A DECOMPOSITION ALGORITHM FOR TRADING ELECTRICITY BETWEEN POWER COMPANIES

by

Diana I. Guettlein

A thesis submitted to the Faculty and Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado

Date 4/1/96

Signed: Diana Guettlein
Diana I. Guettlein

Signed: Robert Underwood
Dr. Robert Underwood
Associate Professor of Mathematics

Golden, Colorado

Date 4/1/96

Graeme Fairweather
Dr. Graeme Fairweather
Head of Mathematical and
Computer Sciences Department

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank Professor Robert Underwood of the Colorado School of Mines for being my advisor on this thesis.

DEDICATION

I would like to dedicate this thesis to my loving husband, Victor, who encouraged me to complete this endeavor.

Chapter 1

INTRODUCTION

The United States is divided up into a power grid with each power company being responsible for supplying electricity to all the consumers within that region. Each power company is autonomous and, therefore, isolated from the other power companies. In general, electricity is produced in the region wherein it is consumed. The two primary methods of meeting increasing demand are to expand the capacity of an established plant, or build another plant. When a new plant is built, it will have a large amount of excess capacity to meet the increasing demand over the next 20-30 years. Ideally, that power company would like to sell the excess electricity to another region. Due to the large capital investment required to build a new plant, it might be advantageous for a different power company to postpone building and import energy from another power company. The trading and its effect on the expansion of energy production will constitute our focus.

The question becomes, what is the best way to model the forementioned problem? An example of such a problem which we will model contains seven variables. Four of these variables vary over four regions, sixty time periods, and thirteen different plant types. In addition, one of the variables is a binary variable which further increases the difficulty and time in solving the problem. For these reasons we have choosen to use a decomposition algorithm. The decomposition algorithm is a method of solving very large linear programming problems. The method consists of breaking the linear programming problem into smaller, more manageable subproblems that are easily solved. This is accomplished by dividing the constraints into two categories: general constraints (or complicating constraints) and special constraints. The special constraints are grouped together into the subproblems that are readily solved. In our energy problem, the natural grouping of constraints is by region. Each region must meet its own demand constraints and generation constraints. These constraints do not have any effect on the other regions. The trading constraint, however, does affect each of the regions simultaneously, and is therefore, the complicating constraint.

In the following chapers we describe (1) the Simplex algorithm, (2) the decomposition algorithm, (3) the model, (4) the heuristic algorithm, (5) the conclusion.

The model is solved using the student version of AMPL software with CPLEX solver. It limits the total number of variables to three hundred and the total number of constraints to three hundred. The problem is scaled down to meet these requirements, however it should have no effect on the model itself.

Chapter 2

A REVIEW OF THE SIMPLEX ALGORITHM

The decomposition algorithm is a based on the simplex algorithm. A quick review of the simplex algorithm is necessary before we discuss the decomposition algorithm in detail. An L.P. is defined as follows.

Minimize: $$z = c^T x \tag{2.1}$$

Subject to: $$Ax = b \tag{2.2}$$

$$x \geq 0$$

where A is an m by n matrix, $x \in R^n$, $b \in R^m$. To start, assume we can partition c, A, and x into basic and nonbasic components as described below. A basic solution consists of a selection of m variables x_B, and n-m variables x_N such that

$$Bx_B + Nx_N = b. \tag{2.3}$$

where $A = [B / N]$ with $B \in R^{mxm}$ and $N \in R^{mxn-m}$ and B nonsingular. With this configuration, it is possible to solve (2.3) by expressing x_B, in terms of x_N.

$$x_B = B^{-1}b - B^{-1}Nx_N \quad . \tag{2.4}$$

The objective function (2.1) can now be redefined in terms of its basic and nonbasic variables,

$$z = c_B^T x_B + c_N^T x_N. \tag{2.5}$$

Substituting equation (2.4) into objective function (2.5), we obtain

$$z = c_B^T\left(B^{-1}b - B^{-1}Nx_N\right) + c_N^T x_N\,,$$

$$z = c_B^T B^{-1}b - c_B^T B^{-1}Nx_N + c_N^T x_N, \tag{2.6}$$

and

$$z = c_B^T B^{-1}b - \left(c_B^T B^{-1}N - c_N^T\right)x_N\,.$$

Let $r_N^T = c_B^T B^{-1}N - c_N^T$, and let $w^T = c_B^T B^{-1}$.

Substituting w^T and r_N^T into equation (2.6) we obtain the final form

$$z = w^T b - r_N^T x_N\,. \tag{2.7}$$

If some $r_{N_i} > 0$, we can decrease z by increasing x_{N_i}. Eventually the increase x_{N_i} is blocked by other constraints otherwise the problem is unbounded. Using this logic, the simplex algorithm moves from one basic solution to another. In that there are a finite number of basic solutions, the algorithm must terminate.

Chapter 3

THE DECOMPOSITION ALGORITHM

3.1 A Brief Overview

We will now describe the decomposition algorithm that was first studied by Philip Wolfe and George B. Dantzig. The decomposition algorithm is a method of solving very large linear programs that have a special structure. This special structure allows the problem to be broken up into "smaller linear programs whose iterated solutions solve the given problem" (Dantiz and Wolfe 160). The linear program may be described as composed of separate linear programming problems tied together by a few constraints. A diagram of the original problem is given in Figure 1. The constraint matrix has been partitioned into nonzero blocks $A_j + B_j$, with b, $b_1 \ldots b_n$ as the right hand side. $x_1 \ldots x_n$ are the variables and $c_1 \ldots c_n$ are the costs associated with each variable.

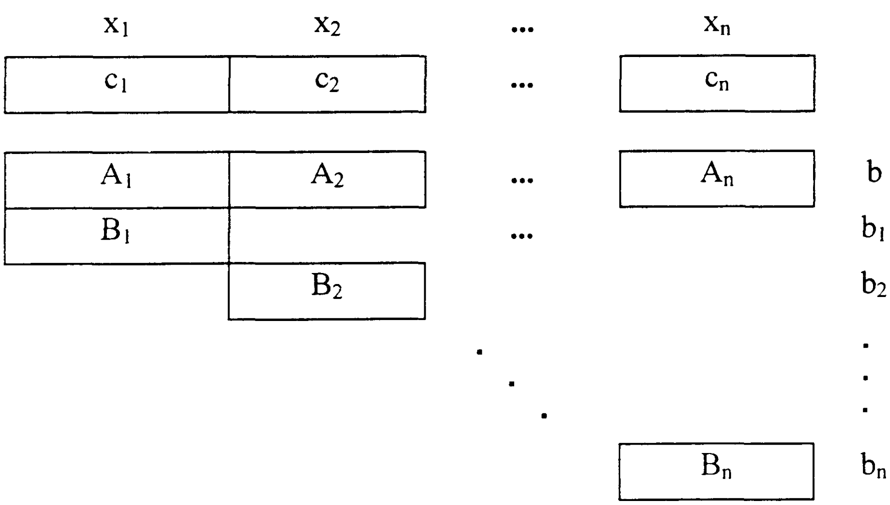

Figure 1. Structure of the Original Problem

For example, a problem with four variables and six constraints may have two constraints that use all four variables. The next four constraints are grouped so that they can be solved independently. No variables from one subproblem interact with variables in any other subproblem.

3.2 A Detailed Description of the Decomposition Algorithm

We will now describe the decomposition algorithm in detail. In matrix notation, the linear program is defined as follows:

Minimize: $z = c^T x$

Subject to: $Ax = b$ (3.1)

$x \in X$

Where A is an m x n matrix and b is an m vector and X is a convex polyhedral set representing constraints of special structure. The constraints $Ax=b$ will be referred to as the complicating constraints. We can represent $x \in X$ as a convex combination of the finite number of extreme points of X. The extreme points of X will be found by solving the decoupled subproblems. The individual points x_1, x_2,...x_t for $x \in X$ can be represented as follows:

$$x = \sum_{j=1}^{t} \lambda_j x_j \tag{3.2}$$

$$\sum_{j=1}^{t} \lambda_j = 1 \qquad \lambda_j \geq 0 \qquad j = 1,2,\ldots,t$$

Substituting for x into equation (3.1), we obtain the following "Master Problem":

Minimize: $$z = \sum_{j=1}^{t} \left(c^T x_j\right) \lambda_j$$

Subject to: $$\sum_{j=1}^{t} \left(A x_j\right) \lambda_j = b \tag{3.3}$$

$$\sum_{j=1}^{t} \lambda_j = 1 \qquad \lambda_j \geq 0 \qquad j = 1,2,\ldots,t$$

Finding the solution of the "Master Problem" is equivalent to finding the solution to the original problem.

Now, we will perform a change of variables.

Let $\hat{c}_j = c^T x_j$

Let $\hat{b} = \begin{bmatrix} b \\ 1 \end{bmatrix}$

Let $\hat{A} = \begin{bmatrix} Ax_1 \ldots Ax_t \\ 1 \ldots\ldots\ldots\ldots 1 \end{bmatrix}$

Rewriting the "Master Problem" (3.3) in terms of the new variables we obtain

Minimize: $$z = \hat{c}^T \lambda$$

Subject to: $$\hat{A}\lambda = \hat{b} \tag{3.4}$$

$$\sum_{j=1}^{t} \lambda_j = 1 \qquad \lambda_j \geq 0 \qquad j = 1,2,\ldots,t$$

where our decision variable is λ.

Applying the simplex algorithm to the master problem (3.4) we obtain:

$$z = \hat{c}_B^T \lambda_B + \hat{c}_N^T \lambda_N , \tag{3.5}$$

$$\hat{B}\lambda_B + \hat{N}\lambda_N = \hat{b}, \tag{3.6}$$

$$\lambda_B = \hat{B}^{-1}b - \hat{B}^{-1}\hat{N}\lambda_N , \tag{3.7}$$

$$z = \hat{c}_B^T\left(\hat{B}^{-1}\hat{b}\right) + \left(\hat{c}_N^T - \hat{c}_B^T\hat{B}^{-1}\hat{N}\right)\hat{\lambda}_N . \tag{3.8}$$

Denote

$$\hat{r}_N^T = \hat{c}_B^T \hat{B}^{-1}\hat{N} - \hat{c}_N^T . \tag{3.9}$$

Substituting $\hat{r}_N^T$ from (3.9) into the objective equation (3.8) we obtain

$$z = \hat{c}_B^T\left(\hat{B}^{-1}\hat{b}\right) - \hat{r}_N^T\hat{\lambda}_N . \tag{3.10}$$

We define the following:

$$\hat{r}_B^T = \hat{c}_B^T\hat{B}^{-1}\hat{B} - \hat{c}_B^T \ , \tag{3.11}$$

$$\hat{r} = \left(\hat{r}_B, \hat{r}_N\right), \tag{3.12}$$

and observe that

$$\hat{r} = \hat{c}_B^T\hat{B}^{-1}\hat{A} - \hat{c}^T . \tag{3.13}$$

Since, $\hat{r}_B^T = 0$, we can replace $\hat{r}_N^T$ in equation (3.10) with $\hat{r}^T$ to obtain

$$z = \hat{c}_B^T\left(\hat{B}^{-1}\hat{b}\right) - \hat{r}^T\hat{\lambda}_N . \tag{3.14}$$

We will adhere to the following strategy. Select the pivot column j* according to

$$\hat{r}_{j^*} = \max_j\left\{\hat{r}_j\right\} \tag{3.15}$$

This strategy corresponds to choosing the column which will provide the largest local decrease in z.

Maximizing $\hat{r}$ is accomplished in a separate linear programming problem which we refer to as the subproblem. From (3.13) we see that maximizing $\hat{r}$ corresponds to

$$\underset{j}{\text{Maximize}} \quad \hat{c}_B^T \hat{B}^{-1} \hat{A}_j - \hat{c}_j^T . \tag{3.16}$$

Let

$$\hat{c}_B^T \hat{B}^{-1} = (w^T \; \alpha) \tag{3.17}$$

where w^T corresponds to the first m-1 rows and α to the last row. It should be noted the w^T is the dual for " Ax_j " constraints and α is the dual for the λ constraint.

Recall that

$$\hat{A} = \begin{bmatrix} Ax_j \\ 1 \end{bmatrix} \tag{3.18}$$

for the columns, j=1,...,t.

Substituting $(w^T \; \alpha)$ from equation (3.17) and the description of $\hat{A}$ given by (3.18) into equation (3.16), we obtain

$$\underset{j}{\text{Maximize}}: \qquad \left(w^T \alpha\right) \begin{bmatrix} Ax_j \\ 1 \end{bmatrix} - c^T x_j . \tag{3.19}$$

Simplifying equation (3.19) we obtain

$$\underset{j}{\text{Maximize}}: \qquad \left(w^{T} A x_{j} + \alpha - c^{T} x_{j}\right), \tag{3.20}$$

Subject to: $\qquad x \in X$,

where x_j is an extreme point of X.

The subproblem consists of solving (3.20) given w^T and α. Having found a new extreme point x_{j*} for these values of w^T and α, a new column associated with this extreme point, $\begin{bmatrix} r_{j*} \\ Ax_j \\ 1 \end{bmatrix}$, is added to the master problem, which is then solved using the simplex method. New values for w^T and α are generated. These values are passed back to the subproblem. The subproblem and master problem are then resolved alternately until the optimal value to the master problem is found.

The test for optimality of the master problem, and thus the overall problem, is performed in the subproblem. Let x_k be the optimal solution to this subproblem with the objective value r_k. If $r_k > 0$, then the master problem is not optimal and the variable λ_k enters the basis in the master problem. If $r_k \le 0$, then the basic feasible solution (λ_B, λ_N) of the master problem is optimal.

It should be noted that the interim solution will not be any worse than the preceding solution because a new extreme point is added with each iteration. A second observation is that an interim solution may approach the optimal solution very rapidly, while, the progress made for each continuing iteration may be extremely small. Once that point is reached, there is a diminishing return in terms of CPU time expended to improve the solution. An exit routine may be included that tests the difference between the solution and the lower bound to see if it is within a specified error tolerance. A discussion of the lower bound follows.

3.3 The Lower Bound

We will now derive a lower bound.

The subproblem is defined as:

Maximize: $\left(w^T A - c^T\right)x + \alpha$,

Subject to: $x \in X$.

Let the optimal objective be $\hat{r}_j$. Let x be any feasible solution to the whole problem with Ax=b and $x \in X$. We obtain

$$\left(w^T A - c^T\right)x + \alpha \le \hat{r}_j, \tag{3.21}$$

$$w^T Ax - c^T x + \alpha \le \hat{r}_j,$$

$$-c^T x \le -w^T Ax - \alpha + \hat{r}_j,$$

$$c^T x \ge w^T Ax + \alpha - \hat{r}_j.$$

Now we substitute for Ax=b into the equation

$$c^T x \ge w^T b + \alpha - \hat{r}_j. \tag{3.22}$$

Substituting equation (3.17) into equation (3.22) we obtain

$$c^T x \ge \hat{c}_B^T B^{-1}\hat{b} - \hat{r}_j \tag{3.23}$$

Since the expression (3.23) is true for each $x \in X$ and $Ax=b$, we will evaluate the expression at its minimum.

Minimum: $c^T x \geq \hat{c}_B^T B^{-1}\hat{b} - \hat{r}_j$ (3.24)

When: $Ax=b$ and $x \in X$

After you have taken the minimum of equation (3.24), the value of the objective function $c^T x$ is greater than or equal to $\hat{c}_B^T B^{-1}\hat{b} - \hat{r}_j$. Therefore, $\hat{c}_B^T B^{-1}\hat{b} - \hat{r}_j$ is the lower bound on the overall objective function, where $\hat{r}_j$ is the value of subproblem objective function. Note that $\hat{c}_B^T B^{-1}\hat{b}$ is the upper bound since we obtain a nonincreasing objective function.

3.4 Example

Now we will do a small example to illustrate the decomposition algorithm and also to look at the lower bounds of the optimal solution.

The L.P. is defined as:

$$
\begin{array}{llllll}
\text{Minimize:} & -4x_1 - 2x_2 & - 2x_3 & - x_4 & & \\
\text{Subject To:} & 2x_1 + & x_3 & & \leq & 36 \\
& x_1 + 3x_2 + & & 2x_4 & \leq & 22 \\
& 2x_1 & & & \leq & 20 \\
& -x_1 + 2x_2 & & & \leq & 18 \\
& & -2x_3 & + x_4 & \leq & 22 \\
& & 2x_3 & + x_4 & \leq & 30
\end{array}
$$

The first two constraints are the complicating constraints corresponding to matrices A_1 and A_2 in Figure 1. The next two sets of constraints correspond to the matrices B_1 and B_2 in Figure 1. B_1 and B_2 are the constraints for the subproblems. Figure 2 is a graph of the constraint regions for B_1 and B_2.

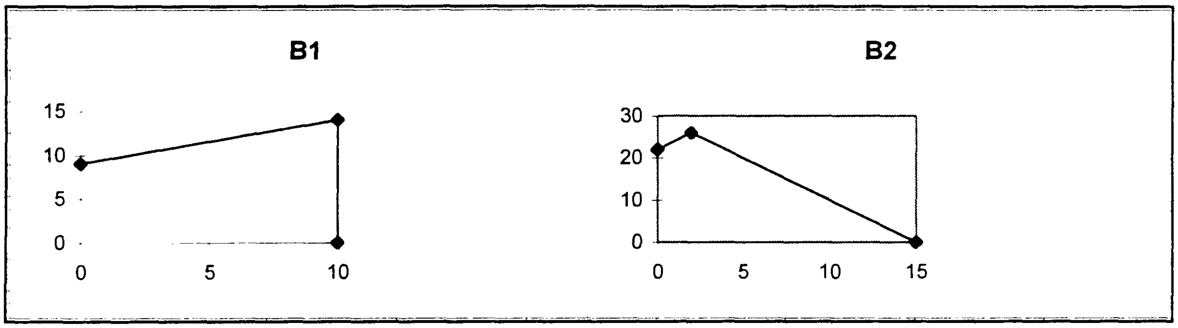

Figure 2. Representation of *X* by two sets.

Step I - Reformulate the problem

Minimize: $\sum_{j=1}^{t}\left(c^T x_j\right)\lambda_j$

Subject to: $\sum_{j=1}^{t}\left(Ax_j\right)\lambda_j + s = b$ (3.17)

$$\sum_{j=1}^{t}\lambda_j = 1 \qquad \lambda_j \geq 0 \qquad j = 1,2,\ldots,t$$

It can be seen from the graphs in Figure 2 that an initial extreme point for the subproblems is $x_1 = (0,0,0,0)$, which is the origin. For the starting basis we will use s_1, s_2 and λ which form the identity matrix. The cost of $c^T x_1 = 0$. The dual vectors are calculated as follows: $\left(w^T, \alpha\right) = \hat{c}_B^T B^{-1} = 0B^{-1} = 0$.

Step II - Formulate and solve the subproblem

Maximize: $\left(w^T A - c^T\right)x + \alpha$

Subject to: $x \in X$

Here $(w_1, w_2, \alpha) = (0, 0, 0)$. Therefore the subproblem is as follows:

$$
\begin{aligned}
\text{Maximize:}\quad & 4x_1 + 2x_2 + 2x_3 + x_4 \\
\text{Subject To:}\quad & 2x_1 \leq 20 \\
& -x_1 + 2x_2 \leq 18 \\
& x_i \geq 0 \\
& x_3, x_4 \text{ constant}
\end{aligned}
$$

and

$$
\begin{aligned}
\text{Maximize:}\quad & 4x_1 + 2x_2 + 2x_3 + x_4 \\
\text{Subject To:}\quad & -2x_3 + x_4 \leq 22 \\
& 2x_3 + x_4 \leq 30 \\
& x_i \geq 0 \\
& x_1, x_2 \text{ constant}
\end{aligned}
$$

The optimal solution to the subproblem is $x_2 = (10, 7, 15, 0)$ with the objective value $r_2 = 84$.

Step III - A column $\begin{bmatrix} r_2 \\ Ax_2 \\ 2 \end{bmatrix}$ is added to the master problem, which is then solved. The duals are passed back to the subproblem and steps II and steps III continue until the solution to the subproblem objective function, $r_j = 0$. At that point, the basic feasible solution, (λ_B, λ_N), to the master problem is optimal.

The solutions for each iteration are listed in table 1.

Subproblem Objective	Master Problem Objective	$x = (x_1, x_2, x_3, x_4)$	Lower bound
	0	0, 0, 0, 0	
54 + 30 = 84	-46.5	10, 7, 15, 0	-84
28.1 + 22.2 = 50.3	-50.0	10, 7, 2, 26	-96.8
13.8 + 0 = 13.8	-58.1	10, 0, 0, 0	-63.8
2.6 + 0 = 2.6	-58.7	10, 7, 0, 0	-60.7
0			-58.7

Table 1. Solutions to the example problem.

The final solution to the overall problem will be the last value of the objective function of the master problem. The best known feasible solution is

$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 + \lambda_4 x_4,$$

$$x = .356(10,7,15,0) + 0(10,7,2,26) + .429(10,0,0,0) + .216(10,7,0,0),$$

$$x = (10,4,5.3,0).$$

The optimal objective value is -58.7 and the final value of x is calculated to be $x = (10,4,5.3,0)$.

A plot is given in Figure 3 of the lower bounds and the upper bounds. One can observe that the value of the objective function is in fact getting smaller at each iteration. The lower bound actually decreases in value at the second iteration. However, it increases at each successive iteration thereafter.

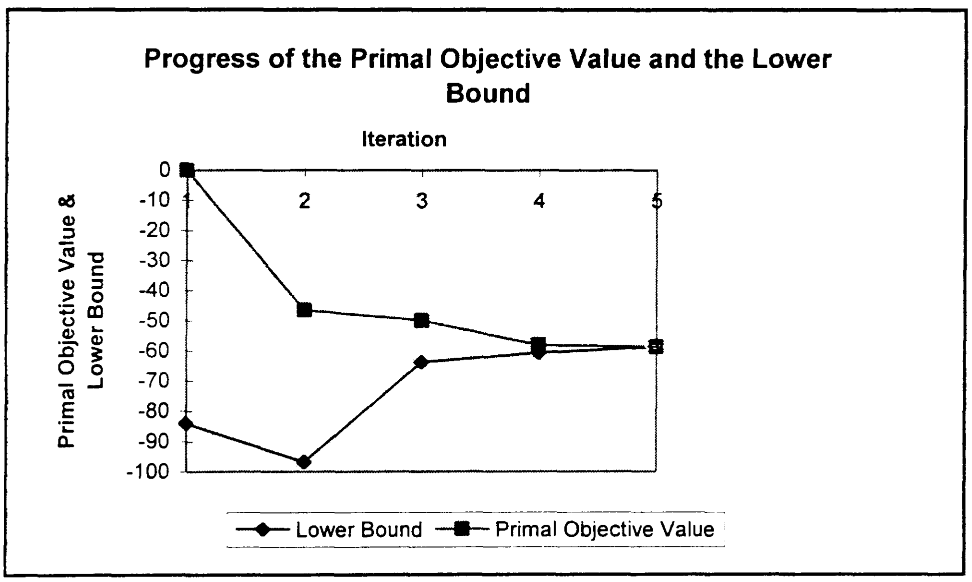

Figure 3. Graph of Primal Objective Value and Lower Bound

Chapter 4

DISCUSSION OF THE MODEL FOR TRADING ELECTRICITY BETWEEN POWER COMPANIES

4.1 Introduction

The linear programming problem solved herein optimizes the expansion of energy production within the various energy districts. Using the forecasted economic data for future usage within each district and the cost of producing each type of energy, we minimize the total cost of energy production while adhering to various capacity requirements and environmental restrictions. The model is decomposed according to regions. The subproblem is solved for each region separately. The master problem takes the solutions from the subproblem and minimizes the total cost while adhering to the complicating constraint which states that the sum of electricity imported minus the sum of electricity exported must equal zero.

4.2 Description of the Variables

This problem consists of seven variables. The variable c is the total energy capacity in megawatts. The variable n is the amount of increase in capacity in megawatts. The third variable d represents the amount of production energy in megawatts that is decommissioned. The variable g is the actual amount in million kilowatt hours produced. These four variables vary over time, region and plant type. The variable import is the amount of energy in megawatts imported from another region. The variable export is the amount of energy in megawatts exported to another region. These two variables vary over time and region only. The units for all of these variables are in megawatts, except for the variable *g*, whose units are kilowatt hours. The units for the variables in a constraint must be the same, therefore the amount of electricity in megawatts is converted to kilowatt hours by multiplying it by a conversion factor of 8760/1000.

The last variable, build, is a binary variable that regulates whether new capacity(n) is built or not. It is a switch that is turned on only when new capacity(n) is greater than some lower bound. Ideally, it may be optimal to add a small amount of new capacity, for example add 25% of the capacity of a new generator, however, this is an obvious impossibility. A generator will add a certain amount of new capacity. We don't know what these incremental amounts of new capacity are because different types of energy production, different efficiencies, different equipment, etc are involved. Nevertheless, a lower bound was set on the amount of new capacity added to try to make the model more realistic.

4.3 Detailed Description of the Model

In this mixed integer programming problem we are minimizing the total cost of producing energy. These costs in the objective function can be broken down as follows: the cost of adding new capacity, the operating fixed costs, the operating variable costs, the fuel variable costs, the cost of importing energy and the profit from exporting energy. The cost of adding new capacity is the cost of construction times the amount of new capacity, *n*, added. The operating fixed costs are set expenses that do not change and are based on the total capacity, *c*, of the plant. Variable costs are incurred when the energy is produced and will vary with the level of production. The operating variable costs, the fuel variable costs are variable costs that are associated with *g*, the actual amount of energy produced. The import cost is the cost for importing energy from another region. There is a negative cost, or profit, for exporting energy.

There are three demand constraints in the model. The constraint (4.8) states that the actual amount of energy generated must be less than the capacity minus the energy exported. For the constraint (4.9), the outage_rate is a fractional amount of time that the energy demand will exceed the supply, thus resulting in outages. (1 - outage_rate[R]) gives the percentage of the time that the sum of the capacity plus the energy imported must be greater than the projected peak demands for that region. The constraint (4.11) states that the actual amount of energy produced plus the energy imported must exceed the demand in that region for each time period.

The next set of constraints determine whether new capacity will be added. The constraint (4.2) states that the variable n must be less than some arbitrarily large number when the binary variable, build, is turned on, or it must be less than or equal to zero when build is turned off. The constraint (4.3) forces n to be greater than a specified lower bound when build is turned on and greater than or equal to zero when build is turned off. When combined with the previous constraint n must be greater than the lower bound or equal to zero. When n does not equal zero, then build must equal one.

Another set of constraints are placed on the capacity. The constraint (4.4) sets initial values for c for each region and plant type for the year 1990. The constraint (4.5) forces the capacity to equal the sum of the capacity of the previous time period plus the new capacity minus any capacity that is decommissioned. The constraint (4.6) forces the amount of energy exported to be less than the capacity of energy production for that region.

There are two constraints associated with the trading. The constraint (4.7) limits the total amount of energy that can be imported. The second constraint states that the total amount of energy imported should equal the total amount of energy exported.

The final constraint (4.13) is an environmental regulation limiting the emission of greenhouse gasses into the environment. The following parameters: emission_rates, emission_rates1, d_weight, i_weight, gram_to_ton, and efficiency contain the data used to describe the greenhouse emission. The amount emitted must be lower than the greenhouse bound.

Without the binary variable, the problem is still a very large problem and would take a considerable amount of time to solve. When the integer variable is added back in, the complexity and time required to solve the problem goes up many fold. In examining the model, all of the constraints except one work with only one region at a time. For this reason, the problem can naturally be decomposed into smaller problems. It will be more efficient to solve these smaller integer programming problems than to solve one very large linear programming problem.

These decomposed problems, called subproblems contain all of the constraints except for the trading constraint, which states that the sum of electricity imported minus the sum of electricity exported must equal zero. This complicating constraint is placed in the master problem. The mathematical model of the decomposed problem is given below.

Unless otherwise stated, the parameters are: $t \in T$, $f \in P$, $r \in R$, $rc \in R$: $rc \neq r$
where

T = Time: 1990, 1995, 2000, 2005, 2010, 2015

P = Plant type: COAL, OILGT, NGGT, NUCLEAR, HYDRO

R = Region: 1, 2.

The variables are expressed in lower case letters and the parameters are expressed in upper case letters.

Maximize Subproblem:

$$\sum_{t} D_t\left(e_{t,r} - i_{t,r}\right) + \alpha - \sum_{t}\sum_{f} CCV_{t,f} * n_{r,t,f} + COF_{t,f} * c_{r,t,f} + COV_{t,f} *$$

$$g_{r,t,f} * 1000 + \frac{CFV_{t,f}}{EFF_f / 3412} * g_{r,t,f} + \sum_{rc}\left(CIMP_{rc} * i_{t,rc}\right) - \left(PEXP * e_{t,r}\right) \qquad (4.1)$$

Subject TO:

$$n_{r,t,f} \leq b_{r,t,f} * 100000 \qquad (4.2)$$

$$n_{r,t,f} \geq b_{r,t,f} * L \qquad (4.3)$$

$$c_{r,1990,f} = c_{0_{r,f}} \qquad (4.4)$$

$$c_{r,t,f} = c_{r,t-1,f} + n_{r,t,f} - d_{r,t,f} \qquad t > 1990 \qquad (4.5)$$

$$e_{t,r} \leq \sum_{f} c_{r,t,f} \qquad t > 1990 \qquad (4.6)$$

$$i_{t,r} \leq 2000 \qquad t > 1990 \qquad (4.7)$$

$$g_{r,t,f} \leq c_{r,t,f} * \frac{8760}{1000} * CAP_f - e_{t,r} * \frac{8760}{1000} \qquad (4.8)$$

$$\left(1 - OUT_r\right) * \sum_{f} c_{r,t,f} + i_{t,r} \geq PD_{r,t} \qquad t > 1990 \qquad (4.9)$$

$$\sum_{i} g_{r,t,i} \leq 0.4 * \sum_{f} g_{r,t,f} \qquad t > 1990,\ i \in PL \qquad (4.10)$$

$$TD_{r,t} \leq \sum_{f} g_{r,t,f} + i_{t,r} * \frac{8760}{1000} \qquad t > 1990 \qquad (4.11)$$

$$n_{r,t,k} = PA_{r,t,k} \qquad k \in HT \qquad (4.12)$$

$$\sum_{f}\sum_{h} g_{r,t,f} * \left[\left(ER_{f,m} + ER1_{f,m}\right) * \left(DW_{H,m} + IW_{H,m}\right)\right] * \frac{GTT}{EFF_f / 3412} \leq GHB$$

$$t > 1995,\ m \in GHG \qquad (4.13)$$

MASTER PROBLEM

Minimize: $$M * \sum_{t} a_t + \sum_{j=1}^{s} \hat{c}_j \lambda_j \tag{4.14}$$

Subject To: $$A_t + \sum_{j=1}^{s} T1_{t,j} * \lambda_j = 0 \tag{4.15}$$

$$\sum_{j=1}^{S} \lambda_j = 1 \tag{4.16}$$

where

$$\hat{c} = \begin{pmatrix} \sum_{t} \sum_{f} CCV_{t,f} * n_{r,t,f} + COF_{t,f} * c_{r,t,f} + COV_{t,f} * g_{r,t,f} * 1000 \\ + \dfrac{CFV_{t,f}}{EFF_f / 3412} * g_{r,t,f} + \sum_{rc} \left(CIMP_{rc} * i_{t,rc} \right) - \left(PEXP * e_{t,r} \right) \end{pmatrix} \tag{4.17}$$

$$T1_{t,s} = \sum_{r} e_{t,r} - \sum_{r} i_{t,r} \tag{4.18}$$

D = trade_dual

α = alpha

e = export

i = import

b = build

CCV = construction_variable_cost

COF = operating_fixed_cost

COV = operating_variable_cost

CFV = fuel_variable_cost

EFF = efficiency

CIMP = import_cost

PEXP = export_profit

L= lower bound on n

I = interval

CAP = capacity_factor

OUT = outage_rate

PD = peak demand

TD = total_demand

PA = plants_added_in_serc

ER = emission_rates

ER1 = emission_rates1

DW = d_weight

IW = i_weight

GTT = gram_to_ton

GHB = greenhouse_bound

GHG = greenhouse gasses

H = horizon

PL = Peak_Load

HT = hydro_type

M = Big_M

s = new_cols

$\hat{c}$ = c_hat

T1 = Trade1

λ = Lambda

The subproblem is solved twice; once for each region. The master problem takes the extreme points generated by the subproblem and forms a convex combination of these points that will minimize the costs while meeting the trading constraint. The solution to the master problem contains the duals trade1_dual and alpha. These duals modify the cost function of the subproblem. The subproblem and master problem are solved iteratively until a suitable stopping point or the optimal solution is found.

4.4 STARTING SOLUTION

In the earlier discussion of the decomposition algorithm, a known starting basic feasible solution was assumed to exist. In the energy problem, however, there is no known starting basic feasible solution, so the Big M method has been employed to start the process. The Big M is a method to take the problem from phase I, where the problem is infeasible, to phase II, where the problem is feasible. It consists of adding artificial variables to the problem to form a basis. A very large cost is placed on these artificial variables so that with each progressive iteration an artificial variable will be removed from the basis. When all of the artificial variables have been removed from the basis, then the problem is feasible. The constraints for the master problem are equality constraints, so artificial variables must be added. These variables absorb the difference between what is actually being imported and exported so that the constraint is not violated. However, a very large penalty is placed on these artificial variables to force them to zero as the problem is being optimized.

On a smaller scale using only four time periods we will solve for the initial starting solution. The objective function, which includes the artificial variables for the master problem, is as follows:

Minimize: $M * \sum_{t=1}^{4} a_t + \hat{c}_1 \lambda_1$ $\quad t \in T, f \in P, r \in R, rc \in R: rc \neq r, j \in new\_cols$

Subject To: $a_t + \sum_{j=1}^{s} T1_{t,j} * \lambda_j = 0$

where

$$\hat{c} = \begin{pmatrix} \sum_t \sum_f CCV_{t,f} * n_{r,t,f} + COF_{t,f} * c_{r,t,f} + COV_{t,f} * g_{r,t,f} * 1000 \\ + \dfrac{CFV_{t,f}}{EFF_f / 3412} * g_{r,t,f} + \sum_{rc} \left(CIMP_{rc} * imp_{t,rc}\right) - \left(PEXP * \exp_{t,r}\right) \end{pmatrix}$$

$$T1_{t,s} = \sum_r \exp_{t,r} - \sum_r imp_{t,r}$$

where $\hat{c}(1)$ is the cost, $c^T x_j$, for the first iteration, a_t are the artificial variables, and M is a vary large cost associated with these artificial variables. The constraints corresponding to the A matrix in the description of the decomposition algorithm is

$$\sum_{r=1}^{2} \left(\exp_{i,t,r} - imp_{i,t,r}\right)\lambda_1 + a_t = 0 \qquad t = 1..4$$

where $i = 1$ for the 1st iteration.

For the first iteration you are assuming that import and export are zero. The basis for the first iteration is as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a(1) \\ a(2) \\ a(3) \\ a(4) \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Table 2. Initial Basis for the Trading Problem.

where $a(t) = 0$ $\lambda_1 = 1$.

The value of the duals which are obtained from the master problem are

$$\begin{bmatrix} [\underline{w},\alpha] = \hat{c}_B B^{-1} = [MMMM\hat{c}(1)]B^{-1} \\ [\underline{w},\alpha] = [MMMM\hat{c}(1)] \\ \underline{w} = MMMM \\ \alpha = \hat{c}(1) \end{bmatrix}$$

These dual values are passed to the subproblem whose objective function has the form

maximize $(w^T A - c)\underline{x} + \alpha = w^T Ax - c^T \underline{x} + \alpha$.

The full objective function to the subproblem are

Maximize:

$$\begin{bmatrix} \left(M * \sum_{r=1}^{2} e_{1,r} - i_{1,r}\right) + \left(M * \sum_{r=1}^{2} e_{2,r} - i_{2,r}\right) + \left(M * \sum_{r=1}^{2} e_{3,r} - i_{3,r}\right) + \left(M * \sum_{r=1}^{2} e_{4,r} - i_{4,r}\right) \\ +\alpha - \sum_{t}\sum_{f} CCV_{t,f} * n_{r,t,f} + COF_{t,f} * c_{r,t,f} + COV_{t,f} * g_{r,t,f} * 1000 + \\ \dfrac{CFV_{t,f}}{EFF_f / 3412} * g_{r,t,f} + \sum_{rc}\left(CIMP_{rc} * i_{t,rc}\right) - \left(PEXP * e_{t,r}\right) \end{bmatrix}$$

To reiterate, this is an example of the initial starting solution only on a smaller scale. It demonstrates what values are being passed to the subproblem and how they affect the subproblem.

Chapter 5
HEURISTIC ALGORITHM

Heuristic Solution

When the suproblem is solved, a solution which is integer but not feasible for the overall program is obtained. The master problem solution is feasible for all the constraints except the integer feature. For this reason, a heuristic method is employed to take the final solution to the master problem and make it satisfy the integer feature and still be feasible.

Part of the heuristic involves adjusting the value of the import and export variables. The constraint in the master problem states that the sum of the electricity imported minus the sum of the electricity exported must equal zero. At the termination of the decomposition algorithm this constraint is satisfied, however, it has one region exporting electricity to itself. This traded electricity is a phantom. It does not exist. The heuristic will take the difference of the electricity imported and exported within the region to obtain the actual amount of electricity traded between the regions. This adjustment will improve the optimal solution because the profit from exporting energy is always smaller than the cost of importing energy. For example, you exported $100,000 worth of electricity and you imported the same amount of electricity at a cost of $120,000. If you neither exported nor imported that electricity, there would be a net gain of $20,000. This is a profitable move, but it is also required to make the output understandable.

Throughout the heuristic, the master problem constraint remains satisfied because the sum of the adjusted values of the electricity imported minus the sum of the electricity exported still equals zero. The import and export variables are then fixed at their adjusted values. The subproblem is then run for each region. The new solutions will satisfy the integer constraint while continuing to satisfy our master problem constraint because the trading variables were held constant. The sum of the objective functions for the two regions is the solution to the overall problem.

The heuristic solution should be close to the solution from the decomposition algorithm. There should be some gains from adjusting the import and export variables. There may be some losses in the process of satisfying the integer constraint. Overall, though, we do not expect these to make a significant impact in the value of the objective function.

Chapter 6

CONCLUSION

The problem was run on an IBM compatible with 486 motherboard at speed of 100 megahertz and 8 megabytes of ram. The solution time for solving both the decomposition problem and the heuristic was 2 minutes and 43 seconds. The actual problem solved contained six time periods, two regions, and five different methods of producing energy. The solution to the trading problem obtained using the decomposition algorithm is 6.11×10^{10} dollars. The solution obtained using the heuristic is 6.13×10^{10} dollars. There is less that a one percent increase in the overall cost function after running the heuristic. The solution to the heuristic is not optimal, however it is acceptable.

Two graphs of the progress of the primal objective and the lower bound are given in Figure 4. As you examine the lower bounds, there is a general increasing trend, however, the data points do bounce around. On iteration 14, you see a sharp spike in the lower bound that would not be expected. Even in the example problem, figure 3, the lower bound does decrease at one point before it starts increasing. One may assume that the lower bound would be a monotonically increasing function. This, however, is not a result from the derivation of the lower bound The result is that the primal objective function will not cross over the lower bound and still be feasible.

The trading problem does not reach true optimality. The solution has come close to this optimal point, but with each successive iteration past the 24 iterations there is not much progress made. The difference between the primal problem and the lower bound is about 13 percent. There is some movement in the lower bound, however the trend is consistent. For this reason we stopped the program and ran the heuristic to obtain a final solution to the overall problem.

Overall the model for the trading of electricity was successful. We successfully decomposed an integer programming problem. We were able to get within a reasonable percentage of optimality. When this is implemented on the full problem, significant decreases in run time using a decomposition algorithm should occur as compared to solving it using the simplex algorithm.

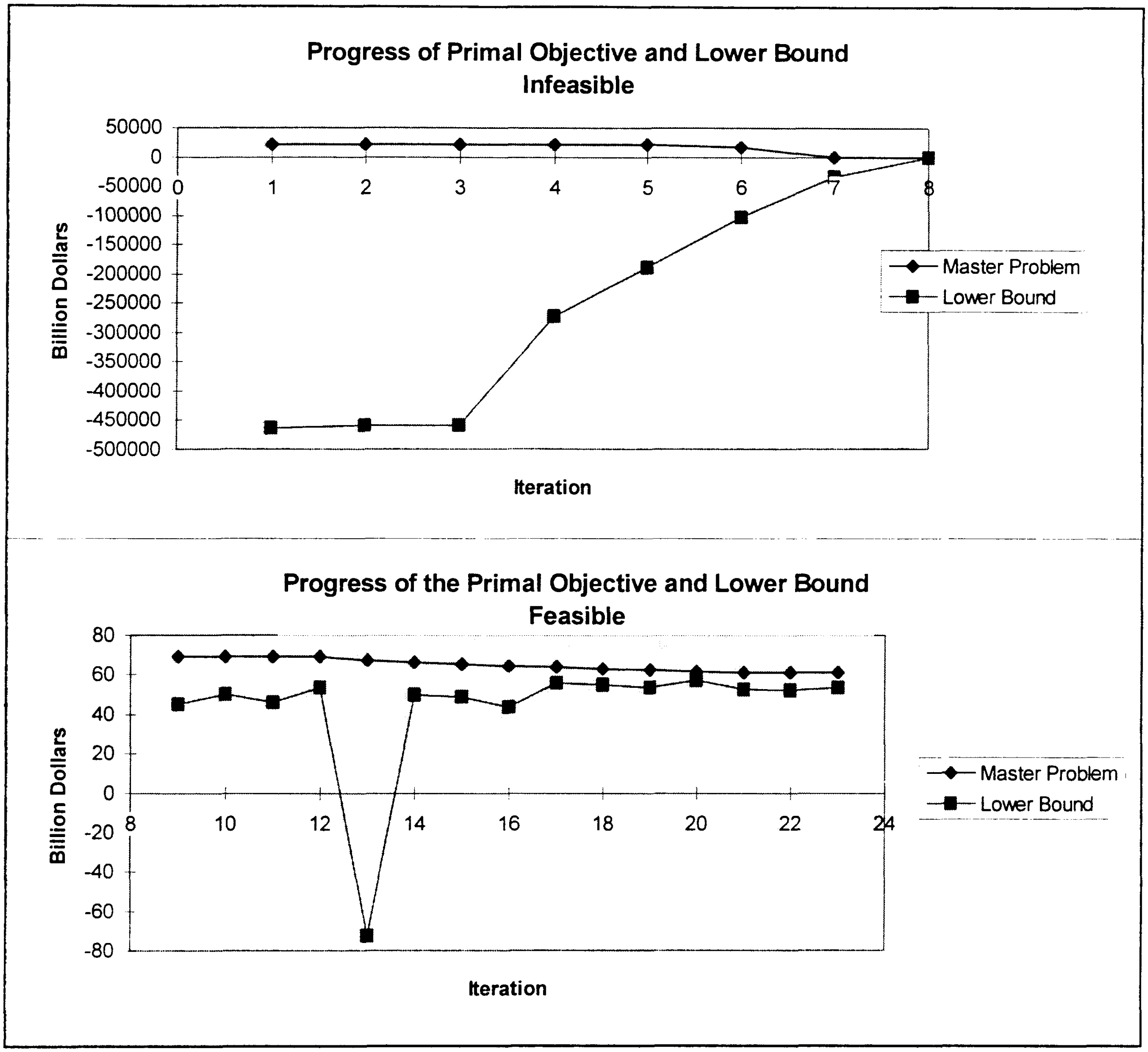

Figure 4. Graph of the Primal Objective Function and the Lower Bound for the Trading Problem.

REFERENCES

[1] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear Programs and Network Flows.* John Wiley & Sons, Inc. ,1990.

[2] Richard Coppins and Nesa Wu. *Linear Programming and Extensions.* McGraw-Hill, Inc., 1981.

[3] G.B. Dantzig and P. Wolfe. "The Decomposition Algorithm for Linear Programming." *Studies in Optimization Volume 10.* Ed. G.B. Dantzig and B.C. Evans. The Mathematical Association of America, 1974.

Appendix A

DESCRIPTION OF THE SOFTWARE

AMPL is an algebraic modeling language for mathematical programming. It allows a program to be written in a form similar to traditional mathematical notation. AMPL translates the program into a form that the solver, CPLEX, can readily use. After CPLEX solves the problem, AMPL will translate the solution back into the modeler's form.

AMPL allows you to program in a modular form. The basic components are the data file, the model file, and the run file. Each file has its own function. The run file controls the flow of the program. The data file contains all of the data. The model file contains the actual model. The power of AMPL lies in that the model is written purely in terms of variables and parameters without specifically encoding the data into the model. Therefore when changes need to be made to the data, you can go directly to the data file rather than having to search through all the code in the model to find out exactly where this data was used. In doing this it decreases your probability of error when modifying data because it is changed in only one place in the data file rather than in multiple places in the model file.

The files included in this software are: energy.dat, energy.mod, energy.run, rsub.run, rmaster.run, dmaster.run, dsub.run. The energy.dat file contains all the data for the various parameters. The energy.mod file is the file that contains the actual model. It also declares the sets, parameters and variables that are used in the model file. Restore is a command that activates constraints. The file rsub.run restores all the constraints used in the subproblem while the file rmaster.run restores all the constraints used in the master problem. Drop is a command that deactivates constraints. The file dsub.run drops all the constraints used in the subproblem while the file dmaster.run drops all the constraints used in the master problem. These files are called in the run file using an include and the filename. The energy.run file controls the flow of the operation. It declares what objective function is to be used and includes files to specify what constraints will be used in solving the linear program. Overall, it controls the iterative process that switches between solving the master problem and the subproblem.

Appendix B

ENERGY SOFTWARE

This appendix contains all of the files that comprise the Energy software.

A.1 Energy.Dat File

```
#January 19, 1996--Final data set

set FUEL_TYPE := COAL OIL GAS NUCL HYDRO;
set GASES  := CH4 N2O NMHC CO NOx CO2 SOx ;
set GREENHOUSE_GASES  := CH4 N2O NMHC CO NOx CO2 ;
set REGION := 1 2;
set TIME := 1990 1995 2000 2005 2010 2015;
set PERIOD := 1 2 3 ;
set OLD_TECH := COAL OILGT NGGT NUCLEAR HYDRO;
set HYDRO_TYPE := HYDRO;
set COAL_FIRED := COAL;
set GAS_FIRED := NGGT;
set OIL_FIRED := OILGT;
set NUCL_FIRED :=NUCLEAR ;
set PEAK_LOAD :=  NGGT OILGT;
set GHG_EFFECT_HORIZON := 20_year 50_year 100_year ;
set LOAD_GROWTH_PATTERN := low_low_low high_high_high ;
set FUEL_GROWTH_SCENARIO := reference high_oil low_oil ;


##########################
#  import/export data  #
##########################
param import_cost :=
        1 120000
        2 140000;


##########################
```

```
#       lambda             #
##########################
param lambda := 1000000;

##########################
#     symbolic  parameters    #
##########################
param  horizon := 20_year ;
param  pattern :=  low_low_low ; #high_high_high ;
param  scenario := reference ;

##########################
#     constant parameters    #
##########################
param  interval := 5 ; # interval in TIME
param  discount_rate := .05 ;
param  d_percent := 0.3;

param  peak_demand0 := # Power demand in MW in 1990
        1        41087
        2      30995;

param  total_demand0 := # 1990 energy demand in million_kWh
        1        211352
        2      154870;

param   SOx_limits :=     # short tons--Phase I reductions
        1        1000000000
        2      812540;

param outage_rate :=
        1        .126
        2      .054;

param   mw_growth_rates :=   # low-low-low growth rates for all three periods
[*,*,low_low_low]:
          1   2 :=
   1      1.00958       1.00941
   2      1.00951       1.00936
   3      1.00658       1.00648
```

```
[*,*, high_high_high]:
            1   2 := #high-high-high mw #growth
   1      1.0258      1.027
   2      1.00958     1.00941
   3      1.00846     1.00833 ;


param   kwh_growth_rates :=    # low growth rates for all three periods
[*,*,low_low_low]:
            1   2 :=
   1      1.01017     1.01017
   2      1.01012     1.01012
   3      1.007       1.007

[*,*, high_high_high]:
            1   2 :=
   1      1.0274      1.0292
   2      1.01017     1.01017
   3      1.009       1.009 ;

param   fuel_escalator :=    # $/MBTU input
[*,*,reference]:
           COAL      GAS      OIL     NUCL :=
   1     1.0075    1.0284   1.015    0.97857
   2     1.016     1.0345   1.0317   1
   3     1.0118    1.0315   1.0234   1

[*,*, high_oil]:
           COAL      GAS      OIL     NUCL :=
   1     1.0064    1.0291   1.0354   0.97857
   2     1.0147    1.038    1.0369   1
   3     1.0105    1.0335   1.0362   1

[*,*,low_oil]:
           COAL      GAS      OIL     NUCL :=
   1     1.0064    1.0099   0.9997   0.97857
   2     1.0157    1.0419   1.0355   1
   3     1.011     1.0258   1.0029   1;
```

```
param   initial_c default 0  (tr) : # initial capacity in 1990
              1 2:=
        COAL     21820  24821
        NGGT       748     0
        OILGT     2208    14
        NUCLEAR  14426 5481
        HYDRO     4742  3794;

param   initial_g default 0  (tr) :== # estimated generation in million kWh in
            1          2:=
        COAL        97013      113391
        NGGT        1626        1503
        OILGT       0            0
        NUCLEAR      92588          29070
        HYDRO       12626     9094;

 param fuel_variable_cost0 default 0 :=   #in $/MBtu  in 1990 value for southeastern
regions
        COAL           1.61
        OILGT          3.05
        NGGT           2.32
        NUCLEAR          0.652
        HYDRO           0;

param capacity_factor default 0 :=  # design value
        COAL        0.8    #.80  *8760*1000
        NGGT        0.10    #. tag guide for peak unit
        OILGT       0.10    #gas turbine oil
        NUCLEAR 0.7    #.70  *8760*1000
        HYDRO       0.46;    #.46  *8760*1000

param   efficiency default 1.0 :=      # these numbers affect the total emissions; note
default
                                   #value for transfer between regions/assumes perfect
efficiency
        COAL         0.36
        NGGT          0.25  # EIA / Annual Outlook for US Elec. Pwr
        OILGT         0.25
        NUCLEAR    0.31
```

```
        HYDRO       0.33;

param construction_variable_cost0 :=    # $/MW
        COAL         425035    # Coal steam from SERC regression
        NGGT         207518     # Gas turbine  form SERC regression
        OILGT        207518    # Gas turbine from SERC regression
        NUCLEAR        850039    #  regression from SERC  or use 1480000
        HYDRO       11020000;

param operating_fixed_cost0 := #$/MW in 1990 value
        COAL          13100
        NGGT            500
        OILGT           500
        NUCLEAR        104900
        HYDRO          6100;

param operating_variable_cost0 :=  #mills/kWh, or 1000$/million_kWh in 1990 value
        COAL       4.100
        NGGT       4.8
        OILGT      4.80
        NUCLEAR     0.7
        HYDRO      1.90;

param  environmental_cost0 :=  #$MBtu fuel input
        COAL       4.137
        NGGT        1.226
        OILGT       1.92
        NUCLEAR   2.189 #2.41 cent/kWh  output, w/o decm , 31% efficiency
        HYDRO       1.06; #0.11-0.12 cents /kWh output, 33% efficiency

param d_weight default 0 :  # GHG equiv. wt from dir effect for 3 different time horizon
              CH4   N2O  NMHC    CO   NOx    CO2     SOx :=
20_year       35    260   0      0     0      1       0
50_year       11    270   0      0     0      1       0
100_year       4    170   0      0     0      1       0 ;

param i_weight default 0 :  # GHG equiv. wt from indir effect for 3 diff. time horizon
           CH4    N2O  NMHC    CO    NOx  CO2    SOx :=
20_year      37    260   28     7    150    1      0
50_year      11    270    8     3     40    1      0
```

```
100_year     4    170    3     2     14    1     0 ;

param emission_rates default 0 :  # in g/MBtu (input) for power plant
                    CH4  N2O  NMHC  CO    NOx    CO2    SOx :=
  COAL              0.7  4    1.    13    227.6  95512  418.5
  NGGT              15.7 2    2.    50.6  90.8   53489  0.3
  OILGT             0.9  2    2.3   15.2  152.3  75101  239.9 # OIL steam
  NUCLEAR           0    0    0     0     0      0      0
  HYDRO             0    0    0     0     0      0      0;

 param emission_rates1 default 0 :  # in g/MBtu produced for upstream
                    CH4    N2O  NMHC  CO    NOx   CO2    SOx :=
  COAL              65.7   0.4  0.4   0.3   5.9   29.3   0
  OILGT             7.9    5.3  3.3   1.5   20.6  141.8  0
  NGGT              16.3   0.7  1.1   0.4   21.9  72.0   0
  NUCLEAR           2.7    0.7  0     0.1   4.6   45.9   0
  HYDRO             0      0    0     0     0     0      0;

param :  capital_escalator   o_m_escalator :=
        1990 0.98202  1.0219
        1995 1.00448  1.02392
        2000 0.99849  1.02054
        2005 1.00232  1.01978
        2010  1.0018  1.01902
        2015 1.00261  1.01827;

param   decommission_variable_cost0   default  0  :=   #$/MW
             NUCLEAR    0;                             # 250000;

param  plants_removed  default  0  :=  # serc scheduled in MW

[1,*,*]:
              COAL      OILGT:=
1995      0          94
2000      33         0

[2,*,*]:
              COAL   OILGT:=
1995          973    0
2000          0      16
```

```
2005          150    0;

param   plants_added_in_serc default 0 :=#  in MW
[1,*,*]:
            NGGT  OILGT     HYDRO  :=
1995         613    1408        1214
2000         186      0            0

 [2,*,*]:
         NGGT   NUCLEAR:=
1995   2063      70
2000   3633      0
2005   952       0;
```

A.2 Energy.Mod File

```
set   FUEL_TYPE ;
set   GASES ;
set   GREENHOUSE_GASES within GASES ;
set   REGION ;
set   REGION2;
set   TIME ;
set   PERIOD ;
set   OLD_TECH ; #PRIMARY_FUELS
set   NEW_TECH ; #SECONDARY_FUELS
set   PLANT_TYPE := OLD_TECH;#union NEW_TECH ;
set   HYDRO_TYPE within PLANT_TYPE ;
set   COAL_FIRED within PLANT_TYPE;
set   GAS_FIRED within PLANT_TYPE ;
set   OIL_FIRED  within PLANT_TYPE ;
set   NUCL_FIRED  within PLANT_TYPE ;
set   PEAK_LOAD within PLANT_TYPE ;
set   GHG_EFFECT_HORIZON ;
set   LOAD_GROWTH_PATTERN ;
set   FUEL_GROWTH_SCENARIO ;

#######################
#Miscellaneous Parameters#
```

```
######################
param  scenario symbolic in FUEL_GROWTH_SCENARIO ;
param  horizon symbolic in GHG_EFFECT_HORIZON ;
param  pattern symbolic in LOAD_GROWTH_PATTERN ;
param  discount_rate;
param  interval ;
param  d_percent;
param  gram_to_ton := 1.102311*10^(-6) ; # gram to short tons conversion factor
param  MBtu_to_kWh := 10^(6)/3412 ; # 1 kWh = 3412 Btu
param  mw_growth_rates{PERIOD, REGION, LOAD_GROWTH_PATTERN} ;
param  kwh_growth_rates{PERIOD, REGION, LOAD_GROWTH_PATTERN} ;
param  outage_rate{REGION} ;
param  efficiency{PLANT_TYPE} ;
param  capacity_factor{PLANT_TYPE} ;
param  capital_escalator{TIME} ; # percentage
param  o_m_escalator{TIME} ;
param  fuel_escalator{PERIOD, FUEL_TYPE, FUEL_GROWTH_SCENARIO} ;
param lower:= 1200;
param alpha default 0;                    # dual of lambda
param trade_dual{TIME};
param final_import{t in TIME,r in REGION};
param final_export {t in TIME,r in REGION};

############
# 1990 data  #
############
param  construction_variable_cost0{PLANT_TYPE} ;# $/MW in 1990 value
param  operating_variable_cost0{PLANT_TYPE} ; #1000$/Million_kWh in 1990 value
param  operating_fixed_cost0{PLANT_TYPE} ; #$/MW in 1990 value
param  decommission_variable_cost0{PLANT_TYPE} ; #$/MW in 1990
param  fuel_variable_cost0{PLANT_TYPE} ; #$/MBtu input in 1990
param  total_demand0{REGION} ; # million kWh in 1990
param  peak_demand0{REGION} ; #MW in 1990
param  environmental_cost0{PLANT_TYPE} ; #$/MBTU input in 1990

####################################
# objective function coefficients #
####################################
param R;
param import_cost{REGION};
```

```
param export_profit:= 100000;
param  construction_variable_cost{t in TIME, f in PLANT_TYPE} := # $/MW
        if t=1990 then construction_variable_cost0[f]
        else construction_variable_cost[t-interval, f]*capital_escalator[t] ;
param  decommission_variable_cost{t in TIME, f in PLANT_TYPE} := # $/MW
        if t=1990 then decommission_variable_cost0[f]
        else decommission_variable_cost[t-interval, f]*capital_escalator[t] ;
param  operating_variable_cost{t in TIME, f in PLANT_TYPE} := # $1000/million kWh
        if t=1990 then operating_variable_cost0[f]
        else operating_variable_cost[t-interval, f]*o_m_escalator[t] ;
param  operating_fixed_cost{t in TIME, f in PLANT_TYPE} := # $/MW
        if t=1990 then operating_fixed_cost0[f]
        else operating_fixed_cost[t-interval, f]*o_m_escalator[t] ;
param  fuel_variable_cost{t in TIME, f in PLANT_TYPE}:= # $/MBTU input in 1990
         if t=1990 then fuel_variable_cost0[f]
        else if t >1990 and t<=2000 and f in COAL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[1, "COAL", scenario]
        else if t >1990 and t<=2000 and f in GAS_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[1, "GAS", scenario]
        else if t >1990 and t<=2000 and f in OIL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[1, "OIL", scenario]
        else if t >1990 and t<=2000 and f in NUCL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[1, "NUCL", scenario]
        else if t>2000 and t<=2010 and f in COAL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[2,"COAL", scenario]
        else if t>2000 and t<=2010 and f in GAS_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[2, "GAS", scenario]
        else if t>2000 and t<=2010 and f in OIL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[2, "OIL", scenario]
        else if t>2000 and t<=2010 and f in NUCL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[2, "NUCL", scenario]
        else if t>2010 and f in COAL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[3,"COAL", scenario]
        else if t>2010 and f in GAS_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[3,"GAS", scenario]
        else if t>2010 and f in OIL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[3, "OIL", scenario]
        else if t>2010 and f in NUCL_FIRED then
                fuel_variable_cost[t-interval, f]*fuel_escalator[3, "NUCL", scenario]
        else fuel_variable_cost[t-interval, f]*o_m_escalator[t] ;
```

```
param  environmental_cost{t in TIME, f in PLANT_TYPE} := #$/MBTU input in 1990
        if t=1990 then environmental_cost0[f]
        else environmental_cost[t-interval, f] * o_m_escalator[t] ;


#################
# RHS resources##
#################
param  total_demand{r in REGION, t in TIME} := # demand for energy in million kWh
        if t=1990 then total_demand0[r]
        else if t >1990 and t<=2000
                then total_demand[r, t-interval]*kwh_growth_rates[1, r, pattern]
        else if t>2000 and t<=2010
                then total_demand[r, t-interval]*kwh_growth_rates[2, r, pattern]
        else if t>2010
                then total_demand[r, t-interval]*kwh_growth_rates[3, r, pattern] ;

param  peak_demand{r in REGION, t in TIME} := # demand for capacity in MW
        if t=1990 then peak_demand0[r]
        else if t>1990 and t<=2000
                then peak_demand[r, t-interval]*mw_growth_rates[1, r, pattern]
        else if t>2000 and t<=2010
                then peak_demand[r, t-interval]*mw_growth_rates[2, r, pattern]
        else if t>2010
                then peak_demand[r, t-interval]*mw_growth_rates[3, r, pattern] ;

param  export_capability{r in REGION, k in REGION2} ; # in MW
param  import_capability{r in REGION, k in REGION2} ; # in MW

#########################
# constraint matrix coefficients##
#########################
param  plants_removed{REGION, TIME, PLANT_TYPE} ; #  serc scheduled in MW
param  plants_added_in_serc{REGION, TIME, PLANT_TYPE} ;# in MW
param  initial_c{REGION, PLANT_TYPE} ;
param  initial_g{REGION, PLANT_TYPE} ;

#########################
```

```
# emission data              ##
##########################
param  d_weight{GHG_EFFECT_HORIZON, GASES} ;
param  i_weight{GHG_EFFECT_HORIZON, GASES} ;
param  emission_rates{PLANT_TYPE, GASES } ; # gram/MBtu input from power plant
param  emission_rates1{PLANT_TYPE, GASES } ; # gram/MBtu input from upstream
param  SOx_limits{REGION} ; #SOx total emissions in short tons in 1990
param  WOOD_thermal_resource := 0.25*1637046000/3412 ; # million kWh/yr
param  BIO_thermal_resource := 0.25*51785500000/3412 ; # million kWh/yr
##########################
# greenhouse data            ##
##########################
param  lambda >= 0 ; #goal weight on greenhouse gases

param  greenhouse_bound := sum{r in REGION, f in PLANT_TYPE,
                              h in GREENHOUSE_GASES}
       initial_g[r,f]*(emission_rates[f,h]
       +emission_rates1[f,h])*
       (d_weight[horizon, h]
       +i_weight[horizon, h])
       *gram_to_ton/(efficiency[f]/3412) ;


##########################
#        variables          ##
##########################


var  c{r in REGION, t in TIME, f in PLANT_TYPE} >= 0, :=0;
                                        # capacity in MWs
var  n{r in REGION, t in TIME, f in PLANT_TYPE} #>= 0;
       >= if f in PLANT_TYPE diff HYDRO_TYPE then plants_added_in_serc[r,t,f],
:=0;
                                        # new mw's of production
var  d{r in REGION, t in TIME, f in PLANT_TYPE} #>= 0;
                              >= if t > 1995 then plants_removed[r, t, f], :=0;
                              # mw's of production decommissioned
var  g{r in REGION, t in TIME, f in PLANT_TYPE} >= 0, :=0;
                                   # actual million_kWh production
```

```
var  import{TIME, REGION } >=0, :=0;  #, <= import_capability[r, k] ;

var  export{TIME, REGION } >=0, :=0; #, <= export_capability[r, k] ;

var build{r in REGION, t in TIME, f in PLANT_TYPE}  binary;


##################
### Subproblem ###
##################

maximize subproblem: # region 1 = VACAR #
        (sum {t in TIME} trade_dual[t]* (export[t,R] - import[t,R])) + alpha
         -
        sum{t in TIME, f in PLANT_TYPE}       # standard costs
        (
        construction_variable_cost[t, f]*n[R, t, f] +
        operating_fixed_cost[t, f]*c[R, t, f] +
        operating_variable_cost[t, f] * g[R, t, f] *1000 +
        (fuel_variable_cost[t, f] / (efficiency[f]/3412)) * g[R, t, f] +
        sum{rc in REGION diff {R}} (import_cost[rc] * import[t,rc]) -
        export_profit * export[t,R]
        );

subject to n_constraint1{t in TIME, f in PLANT_TYPE}: #set for start point.
         n[R, t, f]<= build[R, t, f] * 100000;

subject to n_constraint2{t in TIME, f in PLANT_TYPE}:  #set for start point.
        n[R, t, f]>= build[R, t, f] * lower;

subject to c_constraints{ f in PLANT_TYPE} :  #set for start point.
        c[R, 1990, f] = initial_c[R, f] ;

#balancing constraint on capacity add and decom.
subject to new_capacity_restrictions
   {t in TIME, f in PLANT_TYPE : t > 1990 } :
        c[R, t, f] = c[R, t-interval, f] + n[R, t, f] - d[R, t, f] ;

subject to export_bounds  # you cannot export more the you have
         {t in TIME: t > 1990 } :
```

```
          export[t,R] <=
                 sum{ f in PLANT_TYPE } c[R, t, f] ;

subject to upper_bound_import # limits amount imported
        { t in TIME: t > 1990 } :
          import[t,R] <= 2000;

subject to generation_constraints
         {t in TIME, f in PLANT_TYPE } :
         g[R, t, f] <= c[R, t, f] * (8760/1000) * capacity_factor[f]
        - export[t,R] * (8760/1000) ;

subject to peak_demand_constraints #satisfaction of peak demand
         {t in TIME  : t > 1990} :
         (1 - outage_rate[R])* sum{f in PLANT_TYPE} c[R, t, f] + import[t, R]
         >= peak_demand[R,t];

subject to base_load_constraints #matches technology to demand curve
        {t in TIME  : t > 1990} :
        sum {i in PEAK_LOAD} g[R, t, i]
                <= 0.4*sum {f in PLANT_TYPE} g[R, t, f];

# Each subregion demand must be satisfied separately
subject to total_demand_constraints2 {t in TIME : t > 1990} :
        total_demand[R,t] <=
        (sum{ f in PLANT_TYPE} g[R, t, f]) + (import[t,R]) * (8760/1000) ;

subject to hydro_addition1 # Restriction on hydro additions
         {t in TIME, i in HYDRO_TYPE } :
        n[R, t, i] = plants_added_in_serc[R, t, i] ;

subject to greenhouse_restrictions{t in TIME : t >1995} :
        sum{f in PLANT_TYPE, h in GREENHOUSE_GASES}
                g[R, t, f] * ((emission_rates[f, h] + emission_rates1[f, h])
                * (d_weight[horizon, h]+i_weight[horizon, h]) )
                * gram_to_ton/(efficiency[f]/3412) <= greenhouse_bound;

### MASTER PROBLEM ###
param new_cols integer >=0;
param stored_import{t in TIME,r in REGION,1..new_cols};
```

```
param stored_export{t in TIME,r in REGION,1..new_cols};
param Trade1 {t in TIME,1..new_cols};
param convex_import{t in TIME,r in REGION};
param convex_export{t in TIME,r in REGION};
param c_hat{1..new_cols};
param BigM := 1.0e+10;
var Lambda{1..new_cols}>=0;
var A{t in TIME} >=0;

minimize Total_cost : # cost in units of $
        (BigM * sum {t in TIME}A[t])
        +
    sum{j in 1..new_cols}(c_hat[j]*Lambda[j]);

subject to extreme_point_constraint {t in TIME}:
        (A[t] + sum {j in 1..new_cols} Trade1[t,j]*Lambda[j]) =0;

subject to convex_const:
        sum {j in 1..new_cols} Lambda[j] = 1;
```

A.3 Energy.Run File

```
model energy.mod;
data energy.dat;

let { t in TIME } trade_dual[t] := 0; # initial dual value
let new_cols:=0;

option solver cplex;

printf "\n ITERATION %d\n\n", new_cols+1;

drop extreme_point_constraint; drop convex_const;

let { t in TIME,r in REGION } export[t,r] := 0;
let { t in TIME,r in REGION } import[t,r] :=2000;
fix {r in REGION, t in TIME}import[t,r];
fix {r in REGION, t in TIME}export[t,r];
```

```
objective subproblem;

for {r in REGION}
{
        let R := r;
        solve;
}
display alpha;
repeat
{
        let new_cols := new_cols + 1;
        let  { t in TIME,r in REGION } stored_import [t,r,new_cols] := import[t,r];
        let  { t in TIME,r in REGION } stored_export [t,r,new_cols] := export[t,r];

let  c_hat[new_cols] :=
        sum{ r in REGION, t in TIME, i in PLANT_TYPE}
        (
        construction_variable_cost[t, i]*n[r, t, i] +
        operating_fixed_cost[t, i]*c[r, t, i] +
        operating_variable_cost[t, i] * g[r, t, i] *1000 +
        (fuel_variable_cost[t, i] / (efficiency[i]/3412)) * g[r, t, i] +
        sum{rc in REGION diff {r}} (import_cost[rc] * import[t,rc]) -
        export_profit * export[t,r]
        );

        let { t in TIME} Trade1[t,new_cols] :=
          sum {r in REGION} export [t,r] - sum {r in REGION}  import[t,r];

        include dsub.run;
        include rmaster.run;

        objective Total_cost;
        solve;

        let{ t in TIME:t>1990} trade_dual[t]:= extreme_point_constraint[t].dual;
        let alpha := convex_const.dual;
        display Lambda;
        printf "\nITERATION %d\n\n",new_cols+1;
```

```
        if new_cols >= 24 then break;

        include dmaster.run;
        include rsub.run;

        objective subproblem;

        for {r in REGION}
        {
                let R := r;
                solve;
        }

  };

let { t in TIME, r in REGION}
   convex_import[t,r]:=
     sum{j in 1..new_cols} Lambda[j] * stored_import [t,r,j];
let { t in TIME, r in REGION}
   convex_export[t,r]:= sum{j in 1..new_cols} Lambda[j] * stored_export [t,r,j];

let { t in TIME, r in REGION}
  final_import[t,r]:= convex_import[t,r] - convex_export[t,r];

let { t in TIME, r in REGION}
 final_export[t,r]:= convex_export[t,r] - convex_import[t,r];

display convex_import; display convex_export; display final_import; display
final_export;

let { t in TIME,r in REGION } import[t,r] := final_import[t,r];
let { t in TIME,r in REGION } export[t,r] := final_export[t,r];


for {t in TIME}
{
   for {r in REGION}
        if final_import[t,r] <0 then let import[t,r]:=0
};
```

```
for {t in TIME}
{
   for {r in REGION}
        if final_export[t,r] <0 then let export[t,r]:=0
};

display import;
display export;

include dmaster.run;
include rsub.run;
fix import;
fix export;

objective subproblem;

for {r in REGION}
{
        let R := r;
        solve;
}
        display n; display c; display d; display g;
        printf "    export[t,1] import[t,2] export[t,2] import[t,1] \n";
        printf {t in TIME}: "%3i  %8.2f  %8.2f  %8.2f  %8.2f \n\n",
         t, export[t,1], import[t,2], export[t,2], import[t,1];
```

A.4 Dmaster.Run File

```
fix Lambda;
fix A;

drop extreme_point_constraint;
drop convex_const;
```

A.5 Dsub.Run File

```
## dsub.run

fix c;
fix d;
fix n;
fix g;
fix import;
fix export;
fix build;

drop n_constraint1;
drop n_constraint2;
drop c_constraints;
drop new_capacity_restrictions;
drop export_bounds;
drop generation_constraints;
drop peak_demand_constraints;
drop base_load_constraints;
drop total_demand_constraints2;
drop hydro_addition1;
drop greenhouse_restrictions;
drop upper_bound_import;
```

A.6 Rmaster.Run File

```
## rmaster.run

unfix Lambda;
unfix A;

restore extreme_point_constraint;
restore convex_const;
```

A.7 Rsub.Run File

```
### rsub.run ###

unfix c;
unfix d;
unfix n;
unfix g;
unfix import;
unfix export;
unfix build;

restore n_constraint1;
restore n_constraint2;
restore c_constraints;
restore new_capacity_restrictions;
restore export_bounds;
restore generation_constraints;
restore peak_demand_constraints;
restore base_load_constraints;
restore total_demand_constraints2;
restore hydro_addition1 ;
drop greenhouse_restrictions;
restore upper_bound_import;
```

A.8 Energy.lst File

```
#### ENERGY.LST ####
 ITERATION 1

PLEX 3.0: optimal integer solution; objective -46347910990
PLEX 3.0: optimal integer solution; objective -25351212580
        Solution determined by presolve.
        Lambda [*] :=
        1  1

        ITERATION 2
PLEX 3.0: optimal integer solution within mipgap; objective
CPLEX 3.0: optimal integer solution within mipgap; objective
```

?LEX 3.0: optimal solution; objective 20866972700000
Lambda [*] :=
1 0.52858
2 0.47142

ITERATION 3
?LEX 3.0: optimal integer solution within mipgap; objective
?PLEX 3.0: optimal integer solution; objective 180254250600000
?LEX 3.0: optimal solution; objective 20866972700000
Lambda [*] :=
1 0.52858
2 0.47142
3 0

ITERATION 4
'LEX 3.0: optimal integer solution within mipgap; objective
?PLEX 3.0: optimal integer solution within mipgap; objective
?PLEX 3.0: optimal solution; objective 20866972700000
Lambda [*] :=
1 0.52858
2 0.47142
3 0
4 0

ITERATION 5
'LEX 3.0: optimal integer solution within mipgap; objective
?PLEX 3.0: optimal integer solution within mipgap; objective
?PLEX 3.0: optimal solution; objective 20866972700000
Lambda [*] :=
1 0.52858
2 0.47142
3 0
4 0
5 0

ITERATION 6
LEX 3.0: optimal integer solution within mipgap; objective
PLEX 3.0: optimal integer solution within mipgap; objective
PLEX 3.0: optimal solution; objective 20866972700000
Lambda [*] :=

1 0.52858
2 0.47142
3 0
4 0
5 0
6 0

ITERATION 7
CPLEX 3.0: optimal integer solution within mipgap; objective
CPLEX 3.0: optimal integer solution within mipgap; objective
CPLEX 3.0: optimal solution; objective 14624926590000
Lambda [*] :=
1 0.375351
2 0
3 0.0635517
4 0.200485
5 0.0325275
6 0.1276
7 0.200485

ITERATION 8
CPLEX 3.0: optimal integer solution; objective 27310402350000
CPLEX 3.0: optimal integer solution within mipgap; objective
27330161650000
CPLEX 3.0: optimal solution; objective 69065493340
Lambda [*] :=
1 0.0183567
2 0.314065
3 0
4 0
5 0
6 0
7 0
8 0.667578

ITERATION 9
CPLEX 3.0: optimal integer solution; objective 60071577080
CPLEX 3.0: optimal integer solution; objective 73851810690
CPLEX 3.0: optimal solution; objective 69065493340
Lambda [*] :=

1 0.0183567
2 0.314065
3 0
4 0
5 0
6 4.81638e-19
7 7.56749e-19
8 0.667578
9 0

ITERATION 10
CPLEX 3.0: optimal integer solution; objective 36344284040
CPLEX 3.0: optimal integer solution within mipgap; objective
54681404040
CPLEX 3.0: optimal solution; objective 69065493340
Lambda [*] :=
1 0.0183567
2 0.314065
3 0
4 0
5 0
6 0
7 0
8 0.667578
9 0
10 0

ITERATION 11
CPLEX 3.0: optimal integer solution; objective 34814987350
CPLEX 3.0: optimal integer solution; objective 53165716920
CPLEX 3.0: optimal solution; objective 69065493340
Lambda [*] :=
1 0.0183567
2 0.314065
3 0
4 0
5 0
6 0
7 0
8 0.667578

9 0
10 0
11 0

ITERATION 12
CPLEX 3.0: optimal integer solution; objective 36685719830
CPLEX 3.0: optimal integer solution within mipgap; objective 55388684190
CPLEX 3.0: optimal solution; objective 69065493340
Lambda [*] :=
1 0.0183567
2 0.314065
3 0
4 0
5 0
6 0
7 0
8 0.667578
9 0
10 0
11 0
12 0

ITERATION 13
CPLEX 3.0: optimal integer solution within mipgap; objective
CPLEX 3.0: optimal integer solution; objective 51850337870
CPLEX 3.0: optimal solution; objective 69065493340
Lambda [*] :=
1 0.0183567
2 0.314065
3 0
4 0
5 0
6 0
7 0
8 0.667578
9 0
10 0
11 0
12 0

13 0

ITERATION 14
CPLEX 3.0: optimal integer solution; objective 102081429800
CPLEX 3.0: optimal integer solution within mipgap; objective 108678310700
CPLEX 3.0: optimal solution; objective 67313195880
Lambda [*] :=
1 0.0183567
2 0
3 0
4 0
5 0
6 0
7 0
8 0.082563
9 0
10 0.0214692
11 0.0872121
12 0.0670493
13 0.508631
14 0.214719

ITERATION 15

CPLEX 3.0: optimal integer solution; objective 33103567510
CPLEX 3.0: optimal integer solution; objective 51826337110
CPLEX 3.0: optimal solution; objective 66265739170
Lambda [*] :=
1 0.0183567
2 0.00574729
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0.115936
11 0.0199312

12 0
13 0.585827
14 0.149658
15 0.104544

ITERATION 16
CPLEX 3.0: optimal integer solution; objective 32207846420
CPLEX 3.0: optimal integer solution within mipgap; objective
51566142440
CPLEX 3.0: optimal solution; objective 65573533810
Lambda [*] :=
1 0.0183567
2 0
3 0
4 0
5 0
6 0
7 0.0754804
8 0.0181846
9 0
10 0.090708
11 0
12 0
13 0.473394
14 0
15 0.233116
16 0.0907598

ITERATION 17
CPLEX 3.0: optimal integer solution within mipgap; objective
34184395520
CPLEX 3.0: optimal integer solution; objective 53275486240
CPLEX 3.0: optimal solution; objective 64502643740
Lambda [*] :=
1 0.0183567
2 0.0473022
3 0
4 0
5 0
6 0

7 0
8 0
9 0
10 0
11 0.243114
12 0
13 0.201291
14 0
15 0.195156
16 0.00490915
17 0.289872

ITERATION 18
CPLEX 3.0: optimal integer solution within mipgap; objective
CPLEX 3.0: optimal integer solution; objective 46064550780
CPLEX 3.0: optimal solution; objective 63968759320
Lambda [*] :=
1 0.0183567
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0.241895
12 0
13 0.192965
14 0
15 0.196294
16 0.00230441
17 0.283494
18 0.0646918

ITERATION 19
CPLEX 3.0: optimal integer solution; objective 27163558270
CPLEX 3.0: optimal integer solution; objective 46004798070
CPLEX 3.0: optimal solution; objective 62812339990

Lambda [*] :=
1 0.0183567
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0.0250623
13 0.249598
14 0
15 0.215365
16 0
17 0.140151
18 0.0882399
19 0.263228

ITERATION 20
CPLEX 3.0: optimal integer solution; objective 26659462590
CPLEX 3.0: optimal integer solution; objective 45544552490
CPLEX 3.0: optimal solution; objective 62622231530
Lambda [*] :=
1 0.0183567
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0.228554
14 0

15 0.196844
16 0
17 0.141915
18 0.101913
19 0.26112
20 0.0512968

ITERATION 21
CPLEX 3.0: optimal integer solution within mipgap; objective 24575655220
CPLEX 3.0: optimal integer solution; objective 43542241440
CPLEX 3.0: optimal solution; objective 61424445900
Lambda [*] :=

1 0.0183567	6 0	11 0	16 0	21 0.331413
2 0	7 0	12 0	17 0.217207	
3 0	8 0	13 0.0329785	18 0.0255296	
4 0	9 0	14 0	19 0.3081	
5 0	10 0	15 0	20 0.0664146	

ITERATION 22
CPLEX 3.0: optimal integer solution within mipgap; objective 25638830850
CPLEX 3.0: optimal integer solution; objective 44592355480
CPLEX 3.0: optimal solution; objective 61191662010
Lambda [*] :=

1 0.0183567	6 0	11 0	16 0	21 0.350399
2 0	7 0	12 0	17 0.222508	22 0.0968227
3 0	8 0	13 0	18 0.00360539	
4 0	9 0	14 0	19 0.261774	
5 0	10 0	15 0.0465345	20 0	

ITERATION 23
CPLEX 3.0: optimal integer solution; objective 25796035690
CPLEX 3.0: optimal integer solution; objective 44445112840
CPLEX 3.0: optimal solution; objective 61126307460
12 iterations (1 in phase I)
Lambda [*] :=

1 0.0183567	6 0	11 0	16 0	21 0.382375
2 0	7 0	12 0	17 0.218663	22 0.0619923
3 0	8 0	13 0	18 0	23 0.0185743

```
 4 0         9 0        14 0        19 0.27606
 5 0        10 0        15 0        20 0.0239784

ITERATION 24
CPLEX 3.0: optimal integer solution within mipgap; objective
24551020150
CPLEX 3.0: optimal integer solution; objective 44089913380
CPLEX 3.0: optimal solution; objective 61126307460
Lambda [*] :=
 1 0.0183567   6 0        11 0        16 0        21 0.382375
 2 0           7 0        12 0        17 0.218663  22 0.0619923
 3 0           8 0        13 0        18 0        23 0.0185743
 4 0           9 0        14 0        19 0.27606   24 0
 5 0          10 0        15 0        20 0.0239784


CPLEX 3.0: optimal integer solution within mipgap; objective
19708681530
CPLEX 3.0: optimal integer solution; objective 41634331730
n [1,*,*]
:   COAL  HYDRO   NGGT   NUCLEAR   OILGT    :=
1990  0      0       0       0       0
1995  0   1214   2848.63     0    3113.29
2000  0      0   1724.66     0       0
2005  0      0       0       0       0
2010  0      0       0       0       0
2015  0      0       0       0       0

[2,*,*]
:   COAL  HYDRO   NGGT   NUCLEAR   OILGT    :=
1990  0      0       0       0       0
1995  0      0    2063    1200    1200
2000  0      0    3633       0       0
2005  0      0    1200       0       0
2010  0      0       0       0       0
2015  0      0       0       0       0
;

c [1,*,*]
:   COAL   HYDRO   NGGT   NUCLEAR   OILGT   :=
```

```
1990  21820  4742  748  14426  2208
1995  21820  5956  3596.63  10766.7  5321.29
2000  21787  5956  5321.29  10766.7  5321.29
2005  21787  5956  5321.29  10766.7  5321.29
2010  21787  5956  5321.29  10766.7  5321.29
2015  21787  5956  5321.29  10766.7  5321.29

 [2,*,*]
:    COAL   HYDRO   NGGT   NUCLEAR   OILGT   :=
1990  24821  3794  0  5481  14
1995  24821  3794  2063  845.871  1214
2000  22133.3  3794  5696  0  1198
2005  21832.7  3794  6896  0  1198
2010  21832.7  3794  6896  0  1198
2015  21832.7  3794  6896  0  1198
;

d [1,*,*]
:  COAL HYDRO NGGT  NUCLEAR OILGT  :=
1990  0  0  0  0  0
1995  0  0  0  3659.26  0
2000  33  0  0  0  0
2005  0  0  0  0  0
2010  0  0  0  0  0
2015  0  0  0  0  0

 [2,*,*]
:    COAL   HYDRO NGGT   NUCLEAR  OILGT   :=
1990  0  0  0  0  0
1995  0  0  0  5835.13  0
2000  2687.71  0  0  845.871  16
2005  300.546  0  0  0  0
2010  0  0  0  0  0
2015  0  0  0  0  0
;

g [1,*,*]
:   COAL   HYDRO  NGGT   NUCLEAR OILGT   :=
1990  0  0  0  0  0
1995  129027  21226.6  0  63247.9  0
```

2000	134974	19338.8	0	61360.2	0
2005	134524	20654.9	0	62676.3	0
2010	136539	20749.9	0	62771.3	0
2015	140053	19763.1	0	61784.4	0

[2,*,*]

:	COAL	HYDRO	NGGT	NUCLEAR	OILGT	:=
1990	0	0	0	0	0	
1995	133196	15288.3	0	5186.88	0	
2000	138086	15288.3	0	0	0	
2005	141002	15288.3	0	0	0	
2010	142712	15288.3	0	0	0	
2015	142854	15288.3	0	0	0	

;

	export[t,1]	import[t,2]	export[t,2]	import[t,1]
1990	36.71	36.71	0.00	0.00
1995	316.64	316.64	0.00	0.00
2000	532.13	532.13	0.00	0.00
2005	381.89	381.89	0.00	0.00
2010	371.05	371.05	0.00	0.00
2015	483.70	483.70	0.00	0.00