// -------------------------------------------------------------- -\*- C++ -\*-

// File: fixcost1.cpp

// Version 12.7.1

// --------------------------------------------------------------------------

// Licensed Materials - Property of IBM

// 5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21

// Copyright IBM Corporation 2000, 2017. All Rights Reserved.

//

// US Government Users Restricted Rights - Use, duplication or

// disclosure restricted by GSA ADP Schedule Contract with

// IBM Corp.

// --------------------------------------------------------------------------

//

// fixcost1.cpp -- A production planning problem with fixed costs

/\* ------------------------------------------------------------

Problem Description

-------------------

A company must produce a product on a set of machines.

Each machine has limited capacity.

Producing a product on a machine has both a fixed cost

and a cost per unit of production.

Minimize the sum of fixed and variable costs so that the

company exactly meets demand.

------------------------------------------------------------ \*/

**Q1) Astro And Cosmo**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

//IloInt nbMachines = 1;/\*Machines producing Astro\*/

IloNumVar profit(env, 20.0);

/\*IloNumArray fixedCost(env, nbMachines, 1900.0, 820.0, 805.0, 464.0, 3912.00, 556.0);

IloNum demand = 22.0;\*/

//IloInt nbMachines1 = 1;/\*Machines producing cosmo\*/

IloNumVar profit1(env, 30.0);

IloModel model(env);

IloNumVar x(env, 0, 60);//machine producing astro units

IloNumVar y(env, 0, 50);//machine producing cosmo units

IloExpr tprofit(env);

tprofit = 20 \* x + 30 \* y;

// Objective: maximum prfit

model.add(IloMaximize(env, tprofit));

// Constraint: meet demand

model.add(x + 2 \* y <= 120);

model.add(x <= 60);

model.add(y <= 50);

model.add(x >= 0);

model.add(y >= 0);

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Obj " << cplex.getObjValue() << endl;

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

env.end();

system("pause");

return 0;

}

}

**Q2) Companies Two Products**

int

main(int, char\*\*)

{

IloEnv env;

try {

IloNumVar profit(env, 3000.0);

/\*IloNumArray fixedCost(env, nbMachines, 1900.0, 820.0, 805.0, 464.0, 3912.00, 556.0);

IloNum demand = 22.0;\*/

//IloInt nbMachines1 = 1;/\*Machines producing cosmo\*/

IloNumVar profit1(env, 5000.0);

IloModel model(env);

IloNumVar x(env, 0, IloInfinity);//machine producing astro units

IloNumVar y(env, 0, IloInfinity);//machine producing cosmo units

IloExpr tprofit(env);

tprofit = 3000 \* x + 5000 \* y;

// Objective: maximum prfit

model.add(IloMaximize(env, tprofit));

// Constraint: meet demand

model.add(x <= 4);

model.add(2\* y <= 12);

model.add(3\*x+ 2\*y <= 18);

model.add(x >= 0);

model.add(y >= 0);

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Obj " << cplex.getObjValue() << endl;

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

cout << "The number of Astro unit produced in optimized solution" << cplex.getValue(x) << endl;

cout << "The number of Astro unit produced in optimized solution" << cplex.getValue(y) << endl;

cout << endl;

cout << "----------------------------------------" << endl;

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q3)I Primial Function**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

//IloNumVar profit(env, 3000.0);

/\*IloNumArray fixedCost(env, nbMachines, 1900.0, 820.0, 805.0, 464.0, 3912.00, 556.0);

IloNum demand = 22.0;\*/

//IloInt nbMachines1 = 1;/\*Machines producing cosmo\*/

//IloNumVar profit1(env, 5000.0);

IloModel model(env);

IloNumVar a(env, 0, IloInfinity);//machine producing astro units

IloNumVar b(env, 0, IloInfinity);//machine producing astro units

IloNumVar c(env, 0, IloInfinity);//machine producing astro units

IloNumVar d(env, 0, IloInfinity);//machine producing astro units

IloNumVar e(env, 0, IloInfinity);//machine producing cosmo units

IloExpr tprofit(env);

tprofit = 0.0043\*a + 0.027\*b + 0.025\*c + 0.022\*d + 0.045\*e ;

// Objective: maximum prfit

model.add(IloMaximize(env, tprofit));

// Constraint: meet demand

model.add(a + b + c + d + e <=10);

model.add(-b - c - d <= -4);

model.add(0.6\*a + 0.5\*b - 0.4\*d + 0.3\*e <= 0);

model.add(4\*a + 10\*b - c - 2\*d - 3\*e <= 0);

model.add(a >= 0);

model.add(b >= 0);

model.add(c >= 0);

model.add(d >= 0);

model.add(e >= 0);

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Objective Funtion " << cplex.getObjValue() << endl;

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

cout << "Value of A in optimized solution" << cplex.getValue(a) << endl;

cout << "Value of B in optimized solution" << cplex.getValue(b) << endl;

cout << "Value of C in optimized solution" << cplex.getValue(c) << endl;

cout << "Value of D in optimized solution" << cplex.getValue(d) << endl;

cout << "Value of E in optimized solution" << cplex.getValue(e) << endl;

cout << endl;

cout << "----------------------------------------" << endl;

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**ii)Dual**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

//IloNumVar profit(env, 3000.0);

/\*IloNumArray fixedCost(env, nbMachines, 1900.0, 820.0, 805.0, 464.0, 3912.00, 556.0);

IloNum demand = 22.0;\*/

//IloInt nbMachines1 = 1;/\*Machines producing cosmo\*/

//IloNumVar profit1(env, 5000.0);

IloModel model(env);

IloNumVar a(env, 0, IloInfinity);//machine producing astro units

IloNumVar b(env, 0, IloInfinity);//machine producing astro units

IloNumVar c(env, 0, IloInfinity);//machine producing astro units

IloNumVar d(env, 0, IloInfinity);//machine producing astro units

IloExpr tprofit(env);

tprofit = 10 \* a - 4 \* b;

// Objective: maximum prfit

model.add(IloMinimize(env, tprofit));

// Constraint: meet demand

model.add(a + 0.6\*c + 4\*d >= 0.043);

model.add(a - b + 0.6\*c + 10 \* d >= 0.027);

model.add(a - b - 0.4\*c - d >= 0.025);

model.add(a - b - 0.4\*c - 2 \* d >= 0.022);

model.add(a + 3.6\*c - 3 \* d >= 0.045);

model.add(a >= 0);

model.add(b >= 0);

model.add(c >= 0);

model.add(d >= 0);

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Objective Funtion " << cplex.getObjValue() << endl;

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

cout << "Value of A in optimized solution" << cplex.getValue(a) << endl;

cout << "Value of B in optimized solution" << cplex.getValue(b) << endl;

cout << "Value of C in optimized solution" << cplex.getValue(c) << endl;

cout << "Value of D in optimized solution" << cplex.getValue(d) << endl;

cout << endl;

cout << "----------------------------------------" << endl;

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q4) Lee Hye paper optimization**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float heatdemand[24] = { 130.0, 125.0, 120.0, 130.0, 135.0, 145.0, 155.0, 175.0, 185.0, 190.0, 193.0, 190.0, 190.0, 190.0, 187.0, 185.0, 185.0, 195.0, 200.0, 205.0, 200.0, 195.0, 190.0, 175.0 };//Heat demand w.r.t time

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 151.0, 150.0, 150.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float heatbuy[24] = { 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0 };//heat buying price from DHS w.r.t time

float heatsell[24] = { 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 78.0, 78.0, 78.0, 78.0, 78.0, 78.0 };//heat selling price to DHS w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dg1cost = 135;//Cost for generating per kW power

int dg2cost = 140;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int chp2cost = 145;//Cost for generating per kW power

int hobcost = 80;//Cost for generating per kW power

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray dhsheatbuy(env, totaltime, 0, IloInfinity);//DHS heat bought

IloNumVarArray dhsheatsell(env, totaltime, 0, IloInfinity);//DHS heat sold

IloNumVarArray chp1heat(env, totaltime, 0, IloInfinity);//heat generation at chpA

IloNumVarArray chp2heat(env, totaltime, 0, IloInfinity);//heat generation at chpB

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray dg2power(env, totaltime, 0, 80);//DGB power generation range

IloNumVarArray chp1power(env, totaltime, 30, 60);//CHPA power generation range

IloNumVarArray chp2power(env, totaltime, 50, 100);//CHPB power generation range

IloNumVarArray HOBA(env, totaltime, 0, 80);//HOB heat generation range

IloExpr solutio(env, totaltime);

int i;

for (i = 0; i < totaltime; i++)

{

solutio += chp1cost\*chp1power[i] + chp2cost\*chp2power[i] + dg1cost\*dg1power[i] + dg2cost\*dg2power[i] + hobcost\*HOBA[i] + gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i] + heatbuy[i] \* dhsheatbuy[i] - heatsell[i] \* dhsheatsell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(dg1power[i] <= 100);

model.add(0 <= dg2power[i]);

model.add(dg2power[i] <= 80);

model.add(30 <= chp1power[i]);

model.add(chp1power[i] <= 60);

model.add(50 <= chp2power[i]);

model.add(chp2power[i] <= 100);

model.add(0 <= HOBA[i]);

model.add(HOBA[i] <= 80);

model.add(chp1heat[i] == chp1power[i] \* 1.2);

model.add(chp2heat[i] == chp2power[i] \* 0.8);

model.add(chp1power[i] + chp2power[i] + dg1power[i] + dg2power[i] + Gridpowerbuy[i] + pvpower[i] == electricdemand[i] + Gridpowersell[i]);

model.add(chp1heat[i] + chp2heat[i] + HOBA[i] + dhsheatbuy[i] == heatdemand[i] + dhsheatsell[i]);

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

/\*cout << "DGA" << " " << "DGB" << " " << "CHPA" << " " << "CHPB" << " " << "HOBA" << " " << "Grid" << " " << "PV" << endl;\*/

cout << "Grid sell" << " " << "CHPA heat" << " " << "CHPB heat" << " " << "DHS buy" << " " << "DHS sell" << endl;

for (i = 0; i < totaltime; i++)

{

/\*cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(dg2power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(chp2power[i]) << " " << cplex.getValue(HOBA[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " <<pvpower[i]<< endl;\*/

cout << cplex.getValue(Gridpowersell[i]) << " " << cplex.getValue(chp1heat[i]) << " " << cplex.getValue(chp2heat[i]) << " " << cplex.getValue(dhsheatbuy[i]) << " " << cplex.getValue(dhsheatsell[i]) << endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

//cout << "Value of A in optimized solution" << cplex.getValue(a) << endl;

//cout << "Value of B in optimized solution" << cplex.getValue(b) << endl;

//cout << "Value of C in optimized solution" << cplex.getValue(c) << endl;

//cout << "Value of D in optimized solution" << cplex.getValue(d) << endl;

//cout << endl;

//cout << "----------------------------------------" << endl;

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q5) Lee Hye paper with battery(With loss)**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float heatdemand[24] = { 130.0, 125.0, 120.0, 130.0, 135.0, 145.0, 155.0, 175.0, 185.0, 190.0, 193.0, 190.0, 190.0, 190.0, 187.0, 185.0, 185.0, 195.0, 200.0, 205.0, 200.0, 195.0, 190.0, 175.0 };//Heat demand w.r.t time

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 151.0, 150.0, 150.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float heatbuy[24] = { 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0 };//heat buying price from DHS w.r.t time

float heatsell[24] = { 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 78.0, 78.0, 78.0, 78.0, 78.0, 78.0 };//heat selling price to DHS w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dg1cost = 145;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int hobcost = 80;//Cost for generating per kW power

int i;

for(i = 0; i < 24; i++ )

{

electricdemand[i] = 0.7\*electricdemand[i];

}

float socini = 0.2;

int pbmax = 100;

float effin = 0.95;

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray dhsheatbuy(env, totaltime, 0, IloInfinity);//DHS heat bought

IloNumVarArray dhsheatsell(env, totaltime, 0, IloInfinity);//DHS heat sold

IloNumVarArray chp1heat(env, totaltime, 0, IloInfinity);//heat generation at chpA

IloNumVarArray statoc(env, totaltime, 0, 1);//battery storage capacity

IloNumVarArray besschg(env, totaltime, 0, 100);//battery charging

IloNumVarArray bessdis(env, totaltime, 0, 100);//battery discharging

IloNumVarArray besssys(env, totaltime, 0, 100);//efficient energy

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 30, 100);//CHPA power generation range

IloNumVarArray HOBA(env, totaltime, 0, 80);//HOB heat generation range

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += chp1cost\*chp1power[i] + dg1cost\*dg1power[i] + hobcost\*HOBA[i] + gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i] + heatbuy[i] \* dhsheatbuy[i] - heatsell[i] \* dhsheatsell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(dg1power[i] <= 100);

model.add(0 <= chp1power[i]);

model.add(chp1power[i] <= 100);

model.add(0 <= HOBA[i]);

model.add(HOBA[i] <= 80);

model.add(chp1heat[i] == chp1power[i] \* 1.2);

model.add(0<=statoc[i]);

model.add(statoc[i]<=1);

/\*model.add(0 <= besschg[i]);

model.add(besschg[i]<= pbmax - statoc[i] \* pbmax);

model.add(0 <= bessdis[i]);

model.add(bessdis[i] <= pbmax \* statoc[i]);\*/

/\*model.add(0 <= besssys[i]);

model.add(besssys[i] <= pbmax);\*/

if (i == 0)

{

model.add(statoc[i] == socini + ((0.95\*besschg[i] - (bessdis[i]/0.95)) / pbmax));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (pbmax \* (1 - socini) / 0.95));

model.add(bessdis[i] <= (pbmax \* socini \* 0.95));

}

else

{

model.add(statoc[i] == statoc[i - 1] + ((0.95\*besschg[i] - (bessdis[i] / 0.95)) / pbmax));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (pbmax \* (1 - statoc[i-1])) / 0.95);

model.add(bessdis[i] <= pbmax \* statoc[i-1] \* 0.95);

}

model.add(chp1power[i] + dg1power[i] + Gridpowerbuy[i] + pvpower[i] + bessdis[i] - besschg[i] - Gridpowersell[i] == electricdemand[i] );

model.add(chp1heat[i] + HOBA[i] + dhsheatbuy[i] == heatdemand[i] + dhsheatsell[i]);

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" << " " << "CHPA" << " " << "Grid" << " " << "Grid power sell" << " " << "PV" << " " << "DEmand" << " " << "SOC" << " " << "charging" << " " << "discharging" << endl;

/\*cout << "Grid sell" << " " << "CHPA heat" << " " << "CHPB heat" << " " << "DHS buy" << " " << "DHS sell" << endl;\*/

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << " " << cplex.getValue(statoc[i] \* 100) << " " << cplex.getValue(besschg[i]) << " " << cplex.getValue(bessdis[i]) << " " << gridbuy[i] << " " << gridsell[i]<< endl;

/\*cout << cplex.getValue(Gridpowersell[i]) << " " << cplex.getValue(chp1heat[i]) << " " << cplex.getValue(chp2heat[i]) << " " << cplex.getValue(dhsheatbuy[i]) << " " << cplex.getValue(dhsheatsell[i]) << endl;\*/

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

//cout << "Value of A in optimized solution" << cplex.getValue(a) << endl;

//cout << "Value of B in optimized solution" << cplex.getValue(b) << endl;

//cout << "Value of C in optimized solution" << cplex.getValue(c) << endl;

//cout << "Value of D in optimized solution" << cplex.getValue(d) << endl;

//cout << endl;

//cout << "----------------------------------------" << endl;

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q)HESS AND BESS**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 151.0, 150.0, 150.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

float heatdemand[24] = { 130.0, 125.0, 120.0, 130.0, 135.0, 145.0, 155.0, 175.0, 185.0, 190.0, 193.0, 190.0, 190.0, 190.0, 187.0, 185.0, 185.0, 195.0, 200.0, 205.0, 200.0, 195.0, 190.0, 175.0 };//Heat demand w.r.t time

float heatbuy[24] = { 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 77.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 82.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0 };//heat buying price from DHS w.r.t time

float heatsell[24] = { 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 78.0, 78.0, 78.0, 78.0, 78.0, 78.0 };//heat selling price to DHS w.r.t time

int hobcost = 80;//Cost for generating per kW power

int dg1cost = 145;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int i;

for(i = 0; i < 24; i++ )

{

electricdemand[i] = 0.7\*electricdemand[i];

}

float socini = 0.2;

float hessini =0.2;

int pbmax = 100;

int hbmax = 100;

float effin = 0.99;

float hloss = 0.06;

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray statoc(env, totaltime, 0, 1);//battery storage capacity

IloNumVarArray besschg(env, totaltime, 0, 100);//battery charging

IloNumVarArray bessdis(env, totaltime, 0, 100);//battery discharging

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 30, 100);//CHPA power generation range

IloNumVarArray dhsheatbuy(env, totaltime, 0, IloInfinity);//DHS heat bought

IloNumVarArray dhsheatsell(env, totaltime, 0, IloInfinity);//DHS heat sold

IloNumVarArray chp1heat(env, totaltime, 0, IloInfinity);//heat generation at chpA

IloNumVarArray HOBA(env, totaltime, 0, 80);//HOB heat generation range

IloNumVarArray hess(env, totaltime, 0, 1);//heat storage tank capacity

IloNumVarArray hessstore(env, totaltime, 0, 100);//battery charging

IloNumVarArray hessdis(env, totaltime, 0, 100);//battery discharging

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += chp1cost\*chp1power[i] + dg1cost\*dg1power[i] + hobcost\*HOBA[i] + gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i] + heatbuy[i] \* dhsheatbuy[i] - heatsell[i] \* dhsheatsell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(dg1power[i] <= 100);

model.add(0 <= chp1power[i]);

model.add(chp1power[i] <= 100);

model.add(HOBA[i] <= 80);

model.add(chp1heat[i] == chp1power[i] \* 1.2);

model.add(0<=statoc[i]);

model.add(statoc[i]<=1);

model.add(0 <= hess[i]);

model.add(hess[i] <= 1);

if (i == 0)

{

model.add(statoc[i] == socini + ((effin\*besschg[i] - (bessdis[i]/ effin)) / pbmax));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (pbmax \* (1 - socini) / effin));

model.add(bessdis[i] <= (pbmax \* socini \* effin));

model.add(hess[i]==hessini\*0.97 + (hessstore[i] - hessdis[i]) / hbmax );

model.add(0<=hessstore[i]);

model.add(hessstore[i]<=(hbmax \* (1 - hessini))/0.97);

model.add(0 <= hessdis[i]);

model.add(hessdis[i] <= hbmax \* hessini\*0.97);

}

else

{

model.add(statoc[i] == statoc[i - 1] + ((effin\*besschg[i] - (bessdis[i] / effin)) / pbmax));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (pbmax \* (1 - statoc[i-1])) / effin);

model.add(bessdis[i] <= pbmax \* statoc[i-1] \* effin);

model.add(hess[i] == hess[i-1]\*0.97 + (hessstore[i] - hessdis[i])/hbmax);

model.add(0 <= hessstore[i]);

model.add(hessstore[i] <= (hbmax \* (1 - hess[i-1]))/0.97);

model.add(0 <= hessdis[i]);

model.add(hessdis[i] <= hbmax \* hess[i-1]\*0.97);

}

model.add(chp1power[i] + dg1power[i] + Gridpowerbuy[i] + pvpower[i] + bessdis[i] - besschg[i] - Gridpowersell[i] == electricdemand[i] );

model.add(chp1heat[i] + HOBA[i] + dhsheatbuy[i] + hessdis[i] == heatdemand[i] + dhsheatsell[i] + hessstore[i] );

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" << " " << "CHPA" << " " << "Grid" << " " << "Grid power sell" << " " << "PV" << " " << "DEmand" << " " << "SOC" << " " << "charging" << " " << "discharging" << endl;

/\*cout <<"CHPA heat" << " " << "HOB" << " " << "hess" << " " << "heat stored" << " " << "heat dis" << " " << "DHS buy" << " " << "DHS sell" << " " << "Heat demand" << endl;\*/

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << " " << cplex.getValue(statoc[i] \* 100) << " " << cplex.getValue(besschg[i]) << " " << cplex.getValue(bessdis[i]) << " " << gridbuy[i] << " " << gridsell[i]<< endl;

/\*cout << cplex.getValue(chp1heat[i]) << " " << cplex.getValue(HOBA[i]) << " " << cplex.getValue(hess[i]\*100) << " " << cplex.getValue(hessstore[i]) << " " << cplex.getValue(hessdis[i]) << " " << cplex.getValue(dhsheatbuy[i]) << " " << cplex.getValue(dhsheatsell[i]) << " " << heatdemand[i] << endl;\*/

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

//cout << "Value of A in optimized solution" << cplex.getValue(a) << endl;

//cout << "Value of B in optimized solution" << cplex.getValue(b) << endl;

//cout << "Value of C in optimized solution" << cplex.getValue(c) << endl;

//cout << "Value of D in optimized solution" << cplex.getValue(d) << endl;

//cout << endl;

//cout << "----------------------------------------" << endl;

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q1) Simple system with bess only**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 151.0, 150.0, 150.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dg1cost = 145;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int i;

for(i = 0; i < 24; i++ )

{

electricdemand[i] = 0.7\*electricdemand[i];

}

float socini = 0.2;//already charged battery

int pbmax = 100;//battery maximum capacity

float effin = 0.99;//battery effciency

int pini = 0;//initial power

float ru = 0.6;//rampup factor

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray statoc(env, totaltime, 0, 1);//battery storage capacity

IloNumVarArray besschg(env, totaltime, 0, 100);//battery charging

IloNumVarArray bessdis(env, totaltime, 0, 100);//battery discharging

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 30, 100);//CHPA power generation range

IloNumVarArray rampup(env, totaltime, 0, 200);//CHPA power generation range

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += chp1cost\*chp1power[i] + dg1cost\*dg1power[i] + gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(dg1power[i] <= 100);

model.add(0 <= chp1power[i]);

model.add(chp1power[i] <= 100);

model.add(0<=statoc[i]);

model.add(statoc[i]<=1);

if (i == 0)

{

model.add(statoc[i] == socini + ((effin\*besschg[i] - (bessdis[i]/ effin)) / pbmax));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (pbmax \* (1 - socini) / effin));

model.add(bessdis[i] <= (pbmax \* socini \* effin));

}

else

{

model.add(statoc[i] == statoc[i - 1] + ((effin\*besschg[i] - (bessdis[i] / effin)) / pbmax));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (pbmax \* (1 - statoc[i-1])) / effin);

model.add(bessdis[i] <= pbmax \* statoc[i-1] \* effin);

}

model.add(chp1power[i] + dg1power[i] + Gridpowerbuy[i] + pvpower[i] + bessdis[i] - besschg[i] - Gridpowersell[i] == electricdemand[i] );

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" << " " << "CHPA" << " " << "Grid" << " " << "Grid power sell" << " " << "PV" << " " << "DEmand" << " " << "SOC" << " " << "charging" << " " << "discharging" << endl;

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << " " << cplex.getValue(statoc[i] \* 100) << " " << cplex.getValue(besschg[i]) << " " << cplex.getValue(bessdis[i]) << " " << gridbuy[i] << " " << gridsell[i]<< endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q) Bess RAMPUP**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 151.0, 150.0, 150.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dg1cost = 145;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int i;

int pini = 0;//initial power

float ru = 0.3;//rampup factor

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 0, 100);//CHPA power generation range

IloNumVarArray rampup(env, totaltime, 0, 200);//CHPA power generation range

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += chp1cost\*chp1power[i] + dg1cost\*dg1power[i] + gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(dg1power[i] <= 100);

model.add(0 <= chp1power[i]);

model.add(chp1power[i] <= 100);

if (i == 0)

{

model.add((dg1power[i] - pini) / 100 <= ru);

model.add((chp1power[i] - pini) / 100 <= ru);

}

else

{

model.add((dg1power[i] - dg1power[i-1]) / 100 <= ru);

model.add((chp1power[i] - chp1power[i-1]) / 100 <= ru);

}

model.add(chp1power[i] + dg1power[i] + Gridpowerbuy[i] + pvpower[i] - Gridpowersell[i] == electricdemand[i] );

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" << " " << "CHPA" << " " << "Grid" << " " << "Grid power sell" << " " << "PV" << " " << "DEmand" << endl;

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << " " << gridbuy[i] << " " << gridsell[i]<< endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q) RAMP DOWN**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 151.0, 150.0, 150.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dg1cost = 145;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int i;

int pini = 0;//initial power

float ru = 0.3;//rampup factor

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 0, 100);//CHPA power generation range

IloNumVarArray rampup(env, totaltime, 0, 200);//CHPA power generation range

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += chp1cost\*chp1power[i] + dg1cost\*dg1power[i] + gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(dg1power[i] <= 100);

model.add(0 <= chp1power[i]);

model.add(chp1power[i] <= 100);

if (i == 0)

{

model.add((pini - dg1power[i] ) / 100 <= ru);

model.add((pini - chp1power[i]) / 100 <= ru);

}

else

{

model.add((dg1power[i - 1] - dg1power[i]) / 100 <= ru);

model.add((chp1power[i - 1] - chp1power[i]) / 100 <= ru);

}

model.add(chp1power[i] + dg1power[i] + Gridpowerbuy[i] + pvpower[i] - Gridpowersell[i] == electricdemand[i] );

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" << " " << "CHPA" << " " << "Grid" << " " << "Grid power sell" << " " << "PV" << " " << "DEmand" << endl;

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << " " << gridbuy[i] << " " << gridsell[i]<< endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q) Start up cost**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 151.0, 151.0, 151.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dg1cost = 145;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int i;

int dgini = 190;

int dgsd =170;

int chpini = 300;

int chpsd = 280;

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 0, 100);//CHPA power generation range

IloNumVarArray fused(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray fusedk(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray startup(env, totaltime, 0, 1, ILOINT);//fuse for dg

//IloNumVarArray shutd(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray suchp(env, totaltime, 0, 1, ILOINT);//fuse for dg

//IloNumVarArray sdchp(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += startup[i]\*dgini/\* + shutd[i]\*dgsd \*/+suchp[i]\*chpini/\*+sdchp[i]\*chpsd\*/ +chp1cost\*chp1power[i] + dg1cost\*dg1power[i]+ gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(0 <= chp1power[i]);

//model.add(dg1power[i] <= 100);

//model.add(chp1power[i] <= 100);

model.add(dg1power[i] <= 100 \* fused[i]);

model.add(chp1power[i] <= 100 \* fusedk[i]);

if (i == 0)

{

model.add(startup[i]>=fused[i]-0);

model.add(suchp[i] >= fusedk[i] - 0);

/\*model.add(shutd[i] == 0 - fused[i]);

model.add(sdchp[i] == 0 - fusedk[i]);\*/

}

else

{

model.add(startup[i] >= fused[i] - fused[i - 1] );

model.add(suchp[i] >= fusedk[i] - fusedk[i - 1]);

/\* model.add(shutd[i] == fused[i - 1] - fused[i]);

model.add(sdchp[i] == fusedk[i - 1] - fusedk[i]);\*/

}

model.add(chp1power[i] + dg1power[i] + Gridpowerbuy[i] + pvpower[i] - Gridpowersell[i] == electricdemand[i]);

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" << " " << "CHPA" << " " << "Grid" << " " << "Grid power sell" << " " << "PV" << " " << "DEmand" << " " << "fuse" << endl;

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << " " << cplex.getValue(fused[i]) << " " << cplex.getValue(fusedk[i]) << endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q)RAMP UP AND DOWN**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 150.0, 150.0, 151.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dg1cost = 145;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int i;

int dgini = 190;

int dgsd =170;

int chpini = 300;

int chpsd = 280;

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 0, 100);//CHPA power generation range

IloNumVarArray fused(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray fusedk(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray startup(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray shutd(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray suchp(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray sdchp(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += startup[i]\*dgini + shutd[i]\*dgsd +suchp[i]\*chpini+sdchp[i]\*chpsd +chp1cost\*chp1power[i] + dg1cost\*dg1power[i]+ gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(fusedk[i]\* 5 <= chp1power[i]);

//model.add(dg1power[i] <= 100);

//model.add(chp1power[i] <= 100);

model.add(dg1power[i] <= 100 \* fused[i]);

model.add(chp1power[i] <= 100 \* fusedk[i]);

if (i == 0)

{

model.add(startup[i]>=fused[i]-0);

model.add(suchp[i] >= fusedk[i] - 0);

model.add(shutd[i] >= 0 - fused[i]);

model.add(sdchp[i] >= 0 - fusedk[i]);

}

else

{

model.add(startup[i] >= fused[i] - fused[i - 1] );

model.add(suchp[i] >= fusedk[i] - fusedk[i - 1]);

model.add(shutd[i] >= fused[i - 1] - fused[i]);

model.add(sdchp[i] >= fusedk[i - 1] - fusedk[i]);

}

model.add(chp1power[i] + dg1power[i] + Gridpowerbuy[i] + pvpower[i] - Gridpowersell[i] == electricdemand[i]);

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" << " " << "CHPA" << " " << "Grid" << " " << "Grid power sell" << " " << "PV" << " " << "DEmand" << " " << "fuse" << endl;

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << " " << cplex.getValue(startup[i]) << " " << cplex.getValue(suchp[i]) << " " << cplex.getValue(shutd[i]) << " " << cplex.getValue(sdchp[i]) << endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q) STARTUP AND DOWN**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0,z 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 150.0, 150.0, 151.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dg1cost = 145;//Cost for generating per kW power

int chp1cost = 150;//Cost for generating per kW power

int i;

int dgini = 190;

int dgsd =170;

int chpini = 300;

int chpsd = 280;

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray dg1power(env, totaltime, 0, 100);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 0, 100);//CHPA power generation range

IloNumVarArray fused(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray fusedk(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray startup(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray shutd(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray suchp(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloNumVarArray sdchp(env, totaltime, 0, 1, ILOINT);//fuse for dg

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += startup[i]\*dgini + shutd[i]\*dgsd +suchp[i]\*chpini+sdchp[i]\*chpsd +chp1cost\*chp1power[i] + dg1cost\*dg1power[i]+ gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(fusedk[i]\* 5 <= chp1power[i]);

//model.add(dg1power[i] <= 100);

//model.add(chp1power[i] <= 100);

model.add(dg1power[i] <= 100 \* fused[i]);

model.add(chp1power[i] <= 100 \* fusedk[i]);

if (i == 0)

{

model.add(startup[i]>=fused[i]-0);

model.add(suchp[i] >= fusedk[i] - 0);

model.add(shutd[i] >= 0 - fused[i]);

model.add(sdchp[i] >= 0 - fusedk[i]);

}

else

{

model.add(startup[i] >= fused[i] - fused[i - 1] );

model.add(suchp[i] >= fusedk[i] - fusedk[i - 1]);

model.add(shutd[i] >= fused[i - 1] - fused[i]);

model.add(sdchp[i] >= fusedk[i - 1] - fusedk[i]);

}

model.add(chp1power[i] + dg1power[i] + Gridpowerbuy[i] + pvpower[i] - Gridpowersell[i] == electricdemand[i]);

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" << " " << "CHPA" << " " << "Grid" << " " << "Grid power sell" << " " << "PV" << " " << "DEmand" << " " << "fuse" << endl;

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << " " << cplex.getValue(chp1power[i]) << " " << cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << " " << cplex.getValue(startup[i]) << " " << cplex.getValue(suchp[i]) << " " << cplex.getValue(shutd[i]) << " " << cplex.getValue(sdchp[i]) << endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q)Piecewise linearization**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricdemand[24] = { 185.0, 185.0, 190.0, 190.0, 225.0, 145.0, 250.0, 280.0, 305.0, 310.0, 305.0, 285.0, 275.0, 260.0, 240.0, 195.0, 190.0, 205.0, 250.0, 280.0, 335.0, 340.0, 265.0, 200.0 };//Electicity demand w.r.t time

float gridbuy[24] = { 138.0, 139.0, 142.5, 150.0, 150.5, 151.0, 156.0, 159.0, 161.0, 154.0, 152.0, 152.0, 151.5, 151.0, 150.0, 150.0, 154.0, 156.0, 163.0, 163.5, 163.5, 157.0, 150.0, 147.0 };//buying price from grid w.r.t time

float gridsell[24] = { 130.0, 130.5, 134.0, 138.0, 138.5, 139.0, 143.0, 148.0, 150.0, 142.0, 140.0, 140.0, 139.5, 139.0, 138.0, 138.0, 142.0, 145.0, 150.0, 151.0, 151.0, 147.0, 138.0, 135.0 };//selling price to grid w.r.t time

float pvpower[24] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0, 10.0, 12.0, 18.0, 26.0, 28.0, 24.0, 21.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };//Power generated by PV w.r.t time

int dgrange = 134;//Cost for generating per kW power

int dgrange1 = 139;

int dgrange2 = 146;

int i;

int chprange = 135;

int chprange1 = 145;

int chprange2 = 150;

for (i = 0; i < 24; i++)

{

electricdemand[i] = 0.7\*electricdemand[i];

}

float socini = 0.2;//already charged battery

int pbmax = 100;//battery maximum capacity

float effin = 0.99;//battery effciency

int pini = 0;//initial power

IloModel model(env);

IloNumVarArray Gridpowerbuy(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray dg1power(env, totaltime, 0, 30);//DGA power generation range

IloNumVarArray dg2power(env, totaltime, 0, 30);//DGA power generation range

IloNumVarArray dg3power(env, totaltime, 0, 40);//DGA power generation range

IloNumVarArray chp1power(env, totaltime, 0, 30);//CHPA power generation range

IloNumVarArray chp2power(env, totaltime, 0, 30);//CHPA power generation range

IloNumVarArray chp3power(env, totaltime, 0, 40);//CHPA power generation rang

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += chprange\*chp1power[i]+ chprange1\*chp2power[i] + chprange2\*chp3power[i]+ dgrange\*dg1power[i] + dgrange1\*dg2power[i]+ dgrange2\*dg3power[i]+ gridbuy[i] \* Gridpowerbuy[i] - gridsell[i] \* Gridpowersell[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(dg1power[i] <= 30);

model.add(0 <= dg2power[i]);

model.add(dg2power[i] <= 30);

model.add(0 <= dg3power[i]);

model.add(dg3power[i] <= 40);

model.add(0 <= chp1power[i]);

model.add(chp1power[i] <=30);

model.add(0 <= chp1power[i]);

model.add(chp1power[i] <= 30);

model.add(0 <= chp1power[i]);

model.add(chp1power[i]<=40);

/\*model.add(0 <= statoc[i]);

model.add(statoc[i] <= 1);\*/

/\* model.add((chp1power[i] <= 30) <= (range1[i] == 1));\*/

/\*model.add((chp1power[i] >= 51) <= (range1[i] == 149));\*/

/\*model.add((chp1power[i] >= 51) <= (range2[i] == 149));

model.add((chp1power[i] <= 50) <= (range2[i] == 0));\*/

/\*if (i == 0)

{

model.add(statoc[i] == socini + ((effin\*besschg[i] - (bessdis[i] / effin)) / pbmax));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (pbmax \* (1 - socini) / effin));

model.add(bessdis[i] <= (pbmax \* socini \* effin));

}

else

{

model.add(statoc[i] == statoc[i - 1] + ((effin\*besschg[i] - (bessdis[i] / effin)) / pbmax));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (pbmax \* (1 - statoc[i - 1])) / effin);

model.add(bessdis[i] <= pbmax \* statoc[i - 1] \* effin);\*/

/\*}\*/

model.add(chp1power[i] + chp2power[i] + chp3power[i]+ dg1power[i] + dg2power[i]+ dg3power[i]+ Gridpowerbuy[i] + pvpower[i] /\*+ bessdis[i] - besschg[i] \*/- Gridpowersell[i] == electricdemand[i]);

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" <<" "<< "CHPA" << '/t' << "Grid" << '/t' << "Grid power sell" << '/t' << "PV" << " " << "DEmand" << " " << "SOC" << " " << "charging" << " " << "discharging" << endl;

for (i = 0; i < totaltime; i++)

{

cout << /\*cplex.getValue(chp1power[i]) << " " << cplex.getValue(chp2power[i]) << " " << cplex.getValue(chp3power[i]) << " " << cplex.getValue(chp1power[i]+chp2power[i]+chp3power[i]) <<" " <<cplex.getValue(dg1power[i]) << " " << cplex.getValue(dg2power[i]) << " " << cplex.getValue(dg3power[i]) << " " << cplex.getValue(dg1power[i] + dg2power[i] + dg3power[i]) << " " <<\*/ cplex.getValue(Gridpowerbuy[i]) << " " << cplex.getValue(Gridpowersell[i]) << " " << pvpower[i] << " " << electricdemand[i] << /\*" " << cplex.getValue(statoc[i] \* 100) << " " << cplex.getValue(besschg[i]) << " " << cplex.getValue(bessdis[i]) << \*/" " << gridbuy[i] << " " << gridsell[i] << endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}

**Q) Multiple microgrids**

#include <ilcplex/ilocplex.h>

ILOSTLBEGIN

int

main(int, char\*\*)

{

IloEnv env;

try {

int totaltime = 24;//One day

float electricload1[24] = { 681,695,725,782,795,825,881,869,880,861,892,889,843,822,823,887,849,795,784,787,683,627,668,678};//Electicity load for MG1 w.r.t time

float electricload2[24] = {691,705,755,792,825,885,891,929,900,891,942,939,953,922,923,887,879,895,884,797,783,727,768,758};//Electicity load for mg2 w.r.t time

float electricload3[24] = {820,815,775,872,885,895,921,945,974,1051,1082,1029,943,962,903,897,889,875,804,789,763,723,764,744};//electric load for MG3

float gridbuy[24] = { 84,84,84,84,84,84,97,97,97,97,97,120,120,120,120,120,120,97,97,97,97,97,84,84};//buying price from grid w.r.t time

float gridsell[24] = {64,64,64,64,64,64,77,77,77,77,77,100,100,100,100,100,100,77,77,77,77,77,64,64};//selling price to grid w.r.t time

float rdg1[24] = {35,38,31,24,34,35,46,80,97,113,113,120,99,88,92,72,62,34,41,35,39,29,50,55};//Power generated by Renewable at MG1 w.r.t time

float rdg2[24] = {25,28,21,14,24,25,36,60,117,133,133,140,119,108,122,92,42,24,31,25,29,19,40,45};//renewable power at MG2

float rdg3[24] = {25,28,21,14,24,25,36,70,87,103,103,110,89,78,82,52,42,24,31,25,29,19,40,45};//renewable at MG3

int dg1cost = 76;//Cost for generating per kW power

int dg2cost = 85;

int dg3cost = 88;

int i;

IloModel model(env);

IloNumVarArray Gridpowerbuy1(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell1(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray Gridpowerbuy2(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell2(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray Gridpowerbuy3(env, totaltime, 0, IloInfinity);//Grid power bought

IloNumVarArray Gridpowersell3(env, totaltime, 0, IloInfinity);//Grid power sold

IloNumVarArray statoc(env, totaltime, 0, 1);//battery storage capacity

IloNumVarArray besschg(env, totaltime, 0, 200);//battery charging

IloNumVarArray bessdis(env, totaltime, 0, 200);//battery discharging

IloNumVarArray statoc2(env, totaltime, 0, 1);//battery storage capacity

IloNumVarArray besschg2(env, totaltime, 0, 250);//battery charging

IloNumVarArray bessdis2(env, totaltime, 0, 250);//battery discharging

IloNumVarArray statoc3(env, totaltime, 0, 1);//battery storage capacity

IloNumVarArray besschg3(env, totaltime, 0, 320);//battery charging

IloNumVarArray bessdis3(env, totaltime, 0, 320);//battery discharging

IloNumVarArray dg1power(env, totaltime, 0, 780);//DGA power generation range at MG1

IloNumVarArray dg2power(env, totaltime, 0, 820);//DGA power generation range at MG2

IloNumVarArray dg3power(env, totaltime, 0, 900);//DGA power generation range at MG3

IloNumVarArray S12(env, totaltime, 0, IloInfinity);//inter grid power send from MG1

IloNumVarArray S23(env, totaltime, 0, IloInfinity);//inter grid power send from MG2

IloNumVarArray S13(env, totaltime, 0, IloInfinity);//inter grid power send from MG3

IloNumVarArray S21(env, totaltime, 0, IloInfinity);//inter grid power received to MG1

IloNumVarArray S32(env, totaltime, 0, IloInfinity);//inter grid power received to MG2

IloNumVarArray S31(env, totaltime, 0, IloInfinity);//inter grid power received to MG3

IloExpr solutio(env, totaltime);

for (i = 0; i < totaltime; i++)

{

solutio += dg1cost\*dg1power[i] + gridbuy[i] \* Gridpowerbuy1[i] - gridsell[i] \* Gridpowersell1[i]+ dg2cost\*dg2power[i] + gridbuy[i] \* Gridpowerbuy2[i] - gridsell[i] \* Gridpowersell2[i]+ dg3cost\*dg3power[i] + gridbuy[i] \* Gridpowerbuy3[i] - gridsell[i] \* Gridpowersell3[i];

}

// Objective: minimize cost

model.add(IloMinimize(env, solutio));

// Constraint: meet demand

for (i = 0; i < totaltime; i++)

{

model.add(0 <= dg1power[i]);

model.add(dg1power[i] <= 780);

model.add(0 <= dg2power[i]);

model.add(dg2power[i] <= 820);

model.add(0 <= dg3power[i]);

model.add(dg3power[i] <= 900);

model.add(0 <= statoc[i]);

model.add(statoc[i] <= 1);

model.add(0 <= statoc2[i]);

model.add(statoc2[i] <= 1);

model.add(0 <= statoc3[i]);

model.add(statoc3[i] <= 1);

model.add((S12[i]>=1)==(S23[i]==0));

model.add((S13[i] >= 1) == (S32[i] == 0));

model.add((S21[i] >= 1) == (S13[i] == 0));

model.add((S23[i] >= 1) == (S31[i] == 0));

model.add((S31[i] >= 1) == (S12[i] == 0));

model.add((S32[i] >= 1) == (S21[i] == 0));

model.add((Gridpowerbuy1[i]>0)==(S12[i]==0 && S13[i]==0));

/\*model.add((Gridpowerbuy2[i] >= 0) == (S21[i] == 0 && S23[i] == 0));\*/

/\*model.add((Gridpowerbuy3[i] >= 0) == (S32[i] == 0 && S31[i] == 0));\*/

/\*model.add((S21[i] >= 0 || S31[i] >= 0) == (Gridpowersell1[i]==0));\*/

if (i == 0)

{

model.add(statoc[i] == 0 + ((0.96\*besschg[i] - (bessdis[i] / 0.96)) / 180));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (180 \* (1 - 0) / 0.96));

model.add(bessdis[i] <= (180 \* 0 \* 0.96));

model.add(statoc2[i] == 0 + ((0.96\*besschg2[i] - (bessdis2[i] / 0.96)) / 230));

model.add(0 <= besschg2[i]);

model.add(0 <= bessdis2[i]);

model.add(besschg2[i] <= (230 \* (1 - 0) / 0.96));

model.add(bessdis2[i] <= (230 \* 0 \* 0.96));

model.add(statoc3[i] == 0 + ((0.95\*besschg3[i] - (bessdis3[i] / 0.95)) / 300));

model.add(0 <= besschg3[i]);

model.add(0 <= bessdis3[i]);

model.add(besschg3[i] <= (300 \* (1 - 0) / 0.95));

model.add(bessdis3[i] <= (300 \* 0 \* 0.95));

}

else

{

model.add(statoc[i] == statoc[i - 1] + ((0.96\*besschg[i] - (bessdis[i] / 0.96)) / 180));

model.add(0 <= besschg[i]);

model.add(0 <= bessdis[i]);

model.add(besschg[i] <= (180 \* (1 - statoc[i - 1])) / 0.96);

model.add(bessdis[i] <= 180 \* statoc[i - 1] \* 0.96);

model.add(statoc2[i] == statoc2[i - 1] + ((0.96\*besschg2[i] - (bessdis2[i] / 0.96)) / 230));

model.add(0 <= besschg2[i]);

model.add(0 <= bessdis2[i]);

model.add(besschg2[i] <= (230 \* (1 - statoc2[i - 1])) / 0.96);

model.add(bessdis2[i] <= 230 \* statoc2[i - 1] \* 0.96);

model.add(statoc3[i] == statoc3[i - 1] + ((0.95\*besschg3[i] - (bessdis3[i] / 0.95)) / 300));

model.add(0 <= besschg3[i]);

model.add(0 <= bessdis3[i]);

model.add(besschg3[i] <= (300 \* (1 - statoc3[i - 1])) / 0.95);

model.add(bessdis3[i] <= 300 \* statoc3[i - 1] \* 0.95);

}

model.add(dg1power[i] + Gridpowerbuy1[i] + rdg1[i] + bessdis[i] - besschg[i] - Gridpowersell1[i]+S31[i]+S21[i] == electricload1[i]+S13[i]+S12[i]);

model.add(dg2power[i] + Gridpowerbuy2[i] + rdg2[i] + bessdis2[i] - besschg2[i] - Gridpowersell2[i] + S12[i]+S32[i] == electricload2[i]+S21[i]+S23[i]);

model.add(dg3power[i] + Gridpowerbuy3[i] + rdg3[i] + bessdis3[i] - besschg3[i] - Gridpowersell3[i] + S13[i]+S23[i] == electricload3[i]+ S31[i]+S32[i]);

}

IloCplex cplex(env);

cplex.extract(model);

cplex.solve();

cout << "Solution status: " << cplex.getStatus() << endl;

cout << "Minimized Objective Funtion " << cplex.getObjValue() << endl;

cout << "DGA" <<'\t'<< "CHPA" << '\t' << "Grid" << '\t' << "Grid power sell" << '\t' << "PV" << '\t' << "DEmand" << '\t' << "SOC" << '\t' << "charging" << '\t' << "discharging" << endl;

for (i = 0; i < totaltime; i++)

{

cout << cplex.getValue(dg1power[i]) << '\t' << cplex.getValue(Gridpowerbuy1[i]) << '\t' << cplex.getValue(Gridpowersell1[i]) << '\t' << rdg1[i] << '\t' << electricload1[i] << '\t' << cplex.getValue(statoc[i] \* 100) << '\t' << cplex.getValue(S12[i]) << '\t' << cplex.getValue(S21[i]) << '\t' << cplex.getValue(S13[i]) << '\t' << cplex.getValue(S31[i]) << endl;

}

IloNum eps = cplex.getParam(

IloCplex::Param::MIP::Tolerances::Integrality);

}

catch (IloException& ex) {

cerr << "Error: " << ex << endl;

}

env.end();

system("pause");

return 0;

}