ROYAL HOLLOWAY COLLEGE

Author.................  Degree...............  Year...........

I undertake not to publish either the whole or any part of the
above-mentioned thesis, and not to quote or make a copy of the whole
or any substantial part of it without the consent of the author; and
I further undertake to allow no one else to use this thesis while it
is issued to me.

Date       Signature       Name in block       Address

Undertaking to be signed by reader
of thesis.
REPAIR SCHEDULING FOR GLASS CONTAINER FURNACES

USING

BRANCH AND BOUND TECHNIQUES

by

S. P. M. MOORE
CHAPTER I
SUMMARY
The problem; brief descriptions of the method and the computational procedure.

CHAPTER II
GENERAL THEORY

CHAPTER III
SPECIFIC THEORY
The model; a basic algorithm; explanation of the steps involved; handworked example.

CHAPTER IV
The PROGRAM

CHAPTER V
HOW TO USE THE PROGRAM

CHAPTER VI
RESULTS
Two small illustrative examples; results for a 'real' problem; time analysis.

CHAPTER VII
CONCLUSION
Advantages and disadvantages; possible extensions to the program.

(A flow chart of the final program accompanies this report and a listing of the program is in the folder)
CHAPTER 1

The Summary

The Problem

Given a set of glass furnaces, producing 3 colours of glass and grouped together in factories, to find a method of calculating a stable pattern of timing of normal rebuilds over a seven year period.

Objective

To minimize the total cost of production and repairs

Constraints

(1) No two furnaces in the same factory can be rebuilt in the same quarter.

(2) No two amber or green furnaces to be rebuilt in the same quarter; no more than two white furnaces to be rebuilt in the same quarter.

(3) Given sales demands for each colour glass in each quarter. The sum of the production over 3 consecutive quarters must be greater than or equal to the sum of the demand over these 3 consecutive quarters.

Initially an analytic approach of monitoring the variables that effect cost and production was tried. It became apparent that some essential data on furnace behaviour was either not in existence or at a very experimental stage - in particular data on furnace life expectancy and ageing effects on costs and production. In its place the following method evolved. I would be given a number of optional schedules for each furnace along with data on production and costs. Out of these one has
to choose ONE schedule for each furnace in such a way that we have a minimum cost combination that satisfies all the constraints.

We now have a different problem to solve.

The simplified problem

Given a set of glass furnaces, producing 3 colours of glass and grouped together in factories, and given one set of possible repair schedules for EACH furnace. To find a method of choosing one repair schedule for each furnace to give a stable pattern of timing of normal rebuilds over a seven year period.

Objective and Constraints are the same as before.
A brief description of the method used - Branch and Bound

We are given one set of repair schedules for each furnace, with relevant details on costs, production figures and sales demands. We have to choose one schedule for each furnace while minimizing costs and satisfying the constraints. We cannot evaluate each combination and test for infeasibility. Basically we are dealing with a very large set of solutions which we want to examine more closely and from which we want to pick the best. What we do, is to divide this large set into two smaller sets and then to divide these sets again until a solution is isolated. This is the branching part of the method.

For each of our subsets of the original set we can calculate a lower bound on the costs of solutions in this set. Having already found a solution we can compare its cost with the lower bound. If this lower bound is greater than the cost of the solution this set can be rejected. In this way sets of solutions can be thrown away without having to be examined individually. This is the bounding part of the method.
Summary of Computational Procedure

(1) STAGE 1: To find an initial solution.

As has been mentioned before, as soon as we have a solution - not necessarily the best - we can start rejecting sets of solutions and so speed up the search for an optimum. Stage 1 finds an initial solution as fast as it can while at the same time trying to find one with a small cost.

(2) STAGE 2

Having found an initial solution in stage 1, we now either reject or investigate further all the sets of solutions.

Results

In the chapter on results a few small examples are given as illustration and a final 'real life' run is made. As far as the results allow a time analysis is made.

Conclusion

The model although, only a rough picture of reality, allows the incorporation of many refinements. These are discussed in the final chapter, along with comments on advantages, disadvantages and time.
CHAPTER II

General Problem

Given a system comprised of n-stages, we want to choose a line of action (or take a decision), from a set of options, for each stage: in such a way that we minimize the total 'cost' of the system, while satisfying any constraints that are imposed. Let a solution be any set of n decisions (one for each stage) which satisfies the constraints. A cost is attached to the making of a decision (or committing an option), so every solution has a cost. We seek the minimum cost solution.

Example: n-stages, m options for each stage.

Stage  Decision Options for each stage
1      (D_1,1; D_1,2; D_1,3; ... ; D_1,m)
2      (D_2,1; D_2,2; D_2,3; ... ; D_2,m)
3      (D_3,1; D_3,2; D_3,3; ... ; D_3,m)
4      (D_n,1; D_n,2; D_n,3; ... ; D_n,m)

where $D_{i,j}$ represents the decision that commits the $j$th option (from those available for the $i$th stage) to stage $i$.

A solution in a set of decisions

$$(D_{1,j_1}; D_{2,j_2}; D_{3,j_3}; ... ; D_{n,j_n})$$

that satisfies all the constraints.

Let the cost of $D_{ij}$ be $C(D_{ij})$. Hence the cost of
the above solution is
\[ \sum_{i=1}^{n} C(D_i, j_i) \]
assuming additive 'costing'.

**General Theory** (As applied to this type of problem).

As has already been mentioned, in any sizeable problem the number of solutions is too large to allow for individual evaluation. We use the following method to divide the set of all solutions into more manageable subsets.

**BRANCHING:**

Let node 1 represent the set of all solutions. Consider the decision \( D_{i,j} \). Let node 2 represent the set of all those solutions WITHOUT \( D_{i,j} \) as the committed decision for stage \( i \); and let node 5 represent the set of all those solutions WITH \( D_{i,j} \) as the committed decision for stage \( i \).

We now have two disjoint subsets of node 1, the union of which equals the set represented by node 1. This process is repeated to give smaller and smaller sets which can be represented in the following way.
where

Node 4 represents all those solutions with decision \( D_{i,j} \) taken in stage \( i \); and decision \( D_{i_1,j_1} \) \textbf{NOT} taken in stage \( i \).

Node 9 represents all those solutions with decision \( D_{i,j} \) taken in stage \( i \); \( D_{i_1,j_1} \) \textbf{NOT} taken in stage \( i \); \( D_{i_2,j_2} \) taken in stage \( i_2 \); \( D_{i_3,j_3} \) taken in stage \( i_3 \).

Node 6 represents all those solutions with decision \( D_{i,j} \) taken in stage \( i \); \( D_{i_1,j_1} \) \textbf{NOT} taken in stage \( i \); \( D_{i_2,j_2} \) \textbf{NOT} taken in stage \( i_2 \).

Thus this method of splitting the sets can be represented as above by a tree. Note that the union of all the sets represented by the terminal nodes (2, 6, 8, 9, 5) always equals the original set (node 1).

**Bounding**

As we are minimizing costs, we calculate, for each set, a lower bound on the costs of the solutions in that set. When we have found a single solution we can compare the cost of this solution with the lower bounds of the terminal nodes. All those terminal nodes which have a lower bound greater than the costs of our present solution cannot contain a better solution, hence such sets are not considered further.
FINDING THE INITIAL SOLUTION

Before the bounding process can take place, we must have a solution. We can get this quickly by 'branching right', in other words you divide node 1, then subdivide the node resulting from committing a decision. If you continue in this way, i.e. continuing to commit decisions, you reach a node which in fact represents the set containing only one solution - a solution node.

FINDING THE BEST SOLUTION

Having found an initial solution, you search the terminal nodes - from right to left say - rejecting as many as possible by the bounding procedure, and branching on those not rejected so creating more terminal nodes. If at any stage a new solution node is found with a lower cost than the previous 'best' solution found so far, make this new solution the 'best' and continue. This process of rejecting, exploring and updating the 'best' solution continues until there are no more terminal nodes to be investigated, in other words at this stage all the terminal nodes are either solution nodes or rejected nodes.

INCORPORATION OF CONSTRAINTS

There are two basic classes of constraints, namely
(a) Those which directly limit the choice of combinations of options.
(b) Those which indirectly limit the choice of certain combinations.
To be more explicit, constraints of type (a) take the form of directly excluding some combinations of options from solutions i.e. $D_{1,3}$ and $D_{4,5}$ cannot appear simultaneously in the same solution. An obvious example of this, is that only one of $D_{1,1}$, $D_{1,2}$, ..., $D_{1,m}$ can appear in any one solution.

More general constraints such as, minimum production and maximum cost requirements, are grouped under type (b). For each node you calculate say, an upper bound on production and a lower bound on costs, and if the upper bound on production is LESS THAN the required limit, or if the lower bound on costs is GREATER THAN the maximum allowed, the node under consideration can be rejected.

**RULE FOR FORMULATING CONSTRAINTS OF TYPE (b)**

When incorporating any constraint of type (b), a general rule must be obeyed.

If node X FAILS a constraint, then any nodes that represent sets which are subsets of the set represented by X, must also FAIL the constraint, i.e. if we have

![Diagram](image)

If X fails then Y and Z and any other nodes derived from them must fail the constraint.

This is an obvious rule, but when the constraints are formulated in the model of the problem it is quite possible that one of them, while appearing to be sound, will violate this rule, producing disastrous results.
Conclusion

All the previous material is common to any branch and bound problem. But we still have 3 basic points that can only be decided for each individual problem:

(1) How to choose the decision which is used to divide the sets — branching decision.

(2) How to formulate the bounds for the nodes. (Resulting from constraints, type (b)).

(3) How to strike out options that are infeasible for choosing as a branching decision for each node. (Resulting from constraint, type (a)).

The speed of the process depends on two things:

(1) Choice of branching decision — the nearer you can come to the 'best' solution initially, the faster you can reject terminal nodes.

(2) The nearer the lower bound of a node, is to its greatest lower bound the better. Since this also makes for faster rejection of terminal nodes.
CHAPTER III

Restating the problem:

Given a set of glass furnaces, producing three colours of glass and grouped together in factories also one set of possible rebuild schedules for each furnace (together with production figures and costs), choose one repair schedule for each furnace to give a stable timing of normal repairs over the next seven years.

Objective: To minimize total costs of production plus repairs.

Constraints:

1. No two furnaces in the same factory can be rebuilt in the same quarter.

2. No two furnaces of amber or green can be rebuilt in the same quarter; no more than two white furnaces to be rebuilt in the same quarter.

3. Given sales demands for each colour glass for each quarter. The sum of the production over any three consecutive quarters must be greater than or equal to the sum of the demand of these three quarters.

Note: We are given an equal number of schedules per furnace, these are represented as in (4). Say we have 8 furnaces and
THE MODEL

First the furnaces are numbered 1, 2, 3, ....

1) Colour sets:

These sets of furnace numbers are formed to identify the colour of a furnace. Any two furnaces whose numbers belong to the same colour set have the same colour.

2) Factory sets:

Similarly these are sets of furnace numbers telling us which furnaces are in which factories.

3) Time periods:

We are dealing with an overall timespan of seven years. This is initially broken up into smaller units of time - time periods.

4) Representation of each repair schedule:

Assume we have 10 time periods. Then a schedule is represented by a list of 10 zeros or ones.

i.e. 0010000100

A 1 in time period \( k \) represents a rebuild in that time period.

So in the example, for this schedule, rebuilds took place in time periods 3 and 8.

5) Options:

We are given an equal number of schedules per furnace, above we dealt with seven in the case solution but these are represented as in (4). Say we have 8 furnaces and
6 options per furnace.

Repair schedule (3, 5) represents the 5th option for furnace 3.

6) Production figures of each schedule:

For each repair schedule we need the production figures for each time period. So corresponding to schedule 0010000100 we might have

20, 15, 5, 10, 15, 10, 5, 10, 15

Note that production is reduced in time periods in which a rebuild took place.

7) Cost of Schedules:

Each schedule has a cost, which is the expense of all production AND REPAIRS for that schedule.

8) Sales Demands:

These are given for each time period for each colour glass.

INTEGRATION OF CONSTRAINTS

Constraints (1) and (2) are of type (a). They directly forbid certain combinations of options in any solution.

E.g. let an option for: Furnace 1 be 0010000100

Furnace 3 be 0001000100

and let furnaces 1 and 3 be in the same factory. Now the above two options could never be in the same solution as a
repair takes place in the 8th time period for them both.

As we have seen already, a node represents the set of all solutions with some committed decisions and some rejected ones. To represent the illegal choices, which will be different for different nodes, I set the cost of an illegal choice to infinity. To do this I have to carry a separate set of costs for each node.

Example

Set of all solutions.

In node 2: cost of decision $D_{ij}$ must be set to 'infinity'.

In node 3: for furnace $i$ the cost of all options, except the $j^{th}$, must be set to 'infinity', since I can only choose one option per furnace.

SALES CONSTRAINT:

This is a constraint of type (b). Each node is treated separately. I construct an UPPER BOUND on the production of each colour glass in each time period. The upper bounds are then summed over the necessary time periods and the resulting figures are compared with the summed sales demands, and on this basis the node is tested.
If a node fails, the combination of committed and rejected decisions which define this node is impossible.

**Example:**

![Set of all solutions.](image)

Assume that node 6 fails the sales constraint. Node 6 is rejected, and the set of all solutions defined by

- **Option** $D_{i,j}$ CHOSEN for furnace $i$
- **Option** $D_{k,l}$ CHOSEN for furnace $k$
- **Option** $D_{m,n}$ NOT CHOSEN for furnace $m$

is discarded.

**Algorithm to Find the Best Solution**

**Stage 1. To find an initial solution:**

Read in and write out data. Set up node 1, test it with sales constraint and find a lower bound on cost.
(a) Branch on node 1. All nodes are explored remembering

(b) Choose decision to branch with and number new nodes.

(c) Find the sets of feasible options remaining open to the new nodes, and represent them by setting the cost of infeasible options to 'infinity' for each new node.

(d) Find the lower bounds on the cost of solutions for each new node. Test each new node with the sales constraint.

(e) Choose a branching node - check to see if it is a solution node

IF NOT GO TO (b)

(f) Otherwise

Stage 2. Search the remaining terminal nodes for the best solution.

Store solution

(i) Select the next branching node that is feasible for stage 2.

(ii) Develop it (a) choose a branching decision
(b) update costs of schedules FOR EACH NEW NODE
(c) form lower bounds on costs
(d) test with sales constraint

(iii) Choose next branching node. Check if it is a solution node.

IF NOT GO TO (ii)

(iv) Otherwise: if this solution is better than the 'best' one found so far, make it the 'best' one, otherwise reject it. GO TO (i).
Continue until all terminal nodes are explored remembering that each branching creates two new terminal nodes and destroys one. Ideally, nodes that cannot contain useful solutions are identified.

The steps of this algorithm are now explained in detail. Then follows a handworked example to illustrate the whole method. This example is simplified slightly but it serves its purpose.

**Criteria for rejecting a node**

If the node fails the sales constraint or if its lower bound on costs is infinite, then it is rejected and classified as infeasible.

**Defining among feasible terminal nodes**

In stage 1 we want to find an initial solution quickly. To do this we branch on the feasible node that is 'Nearest to a solution'. Now any node is defined by a set of committed and rejected options. The node which has the greatest number of committed options is 'nearest to a solution'.

**Example**

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

Graph shows the branching process with nodes and edges.
METHODS USED IN EACH STEP OF THE ALGORITHM

(1) CHOOSING A BRANCHING NODE:

In both stages of the problem there are two parts to this choice. Firstly, nodes that cannot contain useful solutions must be rejected. Secondly, a choice must be made among the remaining terminal nodes.

STAGE 1

Criteria for rejecting a node

If the node fails the sales constraint or if its lower bound on costs is infinite, then it is rejected and classified as infeasible.

Choosing among feasible terminal nodes

In stage 1 we want to find an initial solution quickly. To do this we branch on the feasible node that is 'Nearest to a solution'. Now any node is defined by a set of committed and rejected options. The node which has the greatest number of committed options is 'nearest to a solution'.

Example

```
1
 / \
2 3
 / \
4 5
 / \
6 7
 / \
8 9
```

Branch on Node 1

This strategy for obtaining an initial solution is called 'branching'. The idea is to go the nearest we can to a solution. We are trying all the time to develop the tree in this direction.

STAGE 1

Criteria for rejecting a node

(a) All nodes that fail the sales constraint are rejected.
Above each node is a set of three numbers:

- **X** - node number
- **Y** - lower bound on costs of solutions in this node
- **Z** = 1 .... node X **PASSES** sales constraint
- **Z** = 0 .... node X **FAILS** sales constraint

Consider the development of this tree.

1. Branch on node 1 to give new nodes 2 and 3.
2. Node 3 is feasible and is 'nearest a solution'.
   - Branch on node 3 to give new nodes 4 and 5.
3. Similarly branch on node 5 to give new nodes 6 and 7.
4. Now node 7 is 'nearest to a solution' but it is infeasible.
   - The next best feasible terminal node is node 6.
   - Branch on node 6 to give new nodes 8 and 9.
5. Again node 9 is nearest a solution but it is infeasible.
   - The next best node would be node 8 but this also is infeasible.
   - The terminal node that is nearest a solution and feasible is node 4.

Branch on Node 4.

This strategy for obtaining an initial solution is called 'branching right'. The further right and down the tree we go the nearer we are to a solution. We are trying, all the time, to develop the tree in this direction.

**Stage 2:**

**Criteria for rejecting a node:**

(a) All nodes that fail the sales constraint are rejected.
In stage 1 we found an initial solution. Any node with a lower bound on costs which is greater than or equal to the cost of this solution, cannot contain a better result. Hence it is rejected.

Condition (b) can be slightly altered so that a set of 'best' solutions can be found and retained.

**Choosing among the feasible terminal nodes**

In stage 2, having found an initial solution in stage 1, we search the feasible terminal nodes for better solutions. This is done systematically from right to left, by either rejecting or developing each terminal node until all possibilities have been exhausted.

The method used in stage 1 is used again here. Assume we have found a solution node, this is stored if it is the best one so far; we then reject this node and continue as if in stage 1 to find the next solution.

An example of this follows on the next page.
Example A 5 Furnace Problem

Now in stage 2, with the relevant criteria applying.

(1) As in stage 1, if a node was rejected, node 12 would be the next choice, but node 12 is infeasible in stage 2, since it cannot be part of a BEST solution. The next choice is node 10. Again this cannot be a BEST solution as its lower bound on cost is greater than 710.

This brings us to node 6. This is feasible, so develop it.

(2) Branching on node 6, we see no nodes 14 and 15. Continuing as in stage 1 we aim for a root solution. Using the criteria for stage 2, node 15 is the feasible node nearest a solution. Branch on node 15 to give nodes 16 and 17.

(3) Node 17 is feasible and is a solution node. Its cost is less than our previous best solution. We retain the previous root solution represented by node 11.

Now reject node 17 and find the next solution.

Node 16 is the next feasible node that is reached. Branch on node 16 to give nodes 18 and 19.

Node 18 is infeasible

Node 14 is infeasible

Node 15 is infeasible

Node 11 is infeasible.

Let node 13 represent our initial solution. This is stored and the node rejected and we find the next solution. We are
now in stage 2, with the relevant criteria applying.

(1) As in stage 1, if node 13 was rejected, node 12 would be the next choice, but node 12 is infeasible in stage 2, since it cannot contain a BETTER solution. The next choice is node 10. Again this cannot give a BETTER solution as its lower bound on costs is greater than 710.

This brings us to node 8. This is feasible, so develop it.

(2) Branching on node 8 gives new nodes 14 and 15. Continuing as in stage 1 we aim for a quick solution. Under the criteria for stage 2, node 15 is the feasible node nearest a solution. Branch on node 15 to give nodes 16 and 17.

(3) Node 17 is feasible and it is a solution node. Its cost is less than our previous best solution. So retain the solution represented by node 17.

Now reject node 17 and find the next solution.

Node 16 is the next feasible node that is nearest a solution.

Branch on node 16 to give nodes 18 and 19.

(4) Node 19 is feasible and is a solution node. Retain this solution, reject node 19 and continue.

Node 18 is infeasible
Node 14 is infeasible
Node 4 is infeasible
Node 2 is infeasible.
Hence the problem is finished. Notice that the direction of search is from right to left and it is essentially the method that was used in stage 1 applied many times.

For a small example with 3 furnaces and 3 options per furnace:

<table>
<thead>
<tr>
<th>Furnace</th>
<th>Option 1</th>
<th>Option 2</th>
<th>Option 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>120</td>
<td>110</td>
<td>140</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>100</td>
<td>105</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>105</td>
<td>110</td>
</tr>
</tbody>
</table>

For each furnace:

Remember two points:

(a) As soon as an option is committed it knocks out all other options for the furnace, and probably some choices for other furnaces.

(b) Ultimately we are seeking the minimum cost solution, and the nearer our first solutions come to this the faster the problem will be finished. At each iteration we therefore branch with the option of minimum cost for its furnace.

The Minimum Penalty Criteria

Consider branching with the minimum cost option of furnace 1; i.e., (1,1)

The minimum penalty incurred by OPT committing (1,1) is \((110 - 100) = 10\).

Similarly:

- The minimum penalty associated with (2,1) is \((100 - 90) = 10\).
- The minimum penalty associated with (3,1) is \((105 - 80) = 25\).
(2) **CHOOSING A DECISION TO BRANCH WITH**

Having chosen a branching node we want to choose a decision on the basis of which to split the branching node. Consider a small example with 3 furnaces and 3 options per furnace.

**Cost of options**

<table>
<thead>
<tr>
<th>Furnace 1</th>
<th>100</th>
<th>110</th>
<th>140</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furnace 2</td>
<td>90</td>
<td>100</td>
<td>105</td>
</tr>
<tr>
<td>Furnace 3</td>
<td>80</td>
<td>105</td>
<td>110</td>
</tr>
</tbody>
</table>

Remember two points:

(a) As soon as an option is committed it knocks out all other options for the furnace, and probably some choices for other furnaces.

(b) Ultimately we are seeking the minimum cost solution, and the nearer our first solutions come to this the faster the problem will be finished. At each iteration we therefore branch with the option of minimum cost for its furnace.

**The Maximum minimum penalty criteria**

Consider branching with the minimum cost option of furnace 1: i.e. (1,1)

The minimum penalty incurred by NOT committing (1,1) is \( (110 - 100) = 10 \).

Similarly:

The minimum penalty associated with (2,1) is \( (100 - 90) = 10 \)

The minimum penalty associated with (3,1) is \( (105 - 80) = 25 \)
Committing any other option than \((3, 1)\) may knock out \((3, 1)\) hence incurring a penalty of 25. This is the maximum penalty associated with any of the minimum cost options.

To avoid the possibility of incurring this maximum penalty we branch with \((3, 1)\).

To choose the branching decision:

(a) Select for each uncommitted furnace the minimum cost option for each furnace.

(b) For each of these options, calculate the minimum penalty incurred by avoiding it.

(c) Branch on the minimum cost option which incurs the maximum minimum penalty.

Branch on decision \((3, 1)\) to give new nodes 3 and 5.

Feasible options for new node 2:

Node 2 represents all those solutions without option 1 taken for furnace 3.

Hence \((3, 1)\) becomes infeasible. Represent this by setting its cost to 'infinity'.

Costs of options for node 2

<table>
<thead>
<tr>
<th>Furnaces</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>110</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>100</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>105</td>
<td>110</td>
<td></td>
</tr>
</tbody>
</table>

Feasible options for new node 3:

Node 3 represents all solutions with option 1 taken for furnace 3. Thus the following...
(3) REPRESENTATION OF INFEASIBLE OPTIONS FOR EACH NODE

Consider the above example. Let the table of costs for node 1 be:

<table>
<thead>
<tr>
<th>OPTION COSTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FURNACES 1 100 110 140</td>
</tr>
<tr>
<td>2 90 100 105</td>
</tr>
<tr>
<td>3 80 105 110</td>
</tr>
</tbody>
</table>

Assume that we cannot have more than one of the following options in any one solution:

(3,1), (1,1), (2,3)

Branch on decision (3,1) to give new nodes 2 and 3.

Feasible options for new node 2:
New node 2 represents all those solutions without option 1 taken for furnace 3.

Hence (3,1) becomes infeasible. Represent this by setting its cost to 'infinity'.

Costs of options for node 2

<table>
<thead>
<tr>
<th>OPTION COSTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FURNACES 1 100 110 140</td>
</tr>
<tr>
<td>2 90 100 105</td>
</tr>
<tr>
<td>3 ∞ 105 110</td>
</tr>
</tbody>
</table>

Feasible options for new node 3:

Node 3 represents all solutions with option 1 taken for furnace 3. Thus the following
(a) \((3,2)\) and \((3,3)\) since only one option can be chosen for each furnace.

(b) \((1,1)\) and \((2,3)\) since only one option of the set \((3,1), (1,1), (2,3)\) can be in any one solution.

Costs of Options for Node 3

<table>
<thead>
<tr>
<th>FURNACES</th>
<th>OPTION</th>
<th>COSTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>110</td>
<td>140</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>(\infty)</td>
</tr>
</tbody>
</table>

In the example at the end of this chapter 'infinity' is represented by \(10^{10}\).
(4) **TO FIND A LOWER BOUND ON THE COSTS OF SOLUTIONS FOR A GIVEN NODE**

For each furnace choose the option of minimum cost. Then, sum the costs of these chosen options. This gives a lower bound for this node.

For the previous example:

- **Node 1**: L.B. = 270
- **Node 2**: L.B. = 295
- **Node 3**: L.B. = 280

This method always gives a lower bound, but for a solution node it will give the exact cost of the solution since there is only one feasible option for each furnace.

Options with infinite cost (i.e., $10^{10}$) are illegal at this stage.
We are given the sales demand for each colour and time period and the production figures corresponding to each option.

We first construct an upper bound on production of each colour glass in each time period. We compare these upper bounds with the sales demands (each summed over consecutive time periods) to test the node.

**Construction of an upper bound**

Consider the following example with 3 furnaces, 3 options per furnace and 5 time periods.

<table>
<thead>
<tr>
<th>Furnace</th>
<th>Option</th>
<th>Time Period</th>
<th>Production</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 3 10 13 20</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2 20 15 10 5</td>
<td>10^10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>15 10 15 15 5</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6 8 10 12 10</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12 10 8 6 6</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8 6 10 10 6</td>
<td>10^10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>8 8 5 8 8</td>
<td>10^10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5 8 7 5 3</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5 8 7 6 4</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

Options with infinite cost (i.e. 10^10) are illegal at this stage.
For simplicity assume all three furnaces make the same colour glass.

(a) Find for each furnace the maximum possible production in each time period. Production figures of illegal options are classified as impossible.

i.e. FURNACE 1 15 10 15 20 15
FURNACE 2 12 10 10 12 10
FURNACE 3 5 8 7 6 4

(b) To obtain an upper bound on production of each colour glass in each time period.

For each time period, for each colour, sum the maximum possible production figures for each furnace making this colour glass.

i.e. as we have only one colour, we just make a column sum of the above figures.

<table>
<thead>
<tr>
<th>Time Period</th>
<th>UPPER BOUND ON</th>
<th>PRODUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>38</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>29</td>
</tr>
</tbody>
</table>

Testing the Node

Assume that for this example we have the following sales demand in each time period.

<table>
<thead>
<tr>
<th>Time Period</th>
<th>SALES DEMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31, 27</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
</tbody>
</table>

We now compare the following,
Upper bound on production in T.P. 1 WITH Sales demand on TP 1

For simplicity we will have 2 furnaces, 3 options per furnace and 5 time periods.

1+2 WITH TP 1+2
1+2+3 WITH TP 1+2+3
2+3+4 WITH TP 2+3+4
3+4+5 WITH TP 3+4+5

Except for the first two time periods we are comparing upper bounds on production with sales demand, summed over 3 consecutive time periods.

E.g.,

<table>
<thead>
<tr>
<th>Time</th>
<th>Production Bounds summed as above</th>
<th>Sales demands summed as above</th>
<th>Result of comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>31</td>
<td>PASS</td>
</tr>
<tr>
<td>2</td>
<td>32 + 28</td>
<td>60</td>
<td>PASS</td>
</tr>
<tr>
<td>3</td>
<td>32 + 28 + 32</td>
<td>92</td>
<td>PASS</td>
</tr>
<tr>
<td>4</td>
<td>28 + 32 + 38</td>
<td>98</td>
<td>PASS</td>
</tr>
<tr>
<td>5</td>
<td>32 + 38 + 29</td>
<td>99</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

So this node passes in all time periods except 5, therefore it FAILS the sales constraint.

If any node fails the constraint then so does any solution which belongs to the node. It is possible for a node to pass the constraint while containing NO solutions which also do this. If this happens the constraint rejects all these solutions at some later stage.

When this method is applied to a solution node, the upper bounds are the actual production figures, so we are certain that any solution which passes the sales constraint meets the sales demands.
EXAMPLE:

For simplicity we will have

3 furnaces, 3 options per furnace, 5 time periods.

Furnaces produce 1 colour set (i.e. all furnaces produce the same coloured glass - colour is amber).

1) Total production in time period = sales demand in that period.
2) All furnaces in the same factory.
3) Each furnace has repair schedules for each time period.

This is a very simple problem but it will serve to illustrate the main points of the problem.

DATA

<table>
<thead>
<tr>
<th>Furnace</th>
<th>Option</th>
<th>Repair Schedules (time periods)</th>
<th>Production Figures for each time period</th>
<th>Original Cost of each option</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1 0 0 0 0</td>
<td>5 8 10 10 10</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0 0</td>
<td>10 8 5 7 8</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 0 1</td>
<td>10 10 8 6 5</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 0 1</td>
<td>8 5 10 10 5</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 1 0</td>
<td>10 10 8 5 10</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>0 1 0 0 0</td>
<td>10 5 8 10 10</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1 0 0 1</td>
<td>5 8 10 10 5</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0 0 0</td>
<td>10 5 8 10 10</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>1 0 0 0 0</td>
<td>5 8 10 10 10</td>
<td></td>
<td>18</td>
</tr>
</tbody>
</table>

Note that in the example we have only one colour of glass, normally we have 5 colours and each colour is used in 1 furnace.

Using the algorithm we have formed the 'tree' to illustrate the development of the process. Again, three numbers will be associated with each node, giving:

SALES DEMANDS IN EACH TIME PERIOD.

a) Node Number
b) Lower Bound
    c) 0 if node fails sales constraint
    d) 1 if node Fails sales constraint

24 23 25 25 26
THE PROBLEM: find an initial solution

To find a minimum cost solution such that no two furnaces have a rebuild in the same time period,

a) lower bound costs = 14 + 10 + 15 = 49

b) i) Total Production in time period 1 ≥ sales demand in 1

while for each

ii) \( t_p \) for 1 + 2 ≥ 1 + 2

iii) \( t_p \) for 1 + 2 + 3 ≥ 1 + 2 + 3

colour

iv) \( t_p \) for 2 + 3 + 4 ≥ 2 + 3 + 4

v) \( t_p \) for 3 + 4 + 5 ≥ 3 + 4 + 5

This form of a sales constraint is parallel to the one used in the main problem, i.e. for each colour for each time period.

a) for T.P. 1 total production in T.P. 1 ≥ sales demand in T.P. 1

b) \( t_p \) for 2 ≥ 1 + 2

c) for any T.Period K \( (K-2)+(K-1) \) ≥ \( (K-2)+(K-1) \)

Note that in the example we have only one colour of glass, normally we have 3, and each colour is treated separately.

Using the algorithm we now form the 'tree' to illustrate the development of the process. Again, three numbers will be associated with each node, giving

a) Node Number

b) Lower bound on costs

c) 0 if Node FAILS sales constraint

1 if Node PASSES sales constraint
Stage 1. To find an initial solution

Setting up Node 1:

a) Lower bound costs = 14 + 10 + 15 = 39

b) Upper bounds on production in each time period,
   30; 28; 30; 30; 30 (derivation explained in other nodes)
   hence node 1 obviously satisfies sales constraint.

A) Choose Node to branch on - Node 1.

B) Choose decision to branch with

   for FURNACE 1: minimum penalty incurred by avoiding the
   cheapest feasible option is \((15 - 14) = 1\)

   FURNACE 2: Similarly minimum penalty is \((13 - 10) = 3\)

   FURNACE 3: Similarly minimum penalty is \((18 - 15) = 3\).

Choose to branch on option which gives the maximum, minimum
penalty (in this case we have a draw \((2,3)\) and \((3,2)\))

Choose \((2,3)\) - i.e. option 3: FURNACE 2

So we have

C) To form the sets of feasible options for each new node by
   'updating' the costs of schedules:

New Node 2: represents all solutions WITHOUT option 3 for
furnace 2:
   So option \((2,3)\) is infeasible set its cost to 10

34
Cost of repair schedules of New Node 2:

<table>
<thead>
<tr>
<th>OPTIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furnace 1</td>
<td>14</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20</td>
<td>15</td>
</tr>
</tbody>
</table>

New Node 3: represents all solutions WITH option 3 TAKEN for furnace 2.

So a) (2,1) and (2,2) become infeasible.

b) (3,2) becomes infeasible as it has a repair in the same time period as (2,3).

Cost of repair schedules for New Node 3

<table>
<thead>
<tr>
<th>OPTIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furnace 1</td>
<td>14</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10^10</td>
<td>10^10</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20</td>
<td>10^10</td>
</tr>
</tbody>
</table>

D) i) To find lower bounds on costs of solutions for New Nodes 2 and 3.

New Node 2:

Lower Bound (L.B.) = 14 + 13 + 15 = 42

New Node 3:

L.B. = 14 + 10 + 18 = 42

Similarly, select maximum possible production for each furnace for each time period.
ii) Testing the new nodes with the sales constraints:

**New Node 2:**

First, for each furnace choose the maximum production element from each column - but it must not come from an infeasible repair schedule. (There are costs of repair for the node under consideration to tell.

| Furnace 1 | 10, 10, 10, 10, 10 |
| Furnace 2 | 10, 10, 10, 10, 10 |
| Furnace 3 | 10, 10, 10, 10, 10 |

Now form an upper bound on production in each T.P. by summing these columns.

<table>
<thead>
<tr>
<th>Time Period</th>
<th>Upper bounds on Production summed for each T.P.</th>
<th>Sales demand summed for each T.P.</th>
<th>Result for each T.P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>24</td>
<td>PASS</td>
</tr>
<tr>
<td>2</td>
<td>30 + 28 = 58</td>
<td>24 + 23 = 47</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>30 + 28 + 30 = 88</td>
<td>24 + 23 + 25 = 72</td>
<td>&quot;</td>
</tr>
<tr>
<td>4</td>
<td>28 + 30 + 30 = 88</td>
<td>23 + 25 + 25 = 73</td>
<td>&quot;</td>
</tr>
<tr>
<td>5</td>
<td>30 + 30 + 30 = 90</td>
<td>25 + 25 + 26 = 76</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

So choose a branching decision by maximising, minimise.

**Node 2 passes:**

**New Node 3:**

Similarly, select maximum possible production for each furnace for each time period.
FURNACE 1: 10, 10, 10, 10, 10
FURNACE 2: 10, 5, 8, 10, 10 (Note that for furnace 2 (2,3) is only feasible schedule
FURNACE 3: 5, 8, 10, 10, 10 hence we use it's production figures)

(When deriving these figures you need to refer to the table of costs of repairs for the node under consideration to tell you which schedules are feasible), hence the Upper bound on production in each time period is: 25, 25, 28, 50, 50 as each of these is greater than the corresponding sales demand this node must PASS the sales constraint.

(c) Updating costs for the new nodes:

B) Branch on Node 3

Obviously node 3 is not a solution node:

hence we have so far

GO TO (b)

2nd ITERATION

(B) To choose a branching decision by maximum, minimum penalty method:

Consider 'costs' for node 3.

(i) Set (3,1) and (3,2) to 10^10 (if not already at it).
(ii) Set (1,3) to 10^10 since (7,4) and (3,3) both have

require in time period 1.
As furnace 2 is committed for this node it need not be considered.

**FURNACE 1:** Min. penalty incurred by avoiding (1,1) is 1

**FURNACE 3:** Min. penalty incurred by avoiding (3,3) is 2

Hence branch using (3,3) and form new nodes 4 and 5.

(C) Updating costs for the new nodes:

**New Node 4:** represents all solutions with option 3 for furnace 2 and WITHOUT option 3 for furnace 3.

Take the costs of node 3 and set the cost of (3,3) to $10^{10}$.

**New Node 5:** represents all solutions with option 3 taken for furnace 2 and option 3 taken for furnace 3.

Take set of costs of node 3 and

(i) Set (3,1) and (3,2) to $10^{10}$ (if not already at it)

(ii) Set (1,1) to $10^{10}$ since (1,1) and (3,3) both have repairs in time period 1.
Costs for NEW NODE 5:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FURNACE 1</td>
<td>10</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>&quot; 2</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>&quot; 3</td>
<td>10</td>
<td>10</td>
<td>18</td>
</tr>
</tbody>
</table>

(D) Lower Bounds on costs of solutions for each new node

New Node 4: $14 + 10 + 20 = 44$
New Node 5: $15 + 10 + 18 = 43$

Sales constraint:

Select maximum possible production for each furnace in each time period.

Hence upper bound on production in each T.Period is:

25, 25, 28, 30, 25

<table>
<thead>
<tr>
<th>Test Node 4</th>
<th>Time Period</th>
<th>The upper bounds on Production summed for each T.P</th>
<th>The sales demand summed for each T.P</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>24</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25 + 23</td>
<td>48</td>
<td>47</td>
<td>PASS</td>
</tr>
<tr>
<td>3</td>
<td>25 + 23 + 28</td>
<td>76</td>
<td>24 + 23 + 25 = 72</td>
<td>PASS</td>
</tr>
<tr>
<td>4</td>
<td>23 + 28 + 30</td>
<td>81</td>
<td>23 + 25 + 26 = 76</td>
<td>PASS</td>
</tr>
<tr>
<td>5</td>
<td>28 + 30 + 25</td>
<td>83</td>
<td>25 + 25 + 26 = 76</td>
<td>PASS</td>
</tr>
</tbody>
</table>
NODE 4 PASSES.

Considering cost for new node 5.

New node 5: unobstructed furnace is 5, so branch to (1,3).

Max. possible production for each furnace for each time period.

(5) Update costs.

FURNACE 1 10 10 8 7 8
n 2 10 5 8 10 10
n 3 5 8 10 10 10

Upper bound on production in each T.P

2, 25, 23, 26, 25, 28

Test the node:
as all upper bounds on production greater than corresponding sales demands.

NODE 5 PASSES.

(5) Branch on node 5.

5 is not a solution node:

Maximum possible production for each furnace in each T.P.
ITERATION 3

(B) Considering costs for new Node 5:

the only uncommitted furnace is 1: so branch on (1,3)
giving new Nodes 6 and 7.

(C) Update costs:

New Node 6:

Take costs for Node 5 and set (1,3) to 10^{10}

\begin{array}{ccc}
1 & 2 & 3 \\
FURNACE 1 & 10^{10} & 16 & 10^{10} \\
2 & 10^{10} & 10^{10} & 10 \\
3 & 10^{10} & 10^{10} & 18 \\
\end{array}

New Node 7:

Take costs for Node 5 and set (1,2) to 10^{10}

\begin{array}{ccc}
1 & 2 & 3 \\
FURNACE 1 & 10^{10} & 10^{10} & 15 \\
2 & 10^{10} & 10^{10} & 10 \\
3 & 10^{10} & 10^{10} & 18 \\
\end{array}

(D) Lower bounds on costs:

Node 6: \quad 16 + 10 + 18 = 44

Node 7: \quad 15 + 10 + 18 = 43

Sales constraint

Node 6:

Maximum possible production for each furnace in each T.P.:
Upper bound on production in each T.P.:

25, 21, 23, 27, 28

Test the node

T.P. 1: 25 : 24
2: 25 + 21 = 46; 24 + 25 = 49 F.A.I.L.E.D

Node 6 FAILS in Time Period 2.

Node 7: Maximum possible production for each furnace in T.P.

Furnace 1: 10 10 8 6 5
2: 10 5 8 10 10
3: 5 8 10 10 10

Upper bound of production

25, 25, 26, 26, 25

Node 7 PASSES.

Starting at Node 7:

B) BRANCH ON Node 7 - but:

Node 7 is a solution node

Note that the lower bound on costs for this node is in fact the actual cost of the solution; and the upper bound on production in each time period, is in fact the production for this solution.
Selection of branching decision gives \((3,3)\): form new nodes 3 and 9.

(a) Form of subtree for new node

New Node 6:

(i) Select next branching node that is feasible under Stage 2 criteria

Starting at Node 7:

(a) Reject Node 6: as fails sales constraint

4: as lower bound on costs = 44

while cost of present best solution is 43.

Develop Node 2: as it has a lower bound on costs 43.

Branch on Node 2:

(ii) (a) Choosing a branching decision

Costs of options for Node 2 are

New Node 3: 25, 23, 50, 30, 50

Node 6 PASS.
Selection of branching decision gives (3,2): form new nodes 8 and 9.

(b) Costs of options for new nodes:

New Node 8:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FURNACE 1</td>
<td>14</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>13</td>
<td>10^10</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>10^10</td>
<td>18</td>
</tr>
</tbody>
</table>

New Node 9:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FURNACE 1</td>
<td>14</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>10^10</td>
<td>13</td>
<td>10^10</td>
</tr>
<tr>
<td>3</td>
<td>10^10</td>
<td>15</td>
<td>10^10</td>
</tr>
</tbody>
</table>

(c) Lower bounds on costs:

Node 8: \(14 + 13 + 18 = 45\)

Node 9: \(14 + 13 + 15 = 42\)

(d) Sales constraints:

Maximum possible production for each furnace for each T.P., summed to give the upper bound of:

New Node 8: 25, 28, 30, 30, 30

Node 8 PASSES.
New Node 9: 30, 25, 26, 25, 30

Node 9 PASSES.

iii) Branch right on Node 9

We have no new solution node:

- represents development in stage 2.

**DEVELOPE NODE 9:**

From costs table for node 9: Branch on (2,2) as minimum penalty incurred by avoiding it is \((10^{10} - 13)\).

To give new nodes 10 and 11.

Costs of options for new nodes:
we have no new solution node.

**NODE 10**

**OPTION**

1 2 3

**FURNACE 1**

1 14 16 15

2 10^10 10^10 10^10

3 10^10 15 10^10

**OPTION**

1 2 3

**FURNACE 1**

1 14 16 15

2 10^10 13 10^10

3 10^10 15 10^10

**Lower bounds on costs:**

Node 10 = 24 + 10^10

Node 11 = 42

**Sales Constraint:**

**Node 10**

Maximum possible production for each furnace in each T.P.

**FURNACE 1**

1 10 10 10 10 10

2 0 0 0 0 0

3 10 5 8 10 10

Upper bound 20, 15, 18, 20, 20

In fact node 10 FAILS.

**Node 11**

Upper bounds are:

30, 25, 26, 25, 30

Node 11 FAILS.
We have no new solution node.

Branch with \((1,1)\) giving nodes 12, 13.

Costs of options for new nodes.

**NODE 12**

<table>
<thead>
<tr>
<th>OPTION</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FURNACE 1</td>
<td>10^{10}</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>10^{10}</td>
<td>13</td>
<td>10^{10}</td>
</tr>
<tr>
<td>3</td>
<td>10^{10}</td>
<td>15</td>
<td>10^{10}</td>
</tr>
</tbody>
</table>

**NODE 13**

<table>
<thead>
<tr>
<th>OPTION</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FURNACE 1</td>
<td>14</td>
<td>10^{10}</td>
<td>10^{10}</td>
</tr>
<tr>
<td>2</td>
<td>10^{10}</td>
<td>13</td>
<td>10^{10}</td>
</tr>
<tr>
<td>3</td>
<td>10^{10}</td>
<td>15</td>
<td>10^{10}</td>
</tr>
</tbody>
</table>

**Lower bounds on costs:**

Node 12: 43

Node 13: 42

The cost of the solution represented by node 13 is 42.

**Sales Constraint:**

Starting from Node 13:

Node 12: Upper bound on production rejects Node 12 as fails sales constraint

Node 12 fails.
The problem is finished.

Node 13

The 1st solution represented by node 13 has cost 42.

Node 13 passes

and we have a new SOLUTION NODE COST: 42

The 2nd solution represented by node 17 has cost 43.

The cost of the solution represented by node 15 is 42, better than the previous one, so retain it.

Starting from node 15:

Reject node 12 as FAILS sales constraint

Reject node 10 as FAILS sales constraint and has $\infty$ cost

Reject node 8 as lower bound on cost is greater than best solution so far.
Hence the problem is finished.

The 1st solution represented by node 13 has cost 42, production 25, 23, 26, 25, 30, and is:

(a) An initial feasibility test on Node 1.
(b) The introduction of array INC(I) which gives the committed option for each furnace I for each node k.
(c) Disc storage is used for arrays INC, COST.

The 2nd solution represented by node 7 has cost 43, and is:

(a) A check on production 25, 23, 26, 26, 25 etc.

and is:

(a) Options for printing:
(i) Print out of information for each node.
(ii) Print 2 only for solution nodes only.
(iii) An option for defining 3, set of retained solutions.

So for the best solution, rebuild FURNACE 1 in Time Period 1 and the report is supplied with the following:

```
  "  2  "  "    4
  "  3  "  "    2
```

For the 2nd best solution, rebuild FURNACE 1 in Time Period 5 and the report is supplied with the following:

```
  "  2  "  "    2
  "  3  "  "    1
```
The algorithm described in the previous chapter forms a basis for the final program. The only differences are:

(a) An initial feasibility test on Node 1.
(b) The introduction of array ICD(I) which gives the committed option for each furnace I for each Node N.
(c) Disc storage is used for arrays ICD, CSTREP because each array has to be stored for each node.
(d) A check to ensure that we do not generate more than 8 nodes.
(e) Options to give either
   (i) Print out of information for each node.
   or (ii) Print out for solution nodes only.
(f) An option for defining the set of retained solutions.

A flow chart for the basis algorithm is given on the next page, and a copy of the full flow chart is supplied with the report.
The Main Variables

$I_1$ - the number of furnaces

$J_1$ - the number of options per furnace

$K_1$ - the number of time periods

$C_1$ - the number of colours

$D_1$ - the maximum number of furnaces of any one colour

$F_1$ - the number of factories

$F_{H_1}$ - the maximum number of furnaces in any one factory.

$S$ - the maximum number of nodes we can generate.

$M_{R_1}$ - the maximum number of time periods in which a furnace is in repair for any option.

$I_Q$ - the number of 'best' solutions required.

$I_{OPT} (=1)$ gives the best $I_Q$ of all possible solutions.

$I_{OPT} (=0)$ gives all the solutions of the cheapest cost (up to $I_Q$ in number). If there are less than $I_Q$ such solutions, it fills out the numbers by selecting the best among those other solutions found up until the time when the last minimum cost solution was found.

**COLOUR SETS**

$COL(C,B)$ - gives the number of the $B^{th}$ furnace of colour $C$.

**FACTORY SETS**

$FACT(FG, PH)$ - gives the number of the $PH^{th}$ furnace in factory $FG$.

For both the above sets, spaces in the arrays are filled by
zeros, since there are not equal numbers of furnaces in each
colour set or factory set.

**Repair Schedules**

\( R(i, j, k) \) - for each furnace \( i \), for each option \( j \), gives
the numbers of the time periods in which a
repair took place.

**Production Figures**

\( P(i, j, k) \) - gives for each furnace \( i \), for each option \( j \),
for each time period \( k \) - the production figure.

**Sales Demands**

\( S(c, k) \) - gives the demand for colour glass \( c \) in time period
\( k \).

**Sales Constraint**

\( A(c, k) \) - weighting factors in the sales constraint, given
\( I(c, k) \)

**Cost of Repairs**

\( C(i, j) \) - gives the cost of option \( j \) for furnace \( i \).

This varies from node to node and as it is necessary to trace
this array for all nodes. The version for each node is stored
on disc.

**Committed Decisions**

\( I(i) = 0 \) means that furnace \( i \) is uncommitted.

\( = j \) means that option \( j \) is committed for furnace \( i \).
Again this array varies for different nodes. So for each node the array is stored on disc.

**Lower bounds on costs**

\( W(N) \): gives the lower bound on costs of solutions in node \( N \).

**SALES CONSTRAINT**

\( IR(N) \): gives the result of the sales constraint applied to node \( N \).

- 0 ... FAILED
- 1 ... PASSED

**Tree Structure**

\( PAR(N) \): gives the parent node of \( N \).

**Set of 'Best' Solutions**

\( IBEST(J) \): gives the node number of the \( J \)th best solution found so far. In stage 1, or stage 2 when \( IQ \) solutions have not been found, spaces in \( IBEST \) are filled with the node number \((S+1)\) which has a cost of \( 10^{10} \) and does not really exist.

**Writing out interim information**

- \( IWRITE = 0 \) Write out \( N, W(N), IR(N) \) and \( ICD(I) \) for solution nodes only.
- \( IWRITE = 1 \) Write out \( N, W(N), IR(N), ICD(I) \) for every node. Also write out each branching node number, branching furnace number and option number.
Dimensions of Arrays

GSTRMPT(I1,J1), MIN(I1), REP(I1), MINDEN(I1), PEN(I1,J1)
IGD(I1), IELR(I1,J1,MN1), ELRFCH(I1,J1,K1), C0L(C1,D1)
FACT(FG1,FG1), UNION(D1 + FH1), K1(K1), SALES(C1,K1)
A(C1,K1), B(C1,K1), Ia(C1,K1), MAXEL(I1,K1), MAXFROD(C1,K1)
SUMPROD(C1,K1), SUMEL(C1,K1), ICON(C1,K1); ISUMPRT(I1,K1)
PAR(S), W(S+1), IR(S), IBEST(IQ), IDUM(IQ)
IFLOAT(C1,K1), IREL(K1), INDEX(2S+1000)

At the beginning of the program maximum dimensions are stated to allow for the running of problems of different sizes.

DISC STORAGE

At the beginning of the program maximum dimensions are stated to allow for the running of problems of different sizes.

DISC STORAGE

GSTRMPT(I1,J1) and GSTRMPT(I1,J1) are stored on disc for each node.

For Node N

GSTRMPT : in stored in the block N

IGD : in stored in the block S + N
(1) **SUBROUTINE MINCOST (R)**

For a given node R, this subroutine finds, for each furnace I, the cost of the cheapest repair pattern and stores it in MIN(I); it also finds the number of the pattern corresponding to this cost and stores it in REP(I). In the event of a draw the first one is chosen. This subroutine is used mainly to help find the lower bound on costs for each node.

**Variables:**
- I: no. of furnaces
- J: no. of options per furnace
- K: no. of time periods
- S: max. number of nodes.

For Node R

$$CSTREP(I,J):$$ gives the cost of option J for furnace I for Node R.

$$MIN(I):$$ gives the cost of minimum cost repair pattern for furnace I.

$$REP(I):$$ gives the number of this pattern.

**DATA FLOW**

**INPUT:**
- I, J, K, S : via COMMON/A1/
- CSTREP(I,J): This is read off disc.
- N : via call statement.

**OUTPUT:**
- MIN(I) & REP(I): via COMMON/A2/.
FLOW CHART

READ IN: N, I, J, K, S, CSTREP (FROM DISC)

LET I RUN OVER EACH FURNACE NUMBER

I = 1

SET UP INITIAL VALUE OF MIN(I) AND REP(I)

MIN(I) = CSTREP(I, I)
REP(I) = 1

FOR EACH OPTION J COMPARE COST WITH PRESENT VALUE OF MIN(I)

J = 2

IS MIN(I), LE. CSTREP(I, J)

MIN(I) = CSTREP(I, J)
REP(I) = J

IS J, EQ. J 1

J = J + 1

IS I, EQ. I

I = I + 1

END
(2) SUBROUTINE SELECTBD(BN, BF, BD)

Given a branching node BN, this subroutine finds an option to branch with by the maximum, minimum penalty method. To identify this decision we need its furnace number and its option number.

BF - furnace number
BD - option number

Variables:
I1 : no. of furnaces
J1 : no. of options per furnace
K1 : no. of time periods
S : max. no. of nodes.
BN: number of branching node
BF: branching furnace
BD: branching option of furnace BF

FOR NODE BN: (the following arrays will vary for each node)
CSTREP(I, J): cost of option J for furnace I
MIN(I): cost of minimum cost repair pattern for furnace I
REP(I): the option number of this repair pattern
PEN(I, J): 'penalty' attached to option J for avoiding option REP(I), for furnace I.
PEN(I, J) = CSTREP(I, J) - MIN(I)

MINPEN(I): the minimum penalty cost for avoiding option REP(I) for furnace I.

MINPEN(I) = \min_{J \not= REP(I)} PEN(I, J)

(where J only corresponds to feasible options. J \not= REP(I)).
MAXMIN: the maximum of these minimum penalties.

IOD(I): a) = 0 : then I is an uncommitted furnace.
b) otherwise it gives the number of the committed option for furnace I.

DATA-FLOW

INPUT: PI1, J1, K1, S : via COMMON/A11/
BN : via call statement

GSTREP(I, J) \{ 
    FOR EACH FURNACE FIND THE MINIMUM PENALTY COST FOR AVOIDING REF(I)
\}

OUTPUT: BF, BD via call statement.

3 POSSIBLE CASES FOR EACH FURNACE I:

1) I A COMMITTED FURNACE ..........................  IOD(I) # 0
2) I AN UNCOMMITTED FURNACE WITH ONLY ONE FEASIBLE OPTION: ..........................  IOD(I) = 0; MIPEN(I) > 10
3) I AN UNCOMMITTED FURNACE WITH 2 FEASIBLE OPTIONS: ..........................  IOD(I) = 0; MIPEN(I) < 10

TAKE EACH FURNACE I DO FOR I = 1, J1

I CASE 2

YES

COMMIT THIS FURNACE AS ONLY ONE FEASIBLE OPTION
BF = I
BD = REF(I)

END
FLOW CHART

READ IN: Cn, I1, J1, K1, S,
CSTREP (OFF DISC)

FIND MINIMUM COST OPTION FOR EACH FURNACE (MINCST)

FOR EACH FURNACE I, OPTION J,
FIND PENALTY FOR AVOIDING REP(I).
PEN(I,J) = CSTREP(I,J) - MIN(I)

FOR EACH FURNACE FIND THE MINIMUM PENALTY COST FOR AVOIDING REP(I)
MINPEN(I)

3 POSSIBLE CASES FOR EACH FURNACE I:
1) I A COMMITTED FURNACE ............... ICD(I) ≠ 0
2) I AN UNCOMMITTED FURNACE WITH ONLY ONE FEASIBLE OPTION ............... ICD(I) = 0; MINPEN(I) > 10^8
3) I AN UNCOMMITTED FURNACE WITH > 2 FEASIBLE OPTIONS ............... ICD(I) = 0; MINPEN(I) ≤ 10^8

TAKE EACH FURNACE I
DO 800 I = 1, J1

IS I CASE 1

YES
FURNACE IS ALREADY COMMITTED
GO TO 800

NO

IS I CASE 2

YES

COMMIT THIS FURNACE AS ONLY ONE FEASIBLE OPTION
BF = I
BD = REP(I)

NO

I IS CASE 3

STORE I WITH MAX. MINPEN FOUND SO FAR

800 CONTINUE

BF = I
BD = REP(I)

END
SUBROUTINE CSTREPN (BN, BF, BD, NCNT)

Given the branching node BN, and branching furnace BF, and branching option, CSTREPN finds the cost of repair schedules for the new nodes and stores the new sets of costs in the correct positions on disc for the new nodes.

Variables:

- I1 : no. of furnaces
- J1 : no. of options per furnace
- K1 : no. of time periods
- S : max. no. of nodes
- G1 : no. of colours
- FG1: no. of factories
- GOL(0,,D) : gives the colour sets
- FACTCFG : gives the factory sets
- IELR(I,J,MN) : gives the time periods in which repairs take place for each option J for each furnace I.

- FH1: max. no. of furnaces of any one colour.
- PH1: max. no. of furnaces in any one factory.
- MN1 : maximum number of time periods in which any furnace is under repair for any option.

- BN: branching node
- BF: branching furnace
- BD: branching option.

- NCNT : the maximum node number that has been created so far
- NN1 : the number of the new 'rejecting' node - NN1 = NCNT + 1
- NN2 : the number of the new 'committing' node - NN2 = NCNT + 2

- COLBF : gives the colour of furnace BF
- FACTBF: gives the factory in which BF is in.

- COL(C,D) : gives the colour sets
- FACT(FG,PH): gives the factory sets

- COLBF : gives the colour of furnace BF
- FACTBF: gives the factory in which BF is in.
UNION(F) : gives the numbers of all the furnaces either of 
the same colour as BF or in the same factory as 
BF or both.

KI(K) : gives, for each time period K, the number of repairs 
that have taken place among furnaces of colour 1 
(white) in time period K.

CSTREP(I,J) : cost of repair schedules for each furnace 

IOCD(I) : gives the numbers of the committed options.

DATA FLOW

INPUT: -BN,BF,BD,MCNT via call statement 
       -I1,J1,K1,G,C1,D1,FG1,FH1,NN1 via COMMON 
       -IELR,COL,FACT via COMMON 
       -CSTREP and IOCD are read in for node BN from disc.

OUTPUT-The new CSTREP sets for the new nodes NN1, NN2 are 
written onto disc from within the subroutine.
PART 2: TO FIND COSTS OF REPAIRS
FOR NODE NNI - REJECTIONS NODE
c
CALCULATE NNI
NNI = NCNT + 1
c
READ IN COSTS OF REPAIR SCHEDULES
FOR NODE BN
c
SET COST OF OPTION 3D
FOR FURNACE BF TO 10^10

c
STORE COSTS OF SCHEDULES
FOR NODE NNI ON DISC
c
END
(4) **SUBROUTINE SALECONS(N, RESULT)**

This subroutine tests the given node \( N \) with the sales constraint. The result is passed into the main program through RESULT.

- RESULT = 0 \( \rightarrow \) NODE \( N \) FAILS
- RESULT = 1 \( \rightarrow \) NODE \( N \) Passes

**Variables:**

- \( N \): number of the node being tested
- \( S \): max. no. of nodes
- \( I1 \): number of furnaces
- \( J1 \): number of options per furnace
- \( K1 \): number of time periods
- \( C1 \): no. of colours
- \( FG1 \): no. of factories
- \( FH1 \): max. no. of furnaces in any one factory
- \( MN1 \): maximum number of time periods in which any furnace is under repair for any option.

**RESULT:**

\[ \pm 0 \] if \( N \) fails
\[ = 1 \] if \( N \) passes.

- **COL(C,D)**: gives the colour sets
- **ELPROD(I,J,K)**: gives the production in time period \( K \), in option \( J \), for furnace \( I \).
- **SALE(C,K)**: gives the sales demand for colour \( C \) in time period \( K \).
$A(C,K); B(C,K); I(A(C,K))$: are the weightings used in the sales constraint.

$CSTREP(I,J)$: gives the cost of option $J$ for furnace $I$.

$ICD(I)=0$ if $I$ is an uncommitted furnace, otherwise $I$ is a committed furnace.

Arrays generated and used in the subroutine:

$MAXEL(I,K)$: gives the maximum possible production in each time period $K$, for each furnace $I$.

$MAXPROD(C,K)$: gives the maximum possible production in each time period $K$, for each colour $C$.

$\text{MAXPROD}(C,K) = \sum_{I} \text{MAXEL}(I,K)$
where $I$ is of colour $C$

$\text{SUMPROD}(C,K)$: gives the maximum possible production in each time period $K$, for each colour $C$, summed over 3 consecutive time periods.

$\text{SUMPROD}(C,1) = \text{MAXEL}(C,1)$

$\text{SUMPROD}(C,2) = \text{MAXEL}(C,1) + \text{MAXEL}(C,2)$

$\text{SUMPROD}(C,K) = \text{MAXEL}(C,K-2) + \text{MAXEL}(C,K-1) + \text{MAXEL}(C,K)$
for any $K$ such that

$3 \leq K \leq K1$

$\text{SUMSL}(C,K)$: gives the sales demand similarly summed

$\text{SUMSL}(C,1) = \text{SALE}(C,1)$

$\text{SUMSL}(C,2) = \text{SALE}(C,1) + \text{SALE}(C,2)$

$\text{SUMSL}(C,K) = \text{SALE}(C,K-2) + \text{SALE}(C,K-1) + \text{SALE}(C,K)$
for any $K$ such that

$3 \leq K \leq K1$
ICON(C,K): results from attaching weights to the summed sales demands.

\[
ICON(C,K) = A(C,K) + B(C,K)ICON(C,K)
\]

ISUMPR(C,K): results from weighting the summed maximum possible production.

\[
ISUMPR(C,K) = IA(C,K)SUMPROD(C,K).
\]

DATA FLOW

INPUT: - Node number N via call statement.

- HELPROD, COL, SALE, A, B, IA via COMMON.
- \{ T1, J1, K1, S, J1, D1, FG1, PF1 \}

- OUTUP, IOD for node N are read off the disc from within the subroutine.

OUTPUT: - RESULT via call statement.

\[
\text{IF} \quad \text{ISUMPR}(C,K) \geq \text{ICON}(C,K) \quad \text{THEN} \quad \text{YES}
\]

\[
\text{IF} \quad \text{RESULT} = 0 \quad \text{THEN} \quad \text{NO}
\]

END

CONTINUE
FLOW CHART

READ IN DATA

FOR EACH FURNACE I, AND TIME PERIOD K, CALCULATE MAXEL(I,K)

FOR EACH COLOUR C, AND TIME PERIOD K
CALCULATE
1) MAXPROD(C,K)
2) SUMPROD(C,K)
3) SUMSL(C,K)
4) ISUMPR(C,K)
5) ICON(C,K)

FOR EACH COLOUR C, AND TIME PERIOD K, TEST THE NODE
RESULT=1
DO 10 C=1,C1
DO 10 K=1,K1

IF ISUMPR(C,K) .GE. ICON(C,K)

NO
RESULT=0

YES
10 CONTINUE

END
SUBROUTINE IFINDBN(NN,BN,STAGE,NISOL,IQ)

Given a node NN, that is infeasible for branching on in the current stage, IFINDBN selects the next branching node.

Variables

NN: as above

STAGE: 1 or 2

BN: gives the number of the branching node that is selected.

IQ: is the number of solutions in the retained set.

IBEST: Is an array which stores the node numbers of the solutions which have retained. In stage 1 when no solutions have been found, all entries in IBEST are (S + 1). The cost of the non-existent node (S + 1) is $10^{10}$.

PAR(N): gives the parent node of N.

W(N): gives the lower bound on costs for node N.

IR(N): = 0 if N fails the sales constraint.

= 1 if N passes the sales constraint.

NISOL:

(a) If IOPT = 1 (We want to find the best IQ of all solutions).

NISOL is the cost of the worst solution in array IBEST

(b) If IOPT = 0 (We want to find up to IQ solutions with the minimum cost and if there exists less than IQ of these, we fill up IBEST with the best of the solutions found up until the time at which the last Minimum cost solution was found).
NISOL is the cost of the best solution in array IBEST.

DATA FLOW

INPUT:  - NN, STAGE, NISOL, IQ via call statement
        - arrays PAR, M, IR, IBEST via common

OUTPUT: - BN via call statements.

(If BN = 1 then the program is finished)
Having found a new solution this subroutine reorders and retains the set of best solutions.

Variables

NN: is the node number of the new solution
IQ: number of solutions in the retained set
W(N): gives the lower bound on costs of solutions in node N. For a solution node this is the actual cost of the solution.
IBEST: is the set of retained solutions to be updated
IDUM: is a dummy set used in the transition.

DATA FLOW

INPUT: NN, IQ via call statement
W, IBEST via common blocks.

OUTPUT: array IBEST via common
FLOW CHART

READ IN DATA

SET ARRAY IDUM EQUAL TO ARRAY IBEST

C. SEARCH THROUGH IBEST UNTIL A SOLUTION IS FOUND WITH A GREATER COST THAN THE NEW SOLUTION. IF ONE IS NOT FOUND IBEST REMAINS UNCHANGED

I=1

II=IBEST(I)

W(II) .GT. W(NN)

REPLACE Ith BEST SOLUTION NODE B) NN AND RE-SORT THE REST

IBEST(I)=NN

J=I+1

IBEST(J)=IDUM(J)

J=J+1

IS J.EQ. IQ

END
Flow Chart

READ IN DATA

SET ARRAY IDUM EQUAL TO ARRAY IBEST

C. SEARCH THROUGH IBEST UNTIL A SOLUTION IS FOUND WITH A GREATER COST THAN THE NEW SOLUTION. IF ONE IS NOT FOUND IBEST REMAINS UNCHANGED

I=1

I= I+1

II= IBEST(I)

IS W(II) GT W(NN)

REPLACE Ith BEST SOLUTION NODE BY NN AND RE-SORT THE REST

IBEST(I)= NN

J= I+1

J= J+1

IBEST(J)= IDUM(J-1)

END

IS J.EQ. IQ

YES
CHAPTER V

How to use the program

INPUT

For each new problem there are two parts to setting up the program. Firstly some common blocks have to be set and secondly data cards need to be punched and arranged. Data for every problem must be compiled such that

1. Every furnace has an **EQUAL** number of options

2. Every repair schedule has an **EQUAL** number of time periods.

3. The costs of each schedule must either be
   
   (a) Lie between 0 and $10^5$ for a feasible option
   
   or (b) Equal $10^{10}$ for an infeasible option.

Case (b) can arise if you have a furnace that has only one possible option from the outset. All other options are then labelled as infeasible by giving them this cost.

1) COMMON BLOCKS

The dimensions of the arrays CSTREP and ELPROD must be specified exactly throughout the program this is to save storage space when using ELPROD and to enable easy use of disc storage for CSTREP.

For a problem with

- X Furnaces
- Y Options per furnace
- Z Time periods

The dimensions of the variables (all integer):
COMMON/A1/CSTREP(X,Y), must go at the beginning of the main program and subroutines MINGST, SELCTBD, CSTREP, SALEUNS.

COMMON/A5/ELFPROD(X,Y,Z) must go at the beginning of the main program and subroutine SALEUNS.

2) DATA CARDS

Punch and arrange the data cards in the following order.

(a) Parameter Card:

This reads in the variables (all integer):

I1,J1,K1,NN1,S,C1,D1,FH1,PH1,IP,T,RTO in format

(I2,1X,I2,1X,I2,1X,I2,1X,I2,1X,I2,1X,7(I2,1X) )

Limits on size of Variables:

Common blocks have been constructed to allow for the following ranges of these parameters. To enable these limits to be exceeded, the maximum dimensions of all arrays must be altered.

1 ≤ I1 ≤ 30
1 ≤ J1 ≤ 10
1 ≤ K1 ≤ 40
1 ≤ NN1 ≤ 5

1000 ≤ S ≤ 3000
C1 = 3
1 ≤ D1 ≤ 20
1 ≤ FH1 ≤ 8

1 ≤ PH1 ≤ 10
1 ≤ IQ ≤ 20
ICOPT = 0 or 1
IWRITE = 0 or 1

The size of S required depends on the problem. To increase S, the dimensions of PAR, S, IRT, and INDEX must be changed and the CALL OPMNS statement altered. A large S should not be put in for problems that do not need it because an increase in S results in a large increase in storage requirements.
b) **Colour Sets**

Three cards are required, one for each colour. Colour 1 corresponds to white flint, 2 to green, and 3 to amber.

On each card there are \( D_1 \) entries in \((I_2, I_X)\). Blank entries are filled with zeros.

E.g. 10 furnaces, 3 colours, a maximum of 4 furnaces of any one colour

\[ I_1 = 10, \quad C_1 = 3, \quad D_1 = 4 \]

Let the furnaces be arranged thus:

<table>
<thead>
<tr>
<th>Colour 1</th>
<th>1, 2, 7, 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colour 2</td>
<td>3, 4</td>
</tr>
<tr>
<td>Colour 3</td>
<td>6, 9, 9, 5</td>
</tr>
</tbody>
</table>

The 3 data cards are then:

```
 01 02 07 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

All starting in column 1.

**Factory Sets**

There are \( F_1 \) factories, we need one card for each factory. On each card there are \( F_1 \) entries in \((I_2, I_X)\), blank entries being filled with zeros.

E.g. 10 furnaces, 4 factories, \( F_1 = 3 \)

<table>
<thead>
<tr>
<th>FACTORY 1</th>
<th>1, 2, 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FACTORY 2</td>
<td>4</td>
</tr>
<tr>
<td>FACTORY 3</td>
<td>5, 6, 7</td>
</tr>
<tr>
<td>FACTORY 4</td>
<td>8, 9, 10</td>
</tr>
</tbody>
</table>
We would need 4 cards:

The production figures are in '000' ton units.

01 02 03
04 00 00
05 06 07 all starting in column 1
08 09 10

For each furnace we have a set of up to two data cards. Each set has J1 groups of M1 entries in (I2, 1x). Blank entries are filled with zeros.

For example, J1 = 6, M1 = 3

We have the following schedule options for a particular furnace.

0 0 1 0 0 0 1 0 0 1
0 0 0 1 0 0 0 1 0 0
0 0 0 0 1 0 0 0 1
1 0 0 0 1 0 0 0 1
0 1 0 0 0 1 0 0 0 1

The corresponding set of cards for this furnace would consist of 1 card: (a maximum of 25 entries on any card)

03 07 10 04 08 00 05 09 00 06 10 00 01 05 09 02 06

Production Figures

For each option for each furnace we have a set of up to 2 data cards with an entry in (I3, 1x) for each time period. If 2 cards are to be used the first must contain 20 such entries.
There must not be more than 20 entries on any card.

The production figures are in '100' ton units.

**COST OF REPAIR SCHEDULES:**

For each furnace we have a set of up to 2 cards with an entry in (I41, 1X) for each schedule option. No more than 5 such entries on any one card.

The cost units are '$10,000'. An infeasible option is represented by making its cost $10^{10}$.

**SALES DEMANDS:**

For each colour we have a set of up to 2 cards with an entry in (I3, 1X) for each time period. No more than 20 such entries on any one card.

Sales demands are in units of '1000' tons.

**SALES CONSTANTS:**

For each colour we have 3 groups of up to 2 cards in each group. Each group has K1 entries in (I3, 1X). There is a maximum of 20 such entries per card.

E.g. Let our weighting factors for each time period and each colour in the sales constraint be

$$10 \times \left( \text{TOTAL PRODUCTION summed over } 3 \text{ consecutive T.P.} \right) \leq 0 + 102 \left( \text{SALES DEMAND summed over } 3 \text{ consecutive T.P.} \right)$$

Where production figures are in '100' ton and sales demands in '1000' ton units.
Hence $IA(C, K) = 10$ for any colour $C$ or time period $K$.

Let there be 10 time periods. The data cards for each colour would then be:

<table>
<thead>
<tr>
<th>Card 1</th>
<th>Card 2</th>
<th>Card 3</th>
<th>Card 4</th>
<th>Card 5</th>
<th>Card 6</th>
<th>Card 7</th>
<th>Card 8</th>
<th>Card 9</th>
<th>Card 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>010</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>010</td>
</tr>
</tbody>
</table>

**INITIALLY COMMITTED FURNACES**

If a furnace, $I$, is committed the number of the committed option is entered in $ICD(I)$, otherwise $ICD(I) = 0$.

One card is required with up to 30 entries in (12).

Usually all furnaces will be initially uncommitted so this card will consist of zeros.

**To write out data only**

This is easily done by inserting the card

```
GO TO 2000
```

(4) Problem INFEASIBLE CANNOT FIND A-ECIA. NODE

This is printed out when subroutine IMPEAV selects node 1 as the next branching node in STAGE 1.

(e) PROBLEM FEASIBLE AND FINISHED.
OUTPUTING OUT.THE SOLUTION

The intermediate output and the retained set of solutions are defined by the options IOPT and IWRITE.

The following print outs can be obtained.

(a) INITIAL NODE HAS INFINITE COST
- this means that the lower bound on costs of solutions for the initial node is $>10^{10}$

(b) NODE 1 FAILS THE SALES CONSTRAINT
- Your initial sales demands cannot possibly be met by production

(c) NRMT is GE.(S-2) REDUCE NUMBER OF CHOICES
- The program has generated more than the maximum number of nodes. To reduce the number generated three things can be done.

For (a) Reduce the number of options per furnace

(b) Increase $S$ and the dimensions of arrays $KPAR,W,IN$ and INDEX and change the CALL OPENMS statement.

(c) Reduce sales demands if many nodes are failing the constraint.

(d) Problem INFEASIBLE CANNOT FIND A SOLN. NODE
- This is printed out when subroutine IFINDBN selects node 1 as the next branching node in STAGE 1.

(e) PROBLEM FEASIBLE AND FINISHED.
PRINTING OUT THE SOLUTION

For each solution the node number and cost are given, followed by the numbers of the committed options.

Each solution is then printed out in matrix form and this is followed by the production figures in each time period for each colour.

To give an indication of the safety margin between production and sales demand the array $\text{IFLOAT}(C,K)$ is printed out for each solution where:

$$\text{IFLOAT}(C,K) = \text{ISUMPR}(C,K) - \text{ICON}(C,K)$$

$$\text{ISUMPR}(C,K) = \text{IA}(C,K) \times \text{SUMPROD}(C,K)$$

$$\text{ICON}(C,K) = \text{A}(C,K) + B(C,K) \times \text{SUMSL}(C,K)$$

Notice that in general any change in production or sales for a particular time period will alter $\text{IFLOAT}(C,K)$ for 3 time periods.

If $\text{IOPT} = 0$ and there only exists one solution with the optimum cost and if this is found first the program will only find this solution.

(This is slightly modified for time periods 1 and 2).
Increasing the Sales Demands

For each retained solution we have the array IFLOAT(G,K). This indicates the 'safety margin' in the sales constraint. Exact increases that can be made in the sales demands are easily calculated. For the general time period K,

\[ \text{IFLOAT}(G,K) = 10 \times \text{production of colour } C \text{ by this solution, summed over T.P. } K, K-1, K-2 \]

If \( \text{IFLOAT}(G,K) \) becomes negative the solution becomes infeasible. Let the sales demand be increased by \( X \) for a time period \( K \) and a colour \( C \). Then

\[ \text{IFLOAT}(G,K) \]
\[ \text{IFLOAT}(G,K-1) \]
\[ \text{IFLOAT}(G,K-2) \]

will all decrease by \( 102X \).

(This is slightly modified for time periods 1 and 2).
Example 1

This is a very small problem with an output designed to illustrate the updating of the cost of repair schedules by subroutine GSTREPN. For each node the array CSTREP is printed out.

PARAMETERS

5 furnaces, 3 options per furnace, 5 time periods,

\( O_1 = 3, D_1 = 3, FG1 = 3, FH1 = 3, IO = 5, IOPT = 1, MN1 = 2 \)

\( S = 2000. \)

TIME

PERIPHERAL PROCESSORS 57.345 secs

CENTRAL PROCESSOR 8.339 secs

(for 25 nodes)

Loading time accounts for most of this

i.e. PP 41.297 secs

CP 7.629 secs

A diagram of the tree representing node generation follows.
INITIAL DATA FOR A PROBLEM WITH 5 FURNACES, 3 REPAIRS/FURNACE, 5 TIME PERIODS

3 COLOURS
MAX. NO. OF FURNACES OF ONE COLOUR 3

3 FACTORIES
MAX. NO. OF FURNACES IN ANY FACTORY 3

MAX. NO. OF REPAIRS IN ANY ONE OPTION = 2

NO. OF SOLUTIONS IN THE 'BEST' SET = 5
ILOPT DEFINES WHICH 'BEST' SET. IOPT = 1

COLOUR SETS

| COLOUR SET 1 | 1 | 2 | 3 |
| COLOUR SET 2 | 4 | 0 | 0 |
| COLOUR SET 3 | 5 | 0 | 0 |

FACTORY SETS

| FACTORY SET 1 | 1 | 4 | 0 |
| FACTORY SET 2 | 3 | 0 | 0 |
| FACTORY SET 3 | 5 | 2 | 0 |

REPAIR SCHEDULES

| FURNACE 1 | OPTION 1 | 5 | 0 |
| OPTION 2 | 1 | 0 |
| OPTION 3 | 4 | 0 |
| FURNACE 2 | OPTION 1 | 4 | 0 |
| OPTION 2 | 1 | 5 |
| OPTION 3 | 2 | 0 |
| FURNACE 3 | OPTION 1 | 5 | 0 |
| OPTION 2 | 1 | 4 |
| OPTION 3 | 3 | 0 |
| FURNACE 4 | OPTION 1 | 4 | 0 |
| OPTION 2 | 2 | 0 |
| OPTION 3 | 1 | 5 |
| FURNACE 5 | OPTION 1 | 5 | 0 |
| OPTION 2 | 3 | 0 |
| OPTION 3 | 1 | 5 |

COSTS OF REPAIR SCHEDULES

<p>| FURNACE 1 | 100 | 150 | 125 |
| FURNACE 2 | 80  | 70  | 80  |
| FURNACE 3 | 150 | 180 | 140 |
| FURNACE 4 | 180 | 180 | 200 |
| FURNACE 5 | 200 | 220 | 250 |</p>
<table>
<thead>
<tr>
<th>FURNACE 1</th>
<th>OPTION</th>
<th>1</th>
<th>10</th>
<th>15</th>
<th>15</th>
<th>12</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTION</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>12</td>
<td>12</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>OPTION</td>
<td>3</td>
<td>12</td>
<td>15</td>
<td>15</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>FURNACE 2</td>
<td>OPTION</td>
<td>1</td>
<td>7</td>
<td>10</td>
<td>8</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>OPTION</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>OPTION</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>FURNACE 3</td>
<td>OPTION</td>
<td>1</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>OPTION</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>OPTION</td>
<td>3</td>
<td>15</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>FURNACE 4</td>
<td>OPTION</td>
<td>1</td>
<td>20</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>OPTION</td>
<td>2</td>
<td>15</td>
<td>10</td>
<td>20</td>
<td>25</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>OPTION</td>
<td>3</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>FURNACE 5</td>
<td>OPTION</td>
<td>1</td>
<td>15</td>
<td>20</td>
<td>20</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>OPTION</td>
<td>2</td>
<td>20</td>
<td>15</td>
<td>5</td>
<td>15</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>OPTION</td>
<td>3</td>
<td>5</td>
<td>15</td>
<td>20</td>
<td>20</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**SALES DEMANDS**

| COLOUR 1 | 2 | 2 | 2 | 2 | 2 |
| COLOUR 2 | 2 | 2 | 2 | 2 | 2 |
| COLOUR 3 | 2 | 2 | 2 | 2 | 2 |

**SALES CONSTANTS**

<table>
<thead>
<tr>
<th>COLOUR 1</th>
<th>A(C,K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COLOUR 2</th>
<th>B(C,K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>102</td>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COLOUR 3</th>
<th>I(C,K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>

**INITIALLY COMMITTED FURNACES**

<table>
<thead>
<tr>
<th>NODE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOWER BOUND ON COSTS =</td>
<td>690 IR(N) = 1</td>
</tr>
</tbody>
</table>

**BRANCHING NODE**

| NODE | 1 | BR. FURN. | 1 | BR. DEC. | 1 |

**DATA FOR NEW NODE**

<table>
<thead>
<tr>
<th>NODE</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT NODE OF</td>
<td>2 IS 1</td>
</tr>
<tr>
<td>LOWER BOUND FOR NODE</td>
<td>2 EQUALS 715</td>
</tr>
<tr>
<td>FOR NODE 2 RESULT EQUALS 1</td>
<td></td>
</tr>
<tr>
<td>IGD(IF)</td>
<td>0</td>
</tr>
<tr>
<td>COST OF REPAIRS</td>
<td>1000000000</td>
</tr>
<tr>
<td></td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>200</td>
</tr>
</tbody>
</table>

**DATA FOR NEW NODE**

<table>
<thead>
<tr>
<th>NODE</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT NODE OF</td>
<td>3 IS 1</td>
</tr>
<tr>
<td>LOWER BOUND FOR NODE</td>
<td>3 EQUALS 690</td>
</tr>
</tbody>
</table>
**FOR NODE** 3 **RESULT EQUALS** 1

<table>
<thead>
<tr>
<th>ICD(i)</th>
<th>1 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COST OF REPAIRS</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>10000000000</td>
</tr>
<tr>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>150</td>
<td>180</td>
</tr>
<tr>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>200</td>
<td>220</td>
</tr>
</tbody>
</table>

**BRANCHING NODE** 3 **BR. FURN.** 5 **BR. DEC.** 1

**DATA FOR NEW NODE** 4

**PARENT NODE OF** 4 **IS** 3

**LOWER BOUND FOR NODE 4 EQUALS** 710

**FOR NODE 4 RESULT EQUALS** 1

<table>
<thead>
<tr>
<th>ICD(i)</th>
<th>1 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COST OF REPAIRS</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>10000000000</td>
</tr>
<tr>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>150</td>
<td>180</td>
</tr>
<tr>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>10000000000</td>
<td>220</td>
</tr>
</tbody>
</table>

**DATA FOR NEW NODE** 5

**PARENT NODE OF** 5 **IS** 3

**LOWER BOUND FOR NODE 5 EQUALS** 710

**FOR NODE 5 RESULT EQUALS** 1

<table>
<thead>
<tr>
<th>ICD(i)</th>
<th>1 0 0 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>COST OF REPAIRS</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>10000000000</td>
</tr>
<tr>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>150</td>
<td>180</td>
</tr>
<tr>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>200</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

**BRANCHING NODE** 5 **BR. FURN.** 3 **BR. DEC.** 3

**DATA FOR NEW NODE** 6

**PARENT NODE OF** 6 **IS** 5

**LOWER BOUND FOR NODE 6 EQUALS** 710

**FOR NODE 6 RESULT EQUALS** 1

<table>
<thead>
<tr>
<th>ICD(i)</th>
<th>1 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COST OF REPAIRS</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>10000000000</td>
</tr>
<tr>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>150</td>
<td>180</td>
</tr>
<tr>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>200</td>
<td>10000000000</td>
</tr>
</tbody>
</table>
PARENT NODE OF 7 IS 5.
LOWER BOUND FOR NODE 7 EQUALS 700.
FOR NODE 7 RESULT EQUALS 1.

ICD(ii) 1 0 3 0 1
COST OF REPAIRS

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>10000000000</th>
<th>10000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80</td>
<td>10000000000</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>10000000000</td>
<td>10000000000</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>180</td>
<td>180</td>
<td>10000000000</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

BRANCHING NODE 7 BR. FURN. 2 BR. DEC. 1

DATA FOR NEW NODE 8
PARENT NODE OF 8 IS 7.
LOWER BOUND FOR NODE 8 EQUALS 700.
FOR NODE 8 RESULT EQUALS 1.

ICD(ii) 1 0 3 0 1
COST OF REPAIRS

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>10000000000</th>
<th>10000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80</td>
<td>10000000000</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>10000000000</td>
<td>10000000000</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>180</td>
<td>180</td>
<td>10000000000</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

DATA FOR NEW NODE 9
PARENT NODE OF 9 IS 7.
LOWER BOUND FOR NODE 9 EQUALS 700.
FOR NODE 9 RESULT EQUALS 1.

ICD(ii) 1 1 3 0 1
COST OF REPAIRS

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>10000000000</th>
<th>10000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80</td>
<td>10000000000</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>10000000000</td>
<td>10000000000</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>180</td>
<td>180</td>
<td>10000000000</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

BRANCHING NODE 9 BR. FURN. 4 BR. DEC. 1

DATA FOR NEW NODE 10
PARENT NODE OF 10 IS 9.
LOWER BOUND FOR NODE 10 EQUALS 700.
FOR NODE 10 RESULT EQUALS 1.

ICD(ii) 1 1 3 0 1
COST OF REPAIRS

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>10000000000</th>
<th>10000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICD(1)</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>--------</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>COST OF REPAIRS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1000000000</td>
<td>1000000000</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>1000000000</td>
<td>1000000000</td>
<td></td>
</tr>
<tr>
<td>1000000000</td>
<td>1000000000</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>1000000000</td>
<td>1000000000</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>1000000000</td>
<td>1000000000</td>
<td></td>
</tr>
</tbody>
</table>

11 is a solution node.

<table>
<thead>
<tr>
<th>DATA FOR NEW NODE</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT NODE OF</td>
<td>12 IS</td>
</tr>
<tr>
<td>LOWER BOUND FOR NODE</td>
<td>12 EQUALS</td>
</tr>
<tr>
<td>FOR NODE</td>
<td>12 RESULT EQUALS</td>
</tr>
<tr>
<td>ICD(1)</td>
<td>1</td>
</tr>
<tr>
<td>COST OF REPAIRS</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1000000000</td>
</tr>
<tr>
<td>80</td>
<td>1000000000</td>
</tr>
<tr>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>200</td>
<td>1000000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA FOR NEW NODE</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT NODE OF</td>
<td>13 IS</td>
</tr>
<tr>
<td>LOWER BOUND FOR NODE</td>
<td>13 EQUALS</td>
</tr>
<tr>
<td>FOR NODE</td>
<td>13 RESULT EQUALS</td>
</tr>
<tr>
<td>ICD(1)</td>
<td>1</td>
</tr>
<tr>
<td>COST OF REPAIRS</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1000000000</td>
</tr>
<tr>
<td>80</td>
<td>1000000000</td>
</tr>
<tr>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>1000000000</td>
<td>180</td>
</tr>
<tr>
<td>200</td>
<td>1000000000</td>
</tr>
</tbody>
</table>

13 is a solution node.
DATA FOR NEW NODE 14

PARENT NODE OF 14 IS 8
LOWER BOUND FOR NODE 14 EQUALS 10000000620

FOR NODE 14 RESULT EQUALS 1

ICD(I) 1 0 3 0
COST OF REPAIRS

<table>
<thead>
<tr>
<th>100</th>
<th>10000000000</th>
<th>10000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000000</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
<tr>
<td>10000000000</td>
<td>10000000000</td>
<td>140</td>
</tr>
<tr>
<td>180</td>
<td>180</td>
<td>10000000000</td>
</tr>
<tr>
<td>200</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

DATA FOR NEW NODE 15

PARENT NODE OF 15 IS 8
LOWER BOUND FOR NODE 15 EQUALS 700

FOR NODE 15 RESULT EQUALS 1

ICD(I) 1 3 3 0
COST OF REPAIRS

<table>
<thead>
<tr>
<th>100</th>
<th>10000000000</th>
<th>10000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000000</td>
<td>10000000000</td>
<td>80</td>
</tr>
<tr>
<td>10000000000</td>
<td>10000000000</td>
<td>140</td>
</tr>
<tr>
<td>180</td>
<td>180</td>
<td>10000000000</td>
</tr>
<tr>
<td>200</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

BRANCHING NODE 15 BR. FURN. 4 BR. DEC. 1

DATA FOR NEW NODE 16

PARENT NODE OF 16 IS 15
LOWER BOUND FOR NODE 16 EQUALS 700

FOR NODE 16 RESULT EQUALS 1

ICD(I) 1 3 3 0
COST OF REPAIRS

<table>
<thead>
<tr>
<th>100</th>
<th>10000000000</th>
<th>10000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000000</td>
<td>10000000000</td>
<td>80</td>
</tr>
<tr>
<td>10000000000</td>
<td>10000000000</td>
<td>140</td>
</tr>
<tr>
<td>10000000000</td>
<td>180</td>
<td>10000000000</td>
</tr>
<tr>
<td>200</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

DATA FOR NEW NODE 17

PARENT NODE OF 17 IS 15
LOWER BOUND FOR NODE 17 EQUALS 700

FOR NODE 17 RESULT EQUALS 1

ICD(I) 1 3 3 1
COST OF REPAIRS

<table>
<thead>
<tr>
<th>100</th>
<th>10000000000</th>
<th>10000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000000</td>
<td>10000000000</td>
<td>80</td>
</tr>
<tr>
<td>10000000000</td>
<td>10000000000</td>
<td>140</td>
</tr>
<tr>
<td>180</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
<tr>
<td>200</td>
<td>10000000000</td>
<td>10000000000</td>
</tr>
</tbody>
</table>
DATA FOR NEW NODE 21

PARENT NODE OF 21 IS 6
LOWER BOUND FOR NODE 21 EQUALS 710
FOR NODE 21 RESULT EQUALS 1

ICD(I) 1 0 1 0
COST OF REPAIRS

<table>
<thead>
<tr>
<th></th>
<th>0000000000</th>
<th>0000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>80</td>
<td>0000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>150</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>180</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>200</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
</tbody>
</table>

BRANCHING NODE 21 BR. FURN. 2 BR. DEC. 1

DATA FOR NEW NODE 22

PARENT NODE OF 22 IS 21
LOWER BOUND FOR NODE 22 EQUALS 710
FOR NODE 22 RESULT EQUALS 1

ICD(I) 1 0 1 0
COST OF REPAIRS

<table>
<thead>
<tr>
<th></th>
<th>0000000000</th>
<th>0000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>1000000000</td>
<td>1000000000</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>180</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>200</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
</tbody>
</table>

DATA FOR NEW NODE 23

PARENT NODE OF 23 IS 21
LOWER BOUND FOR NODE 23 EQUALS 710
FOR NODE 23 RESULT EQUALS 1

ICD(I) 1 1 1 0
COST OF REPAIRS

<table>
<thead>
<tr>
<th></th>
<th>0000000000</th>
<th>0000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>1000000000</td>
<td>1000000000</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>180</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
<tr>
<td>200</td>
<td>1000000000</td>
<td>1000000000</td>
</tr>
</tbody>
</table>

BRANCHING NODE 23 BR. FURN. 4 BR. DEC. 1

DATA FOR NEW NODE 24

PARENT NODE OF 24 IS 23
LOWER BOUND FOR NODE 24 EQUALS 710
FOR NODE 24 RESULT EQUALS 1

ICD(I) 1 1 1 0
DATA FOR NEW NODE 25

PARENT NODE OF 25 IS 23
LOWER BOUND FOR NODE 25 EQUALS 710
FOR NODE 25 RESULT EQUALS 1

ICD(I) 1 1 1 1

COST OF REPAIRS

<table>
<thead>
<tr>
<th>COST</th>
<th>100</th>
<th>200</th>
<th>150</th>
<th>180</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000000</td>
<td>1000000000</td>
<td>1000000000</td>
<td>1000000000</td>
<td>1000000000</td>
<td></td>
</tr>
</tbody>
</table>

25 IS A SOLN. ODE

PROBLEM IS FEASIBLE AND FINISHED

1 TH BEST SOLN. GIVEN BY NODE 11 COST IS 700
COMMITTED DECISION FOR FURNACE 11S 1
COMMITTED DECISION FOR FURNACE 21S 1
COMMITTED DECISION FOR FURNACE 31S 3
COMMITTED DECISION FOR FURNACE 41S 1
COMMITTED DECISION FOR FURNACE 51S 1

2 TH BEST SOLN. GIVEN BY NODE 13 COST IS 700
COMMITTED DECISION FOR FURNACE 11S 1
COMMITTED DECISION FOR FURNACE 21S 1
COMMITTED DECISION FOR FURNACE 31S 3
COMMITTED DECISION FOR FURNACE 41S 2
COMMITTED DECISION FOR FURNACE 51S 1

3 TH BEST SOLN. GIVEN BY NODE 17 COST IS 700
COMMITTED DECISION FOR FURNACE 11S 1
COMMITTED DECISION FOR FURNACE 21S 3
COMMITTED DECISION FOR FURNACE 31S 3
COMMITTED DECISION FOR FURNACE 41S 1
COMMITTED DECISION FOR FURNACE 51S 1

4 TH BEST SOLN. GIVEN BY NODE 19 COST IS 700
COMMITTED DECISION FOR FURNACE 11S 1
COMMITTED DECISION FOR FURNACE 21S 3
COMMITTED DECISION FOR FURNACE 31S 3
COMMITTED DECISION FOR FURNACE 41S 2
COMMITTED DECISION FOR FURNACE 51S 1

5 TH BEST SOLN. GIVEN BY NODE 25 COST IS 710
COMMITTED DECISION FOR FURNACE 11S 1
COMMITTED DECISION FOR FURNACE 21S 1
COMMITTED DECISION FOR FURNACE 31S 1
COMMITTED DECISION FOR FURNACE 41S 1
COMMITTED DECISION FOR FURNACE 51S 1
Example 2 (Results in accompanying folder)

Again this is a small problem. It illustrates the print out given when IWRITO = 1. It is possible to work through the program by hand as only 35 nodes are generated.

PARAMETERS

5 furnaces, 3 options per furnace, 5 time periods
G1 = 3; D1 = 3; FG1 = 3; FH1 = 3; MN1 = 3; S = 3000; IQ = 5;
IOPT = 1; IWRITO = 1.

TIME

PERIPHERAL PROCESSORS 56.746 secs
CENTRAL PROCESSORS 9.149 secs

again the majority of time was taken in loading. The tree representing the solving of the problem, follows.
Example 2

Node Development
Example 3

This is a problem with 'real' data. There are 18 furnaces, 6 options per furnace and 28 time periods. Run under the present program, it was NOT finished in 380 secs and 7000 nodes. The program in its present form is rapidly becoming an infeasible proposition. All the solutions we found in the first 150 nodes while the remaining 6850 were unproductive. This unsatisfactory state of affairs was caused by the fact that most furnaces had several options of minimum cost, thus the lower bounds on the new nodes increase very slowly with branching. As can be seen, much fruitless branching is required before infeasible solutions are weeded out.

To speed up the program and reduce the storage the following method is used:

1. For the first 250 nodes proceed as before.

2. After this reduce NISOL by approximately 0.2% (approximation due to integer rounding and tends to ≥ 0.2%). In other words we only retain nodes which are potentially at least 0.2% better than our present value of NISOL (by cost).

This 0.2% works out at less than £40,000 out of nearly £18,000,000 and it is also spread over a seven year period. This margin of error can be easily altered. In the main
program there are two pairs of cards.

IF (I0PT.EQ.0) NISOL = (W(IP1) - W(IP1/500)

IF (I0PT.EQ.1) NISOL = (W(IP2) - W(IP2)/500)

This factor of 500 accounts for the .2% margin, increasing it will decrease the error and visa versa.

The problem was then solved in 27.040 secs, - a considerable reduction. The program, was run again with a margin of error of .5% (corresponding to a factor of 200). This was completed in 25.206 secs. (Giving the same solutions)

The size of the factor (i.e. error allowed) must be calculated with respect to the accuracy of the data and the range of the costs. Widely ranging costs might not even need it, while nearly equal cost figures may require more. But this can easily be tested.

To remove this margin completely remove the elements, W(IP1)/500 and W(IP2)/500 from the above cards.

NOTE: This method of decreasing time will NOT be effective if
(a) We cannot find an initial solution.
(b) I0PT = 1 and we cannot find IQ solutions.

This is the major factor effecting the speed of the program. Once a solution has been found the program rejects a note by comparing its lower bound on costs with the cost of the best solution found so far. The nearer this lower
Advantages and Disadvantages

As I have said before the general method is basically robust. The simplicity of the theory allows easy modification to the program, but the usefulness of the results is limited by the original sets of options. This limitation can be partially removed by running the program several times with different sets of data. The main concern is the speed of the program.

Running Time

The speed of the program is governed by two main factors.

(a) The Choice of the Branching Decision

The present method of choosing a branching decision by the maximum minimum penalty method does not take into account the fact that committing an option for one furnace might knock out cheap options of other furnaces. To occasionally this method of choice might force the program to investigate a potentially less profitable node.

(b) Construction of the lower and upper bounds on costs and production respectively.

This is the major factor effecting the speed of the program. Once a solution has been found the program rejects a node by comparing its lower bound on costs with the cost of the best solution found so far. The nearer this lower
bound is to the greatest lower bound the quicker
infeasibilities are rejected. Similarly for the sales
constraint the lower the upper bound the quicker the
rejection.

The nearer these bounds are to their limits the more
time is saved, since the program avoids much fruitless
searching. A better method of calculating a lower bound
on costs would cut computing time very considerably.

Extensions to the program

Although the basic model is a simplified version of real
life, many useful 'realities' can be incorporated.

(a) Representation of any sort of rebuild

In our present model we only consider normal rebuilds.
But these are only represented as a '1' in a particular
time period. There is no reason why any rebuild cannot be
represented in such a way. Two things must be remembered
when using this extension. First, the constraints will treat
all kinds of rebuilds as the same, hence they will forbid
minor repairs taking place simultaneously on more than one
furnace in any factory — second, there is a limit on the
maximum number of time periods in which any furnace can be
under rebuild in any time period. At present this limit is
5. To increase this, the dimensions of IELR must be altered.
(b) Opening and closing furnaces

This is easily done by giving a furnace 0 production in all time periods when it is closed and by entering zeros in its repair schedule options for these time periods.

(c) Changing the colour of a furnace

(i) When the change can only take place in one time period, k say.

To do this you must introduce a dummy furnace with the new colour. The old furnace is then closed and the dummy one opened at the beginning of timer period k.

To find the final result for this furnace you combine the schedules for the old and the dummy furnace.

(ii) When the change can take place in more than one time period.

These are two ways of approaching this problem. You can either run the program several times and for each run you have a different time period in which the change takes place (as in (i)).

The second way is similar to (i) but a constraint is added to ensure that if an option is committed for one of the pair of furnaces then only matching options remain feasible in the other furnace.

E.g. If for a particular node the original furnace is closed at the end of time period k then all options for the dummy furnace that do not represent an 'opening' at the beginning of time period (k+1) are made infeasible.
(d) **Unequal time periods — 'Focusing on Problem Periods'**

Initially all time periods were of equal length but this is not necessary as the program only deals in time periods and does not recognize their length. This enables one to put a problem period under a microscope by splitting it into many time periods.

Whenever using any of these extensions one must remember that the program deals in time periods not in years, also that their are limits on the parameters. These limits can be altered by altering the dimension of the arrays.

An analytic approach to the problem would have been ideal as long as the theories on furnace behaviour, used to construct the model, are correct.

With the introduction of furnaces with longer campaigns, gathering of data will not be interfered with by design changes every 3 or 4 years, the analytic approach should then become possible.