

Mine Planning and Scheduling at RTZ Technical Services

Mike Pegman, Vine Solutions Ltd Tel 44 191 416 6389, mp@vinesolutions.co.uk
Nigel Forward, Brett King, Dave Teal, RTZ Technical Services Ltd

Abstract

This paper describes work undertaken by Vine Solutions Ltd with RTZ Technical Services in 1996. Two projects are described in outline and brief technical descriptions of the projects are given.

Both projects are in the application area of open-pit mine planning and scheduling. One deals with short term operations scheduling for a period of weeks, the other longer term planning. Common factors are the selection of material for mining from a range of alternatives and the subsequent sequencing of the chosen material. In addition, this sequencing must be converted into a viable operational schedule of timed activities. Various constraints are modelled, including physical limitations of mining, machinery and processing constraints, operating costs and related financial information is included to support the calculation of cash flows.

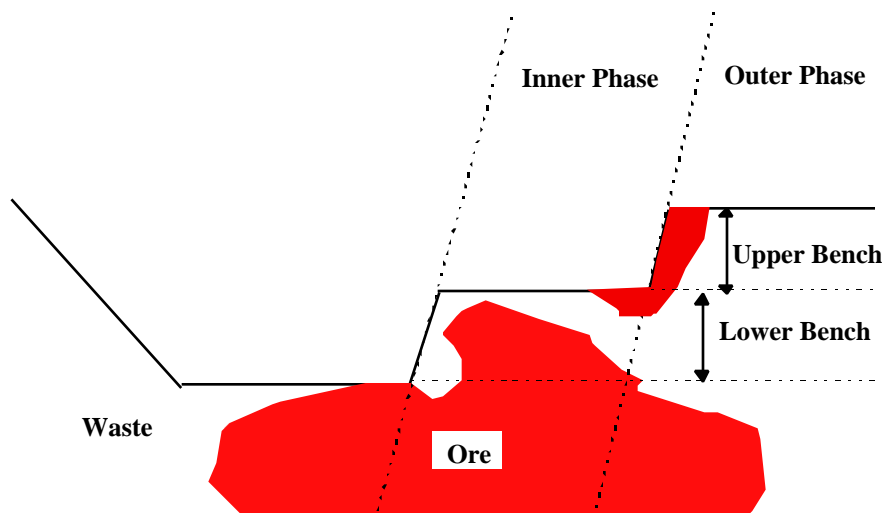
The paper concentrates on three areas: the material selection problem, discussing two implemented models; the waste removal problem, presenting non-monotonic planning search code; and the joint resource allocation and scheduling problem associated with operations mine scheduling.

ILOG Solver is used heavily in the block selection process, for the representation of mining constraints and for specialised planners. ILOG Schedule is used for the resource allocation and scheduling sub-problems. Both projects are in prototype form and are undergoing trials on behalf of their respective user groups.

The paper presents sufficient detail to act as a study of model design and also of the search space engineering issues which arise in practical Solver / Schedule projects.

1. Description of the Application

The application area is that of mining for various minerals by means of removing material from the surface to create 'pits' in which mining machinery, such as large mechanical shovels, is located. These '**open pits**' may measure kilometres across. Mining is performed in order of '**phases**', starting with an inner phase. These phase boundaries delineate successively larger pits. It is not required that the whole of an inner phase be mined before proceeding to a later phase. Mining activity is arranged in horizontal levels, say 10 meters apart, called '**benches**'. A bench is divided into '**blocks**' which are the units of rock to be scheduled. Each block has characteristics such as the quantity of metal or metals contained within it.



The decision over which blocks of rock to mine is made first at the bench and then at the block level. Depending on the timescale of mine planning the size of the blocks varies dramatically.

Each rock contains a certain proportion of metal. Correspondingly it is considered to be '**waste**' and is dumped or put into stock piles. The proportion of mineral (i.e. the **grade**) at which rock is considered to be '**ore**' is known as the '**cut-off grade**'. The cut-off grade which is used by the mine is known for each time period of operation of the mine. Rock containing more than this proportion of metal is called '**ore**' and is '**milled**' or processed yielding product, that is mineral in some form. The ore is shown for diagrammatic purposes as a grey

grade of the deposit will vary within the deposit. Note that there may be ‘separate’ bodies which are mined together.

The rock, which consists of both waste and ore, is mined by blasting blocks and then shovelling the rock into trucks which take it to either a waste dump or to the mill. The ‘mill’ is an extraction plant which processes ore to yield a product. The product may be shipped to customers by train (in the case of iron ore) or may be processed further at the mine site.

The projects discussed address two types of activity. The first is the selection of which blocks are to be mined, and in which order they are mined, and over what time period. This is discussed in some detail in the first part of the paper; two aspects are discussed - the block selection problem and the waste removal problem. The second is the allocation of mining machinery to each block and the scheduling of the necessary mining activity. After the creation of a list of blocks to be mined, a resource allocation process occurs and then mining activities are placed in time on the chosen resources.

2. Material Selection

The selection of blocks can be considered to be driven by a *quality profile* which must be achieved after a *certain target mass of ore* comprised of a number of blocks are mined over a *specific time scale*. The *order in which blocks* may be mined is determined by a number of constraints, arising from both physical and resource limitations.

The **target mass** and **timescale** over which it is to be achieved may be set (in short term planning) by the arrival of a train which will transport the material to market. In longer term planning the timescale is determined by the operational mass targets for milled ore and consequent metal yield which are set by the mine management.

The target mass, that is the total mass of the selected blocks, must be within the specified bounds.

$$Mass_b$$

allselectedblocks_b

The **quality profile** is set by the ‘customer’ of the mine. It is specified in terms of the proportions by weight of metal (e.g. iron) and associated material types (e.g. aluminium oxide) in the product. The amounts of each of the quality parameters is calculated by using a weighted mean of those constituents in the blocks. For each of the required quality parameters the parameter must lie within a tolerance specified by the customer. The weighted mean is defined by expressions of the form:

$$Mass_b \times QualityParam_b \Bigg/ \frac{Mass_b}{allselectedblocks_b}$$

As can be seen, the bounds of these expressions are not strongly effected by changing the bounds of $Mass_b$, and as a result the quality specification ranges on this weighted mean propagate poorly. The constraints are rather weak, and the search space is not substantially pruned until fairly late in the search tree.

2.1 Meta Constraint Model

This section presents the first of two implemented models of the block selection problem. The first is based on meta-constraints and array sums.

2.1.1 Blocks

Each block of rock is represented by an object in the model which maintains a collection of member constrained integer variables. Each variable is initialised with a two element domain, passed via an array, holding the values 0 (mass if the block is not selected) or M (mass of that parameter if the block is selected). This model reflects the fact that if a block is selected it contributes to the quality profile, if it is not selected it contributes M to the relevant quality parameter. The block’s selection status is represented by a boolean variable ‘selected’.

```
BlockCI::BlockCI(
    BlockData srcBlock,
    ...,
    IlcIntArray mass_array,
    IlcIntArray fe_array,
    IlcIntArray quality_param_array,
    ...
)
: - - - - -
```

```

    _mass(mass_array),
    _fe(fe_array),
    _qualityParam1(quality_param_array),
    _srcBlock(srcBlock)
    {
        if (srcBlock.isAlreadySelected())
            selected.setValue(1);

        IlcTell((_mass > 0) == (selected == 1));
        IlcTell((selected==1) == (_fe > 0));
        IlcTell((selected==1) == (_qualityParam1 > 0));
        ...
    }

```

2.1.2 Quality Parameters and Targets

In this model the target mass is an array sum over the BlockCI::_mass variables and the weighted mean quality parameters are array sums of each divided by the mass sum.

```

IlcIntVarArray _targetMass(NUMBER_OF_BLOCKS);
IlcIntVarArray _targetFe(NUMBER_OF_BLOCKS);
for (i=0; i< NUMBER_OF_BLOCKS; i++) {
    _targetMass(i) = blocks[i]->getMass();
    _targetFe(i)   = blocks[i]->getFe();
    ...
}
IlcIntVar _totalMass = IlcSum(_targetMass);
_totalMass.setRange(SPEC_MASS_MIN, SPEC_MASS_MAX);
...

```

2.1.3 Physical Mining Constraint

In addition to the above constraints, blocks are selected in a way which corresponds to physical access constraints between blocks. For instance it is not possible to mine a block unless the blocks in front of it and to the left and right are also mined. These constraints are implemented by the following meta-constraint, posted after the construction of the blocks.

```

if (IsBlockedBy(block, blockedBy){
    IlcIfThen( (block->getSelected() == 1),
              (blockedBy->getSelected() == 1));
}

```

This accessibility constraint has the desirable property that the selection by the search code of a block behind a currently exposed face of the mine automatically selects the blocks nearer that face. Equally the decision *not* to select a block at the face forces the consequent that the blocks behind it are also not selected.

2.1.4 Search Code and Propagation

Each variable in each block has only two possible values. This is both a benefit and a weakness. The benefit is that the above constraints are able to propagate to instantiation once any one domain is known. The weakness is that the search space is represented by an array of ‘boolean’ *_selected* variables which comprise the primary decision variables of the problem. (A naïve generate function would simply call instantiate on each of these variables down the whole array.) Because few constraints operate between these boolean variables, the search tree is large and little pruning occurs.

These issues have led to a strategy for the search code which relies on the rapid identification of a candidate solution and then its incremental improvement. The search code essentially makes the ranges of the quality parameters variable (rather than constant as the above SPEC_MASS_MIN, SPEC_MASS_MAX). These variables are initially set to be relaxed (i.e. a wide range of the quality specification is acceptable). A ‘solution’ is quickly found and the search code proceeds to ‘optimise’ the quality ranges in turn. The currently found ranges are kept for each cycle and the ranges are reduced and imposed as new constraints on a new attempt to solve the problem. The search process is:

- 1) generate a solution (i.e. generate over selection variables)

```

    if no initial solution then FAIL

2) improve on target qualityParam[i]
   (decreasing N) until it is in range
   if target[0] not in range FAIL
   if cannot get target[i] (i>0) in range SUCCEED with target[i-1]

3) do (2) for each target, i, arranged in order of importance to user
   (i.e. all targets now in range)

```

Various variable ordering strategies have been applied within the solution generator. Even so, the search space remains large and run times can be considerable. Accordingly a mechanism was introduced to fail the search to improve the target ranges after a certain time period. The search for an incremental improvement in a quality parameter at step 2 above is considered to have failed if the search takes longer than a certain time period. Thus the code facilitates the compromise between solution quality and solution time.

2.2 *Element Constraint Model*

An alternative model of the block selection problem was formulated in a later project. This model is based on element constraints over incremental sums.

In this version of the problem, which was at a lower level of detail and thus had much larger blocks, the blocks were arranged in independent lists in order of allowable mining access. There are as many lists as there are different access routes to the blocks.

The selection of blocks is represented by an index into a list of blocks in order of mining access. All the blocks at lower index values are selected and thus contribute to the quality parameters, those at higher index values are not selected and hence do not contribute.

Each property of a block can be seen as contributing to a separate cumulative sum. A list of the cumulative sum of the blocks' parameters up to the index is created, for all possible indices. These lists can be represented as integer arrays of cumulative mass, cumulative metal, or other cumulative quality parameter values. There is one cumulative sum for each quality parameter.

```

_cumMass1 = IlcIntArray(_nBlocks1);
for (bi=0; bi<_nBlocks; bi++) {
    MASS_UNIT blkMass = _blockList1[bi].mass();
    if (bi==0)
        _cumMass1[bi] = blkMass;
    else
        _cumMass1[bi] = _cumMass1[bi-1] + blkMass;
}

```

The block selection model is formulated in terms of element constraints, one for each quality parameter for each access route.

```

for (all access routes, i=1,2,...){
    IlcIntVar selectedBlocki(0, NUMBER_BLOCKS_IN_ACCESS_ROUTE-1);
    IlcIntVar selectedMassi = _cumMassi[selectedBlocki];
    ...
}

```

The totals across all access routes are again array sums of the relevant parameters e.g. selectedMass1 + selectedMass2 +

This model has fewer constraints and fewer decision variables and hence offers higher time performance. However in the form above the representation of access constraints is less flexible. Further access constraints were represented by expressing constraints between the index variables used above. This is discussed later.

A naïve search process was implemented for this representation. The simple approach, vis. the instantiation of the index variables for all access routes for all time periods, led to very protracted search times. This simple approach was replaced by the much more involved incremental planner search code discussed later.

3. Problem Constraints

This section describes some mining constraints drawn from the applications, then gives an overview of their

3.1 Phase / Block ordering & access constraints

This section presents the relationships between blocks in terms of physical access by mining equipment. Consider again the diagram of the open-pit mine presented earlier. Each phase represents a set of blocks. We can consider that the blocks can be mined within a phase only in order of benches. That is we must mine in order of depth, taking blocks from a whole bench before proceeding to a lower bench.

There is an array of data for each phase. Each array represents an ordered list of blocks in the order they can be mined. It is considered that block 0, i.e. the innermost block of a bench, represents the entire face of the phase; its shape is an annulus.

Thus for each phase has an array of blocks in the order bench 0 - block 0, block 1, block 2, ..bench 1 - block 0, block 1, A block cannot be mined unless all the blocks at earlier indices in the array have also been mined. The index of the last mined block in a phase is represented as a constrained variable. So the access to blocks within a phase has been *structurally* constrained by the choice of the array and index as the model of the problem.. (This has been described above in the discussion of the element constraint as a representation of the block selection problem.)

However it is also a constraint that mining in an earlier phase must have progressed to a greater depth than that in later phases. This constraint represents the exposing of the face to be mined say in phase 1, bench 1, block 0, by the removal (at least) of *all* of the blocks of phase 0, bench 1. This can be represented as a constraint between the block indices of the two phases. Since the number of blocks in each bench may differ between phases another constraint represents the relationship between block index and bench number.

```
for (all phases, i=0,1,2,...){
    IlcIntVar selectedMassi = _cumMassi[selectedBlocki];
    IlcIntVar selectedBenchi = _benchi[selectedBlocki];
    ...
}

and then:
for (all phases, i=1,2,...){
    IlcIfThen( (selectedBench[i-1] != maxBenchInPhase),
              (selectedBench[i-1] > selectedBench[i]));
}
```

3.2 Mill and Mining Constraints

The total amount of rock selected is the sum over all the phases of the selectedMass_i variables described above. This is represented as a simple array sum.

Over a time period of operation there are capacity ranges which must be met for the output of the mine as a whole (mass of rock) and the amount of material which is milled (mass of ore). As has been discussed, these targets arise for time periods from operational considerations such as the requirement to fill a train or to meet an annual production target.

The constraint may be imposed in stages (as discussed in the section on search code above) or set using a simple modifier.

```
_totalMined.setRange(
    mine.mining(_period).capMineN, mine.mining(_period).capMineX);
_totalMilled.setRange(
    mine.mining(_period).capMillN, mine.mining(_period).capMillX);
```

3.3 Smoothing of production

It is important to keep a smooth flow of material both of waste and of ore through the mill. This objective reflects a desire to avoid both shut down costs and irregular use of high value capital assets. Such constraints are implemented as offset constraints between periods, similar to that shown in the next section.

3.4 Depth of Mining constraints

Mining activities in a phase in a period may need to be restricted to a certain number of benches (i.e. a depth of rock mined in a period). Alternatively the total mass of rock mined from a phase may need to be constrained (i.e. a mass of rock mined in a period).

These are expressed simply as an offset on the relevant mass or bench variables.

```
IlcPost(IlcLeOffset(prevPeriod->bench(),_bench, sinkingRateBench()))
```

3.5 *Origin by pit 'constraints'*

It is required that the selection of blocks achieves a balance between different sources of material. So that say roughly two blocks come from one source to every one from another source. This is really a 'soft constraint' and so was implemented as a search order preference. The selection variables are instantiated in groups drawn from blocks of the correct type. A goal was instantiated the 'boolean' selection variables in groups of the correct ratio. This does *not force* the solution to comply with the preference but represents the search space in terms of appropriate units which were likely to be instantiated together.

3.6 *Scope of operations*

The 'area' of the mine over which equipment is distributed needs to be controlled.

In the short term scheduling model this was represented by restricting the choice of shovel resource with which to mine a selected block to one of a predetermined list. Thus the resource allocation problem consists of the selection of one of a list of shovels which are stated to be potentially available to mine blocks in a given area of the mine.

In the coarser model the same notion is represented without identifying the resources individually. A constraint allows mining activities only within a certain number of phases within a certain period. This constraint is a limit on the count of those phases in which the selected block index has changed in the period (i.e. the selected block index is different from the previous period).

```

IlcConstraint cst;
if (previousPeriod)
    cst = (_selectedBlock[0] != previousPeriod->selectedBlock[0]);
else
    cst = (_selectedBlock[0] != 0L);
IlcIntVar minedPhase = cst;
for (ph=1; ph<mine.nPhase(); ph++) {
    if (previousPeriod)
        cst = ( _selectedBlock[ph] !=
                previousPeriod-> selectedBlock[ph]);
    else
        cst = (_selectedBlock[ph] != 0L);
    minedPhase = minedPhase + cst;
}
IlcPost( minedPhase<= MaxMinedPhasesInPeriod );

```

This fragment of code makes use of the ability to cast a meta-constraint to an integer and then constrain the sum of those integers.

4. Waste Removal Problem - Search Space Engineering

This section describes waste stripping (i.e. the removal of material less than cut off grade). Waste stripping is a process which must be delayed for financial reasons. However sufficient material must be removed in time to facilitate the production of ore. An objective of the mine plan is to achieve this balance.

As has been described in a previous section, many constraints apply to what mining operations may be undertaken. In particular the total amount of rock mined across all phases must be within mining capacity bounds. The amount of ore mined must be within the limits of the capacity of the mill.

4.1 *Mutually Recursive Planners*

This section gives an overview of the search mechanism implemented for the project. The search process is implemented as two mutually recursive planners. One planner operates forwards in time, seeking to maximise mill production of ore. The objective is to mine as much ore as the constraints allow to achieve the maximum mill production. Whenever this planner finds that it cannot meet this objective, it establishes an objective for another sub-planner. The sub-planner works backwards in time trying to meet the new mining objective. Its objective is to remove waste so that the ore body is exposed. Of course all the mining constraints must be obeyed here as well.

4.2 *Non-monotonic Planning and Plan Patching*

Note that the backward sub-planner is constrained by the same mining capacity and other constraints as the forward planner. Meeting its objective is likely to require some of the capacity which could have been allocated to the forward planner. In the normal course of a constraint program this would lead to failure and a backtrack

process until the forward planner was able to reduce the utilised capacity earlier in the search tree. Such a long backtrack as a routine part of the search process is impractical on grounds that it takes too much time. Labelled OR choice points were considered but it was felt that the search space for that search model was also too large.

To address this issue the search was designed as a series of sub-problems each in a separate search space. This is implemented as separate calls to `IlcSolve` with the state being restored across each goal solve. The forward planner is called in this way, planning for maximum mill utilisation until it cannot meet this objective. The waste removing objective is established; the solution so far is cached in non-backtracking memory and the state restored. Then the sub-planner is called, which may include recursive calls to the forward planner.

If the sub-planner finds that its solution is incompatible with that of the cached results of the previous plan, it is necessary to patch the plan. This plan patching is necessary because the separate planners are now proceeding towards a solution in a non-monotonic fashion. That is, a decision of a previous planner may have to be *undone* and the consequent corrections propagated. This plan patching is achieved by calling the forward planner from the point of the plan failure forwards. Thus the overall planning process takes the form:

1. Forward Plan, keep plan state in cache, establish waste removal objective.
2. Backward sub-plan, comparing this plan with the previous version.
3. If, during the sub-planning, a difference is identified, fix the plan by calling the forward planner recursively from the plan failure.

This algorithm is producing acceptable run times (around two minutes for problems of a size which were not terminating using naïve search).

5. Operational Mine Scheduling

This section presents the application of ILOG Schedule to the operational mine scheduling sub-problem. Recall that after the creation of a list of blocks to be mined, a resource allocation process is undertaken and then the mining activities are placed in time on the chosen resources.

The application allocates mining machinery (a shovel) to each block and then schedules the necessary mining activity using that resource. All *ore* mining activities require a shovel *and* the mill. Mining may only proceed when the mill has sufficient capacity. Since waste is not milled, mining *waste* blocks requires only shovels. Consequently it is possible to schedule waste block mining whilst the mill is not available.

An additional complication is the fact that ore mining activities may be suspended by scheduled mill down-time. Once the mill restarts, the ore mining activity is restarted. During the down-time the same shovels may be employed in mining waste.

5.1 Schedule Model

This section presents the basic model of the application in terms of Schedule objects.

A mill is represented as an `IlcDiscreteResource`. The capacity of the mill is represented as a time table which has a maximum determined by the amount of ore it can handle. The time table is used to represent periods of down-time corresponding to maintenance or break down time (using `setCapacityMax` member function).

Shovels are unique and have a rate at which they can dig. They are implemented as a sub-class of `IlcUnaryResource`. Shovels may have planned maintenance periods - these are represented by dummy activities which require the resource.

A block selected earlier in the planner must be dug by an allocated shovel and is represented by a 'Dig' object which contains relevant `IlcIntervalActivities`, as discussed later. Blocks of rock have an 'ease of digging' parameter. Using shovel rate and ease of digging the amount of rock per unit time is calculated. This determines the length of time required to dig the block; i.e. is the duration of the digging activity (see `digTime` in the code below).

Ore mining activities require the mill. The capacity which is required is a property of the block of rock and of the shovel being used to perform the digging. Notice that the throughput is represented as an element constraint 'driven by' the choice of resource.

```
IlcIntArray shovelThroughPut(NSHOVELS);
for (all shovels i)
    shovelThroughPut[i]=shoveli.getDiggingRate()*block.get_digability();
```

```
... ..
```

```

    IlcPost(digTime== block.getMass() / throughput);

    IlcPost(dig.requires(mill, throughput));

```

5.2 Shovel Moves

Moving a shovel around the mine is a potentially costly affair. It involves lost production time of major capital asset.

The time of movements depends on factors such as the type of the shovel, vertical movement across bench levels or movement to a different pit. The model of this is a straight forward use of the transition time features of Schedule. A ShovelResourceI is a class derived from the library's UnaryResourceI.

```

    IlcInt ShovelResourceI::getTransitionTime(
    const IlcResourceConstraintI* r1, const IlcResourceConstraintI* r2)
    {
    Block *b1 = ((Dig *)r1->getActivity()->getObject()->get_srcBlock());
    Block *b2 = ((Dig *)r2->getActivity()->getObject()->get_srcBlock());
    if (b1->pit()!=b2->pit())
        return (_isMainShovel ? 12 : 2);
    if (b1->bench()!=b2->bench())
        return (_isMainShovel ? 6 : 1);
    return 0;
    }

```

5.3 Mining Activity

The representation of a mining activity is a class called *Dig*. This class encapsulates the block data (which block is being mined), the list of resources required (mill and shovel choice) and the representation of the mining activity. The mining activity may be suspended up to twice to cope with mill shut-down time. Since Schedule does not allow its activities to be interrupted, the mining activity is represented as three objects of type IlcIntervalActivity.

```

    Dig::Dig(      Block src, IlcDiscreteResource mill,
                  IlcInt nShovels, ShovelResource **shovels)
    :
    srcBlock(src),                // this digs the srcBlock
    shovelChoice(0,nShovels-1),   // resource allocation decision var
    shovelRequired(nShovels,0,1)
    {
    dig[0] = IlcIntervalActivity(_getSchedule());
    dig[1] = IlcIntervalActivity(_getSchedule());
    dig[2] = IlcIntervalActivity(_getSchedule());

    dig[0].setObject(this);
    ...

    IlcIntArray durationArray(3,
        dig[0].getDurationVariable(),
        dig[1].getDurationVariable(),
        dig[2].getDurationVariable());

    digTime = IlcSum(durationArray);
    dig[2]->setEndMax(trainTime) ;

    // component digs must follow in order
    IlcPost(dig[1]->startsAfterEnd(dig[0]));
    IlcPost(dig[2]->startsAfterEnd(dig[1]));

    IlcPost(IlcSum(shovelRequired)==1); // only one shovel required

    for (all digs 0,1,2 and all shovels i)
        IlcPost(dig[0].requires(shovels[i]), shovelRequired[i]);

    if (srcBlock.isOre()) {
        IlcPost(dig[0].requires(mill, throughput));
    }

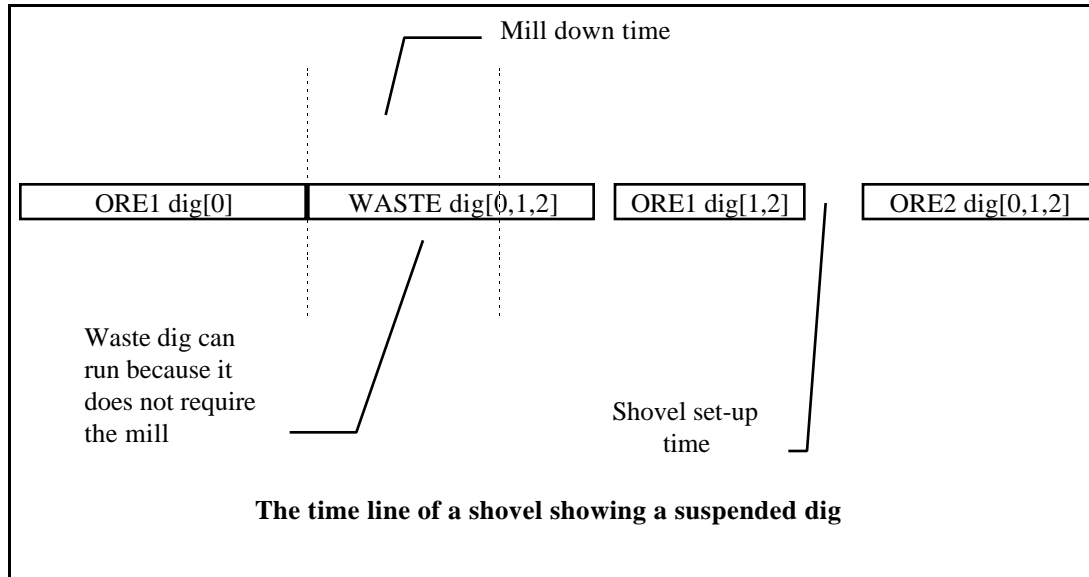
```

```

    }
    ...
    }

```

The above code presents a skeleton of the object which is at the core of the resource allocation and scheduling problem.



The search code first creates a trial resource allocation, assigning shovel to dig. In the above code fragment this primarily involves the instantiation of the shovelChoice decision variable. Then the search schedules the digs using the heaviest loaded shovel first. This choice of resource is made because it is expected that the mill will ultimately limit the possible start times of the dig activities and the heaviest loaded shovel has the least slack in the scheduling of its digs.

For a chosen shovel resource, the possible first digs are searched (using resource constraint member rankFirst, rankNotFirst) to determine a dig ordering on that shovel. As each activity is ranked the distribution of the dig’s duration is allocated to the constituent interval activities. If the current interval activity cannot execute for the whole digTime then the mill must be constraining it. So the code searches for another possible dig to start immediately, which must be a waste dig, and thus utilise the mill down time. After the mill down time and the waste dig, the code ensures that the other activities of the suspended dig are scheduled.

6. Results & Conclusion

The above two models of the block selection problem adequately represent the requirement of the applications. However both suffer from less than ideal propagation characteristics and lead to the need to search over the block selection control variables. Each selection incrementally adds to the sums of the quality parameters and thus a solution is incrementally generated. In the first model, the sum constraints do not propagate to the selection variables of blocks (thus removing blocks which are unacceptable) until the search is quite deep in the search tree. The search tree is large, and although accessibility constraints reduce the viable combinations to some degree, run times can be long (the tens of minutes on Pentium PC hardware for the meta-constraint model and the search code described first).

This leads to several additional ideas to change this state of affairs.

- Exploration of the granularity of the block selection decision variables, trying to reduce combinations by searching over higher level abstractions of the mine, rather than over blocks themselves. In addition various redundant constraints are being explored.
- Stochastic seeding of the search space; the selection of an initial number of blocks can be done randomly.
- It is possible to envisage a custom ‘weighted mean’ constraint which uses information about the available blocks to reduce the ranges of the sums. Essentially this would involve information about the quality parameters of a block taken *as a whole* in addition to reasoning about the values within the domains (c.f. a constraint considering each parameter separately and reasoning only about the bounds).
- Sub-problems could be defined corresponding to imposing the quality requirements over smaller mass targets; in other words the ‘requirement’ that the quality parameters must be met over parts of the problem, and not just ‘by the end’.

Perhaps of most interest is the likelihood that the project can apply ILOG Planner. The block selection problem in general can be formulated as a set of linear inequalities and 0-1 variables. Using Planner these can be incorporated into the rest of the model and the associated mine scheduler implemented, as now in ILOG Solver and Schedule. At the date of writing this work has not been undertaken, but we expect to experiment with the application of ILOG Planner beta test version to this sub-problem. It is hoped that this will greatly improve the performance of the block selection process. It should also enable a more complete optimisation of the quality parameters leading to reduced deviation from specification.

The mine scheduler is expected to be further developed and incorporated into an existing application which is in operational service in iron ore mines. Schedules are expected to be produced automatically which are more efficient than those currently produced using a manual editing process.

Work on the longer term mine planner is continuing. Trials of the system reported in this paper have been positively evaluated and further funding for on-going work is available. RTZ Technical Services and Vine Solutions will continue the joint development to reduce the total project time. It is expected that the system will replace an existing system (coded in Fortran) and will offer higher functionality and greater maintainability and extensibility.

This paper has presented several aspects of the development of the models and the search code of ILOG Solver and Schedule applications. The design and implementation of search code which meets the users' expectations of run-time is perhaps the most significant challenge (as it has been in the author's experience of other Solver projects). As reported there are several ways of enhancing performance. The trend towards 'global constraints' and their specialised algorithms is interesting and welcome. Although the detail of the application is clearly specific to mine planning, it is hoped that the issues raised will be of practical use as a study of model development.