In: STeP 2000 – Millenium of Artificial Intelligence, Proceedings of the 9th Finnish AI Conference. Publications of the Finnish AI Society, No 16, 2000, pp. 175-179.

APPLICATION OF CONSTRAINT LOGIC SOLVING TO GENERATION REJECTION IN ELECTRICAL POWER UTILITY

Steve Chan[†] *and Eero Hyvönen**

[†]Hydro One Inc., Toronto, Canada, ylsteve.chan@ohsc.com
*University of Helsinki, Helsinki, Finland, eero.hyvonen@cs.helsinki.fi

When a high voltage transmission line is at fault condition, it is isolated from the network by the relaying protection system. In order to prevent excess flow in the remaining network, generation must be somehow rejected. This paper considers the problem of deciding optimal strategies of turning generators off. A formulation of the problem is presented and constraint logic solving is used to solve the problem. Initial test results are presented: all globally optimal solutions could be generated in a reasonable execution time.

1. INTRODUCTION

Many industrial problems involve making optimal choices between alternatives under constraints. The simplest way of solving such problems is to enumerate all scenarios and then to select the best one(s). However, this is not usually possible due to combinatorial explosion. Linear programming [Linear, 2000] is computationally fast but deals with only linear constraints and (usually) continuous values. Non-linear and mixed-integer programming algorithms [Nonlinear, 2000] can get stuck with local minima/maxima, the algorithms may not find any solution even when there is one, and even in the best case only one solution is found although there may be several equally optimal ones. In real world situations, the monetary cost of not finding a solution or not finding all equally optimal solutions can be large. Furthermore, traditional mathematical programming techniques and systems [Fourer et al., 1993] do not deal with logical and other non-arithmetic constraint that may be present in the problem.

As an alternative approach, the idea of constraint programming and solving has emerged in the fields of AI and logic programming [Marriot, Stuckey, 1998] and reliable or interval computing [Interval, 2000]. In this approach both arithmetic and non-arithmetic constraints can be incorporated and the reliability guarantee of finding *all global* solutions (within a user-given precision criterion) can be given [Van Hentenryck et al., 1997]. These techniques have usually been applied to continuous domains [Kearfott, 1996] but can be applied to discontinuous domains as well [Heipcke, 1999].

In this paper, the constraint scheme is applied to a generation rejection problem in an electrical power utility. In the following, the problem is first formulated. After this, an implementation for solving it is presented and test cases are presented. Finally, conclusions are drawn and directions of further research are suggested.

2. GENERATION REJECTION PROBLEM

2.1 Characterization of the problem

In an electrical power network where the power generation resources are concentrated in a remote location, high voltage transmission lines are used to transport the power from the generation location to the load area. To increase the reliability of this transportation system, two or more transmission lines are built on different geographic routes. Electrically, they are operating in parallel. When one of the lines is at fault condition, such as short-circuit or open-circuit, the relaying protection system will isolate that line from the network. The power flow, that has been on the faulty line before the fault occurs, will be distributed to the lines remaining in the network. This will increase the flow on the remaining lines and might cause two problems:

- The increase may cause the line flow exceeding its thermal limit. If this condition lasts long enough, the line will be overheated and eventually melt down.
- The increase may also cause the network to become unstable and finally collapse. This can occur in a short time and corrective action must be taken immediately after the fault is detected.

Here is a typical example: A main line (operating at 500 kV) and a secondary line (operating at 115 kV) are connected in parallel in transporting power from the generation location to the load center. When the main line is at fault, the increase in flow on the secondary line can cause instability in the generation area. The remedy is to disconnect (or reject) enough generation, immediately after the fault is detected, to ensure that the flow on the secondary line after the fault is within its stability limit.

This paper concerns the problem of selecting the right amount of generation for rejection. In some practical cases, the number of combinations of generation units that can be selected for rejection is very large. For example, with 22 generation units, the number of combinations is approximately 4.2 millions; it increases exponentially with the number of units.

There is a cost penalty for selecting a generation unit for rejection. This means that when selecting a generator combination for rejection the goal is to minimize its total penalty. Also, there are constraints on selecting a combination, which can be expressed in Boolean or inequality forms.

2.2 The object function to be minimized

Our goal will be to minimize a penalty function of a combination of generation units. There are several components in this function:

The site penalty

This penalty depends on individual generators, not on their combinations. Let $z_1, z_2, \ldots, z_i, \ldots, z_n$ denote the selection of generation units #1, #2,…,#i,…,#n for rejection, where $z_i = 1$ when the i:th generation unit is selected, and $z_i = 0$ when it is not selected. If $p_1, p_2, \ldots, p_i, \ldots, p_n$ denote the site penalties of generation units #1, #2,…,#n, then the total site penalty of a selected combination, sp, can be expressed as:

$$sp = \sum_i z_i \cdot p_i \tag{1}$$

The range penalty

The range penalty refers to the fact that the amount of generation selected may not be exactly equal to the flow of the main line. There is a range of generation amount for rejection within which the network can be maintained stable. This range is divided into three sections. When the selected generation amount is in the center section, there is no penalty, while if it is in the two end sections, a penalty proportional to the deviation from the flow of the main line is added to the object function.

Let $w_1,\dots,w_n$ denote the power generated by generation units #1,…,#n. Then the total generation amount selected for rejection, W, can be expressed as:

$$W = \sum_i z_i \cdot w_i \tag{2}$$

The range of generation amount is divided into the three sections below:

R_{low} to R_1	Low-end section
R_1 to R_2	Center section
R_2 to R_{high}	High-end section

Where

R_2 is the flow in the main line before the fault occurs and

$R_1 = R_2 - k_2$, $R_{low} = R_1 - k_1$, $R_{high} = R_2 + k_3$, and k_1, k_2 and k_3 are constants depending on the characteristics of the faulty line.

Let r_1 and r_2 be two Boolean variables representing the two end sections of R_{low} to R_1, and R_2 to R_{high} respectively:

$r_1 = 1$ if $W <= R_1$, and
$r_1 = 0$ otherwise.
$r_2 = 1$ if $W >= R_2$, and
$r_2 = 0$ otherwise.

The range penalty, rp, can then be expressed as

$$rp = r_1 \cdot (R_1 - W) + r_2 \cdot (W - R_2) \tag{3}$$

Figure 1 illustrates the situation.

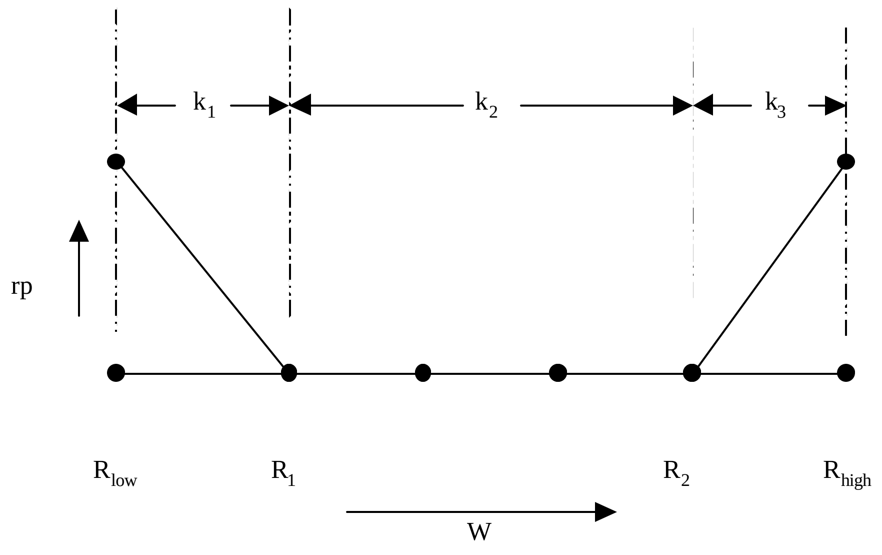

Figure 1. Illustration of range penalty rp as a function of rejected flow W.

Penalties associated with specific combinations

These penalties are due to undesired combinations of generation units. In our case there are two undesired combinations U_1 and U_2.

If generation units #8, #9, #19, and #20 are selected, add 7 to the object function. Let U_1 denote the undesired combination, then

$$U_1 = z_8 \wedge z_9 \wedge z_{19} \wedge z_{20}$$

and its penalty cp_1 can be expressed as:

$$cp_1 = 7 \cdot U_1 \tag{4}$$

The other undesired combination occurs if both units #3 and #4 are selected and both units #5 and #6 are not selected, or vise versa. In this case, penalty 5 is added to the object function. If U_2 denotes the undesired combination, then

$$U_2 = (\neg z_3 \wedge \neg z_4 \wedge z_5 \wedge z_6) \vee (z_3 \wedge z_4 \wedge \neg z_5 \wedge \neg z_6)$$

and its penalty can be expressed as:

$$cp_2 = 5 \cdot U_2 \tag{5}$$

The object function to be minimized is the sum of all penalty components of equations (1), (3), (4), and (5) and can be expressed as:

$$P = sp + rp + cp_1 + cp_2 \tag{6}$$

2.3 Constraints

The object function (6) to be minimized is subject to the following constraints:

Range constraint

The total generation amount selected, equation (2) in section 2.1, must be within the range between R_{low} and R_{high}, and can be expressed as

$$R_{low} \leq W \leq R_{high} \tag{7}$$

Reactive power constraint

The total reactive power remaining after rejection must be greater than a given minimum k_4. Let $v_1, v_2, \ldots, v_i, \ldots, v_n$ denote the reactive power remaining after rejection for units #1, #2,…,#i,…,#n. Then the reactive power constraint can be expressed as:

$$\sum_i v_i - \sum_i z_i \cdot v_i = \sum_i (v_i - z_i \cdot v_i) \geq k_4 \tag{8}$$

Combination constraint

Certain generator combinations may be impossible. In our case, units #16, #17, and #18 are mutually exclusive. This constraints can be expressed as:

$$(\neg z16 \wedge \neg z17) \vee (\neg z16 \wedge \neg z18) \vee (\neg z17 \wedge \neg z18) \tag{9}$$

2.2 Summary of the formulation

The problem can now be summarized as: Minimize (6), subject to constraints of (7), (8), and (9), by selecting sets of z_i's.

3. IMPLEMENTATION AND TESTING

Interval constraint satisfaction problem (ICSP) [Hyvönen, 1992] can be defined as a tuple <E, V> where E is a set of constraints and V is a value assignment for variable domains used in E. In our solution approach, the object function y=f(…) and the constraints of the generation rejection problem are seen as constraint expressions of an ICSP. Variable domain are either real or integer intervals and constraints are either logical or arithmetical constraints. The task is to find the solution(s) of E with minimal y-value.

An ICSP can be solved in the following way [Cleary, 1987; Van Hentenryck et al., 1997]. First, consistency algorithms can be used to refine variable domains by removing infeasible values not participating in possible configurations satisfying E within V. When all constraints have consistent domains, some non-exact domain is selected and forcibly split. Consistency refinements are then performed again, and so on. The process terminates either when the situation is found to be infeasible, or all domains are exact (down to given precision) and constraints are consistent, i.e., when a solution is found. By splitting domains exhaustively by a search algorithm, all solutions can eventually be generated. This basic technique applies to both discontinuous and continuous domains.

The ICE library [Hyvönen, De Pascale, 1995] implemented a consistency algorithm for continuous arithmetic equation and inequality constraints. For the generation rejection problem, this basis has been extended with the following extensions:

- A splitting algorithm for generating solutions.
- Logical constraint expressions can be used and intermingled with arithmetic ones.
- Discrete integer domains and integer constraints can be used.

The splitting algorithm generates a search space. Generation can be controlled by a few parameters: The choice of the next variable to be split, the way in which the domain is split (e.g., symmetric bisection), and the order in which the split domain parts are investigated.

When applying splitting to finding the minimum (or maximum) of a variable, a simple strategy is to select this variable first, instantiate it to its lower (or higher) part, apply consistency algorithm, split the variable again etc. If a situation is found infeasible, then that part of the search tree can be rejected. If a solution is found, then an upper bound for the minimum (or lower bound for the maximum) is found and situations involving values above (below) the bound can be cut off.

Our hypothesis is that this approach is likely to be efficient with non-linear problems whose feasibility area is not large because then consistency algorithms are likely to converge rapidly and the cut-off mechanisms become more effective. The generation rejection problem seems to have such characteristics.

Testing scenarios

We tested the search scheme in finding the optimal solutions, sets of z_i's, for object function (6) subject to constraints (7), (8) and (9). A Pentium 350MHz/64MB PC and Windows NT were used.

Values for the following input variables and constants define a scenario that can be solved: w_i's, p_i's, v_i's for each generator, the pre-fault flow R_2 on the main line, and the constants k_1, k_2, k_3, and k_4. For each scenario, a solution is a set of z_i's, the optimal combination of units, and the total power for rejection (W), assuming that the constraints (7), (8) and (9) are satisfied.

Four tests were conducted using the pre-fault flows of table 1. Table 2 lists the other parameters used in the tests.

Test #	Flow R_2 (MW)
1	150
2	300
3	450
4	600

Table 1. Test scenarios for determining optimal generation rejection.

	Generator i																					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
w_i	62	61	63	64	65	65	58	54	52	53	55	47	45	44	42	40	21	101	46	38	20	54
v_i	11	11	11	11	11	11	11	0	0	11	11	11	11	11	11	0	0	0	0	0	0	0
p_i	-1	0	5	10	15	-2	0	5	10	15	-2	0	5	10	15	-2	0	5	10	15	-2	0
k_1	10																					
k_2	30																					
k_3	10																					
k_4	20																					

Table 2. Common variable values for the test scenarios of table 1.

With a proper adjustment of the parameters in the search engine, the solutions listed in table 3 were obtained. Here Min refers to the objective function minimum. Notice that in tests 1-3 multiple optimal configurations could be found. Column Time lists the total times needed in finding all solutions and proving that there are no other ones.

Test #	R_2 (MW)	Solution #	Generator selection	W	Min	Time (s)
1	150	1	1, 11, 21	137	-6	19.5
		2	6, 11, 21	140		
		3	1, 6, 21	147		
		4	1, 16, 21	122		
		5	6, 16, 21	125		
2	300	1	1, 6, 11, 12, 16, 21	289	-10	14.7
		2	1, 6, 11, 16, 21, 22	296		
3	450	1	1, 2, 6, 7, 11, 12, 21, 22	422	-8	18.0
		2	1, 2, 6, 7, 11, 12, 16, 22	442		
		3	1, 2, 6, 7, 11, 12, 17, 21, 22	443		
4	600	1	1, 2, 3, 6, 7, 8, 11, 12, 16, 21, 22	579	0	174.3

Table3. Optimal generation selections for the test scenarios of tables 1-2.

4. DISCUSSION

The results obtained in the experiment are promising. Multiple solutions could be identified and the guarantee for global optimum can be given. Logical and integer constraints and domains could be used in the model without reformulating them in some arithmetical form.

The execution times of table 3 are a little longer than the desired goal (less than 10s) set beforehand. However, with a 100% faster 700MHz CPU, test cases 1-3 could be solved in 10s without further algorithm/code optimization or problem reformulation. In the worst case (test 4), the desired order of execution time would not be very far away.

Parameter tuning of the search procedure was needed in order find the solutions effectively. Further research is needed in order to automate such tuning based on problem characteristics, if the algorithm is used for solving other kind of problems. Planned future work includes comparizon of the interval solving approach with traditional mixed integer solvers, such as CPLEX and XLSOLVE [Fourer et al., 1993]. Problems such as the generation rejection problem of this paper are challenging to traditional optimization techniques whose iterative algorithms cannot easily make use of the discrete value domains and constraints. Furthermore, with these systems at most one solution can be found while with interval techniques the guarantee for finding all solutions can be given.

ACKNOWLEDGEMENTS

Thanks to Patrick Anam for fruitful comments and help in finishing the paper.

REFERENCES

[Cleary, 1987] Cleary, J., Logical arithmetic. Future Computing Systems 2 (2), 1987.

[Fourer et al., 1993] Fourer, R., Gay, D., Kernighan, B., AMPL. A modeling language for mathematical programming. Boyd & Freaser Publ. Company, 1993.

[Hansen, 1992] Hansen, E., Global optimization using interval analysis. Marcel Dekker, New York.

[Heipcke, 1999] Heipcke, S., An example of integrating constraint programming and mathematical programming. Research report, University of Buckingham, School of Business, Buckingham, U.K., 1999.

[Hyvönen, 1992] Hyvönen, E., Constraint reasoning based on interval arithmetic: the tolerance propagation approach. Artificial Intelligence 58, 71-112.

[Hyvönen, De Pascale, 1995] Hyvönen, E., De Pascale, S., InC++ library family for interval computations. International Journal of Reliable Computing, supplement, Proceedings of Applications of Interval Computations, El Paso, Texas, USA, 1995.

[Interval, 2000] Home page of interval computations research: http://cs.utep.edu/interval-comp/main.html.

[Kearfott, 1996] Kearfott, B., Rigorous global search: Continuous problems. Kluwer, New York, 1996.

[Linear, 2000] Linear programming FAQ:
http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming.

[Marriot, Stuckey, 1998] Marriot, K., Stuckey, P., Programming with constraints. An introduction. The MIT Press, 1998.

[Nonlinear, 2000] Nonlinear programming FAQ:
http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming.

[Van Hentenryck et al., 1997] Van Hentenryck, P., Michel, L., Deville, Y., Numerica. A Modeling Language for Global Optimization. MIT Press, Cambridge, 1997.