An experience in developing an expert system for scheduling problem in electric power system

Md. Sayeed Salam, Khalid M. Nor & Abdul Razak Hamdan
Electrical Engineering Department
University of Malaya
50603 Kuala Lumpur
Malaysia
Bangi,
Email: k3sayeed@cc.um.edu.my

Faculty of Information
Science & Technology, Universiti Kebangsaan Malaysia, 43600
Selangor, Malaysia.

Abstract

The mathematical programming based scheduling algorithms have shortcomings in handling constraints that cannot be expressed in terms of the required mathematical form. Furthermore experienced system operators are often able to tune data so that a schedule can be improved. This paper presents the development of an expert system that has been used as a guide to a mathematical programming based generation scheduling algorithm. The expert system is developed using the knowledge of experience operators and information from the mathematical programming. The expert system takes care constraints that are difficult or impractical to be implemented in a mathematical programming based scheduling algorithm such as cycling of gas turbine and steam turbine units, etc. Expert system is also used to check the feasibility of the solution. The experiences in developing the expert system in the aspects of knowledge acquisition, knowledge base development, inference engine and user interface is discussed in the paper. The developed expert has been integrated into two mathematical programming based algorithms, the dynamic programming and Lagrangian relaxation. Results of the expert system integrated with Lagrangian relaxation algorithm is presented. It was found that the expert system performed its function within acceptable execution time.
1 Introduction

The demand of electric power utilities varies against time periodically. In peak demand the number of generating units selected to supply the demand will be more than when the demand is at the minimum level. Therefore throughout a cyclic demand schedule such as in a day or in a week, plants may be scheduled to be disconnected during certain intervals or not used at all in certain days or run continuously. In order to operate electric power system with minimum cost, an optimum schedule of generating units to be in service to meet the time variant system demand subject to plant and system constraints needs to be determined.

In typical electric power systems, the two main type of generating plants are the thermal plants and the hydro plants. All types of generating units are capacity limited. In the case of hydro units, they are constrained by water availability. For the thermal plants, their schedule is influenced greatly by the boiler-turbine characteristics which impose loading limit and heat rate limit. The limits translate into time constraints such as minimum shutdown time in order to satisfy boiler loading limit. Such constraints force system operator to commit thermal plants before they are scheduled to be in service and hence this scheduling problem is usually called unit commitment. Due to the time variant constraints of the thermal plants as opposed to the hydro plant, thermal plants and hydro plants schedule are solved separately as sub-problems in a hydro thermal electric power system and then coordinated to get the complete results.

The most common mathematical programming techniques to solve the unit scheduling problem is the dynamic programming and Lagrangian relaxation. The techniques have also been extended to solve for the hydrothermal co-ordination [1-4]. An increase in the number of operational constraints causes the execution time of the mathematical programming based unit commitment program to increase exponentially [5]. To limit this excessive time requirement, most complex operating constraints are usually relaxed in the problem formulation. Also, these complex constraints are not often violated or inactive. Therefore, in some cases, the relaxation of complex constraints may lead to infeasible solutions and thus may call for further modification of the answer by enforcing constraints that are violated [6].

Mathematical programming techniques are able to only handle constraints expressable in certain mathematical forms. Therefore there are constraints that cannot be included in the problem formulation. There will be many instances where the commitment results will violate the constraints which are not included in the mathematical programming formulation. The commitment results arising from such a situation is infeasible although the mathematical programming technique has provided the right solution to the problem that it was presented with. This, obviously raise doubting questions from electric power system operators. However to respond to these queries requires some analysis and/or experience. From our own experience, we have noted that there are operators
who are not familiar with mathematical programming have expressed reservation about the practicality of such results.

Another aspect of unit commitment is that some of the constraints involved are heuristic in nature. These constraints may be adjusted or varied without making the system infeasible. In some cases, a small variation in these data often cause a significant change in the commitment result. Using the unit commitment program results as a starting point, experienced operators are frequently able to refine the results by adjusting heuristic data and input parameters.

As unit commitment programming experts and experienced power system operators are able to explain and modify commitment schedules by using heuristic knowledge developed from their past experience, an expert system which contains the knowledge of both the unit commitment programming expert and the experienced power system operator can be used as a post-processor to the mathematical programming based unit commitment program to check and modify commitment results. By iterating between expert and mathematical programming technique, an expert system may help to obtain an operationally feasible and/or preferable optimal schedule.

To meet the above requirements, an expert system has been developed by combining the knowledge of experienced power system operators and unit commitment experts. The developed expert system is used as a preprocessor as well as a postprocessor to the mathematical programming based results by adjusting the input data if necessary.

2 Integration of an expert system with a unit commitment algorithm

The approach that we have taken is to combine the knowledge of experienced power system operators and unit commitment experts into an expert system. The first task of an expert system is to analyze the commitment result. The expert system may then use its rules help to enforce the complex operating constraints external to the unit commitment program by tuning the input data. The system is therefore used as a preprocessor as well as a postprocessor to the mathematical programming algorithm to check and modify commitment results by adjusting the input data if necessary.

The mathematical program is used to generate a commitment schedule along with the dispatches of all generating units. The necessary input data and output data for the expert system knowledge base are then extracted and passed onto the expert system. The expert system will recommend resetting specific input data to the unit commitment program if the result is found operationally infeasible. The expert system recommendations are then passed to the operator for action. The operator will reset the input data and repeat the whole cycle until an operationally feasible and/or preferable solution is found. It is also
possible to automate the execution of the recommendation. The adopted approach is summarized in Figure 1.

Another feature of the expert system is that it can provide consultation sessions to answer an inexperienced operator's questions about the commitment pattern obtained by the program. For every recommendation made by the expert system, reasons can be given as to how the recommendation is derived. This increases the operator's confidence in the program and at the same time saves programmer's valuable time. This feature is very helpful if the operator is inexperienced or has yet to grasp the intricacy of mathematical technique.

Two mathematical programming techniques were developed. They are the dynamic programming and Lagrangian relaxation. The mathematical programming techniques were developed in the C language and the expert system was developed in SICStus Prolog [7]. This paper will concentrate on the expert system development.

3 Development of a unit commitment expert system
The structure of the proposed unit commitment expert system has been shown in Figure 2. This structure was selected because its knowledge base consists of data related knowledge, commitment result related knowledge and procedural knowledge. The driver was necessary for co-ordinating the functions of inference engine and user interface [8].

The following steps were taken for the development of the unit commitment expert system:

i) Operator and Programming Engineer Knowledge Acquisition

ii) Knowledge Base Development

iii) Expert System Shell Development

As the user and unit commitment domain experts were expected to be involved throughout the entire development process, the model shown in Figure 3 was adopted for the development of the unit commitment expert system.

3.1 Operator and programming engineer knowledge acquisition

An initial meeting was held for domain understanding. Experienced unit commitment operators of a practical utility were familiarized with the concept of a unit commitment expert system and with the respective responsibilities of an expert and the knowledge engineer. The operators presented a structured description of the unit commitment domain. A number of execution results of the unit commitment program were shown to them. A number of questions were asked by the operators about the feasibility of the results. Then they stated several general rules presently being used in the unit commitment of the utility. These general rules were further analyzed through introspection technique while the operators were being interviewed. All the general rules
were documented and restated to the operators in the next interview session. After several formal meetings, a number of general rules were finally extracted from the operators. A few of them are given below:

Gen. Rule a: In a day, if a gas turbine unit is switched off, then that unit should not be switched on again in that day, but other gas turbine units can be switched on.
Gen. Rule b: A gas turbine unit should not be committed on any day prior to a steam turbine unit.
Gen. Rule c: A steam unit should not be switched on after shutdown of the same or different steam units within a day.
Gen. Rule d: A gas turbine unit's load should not be varied more than once in every two hours.

The knowledge of both the unit commitment programming experts and the experienced power system operators was then analyzed, expanded and structured into a set of specific rules. This procedure developed several logical decision trees and the end node of each tree yielded a recommendation to an inexperienced operator.

3.2 Knowledge base development

In this work, knowledge was represented by production rules of the following form:
RuleName # if Condition then Conclusion.
where,
  RuleName   - Name that identifies the rule.
  Condition  - A set of simple assertions combined by operators AND, OR
                and NOT.
  Conclusion - Action to be taken by the production.

A portion of the Prolog version developed knowledge base is shown in Fig. 4.

```
rule1 # if
   not(gastr(_,_,switched_off,_,_))
then
   gastromok.

rule2 # if
   gastr(Unit,Plant,switched_off,Time,_) and
   gastr(Unit,Plant,switched_on,Time1,_) and
   with_in(Time1,Time,24) and
   'on scheduled outage'(Unit,Time2,_) and
   mid(Time2,Time,Time1)
then
   must_out(1,Unit,Time1).

rule3 # if
   gastr(Unit,Plant,switched_off,Time,_) and
   gastr(Unit,Plant,switched_on,Time1,_) and
   less_than(Time,Time1) and
   'on scheduled outage'(Unit,Time2,_) and
   not_mid(Time2,Time,Time1)
then
   not_outage(1,Unit,Time,Time1).
```

---

**Figure 4:** Partial example of a knowledge base for gas turbine unit's cycling

There are various sources of factual knowledge used in a unit commitment expert system. Factual knowledge can be given by the user, found from calculation, extracted from the fact, extracted from the data and results of the unit commitment program. Knowledge base partitioning helps to change the knowledge base easily and speed up the search process [9]. To obtain these
advantages, the whole unit commitment knowledge base was analyzed first and then was partitioned by using a completely separate knowledge base stored in separate files.

3.3 Expert system shell development

The development process of the shell was broken down into the following three sub-processes:
   i) Inference Engine Development
   ii) User Interface Development
   iii) Driver Development

3.3.1 Inference engine development

Input to the expert system gives a question to find out an answer. As the condition part of the production rule consists of AND/OR/NOT combinations of sub-conditions, an input question can also consist of AND/OR/NOT combinations of sub-questions. Also an answer to a question may come from various sources. It can be given by the user, found from calculation, extracted from the fact, extracted from the data and results of the unit commitment program. In order to tackle all these, the following principles were used to find an answer Answer to a question Goal:

- If Goal is found as a fact in the knowledge base then Answer is 'Goal is true as extracted from the fact'.
- If Goal is found from mathematical calculation then Answer is 'Goal is true as found from calculation'.
- If Goal is found from the data of the unit commitment program then Answer is 'Goal is true as extracted from the data'.
- If Goal is found from the result of the unit commitment program then Answer is 'Goal is true as extracted from the schedule'.
- If there exists a rule in the knowledge base of the following form

  RuleName # if Condition then Goal

  then Condition has to be explored in order to find answer Answer.
- If Goal is of the form 'Goal1 and Goal2' then Goal1 has to be explored first and then
  if Goal1 is false then Answer is 'Goal is false',
  else Goal2 has to be explored and answers to both Goal1 and Goal2 are to be combined properly to form Answer.
- If Goal is of the form 'Goal1 or Goal2' then Goal1 has to be explored first and then
  if Goal1 is true then Answer is 'Goal is true', or alternatively Goal2 has to be explored and answers to both Goal1 and Goal2 are to be combined properly to form Answer.
- If Goal is of the form 'not( Goal1 )' then Goal1 has to be explored first and then invert the answer to Goal1 to get Answer.
- If Goal is an askable question then user has to be asked for Answer [10].

In a production system, rules can be used in two ways i.e., in forward and backward chaining. In forward chaining, the system utilizes information on the left-hand side to get information on the right. Whereas in backward chaining, the system begins with what it wants to prove, e.g., Goal, and tries to establish the facts it requires to prove Goal. In the unit commitment expert system, it was necessary to establish the Goal in the form of a recommendation, and the system needed to start with the Goal. Use of backward chaining in that case was found to be more cost-effective, because only those rules relevant to establishing it would be executed [11].

The principles outlined above were implemented by developing procedure explore(Goal,Answer) which will search for an answer Answer to the question Goal. This procedure is given in the appendix.

3.3.2 User interface development
This was designed for two purposes. First, for asking the user to obtain information that is unavailable in the data and results of the unit commitment program. Secondly, for showing the result of a consultation session. Hence user interface was developed in two forms:

i) Input user interface
ii) Output user interface

3.3.2.1 Input user interface: The procedure useranswer(Goal,Answer) was developed for input user interface. It finds the solution Answer for a question Goal by asking the user. useranswer will first examine whether this question can be asked of the user. This type of question can be specified by one of the following two forms:

askable(Goal) if the question is of yes/no type and requires some value if the answer is yes.
askable2(Goal) if the question needs some value only.

Both types of question will be asked using window facility of Prolog.

To prevent repeated asking of the user about the same Goal and to properly employ the user supplied information at different places, the user's answer needs to be asserted as elements of a relation. In this shell, this was asserted as the following form:

wasgiven(Goal,TruthValue,Index).
where **Index** is a counter of user supplied answers and **TruthValue** is either true or false.

Another procedure `useranswer(Goal,Answer,N)` was used to generate solutions to **Goal** first by retrieving known solutions indexed from **N** onwards. If all known solutions are retrieved, the user is asked again about the **Goal** and newly supplied solutions are asserted using consecutive index numbers. If there are no more solutions from the user then this state is stored by asserting `end_answer(Goal)` If there are no solutions at all from the user, then this state is stored by asserting `wasgiven(Goal,false,Index)`.

### 3.3.2.2 Output user interface:
If the unit commitment expert system finds the answer, the user may wish to know the reasoning behind the conclusion reached. A suitable way to show this reasoning is to display an AND/OR solution tree comprised of the rule names and subgoals by properly indenting the subtrees.

The procedure `present(Answer)` was developed for output interface. It shows the result of a consultation session and then displays an option through a window. This option queries whether the user wishes to know the reasoning behind the conclusion or not. If the user wishes to know the reasoning, a derivation tree is displayed in a natural language and suitably indented form. The predicate `prom(Goal)` helps to show this derivation tree in natural language form.

### 3.3.3 Driver development
The procedure `unitcom` acts as a driver. It co-ordinates the functions of the three main modules i.e. `explore`, `useranswer` and `present`, of the unit commitment expert system shell and functions through a menu system. It displays a main menu showing the main problems i.e. unit commitment and unit loading. The user needs to choose one option by placing the cursor on that option and pressing the enter key. This main menu appears on screen as follows:

```
Check...
  Unit Commitment
  Unit Loading
  Exit
```

If the user chooses the **Unit Commitment** option, a submenu containing a list of various subproblems will be displayed on the screen as shown below:

```
Check...
Gas Turbine unit_s cycling
Commitment of Gas Turbine unit before Steam Turbine Unit
Steam Turbine unit_s cycling
```
Commitment for adequate voltage control
Commitment at plant 3
Commitment at plant 5
Exit

On the other hand, if the user chooses the *Unit Loading* option, a submenu containing a list of various subproblems will be displayed on the screen as shown below:

Check...
Gas Turbine unit's loadings
Largest loaded unit's loading
Loadings of units for Group Constraints at plant 1
Loadings of units for Group Constraints at plant 2 and plant A
Exit

The user has to choose one subproblem. This starts the execution and generates a recommendation for that subproblem.

All the probable recommendations for a particular subproblem are contained in a list in the knowledge base. The shell checks each of the recommendations from the list and finds a proper recommendation depending on the unit commitment result. Similarly all the probable subproblems for a particular problem are contained in a list in the knowledge base.

During the consultation session of a particular subproblem, only that file containing relevant production rules is consulted. At the end of the recommendation display, the file containing production rules relevant to that subproblem is deleted and the submenu is displayed again on the screen. This process speeds up the search procedure.

The shell can be integrated with the mathematical based program through C language. Alternatively it can be run as an interpreter. Then procedure driver will be invoked to start the consultation session. The window oriented menu system will guide the user in every step.

4 Experimental results

The developed programs have been tested on a utility system and were run on a SUN SPARCstation 10. The scheduling horizon is 24 hours. The test system has 12 hydro and 32 thermal generating units. The demand curves of Wednesday, Saturday and Sunday were used to test the capability of this approach. A summary of the system parameters is given in Table 1. For example, the solution process for the Sunday demand curve is shown below.

| Table 1: Summary of the power system |
The stepwise solution process is presented below:

**Step 1:** Using the Lagrangian relaxation based unit commitment program, a schedule is suggested whose cost of operation is denoted as 1.0 p.u. The CPU time for performing this unit commitment program is 40 seconds.

<table>
<thead>
<tr>
<th>System characteristics</th>
<th>Number of units</th>
<th>Total capacity or requirements (MW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steam units</td>
<td>25</td>
<td>3360</td>
</tr>
<tr>
<td>Gas turbine units</td>
<td>7</td>
<td>280</td>
</tr>
<tr>
<td>Hydro units</td>
<td>12</td>
<td>848</td>
</tr>
<tr>
<td>All units</td>
<td>44</td>
<td>4488</td>
</tr>
<tr>
<td>Peak demand</td>
<td></td>
<td>2198 - 2952</td>
</tr>
<tr>
<td>Minimum demand</td>
<td></td>
<td>1563 - 1658</td>
</tr>
<tr>
<td>Maximum reserve</td>
<td></td>
<td>166 - 284</td>
</tr>
</tbody>
</table>

The result was then analyzed using the expert system. The expert system gave a recommendation to keep thermal unit 3 ‘must run’ at hour 13. The expert system also gave a reasoning, stating violation rules of the steam turbine unit’s cycling which is shown in Figure 5.

**Step 2:** The information to keep thermal unit 3 ‘must run’ at hour 13 is incorporated in the data. The unit commitment program was run. The suggested schedule is shown in Table 2 which corresponds to an operating cost of 1.0002 p.u. In the schedule, units 1 to 25 correspond steam units and units 26 to 32
correspond gas turbine units. The CPU time for performing this unit commitment program is 40 seconds. The inclusion of the must run constraint causes the increase in total cost in this step over the previous step.

Then the result was analyzed using the expert system. In this case, the expert system found no problem in unit commitment and unit loading. Thus the expert system suggested that the schedule obtained in this step was operationally feasible.

Table 2: Commitment result of Sunday: Step 2

<table>
<thead>
<tr>
<th>Thermal units</th>
<th>Hydro units **</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000111111112222222222333</td>
<td>000000000111</td>
</tr>
<tr>
<td>12345678901234567890123456789012</td>
<td>123456789012</td>
</tr>
</tbody>
</table>

The expert system has guided the adjustment of the input data to the unit commitment program to obtain an operationally feasible solution. Each consultation of the subproblem with the expert system took between 5 and 15 seconds.

5 Conclusions
An expert system was used as a postprocessor to the Lagrangian relaxation based unit commitment program. Consultation session with the expert system was very fast. This is because all the relevant data and results of the unit commitment program were transferred to the expert system as knowledge base by the unit commitment program itself. Complex operating constraints, that are not violated frequently and are not computationally viable to be included in the unit commitment program, were enforced by the expert system. The expert system was found to be helpful in obtaining a feasible schedule within a reasonable time. The expert system also shows a reasoning behind a recommendation to increase operator’s confidence in the result.

6 References


7 Appendix

Procedure `explore(Goal,Answer)`
explore(Goal, Goal is true was 'as given by you'): -
  you @ Goal.
explore(Goal, Goal is true was 'as found from calculation'): -
  cal @ Goal.
explore(Goal, Goal is true was 'as extracted from the fact'): -
  fact $ Goal.
explore(Goal, Goal is true was 'as extracted from the data'): -
  data $ Goal.
explore(Goal, Goal is true was 'as extracted from the schedule'): -
  schd $ Goal.
explore(Goal, Goal is TruthValue was 'derived by' Rule from Answer): -
  Rule# if Condition then Goal,
  explore(Condition, Answer),
  truth(Answer, TruthValue).
explore(Goal1 and Goal2, Answer): -!
  explore(Goal1, Answer1),
  continue(Answer1, Goal1 and Goal2, Answer).
explore(Goal1 or Goal2, Answer): -
  exploreyes(Goal1, Answer);
  exploreyes(Goal2, Answer).
explore(Goal1 or Goal2, Answer1 and Answer2): -!
  not(exploreyes(Goal1, _)),
  not(exploreyes(Goal2, _)),
  explore(Goal2, Answer2),
  explore(not(Goal), Answer): -!
  ((explore(Goal, Answer1), !);
  explore1(Goal, Answer1)),
  invert(Answer1, Answer).
explore1(Goal, Goal is false was found).
invert(Goal is true was Found, not(Goal) is false was Found).
invert(Goal is false was Found, not(Goal) is true was Found).
explore(Goal, Goal is Answer was 'as given by you'): -
  useranswer(Goal, Answer).
exploreyes(Goal, Answer): -
  explore(Goal, Answer),
  positive(Answer).
continue(Answer1, _, and Goal2, Answer): -
  positive(Answer1),
  explore(Goal2, Answer2),
  (positive(Answer2), Answer = Answer1 and Answer2; negative(Answer2), Answer = Answer2),
  continue(Answer1, _, and _, Answer1): -

negative(Answer1).

truth(_ is TruthValue was _ , TruthValue):-!.

truth(Answer1 and Answer2, TruthValue):-
  truth(Answer1, true),
  truth(Answer2, true),!,
  TruthValue = true;
  TruthValue = false.

positive(Answer)::-truth(Answer, true).

negative(Answer)::-truth(Answer, false).