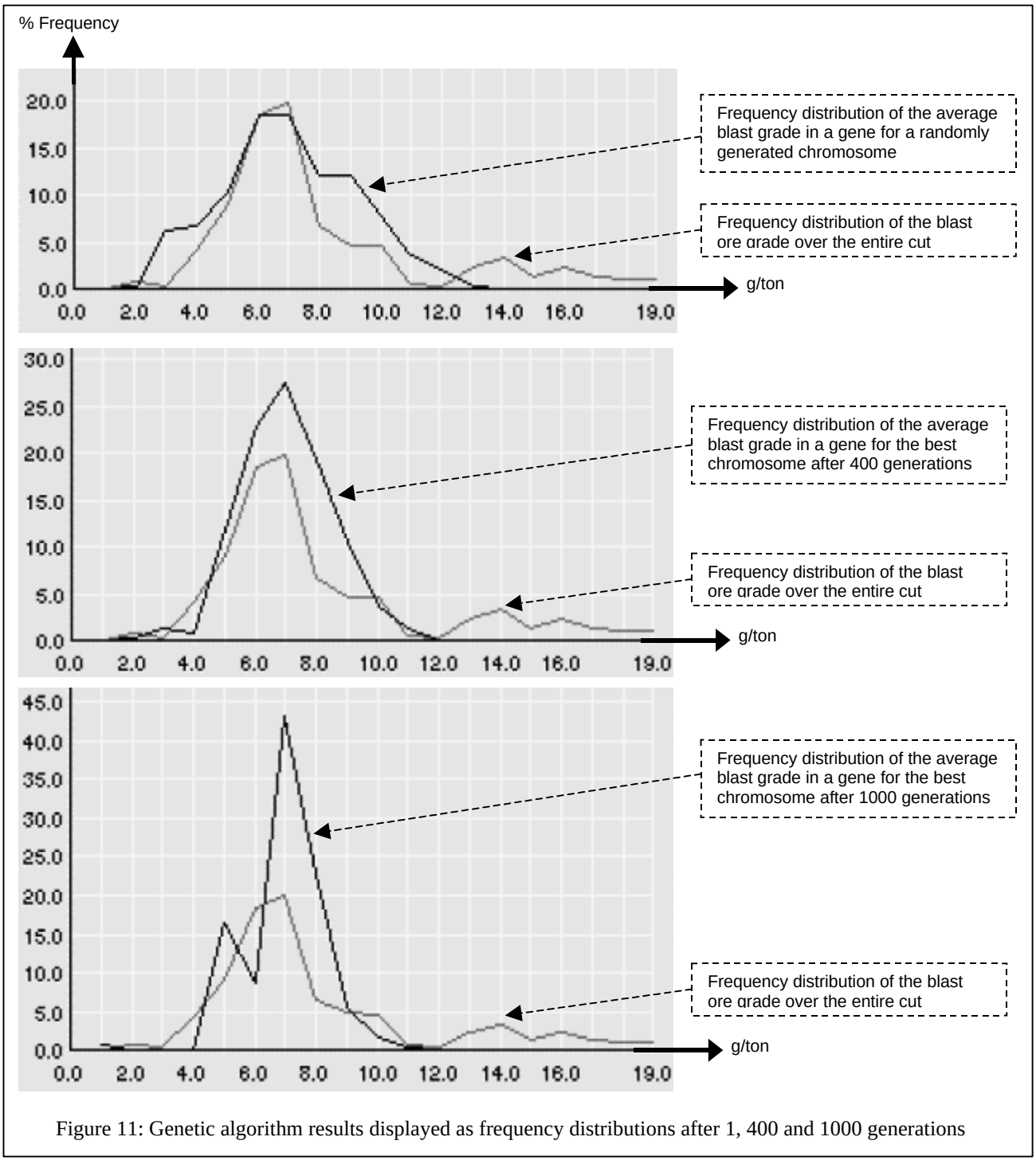

Figure 11: Genetic algorithm results displayed as frequency distributions after 1, 400 and 1000 generations

Table 2: Results of genetic algorithm optimisation for the cut specified in Table 1.

Band Number	**Generation 1**	**Generation 400**	**Generation 1000**
4 (Profitable, but undesirable)	13	3	1
3 (Sufficiently profitable)	92	124	156
2 (Not sufficiently profitable)	76	85	59
1 (Below Pay)	41	6	2

PERFORMANCE OF THE SCHEDULER

A “good” solution depends upon the average grade and the target-limit. As a rule of thumb, if the average grade over all the blasts is just sufficient to, on average, reach the target kgs gold/day, then a “good” solution is one where the number of blasting-periods in Band 1 is less than 10. Under conditions such as those specified in Table 1, a typical number of generations required to reach a “good” solution is of the order of 50 – 400.

For a cut of approximately 1000 blasts, with 150 chromosomes per generation, and running on a Pentium II 400 MHz machine, the average time for a generation was:

- 2 seconds, if pillars were not used (see Figures 1 and 3).
- 10 seconds if pillars are used

This allows a solution to be reached roughly within the half-hour time-frame originally stipulated.

THE EFFECT OF MUTANTS

It was noted that mutants tended to occasionally significantly improve the best chromosome in a new generation – as indicated in Figure 12 below, wheras crossovers tended to frequently slightly improve the best chromosome in new generations. Also, the effect of mutants was normally to reduce the number of genes in the worst band (Band 1 – which is rewarded heavily by the Cost Function) whereas crossovers tend to move genes from Bands 2 and 4 to the most desirable band (Band 3).

As is expected the Cost Function curve in Figure 12 below reaches a plateau fairly soon. (Note that an increasing Cost Function value implies a better solution). However, as soon as a plateau is reached (indicated by no increase in the Cost Function over several generations), the percentage of mutants generated in new generations is gradually increased. This frequently results in a step-up in the Cost Function in the next generation, which implies that a new effective gene is found through mutation. After the step-up, the percentage of crossovers in successive populations is increased to its normal value, and refining of the new improved chromosome occurs. This method of increasing the number of mutations when the Cost Function stagnates results in reducing the number of generations required for a “good” solution by up to an order of magnitude.

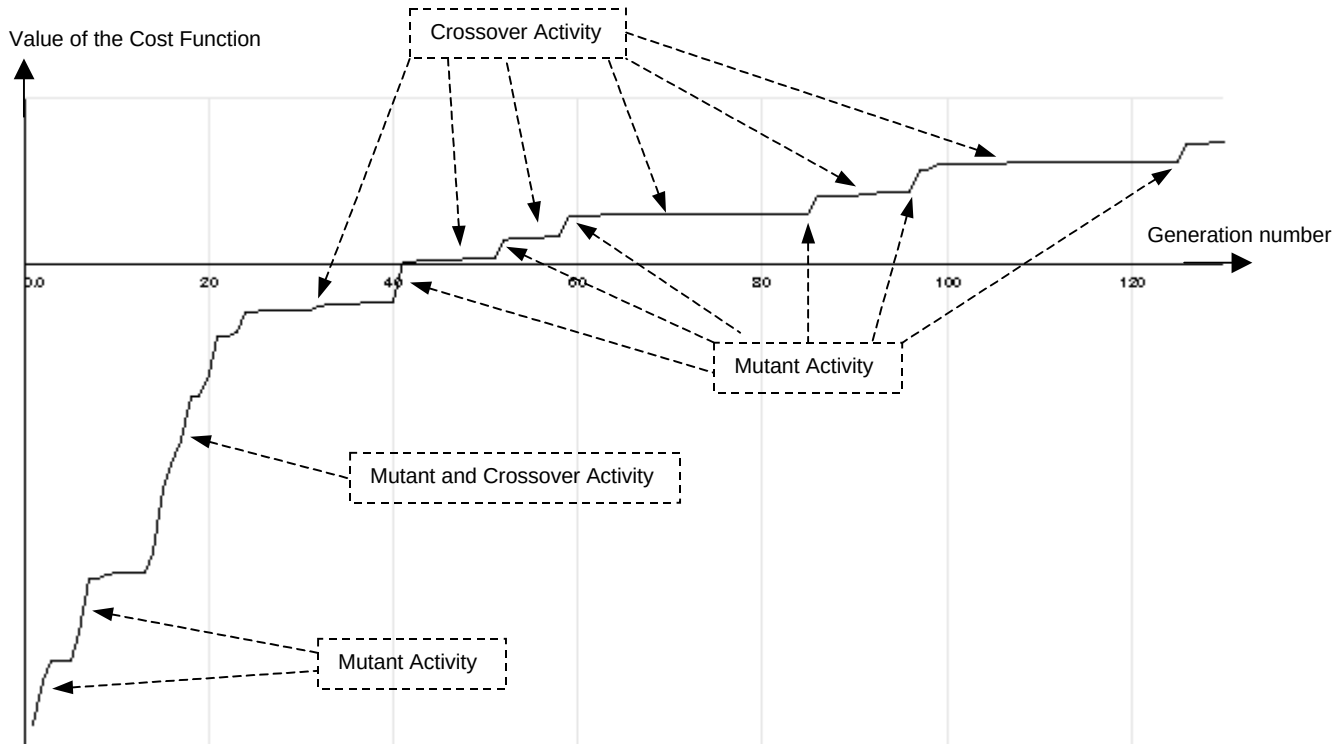

Figure 12: Plot of the effects of Crossover and Mutant activity on the improvement of the Cost Function in successive generations

The reason for the effectiveness of mutants is that an exponential distribution is used to select the gene to mutate. This means that genes near the beginning of the chromosome are chosen. In reality, this corresponds to trying out new permutations of choices of drifts where blasting starts – recall that drifts adjacent to a blasting drift become unavailable for well over a month. Choosing different starting drifts has the most profound possible effect on the resulting schedule. Performing a crossover, however, doesn't result in new starting drifts being chosen – it merely results in previously tried starting drifts being "swopped".

CONCLUSION

A genetic algorithm method was successfully used to generate schedules for mining cuts. Good solutions for the schedule were generated within a reasonable time frame. The following points should be noted:

- The use of G2 as a prototyping, proof-of-concept and user-interface development tool was a crucial factor in rapidly developing the application.
- Approximately 80% of the application development time was spent in analysing, designing & implementing the rules for propagating the domain of available blasts. Implementing the genetic algorithm itself was a relatively simple part of the application.
- Mutating genes at or near the beginning of chromosomes was an essential factor in discovering progressively better chromosomes. Generally speaking, mutants caused occasional large increases in the value of the cost function, while crossovers caused frequent small increases.
- By increasing the percentage of mutated genes during stagnation, the optimisation could frequently be given a "step-up" to a better solution, resulting in far quicker "good" solution arrival times.
- The effective implementation of genetic algorithms in this project was dependent upon a good understanding of the rules of the propagation of available blasts, and of the mining methodology.