

```

// ----- *- C++ -* -----
// File: ilodiet.cpp
// Version 12.6.3
// -----
// Licensed Materials - Property of IBM
// 5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
// Copyright IBM Corporation 2000, 2015. All Rights Reserved.
//
// US Government Users Restricted Rights - Use, duplication or
// disclosure restricted by GSA ADP Schedule Contract with
// IBM Corp.
// -----
// A dietary model.
//
// Input data:
// foodMin[j]      minimum amount of food j to use
// foodMax[j]      maximum amount of food j to use
// foodCost[j]     cost for one unit of food j
// nutrMin[i]      minimum amount of nutrient i
// nutrMax[i]      maximum amount of nutrient i
// nutrPer[i][j]   nutrition amount of nutrient i in food j
//
// Modeling variables:
// Buy[j]          amount of food j to purchase
//
// Objective:
// minimize sum(j) Buy[j] * foodCost[j]
//
// Constraints:
// forall nutr i: nutrMin[i] <= sum(j) Buy[j] * nutrPer[i][j] <= nutrMax[i]
//
#include <ilcplex/ilocplex.h>      // entêtes obligatoires
ILOSTLBEGIN

void usage(const char* name) {
    cerr << endl;
    cerr << "usage: " << name << " [options] <file>" << endl;
    cerr << "options: -c build model by column" << endl;
    cerr << "      -i use integer variables" << endl;
    cerr << endl;
}

void buildModelByRow(IloModel      mod,
                    IloNumVarArray Buy,
                    const IloNumArray foodMin,
                    const IloNumArray foodMax,
                    const IloNumArray foodCost,
                    const IloNumArray nutrMin,
                    const IloNumArray nutrMax,
                    const IloNumArray2 nutrPer,
                    IloNumVar::Type type) {

    IloEnv env = mod.getEnv();
    Buy.clear();

```

```

    IloNumVarArray tmp(env, foodMin, foodMax, type);
    // tableau temporaire de variables créé d'un coup avec bornes
    Buy.add(tmp); // copie dans tableau permanent
    tmp.end(); // détruire objet temporaire

    IloInt i, j;
    IloInt n = foodCost.getSize();
    IloInt m = nutrMin.getSize();

    mod.add(IloMinimize(env, IloScalProd(Buy, foodCost)));
    // création de l'objectif d'un coup

    for (i = 0; i < m; i++) {
        IloExpr expr(env);
        for (j = 0; j < n; j++) {
            expr += Buy[j] * nutrPer[i][j];
            // remplissage de l'expression de la contrainte
        }
        mod.add(nutrMin[i] <= expr <= nutrMax[i]);
        // création et ajout de la contrainte au modèle
        expr.end(); // détruire objet temporaire
    }

    void buildModelByColumn(IloModel      mod,
                           IloNumVarArray Buy,
                           const IloNumArray foodMin,
                           const IloNumArray foodMax,
                           const IloNumArray foodCost,
                           const IloNumArray nutrMin,
                           const IloNumArray nutrMax,
                           const IloNumArray2 nutrPer,
                           IloNumVar::Type type) {

        IloEnv env = mod.getEnv();
        IloInt i, j;
        IloInt n = foodCost.getSize();
        IloInt m = nutrMin.getSize();

        IloRangeArray range (env, nutrMin, nutrMax);
        // tableau temporaire de contraintes (pour créer intersections avec variables)
        mod.add(range);
        IloObjective cost = IloAdd(mod, IloMinimize(env));
        // objectif créé et ajouté au modèle

        for (j = 0; j < n; j++) {
            IloNumColumn col = cost(foodCost[j]);
            // coût de la colonne j dans l'objectif

            for (i = 0; i < m; i++) {
                col += range[i](nutrPer[i][j]);
                // coeff de la colonne j pour la contrainte i
            }
            Buy.add(IloNumVar(col, foodMin[j], foodMax[j], type));
            // création de la variable j et ajout dans tableau
            col.end(); // détruire objet temporaire
        }
        range.end(); // détruire objet temporaire
    }
}

```

```

int main(int argc, char **argv)
{
    IloEnv env; // création de l'environnement

    try {
        const char* filename = "../../examples/data/diet.dat";
        IloBool byColumn = IloFalse;
        IloNumVar::Type varType = ILOFLOAT;
        IloInt i;

        for (i = 1; i < argc; i++) {
            if (argv[i][0] == '-') {
                switch (argv[i][1]) {
                    case 'c':
                        byColumn = IloTrue;
                        break;
                    case 'i':
                        varType = ILOINT;
                        break;
                    default:
                        usage(argv[0]);
                        throw (-1);
                }
            }
            else {
                filename = argv[i];
                break;
            }
        }

        ifstream file(filename);
        if ( !file ) {
            cerr << "ERROR: could not open file '" << filename
                << "' for reading" << endl;
            usage(argv[0]);
            throw (-1);
        }

        // model data

        IloNumArray foodCost(env), foodMin(env), foodMax(env);
        IloNumArray nutrMin(env), nutrMax(env); // tableaux de nombres
        IloNumArray2 nutrPer(env); // matrice 2D de nombres

        //-----
        // Ne pas utiliser car suppose un format de fichier particulier
        // Lire normalement le fichier et stocker dans des IloArray<
        // ou autres structures
        file >> foodCost >> foodMin >> foodMax;
        file >> nutrMin >> nutrMax;
        file >> nutrPer;
        //-----

        IloInt nFoods = foodCost.getSize();
        IloInt nNutr = nutrMin.getSize();

        if ( foodMin.getSize() != nFoods ||
            foodMax.getSize() != nFoods ||
            nutrPer.getSize() != nNutr ||
            nutrMax.getSize() != nNutr ) {
            cerr << "ERROR: Data file '" << filename
                << "' contains inconsistent data" << endl;
            throw (-1);
        }

        for (i = 0; i < nNutr; i++) {
            if (nutrPer[i].getSize() != nFoods) {
                cerr << "ERROR: Data file '" << argv[0]
                    << "' contains inconsistent data" << endl;
                throw (-1);
            }
        }

        // Build model

        IloModel mod(env); // création du modèle
        IloNumVarArray Buy(env); // création du tableau de variables

        if ( byColumn ) {
            buildModelByColumn(mod, Buy, foodMin, foodMax, foodCost,
                               nutrMin, nutrMax, nutrPer, varType);
        }
        else {
            buildModelByRow(mod, Buy, foodMin, foodMax, foodCost,
                            nutrMin, nutrMax, nutrPer, varType);
        }

        // Solve model

        IloCplex cplex(mod); // création de l'objet CPLEX à partir du modèle
        cplex.exportModel("diet.lp"); // écriture du modèle dans un fichier texte

        cplex.solve(); // résolution du modèle

        // récupération de la solution
        cplex.out() << "solution status = " << cplex.getCplexStatus() << endl;
        cplex.out() << endl;
        cplex.out() << "cost = " << cplex.getObjValue() << endl;
        for (i = 0; i < foodCost.getSize(); i++) {
            cplex.out() << " Buy" << i << " = " << cplex.getValue(Buy[i]) << endl;
        }
    }
    catch (IloException& ex) { // gestion des exceptions
        cerr << "Error: " << ex << endl;
    }
    catch (...) {
        cerr << "Error" << endl;
    }

    env.end();
    // destruction de l'environnement ainsi que de tous les objets liés

    return 0;
}

```

```

// ----- *- C++ -* -----
// File: facility.cpp
// Version 12.6.3
// -----
// Licensed Materials - Property of IBM
// 5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
// Copyright IBM Corporation 2000, 2015. All Rights Reserved.
//
// US Government Users Restricted Rights - Use, duplication or
// disclosure restricted by GSA ADP Schedule Contract with
// IBM Corp.
// -----

#include <ilcplex/ilocplex.h>
ILOSTLBEGIN

typedef IloArray<IloNumArray>   FloatMatrix;   // matrice 2D de nombres
typedef IloArray<IloNumVarArray> NumVarMatrix; // matrice 2D de variables

//-----
// Input data:
//
// capacity[j] : capacité de chaque dépôt (en clients)
// fixedCost[j] : coût fixe de chaque dépôt
// cost[i][j] : coût de relier un client i au dépôt j
//
// Variables:
//
// open[j] : 1 si dépôt ouvert, 0 sinon
// supply[i][j] : 1 si client i est relié au dépôt j
//
// objective:
// minimize sum(j) fixedCost(j) * open(j) + sum(i,j) cost(i,j) * supply(i,j)
//
// constraints:
//
// sum(j) supply(i,j) == 1 // chaque client est relié à un dépôt
//
// sum(i) supply(i,j) <= capacity(j) * open(j) // capacité des dépôts
//-----

int main(int argc, char **argv)
{
    IloEnv env; // création de l'environnement
    try {
        IloInt i, j;
        IloNumArray capacity(env), fixedCost(env);
        FloatMatrix cost(env);
        IloInt nbLocations;
        IloInt nbClients;

//-----
// Ne pas utiliser car suppose un format de fichier particulier
// Lire normalement le fichier et stocker dans des IloArray<
// ou autres structures

        const char* filename = "../../examples/data/facility.dat";
        if (argc > 1)
            filename = argv[1];
        ifstream file(filename);
        if (!file) {
            cerr << "ERROR: could not open file '" << filename
                 << "' for reading" << endl;
            cerr << "usage: " << argv[0] << " <file>" << endl;
            throw(-1);
        }

        file >> capacity >> fixedCost >> cost;
//-----

        nbLocations = capacity.getSize();
        nbClients = cost.getSize();

        IloBool consistentData = (fixedCost.getSize() == nbLocations);
        for(i = 0; consistentData && (i < nbClients); i++)
            consistentData = (cost[i].getSize() == nbLocations);
        if (!consistentData) {
            cerr << "ERROR: data file '"
                 << filename << "' contains inconsistent data" << endl;
            throw(-1);
        }

        IloNumVarArray open(env, nbLocations, 0, 1, ILOINT);
        // tableau de variables binaires de taille nbLocations
        NumVarMatrix supply(env, nbClients);
        for(i = 0; i < nbClients; i++)
            supply[i] = IloNumVarArray(env, nbLocations, 0, 1, ILOINT);
            // matrice 2D de variables binaires de taille nbClients * nbLocations

        IloModel model(env);
        for(i = 0; i < nbClients; i++)
            model.add(IloSum(supply[i]) == 1); //  $\sum_j \text{supply}[i][j] = 1$ 
        for(j = 0; j < nbLocations; j++) {
            IloExpr v(env);
            for(i = 0; i < nbClients; i++)
                v += supply[i][j]; //  $\sum_i \text{supply}[i][j]$ 
            model.add(v <= capacity[j] * open[j]); //  $\sum_i \text{supply}[i][j] \leq \text{cap}[j] * \text{open}[j]$ 
        }
        v.end(); // détruire objet temporaire

        IloExpr obj = IloScalProd(fixedCost, open);
        for(i = 0; i < nbClients; i++) {
            obj += IloScalProd(cost[i], supply[i]); //  $\sum_{i,j} \text{supply}[i][j] * \text{cost}[i][j]$ 
        }
        model.add(IloMinimize(env, obj));
        obj.end(); // détruire objet temporaire
    }
}

```

```

IloCplex cplex(env);
cplex.extract(model); // équivaut à IloCplex cplex(model);
cplex.solve();
cplex.out() << "Solution status: " << cplex.getCplexStatus() << endl;

IloNum tolerance =
    cplex.getParam(IloCplex::Param::MIP::Tolerances::Integrality);
cplex.out() << "Optimal value: " << cplex.getObjValue() << endl;
for(j = 0; j < nbLocations; j++) {
    if (cplex.getValue(open[j]) >= 1 - tolerance) {
        cplex.out() << "Facility " << j << " is open, it serves clients ";
        for(i = 0; i < nbClients; i++) {
            if (cplex.getValue(supply[i][j]) >= 1 - tolerance)
                cplex.out() << i << " ";
        }
        cplex.out() << endl;
    }
}
}
catch(IloException& e) { // gestion des exceptions
    cerr << " ERROR: " << e << endl;
}
catch(...) {
    cerr << " ERROR" << endl;
}
env.end();
return 0;
}

```

```

// ----- *- C++ -* ----- // MAIN PROGRAM //
// File: cutstock.cpp
// Version 12.6.3
// -----
// Licensed Materials - Property of IBM
// 5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
// Copyright IBM Corporation 2000, 2015. All Rights Reserved.
//
// US Government Users Restricted Rights - Use, duplication or
// disclosure restricted by GSA ADP Schedule Contract with
// IBM Corp.
// -----
// Problème maître:
//
// données:
// rollWidth longueur du rouleau
// amount[j] nb. min. de chaque morceau
// size[i] taille de chaque morceau
//
// variables:
// Cut[i] patron de coupe i
// price[j] variable duale de chaque contrainte de coupe i
// Use[j] nb. de chaque patron choisi
//
// Objectif:
// minimize sum(i) Cut[i]
//
// Contraintes:
// forall morceau: sum(i) use[i][j] * Cut[i] >= amount[j]
//
//
// Sous-problème:
//
// Objectif:
// minimize sum(j) (-price[j]) + 1
//
// Contraintes:
// sum(j) size[j] * Use[j] <= rollwidth

#include <ilcplex/ilcplex.h> // entêtes obligatoires
ILOSTLBEGIN

#define RC_EPS 1.0e-6

static void readData (const char* filename, IloNum& rollWidth,
                    IloNumArray& size, IloNumArray& amount);
static void report1 (IloCplex& cutSolver, IloNumVarArray Cut,
                   IloRangeArray Fill);
static void report2 (IloAlgorithm& patSolver,
                   IloNumVarArray Use,
                   IloObjective obj);
static void report3 (IloCplex& cutSolver, IloNumVarArray Cut);

int
main(int argc, char **argv)
{
    IloEnv env; // création de l'environnement
    try {
        IloInt i, j;

        IloNum rollWidth;
        IloNumArray amount(env); // tableaux de nombres
        IloNumArray size(env);

        if ( argc > 1 )
            readData(argv[1], rollWidth, size, amount);
        else
            readData("../..../examples/data/cutstock.dat",
                    rollWidth, size, amount);

        // CUTTING-OPTIMIZATION PROBLEM //

        IloModel cutOpt (env); // modèle principal (PM)
        IloObjective RollsUsed = IloAdd(cutOpt, IloMinimize(env));
        // objectif PM créé et ajouté au modèle
        IloRangeArray Fill = IloAdd(cutOpt,
                                    IloRangeArray(env, amount, IloInfinity));
        // tableau de contraintes de coupe créé et ajouté au modèle
        IloNumVarArray Cut(env); // tableau de variables

        IloInt nWdth = size.getSize();
        for (j = 0; j < nWdth; j++) {
            Cut.add(IloNumVar(RollsUsed(1) + Fill[j](int(rollWidth / size[j]))));
            // création des variables initiales et ajout au modèle
        }

        IloCplex cutSolver(cutOpt); // problème maître

        // PATTERN-GENERATION PROBLEM //

        IloModel patGen (env); // modèle secondaire (SP)

        IloObjective ReducedCost = IloAdd(patGen, IloMinimize(env, 1));
        // objectif SP créé et ajouté au modèle
        IloNumVarArray Use(env, nWdth, 0.0, IloInfinity, ILOINT);
        // tableau de variables créé
        patGen.add(IloScalProd(size, Use) <= rollWidth);
        // création des contraintes de patron et ajout au modèle

        IloCplex patSolver(patGen); // sous-problème

        // COLUMN-GENERATION PROCEDURE //

        IloNumArray price(env, nWdth);
        // valeurs duales des contraintes de coupe (PM)
        IloNumArray newPatt(env, nWdth); // nouveaux patrons de coupe
    }
}

```

```

/// COLUMN-GENERATION PROCEDURE ///
for (;;) {
    /// OPTIMIZE OVER CURRENT PATTERNS ///

    cutSolver.solve();          // résolution PM
    report1 (cutSolver, Cut, Fill);

    /// FIND AND ADD A NEW PATTERN ///

    for (i = 0; i < nWdth; i++) {
        price[i] = -cutSolver.getDual(Fill[i]); // var. duales
    }
    ReducedCost.setLinearCoefs(Use, price);
    // modification de l'objectif du SP

    patSolver.solve();        // résolution du SP
    report2 (patSolver, Use, ReducedCost);

    if (patSolver.getValue(ReducedCost) > -RC_EPS) break;
    // critère d'arrêt de la génération de colonnes

    patSolver.getValues(newPatt, Use);
    // récupération de la nouvelle colonne
    Cut.add( IloNumVar(RollsUsed(1) + Fill(newPatt)) );
    // création et ajout de la nouvelle variable au PM
}

cutOpt.add(IloConversion(env, Cut, ILOINT));
// on passe au MILP (PM)

cutSolver.solve(); // résolution en nombres entiers du PM
cout << "Solution status: " << cutSolver.getCplexStatus() << endl;
report3 (cutSolver, Cut);
}
catch (IloException& ex) { // gestion des exceptions
    cerr << "Error: " << ex << endl;
}
catch (...) {
    cerr << "Error" << endl;
}

env.end();

return 0;
}

static void readData (const char* filename, IloNum& rollWidth,
                    IloNumArray& size, IloNumArray& amount)
{
    ifstream in(filename);
    if (in) {
        in >> rollWidth;
        in >> size;
        in >> amount;
    }
}

```

```

else {
    cerr << "No such file: " << filename << endl;
    throw(1);
}
}

static void report1 (IloCplex& cutSolver, IloNumVarArray Cut,
                    IloRangeArray Fill)
{
    cout << endl;
    cout << "Using " << cutSolver.getObjValue() << " rolls" << endl;
    cout << endl;
    for (IloInt j = 0; j < Cut.getSize(); j++) {
        cout << " Cut" << j << " = " << cutSolver.getValue(Cut[j]) << endl;
    }
    cout << endl;
    for (IloInt i = 0; i < Fill.getSize(); i++) {
        cout << " Fill" << i << " = " << cutSolver.getDual(Fill[i]) << endl;
    }
    cout << endl;
}

static void report2 (IloAlgorithm& patSolver, IloNumVarArray Use,
                    IloObjective obj)
{
    cout << endl;
    cout << "Reduced cost is " << patSolver.getValue(obj) << endl;
    cout << endl;
    if (patSolver.getValue(obj) <= -RC_EPS) {
        for (IloInt i = 0; i < Use.getSize(); i++) {
            cout << " Use" << i << " = " << patSolver.getValue(Use[i]) << endl;
        }
        cout << endl;
    }
}

static void report3 (IloCplex& cutSolver, IloNumVarArray Cut)
{
    cout << endl;
    cout << "Best integer solution uses "
        << cutSolver.getObjValue() << " rolls" << endl;
    cout << endl;
    for (IloInt j = 0; j < Cut.getSize(); j++) {
        cout << " Cut" << j << " = " << cutSolver.getValue(Cut[j]) << endl;
    }
}

/* Example Input file:
115
[25, 40, 50, 55, 70]
[50, 36, 24, 8, 30]
*/

```

```

// ----- *- C++ -* -----
// File: foodmanufact.cpp
// Version 12.6.1
// -----
// Licensed Materials - Property of IBM
// 5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
// Copyright IBM Corporation 2000, 2014. All Rights Reserved.
//
// US Government Users Restricted Rights - Use, duplication or
// disclosure restricted by GSA ADP Schedule Contract with
// IBM Corp.
// -----
//
// foodmanufact.cpp - An implementation of an example from H.P.
// Williams' book Model Building in Mathematical
// Programming. This example solves a
// food production planning problem. It
// demonstrates the use of CPLEX's
// linearization capability.

#include <ilcplex/ilocplex.h>
ILOSTLBEGIN

typedef IloArray<IloNumVarArray> NumVarMatrix;
typedef IloArray<IloNumArray> NumMatrix;

typedef enum { v1, v2, o1, o2, o3 } Product;
const IloInt nbMonths = 6;
const IloInt nbProducts = 5;

int
main()
{
    IloEnv env;
    try {
        NumMatrix cost(env, nbMonths);
        cost[0]=IloNumArray(env, nbProducts, 110.0, 120.0, 130.0, 110.0, 115.0);
        cost[1]=IloNumArray(env, nbProducts, 130.0, 130.0, 110.0, 90.0, 115.0);
        cost[2]=IloNumArray(env, nbProducts, 110.0, 140.0, 130.0, 100.0, 95.0);
        cost[3]=IloNumArray(env, nbProducts, 120.0, 110.0, 120.0, 120.0, 125.0);
        cost[4]=IloNumArray(env, nbProducts, 100.0, 120.0, 150.0, 110.0, 105.0);
        cost[5]=IloNumArray(env, nbProducts, 90.0, 100.0, 140.0, 80.0, 135.0);

        // Variable definitions
        IloNumVarArray produce(env, nbMonths, 0, IloInfinity);
        NumVarMatrix use(env, nbMonths);
        NumVarMatrix buy(env, nbMonths);
        NumVarMatrix store(env, nbMonths);
        IloInt i, p;
        for (i = 0; i < nbMonths; i++) {
            use[i] = IloNumVarArray(env, nbProducts, 0, IloInfinity);
            buy[i] = IloNumVarArray(env, nbProducts, 0, IloInfinity);
            store[i] = IloNumVarArray(env, nbProducts, 0, 1000);
        }
        IloExpr profit(env);

        IloModel model(env);

```

```

// For each type of raw oil we must have 500 tons at the end
for (p = 0; p < nbProducts; p++) {
    store[nbMonths-1][p].setBounds(500, 500);
}

// Constraints on each month
for (i = 0; i < nbMonths; i++) {
    // Not more than 200 tons of vegetable oil can be refined
    model.add(use[i][v1] + use[i][v2] <= 200);

    // Not more than 250 tons of non-vegetable oil can be refined
    model.add(use[i][o1] + use[i][o2] + use[i][o3] <= 250);

    // Constraints on food composition
    model.add(3 * produce[i] <=
        8.8 * use[i][v1] + 6.1 * use[i][v2] +
        2 * use[i][o1] + 4.2 * use[i][o2] + 5 * use[i][o3]);
    model.add(8.8 * use[i][v1] + 6.1 * use[i][v2] +
        2 * use[i][o1] + 4.2 * use[i][o2] + 5 * use[i][o3]
        <= 6 * produce[i]);
    model.add(produce[i] == IloSum(use[i]));

    // Raw oil can be stored for later use
    if (i == 0) {
        for (IloInt p = 0; p < nbProducts; p++)
            model.add(500 + buy[i][p] == use[i][p] + store[i][p]);
    }
    else {
        for (IloInt p = 0; p < nbProducts; p++)
            model.add(store[i-1][p] + buy[i][p] == use[i][p] + store[i][p]);
    }

    // Logical constraints
    // The food cannot use more than 3 oils
    // (or at least two oils must not be used)
    model.add((use[i][v1] == 0) + (use[i][v2] == 0) + (use[i][o1] == 0) +
        (use[i][o2] == 0) + (use[i][o3] == 0) >= 2);

    // When an oil is used, the quantity must be at least 20 tons
    for (p = 0; p < nbProducts; p++)
        model.add((use[i][p] == 0) || (use[i][p] >= 20));

    // If products v1 or v2 are used, then product o3 is also used
    model.add(IloIfThen(env, (use[i][v1] >= 20) || (use[i][v2] >= 20),
        use[i][o3] >= 20));

    // Objective function
    profit += 150 * produce[i] - IloScalProd(cost[i], buy[i]) -
        5 * IloSum(store[i]);
}

// Objective function
model.add(IloMaximize(env, profit));

IloCplex cplex(model);

```

```

if (cplex.solve()) {
    cout << "Solution status: " << cplex.getStatus() << endl;
    cout << " Maximum profit = " << cplex.getObjValue() << endl;
    for (IloInt i = 0; i < nbMonths; i++) {
        IloInt p;
        cout << " Month " << i << " " << endl;
        cout << " . buy ";
        for (p = 0; p < nbProducts; p++) {
            cout << cplex.getValue(buy[i][p]) << "\t ";
        }
        cout << endl;
        cout << " . use ";
        for (p = 0; p < nbProducts; p++) {
            cout << cplex.getValue(use[i][p]) << "\t ";
        }
        cout << endl;
        cout << " . store ";
        for (p = 0; p < nbProducts; p++) {
            cout << cplex.getValue(store[i][p]) << "\t ";
        }
        cout << endl;
    }
}
else {
    cout << " No solution found" << endl;
}
}
catch (IloException& ex) {
    cerr << "Error: " << ex << endl;
}
catch (...) {
    cerr << "Error" << endl;
}
env.end();
return 0;
}

```



```

// ----- *- C++ -* -----
// File: ilobendersatsp.cpp
// Version 12.6.3
// -----
// Licensed Materials - Property of IBM
// 5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
// Copyright IBM Corporation 2000, 2015. All Rights Reserved.
//
// US Government Users Restricted Rights - Use, duplication or
// disclosure restricted by GSA ADP Schedule Contract with
// IBM Corp.
// -----
//
// Example ilobendersatsp.cpp solves a flow MILP model for an
// Asymmetric Traveling Salesman Problem (ATSP) instance
// through Benders decomposition.
//
// The arc costs of an ATSP instance are read from an input file.
// The flow MILP model is decomposed into a master ILP and a worker LP.
//
// The master ILP is then solved by adding Benders' cuts during
// the branch-and-cut process via the cut callback functions.
// The cut callback functions add to the master ILP violated Benders' cuts
// that are found by solving the worker LP.
//
// The example allows the user to decide if Benders' cuts have to be separated:
//
// a) Only to separate integer infeasible solutions.
// In this case, Benders' cuts are treated as lazy constraints through the
// class IloCplex::LazyConstraintCallbackI.
//
// b) Also to separate fractional infeasible solutions.
// In this case, Benders' cuts are treated as lazy constraints through the
// class IloCplex::LazyConstraintCallbackI.
// In addition, Benders' cuts are also treated as user cuts through the
// class IloCplex::UserCutCallbackI.
//
//
// To run this example, command line arguments are required:
// ilobendersatsp.cpp {0|1} [filename]
// where
// 0 Indicates that Benders' cuts are only used as lazy constraints,
// to separate integer infeasible solutions.
// 1 Indicates that Benders' cuts are also used as user cuts,
// to separate fractional infeasible solutions.
//
// filename Is the name of the file containing the ATSP instance (arc
costs).
// If filename is not specified, the instance
// ../../../../examples/data/atsp.dat is read
//
//
// ATSP instance defined on a directed graph G = (V, A)
// - V = {0, ..., n-1}, V0 = V \ {0}
// - A = {(i,j) : i in V, j in V, i != j}
// - forall i in V: delta+(i) = {(i,j) in A : j in V}
// - forall i in V: delta-(i) = {(j,i) in A : j in V}
// - c(i,j) = traveling cost associated with (i,j) in A
//
// Flow MILP model
//
// Modeling variables:
// forall (i,j) in A:
//   x(i,j) = 1, if arc (i,j) is selected
//             = 0, otherwise
// forall k in V0, forall (i,j) in A:
//   y(k,i,j) = flow of the commodity k through arc (i,j)
//
// Objective:
// minimize sum((i,j) in A) c(i,j) * x(i,j)
//
// Degree constraints:
// forall i in V: sum((i,j) in delta+(i)) x(i,j) = 1
// forall i in V: sum((j,i) in delta-(i)) x(j,i) = 1
//
// Binary constraints on arc variables:
// forall (i,j) in A: x(i,j) in {0, 1}
//
// Flow constraints:
// forall k in V0, forall i in V:
//   sum((i,j) in delta+(i)) y(k,i,j) - sum((j,i) in delta-(i)) y(k,j,i) =
q(k,i)
//   where q(k,i) = 1, if i = 0
//                 = -1, if k == i
//                 = 0, otherwise
//
// Capacity constraints:
// forall k in V0, for all (i,j) in A: y(k,i,j) <= x(i,j)
//
// Nonnegativity of flow variables:
// forall k in V0, for all (i,j) in A: y(k,i,j) >= 0
//
#include <string>
#include <ilcplex/ilocplex.h> // entêtes obligatoires
ILOSTLBEGIN

// Declarations for functions in this program

void createMasterILP(IloModel mod, Arcs x, IloNumArray2 arcCost);

void createWorkerLP(IloCplex cplex, IloNumVarArray v, IloNumVarArray u,
IloObjective obj, IloInt numNodes);

IloBool separate(const Arcs x, const IloNumArray2 xSol, IloCplex cplex,
const IloNumVarArray v, const IloNumVarArray u,
IloObjective obj, IloExpr cutLhs, IloNum& cutRhs);

void usage(char *progname);

```

```

// Implementation class for the user-defined lazy constraint callback.
// The function BendersLazyCallback allows to add Benders' cuts as lazy
// constraints.
//
// S'applique aux noeuds entiers
//
ILOLAZYCONSTRAINTCALLBACK5(BendersLazyCallback, Arcs, x, IloCplex, workerCplex,
                           IloNumVarArray, v, IloNumVarArray, u,
                           IloObjective, workerObj)
{
  IloInt i;
  IloEnv masterEnv = getEnv();
  IloInt numNodes = x.getSize();

  // Get the current x solution

  IloNumArray2 xSol(masterEnv, numNodes);
  for (i = 0; i < numNodes; ++i) {
    xSol[i] = IloNumArray(masterEnv);
    getValues(xSol[i], x[i]);
  }

  // Benders' cut separation

  IloExpr cutLhs(masterEnv); // expression de la coupe
  IloNum cutRhs;
  IloBool sepStat = separate(x, xSol, workerCplex, v, u, workerObj, cutLhs,
                             cutRhs);
  if ( sepStat ) {
    add(cutLhs >= cutRhs).end();
    // crée la coupe, l'ajoute au modèle et détruit l'objet temporaire créé
  }

  // Free memory

  cutLhs.end();
  for (i = 0; i < numNodes; ++i)
    xSol[i].end();
  xSol.end();

  return;
} // END BendersLazyCallback

void usage (char *progname)
{
  cerr << "Usage:      " << progname << " {0|1} [filename]" << endl;
  cerr << " 0:      Benders' cuts only used as lazy constraints," << endl;
  cerr << "         to separate integer infeasible solutions." << endl;
  cerr << " 1:      Benders' cuts also used as user cuts," << endl;
  cerr << "         to separate fractional infeasible solutions." << endl;
  cerr << " filename: ATSP instance file name." << endl;
  cerr << "         File ../../examples/data/atstp.dat " << endl;
  cerr << "         << "used if no name is provided." << endl;
} // END usage

```

```

// Implementation class for the user-defined user cut callback.
// The function BendersUserCallback allows to add Benders' cuts as user cuts.
//
// S'applique aux noeuds fractionnaires
//
ILOUSERCUTCALLBACK5(BendersUserCallback, Arcs, x, IloCplex, workerCplex,
                   IloNumVarArray, v, IloNumVarArray, u,
                   IloObjective, workerObj)
{
  // Skip the separation if not at the end of the cut loop

  if ( !isAfterCutLoop() )
    return;

  IloInt i;
  IloEnv masterEnv = getEnv();
  IloInt numNodes = x.getSize();

  // Get the current x solution

  IloNumArray2 xSol(masterEnv, numNodes);
  for (i = 0; i < numNodes; ++i) {
    xSol[i] = IloNumArray(masterEnv);
    getValues(xSol[i], x[i]);
  }

  // Benders' cut separation

  IloExpr cutLhs(masterEnv); // expression de la coupe
  IloNum cutRhs;
  IloBool sepStat = separate(x, xSol, workerCplex, v, u, workerObj, cutLhs,
                             cutRhs);
  if ( sepStat ) {
    add(cutLhs >= cutRhs).end();
    // crée la coupe, l'ajoute au modèle et détruit l'objet temporaire créé
  }

  // Free memory

  cutLhs.end();
  for (i = 0; i < numNodes; ++i)
    xSol[i].end();
  xSol.end();

  return;
} // END BendersUserCallback

```

```

int main(int argc, char **argv)
{
    IloEnv masterEnv;    // 2 environnements distincts (1 seul nécessaire)
    IloEnv workerEnv;

    try {
        const char* fileName = "../../examples/data/atsp.dat";

        // Check the command line arguments

        if ( argc != 2 && argc != 3 ) {
            usage (argv[0]);
            throw (-1);
        }

        if ( (argv[1][0] != '1' && argv[1][0] != '0') ||
             (argv[1][1] != '\0' ) ) {
            usage (argv[0]);
            throw (-1);
        }

        IloBool separateFracSols = ( argv[1][0] == '0' ? IloFalse : IloTrue );

        masterEnv.out() << "Benders' cuts separated to cut off: ";
        if ( separateFracSols ) {
            masterEnv.out() << "Integer and fractional infeasible solutions." << endl;
        }
        else {
            masterEnv.out() << "Only integer infeasible solutions." << endl;
        }

        if ( argc == 3 )   fileName = argv[2];

        // Read arc_costs from data file (17 city problem)

        IloNumArray2 arcCost(masterEnv);    // matrice 2D de nombres (coûtsArc)
        ifstream data(fileName);
        if ( !data ) throw(-1);
        data >> arcCost;
        data.close();

        // create master ILP

        IloModel masterMod(masterEnv, "atsp_master");
        IloInt numNodes = arcCost.getSize();
        Arcs x(masterEnv, numNodes);
        createMasterILP(masterMod, x, arcCost);

        // Create worker IloCplex algorithm and worker LP for Benders' cuts
        separation

        IloCplex workerCplex(workerEnv);
        IloNumVarArray v(workerEnv);    // tableaux de variables
        IloNumVarArray u(workerEnv);
        IloObjective workerObj(workerEnv);    // objectif du SP
        createWorkerLP(workerCplex, v, u, workerObj, numNodes);

        // Set up the cut callback to be used for separating Benders' cuts

        IloCplex masterCplex(masterMod);
        masterCplex.setParam(IloCplex::Param::Preprocessing::Presolve, IloFalse);
        // met le presolve à OFF pour le problème maître

        // Set the maximum number of threads to 1.
        // This instruction is redundant: If MIP control callbacks are registered,
        // then by default CPLEX uses 1 (one) thread only.
        // Note that the current example may not work properly if more than 1
        threads
        // are used, because the callback functions modify shared global data.
        // We refer the user to the documentation to see how to deal with multi-
        thread
        // runs in presence of MIP control callbacks.

        masterCplex.setParam(IloCplex::Param::Threads, 1);

        // Turn on traditional search for use with control callbacks

        masterCplex.setParam(IloCplex::Param::MIP::Strategy::Search,
                             IloCplex::Traditional);

        masterCplex.use(BendersLazyCallback(masterEnv, x, workerCplex, v, u,
                                             workerObj));
        if ( separateFracSols )
            masterCplex.use(BendersUserCallback(masterEnv, x, workerCplex, v, u,
                                                 workerObj));

        // Solve the model and write out the solution

        if ( masterCplex.solve() ) {

            IloCplex::CplexStatus solStatus= masterCplex.getCplexStatus();
            masterEnv.out() << endl << "Solution status: " << solStatus << endl;

            masterEnv.out() << "Objective value: "
                << masterCplex.getObjValue() << endl;

            if ( solStatus == IloCplex::Optimal ) {

                // Write out the optimal tour

                IloInt i, j;
                IloNumArray2 sol(masterEnv, numNodes);    // matrice 2D de nombres
                IloIntArray succ(masterEnv, numNodes);
                //matrice 2D de nombres entiers

                for ( j = 0; j < numNodes; ++j)
                    succ[j] = -1;

                for ( i = 0; i < numNodes; i++) {
                    sol[i] = IloNumArray(masterEnv);
                    masterCplex.getValues(sol[i], x[i]);    // récupération solution
                    for(j = 0; j < numNodes; j++) {
                        if ( sol[i][j] > 1e-03 ) succ[i] = j;
                    }
                }
            }
        }
    }
}

```

```

    }
}

masterEnv.out() << "Optimal tour:" << endl;
i = 0;
while ( succ[i] != 0 ) {
    masterEnv.out() << i << ", ";
    i = succ[i];
}
masterEnv.out() << i << endl;
}
else {
    masterEnv.out() << "Solution status is not Optimal" << endl;
}
}
else {
    masterEnv.out() << "No solution available" << endl;
}
}

catch (const IloException& e) { // gestion exceptions
    cerr << "Exception caught: " << e << endl;
}
catch (...) {
    cerr << "Unknown exception caught!" << endl;
}

// Close the environments

masterEnv.end();
workerEnv.end();
return 0;
} // END main

// This routine creates the master ILP (arc variables x and degree constraints).
//
// Modeling variables:
// forall (i,j) in A:
//     x(i,j) = 1, if arc (i,j) is selected
//     = 0, otherwise
//
// Objective:
// minimize sum((i,j) in A) c(i,j) * x(i,j)
//
// Degree constraints:
// forall i in V: sum((i,j) in delta+(i)) x(i,j) = 1
// forall i in V: sum((j,i) in delta-(i)) x(j,i) = 1
//
// Binary constraints on arc variables:
// forall (i,j) in A: x(i,j) in {0, 1}
//
void createMasterILP(IloModel mod, Arcs x, IloNumArray2 arcCost)
{
    IloInt i, j;
    IloEnv env = mod.getEnv();
    IloInt numNodes = x.getSize();

```

```

// Create variables x(i,j) for (i,j) in A
// For simplicity, also dummy variables x(i,i) are created.
// Those variables are fixed to 0 and do not participate to
// the constraints.

char varName[100];
for (i = 0; i < numNodes; ++i) {
    x[i] = IloIntVarArray(env, numNodes, 0, 1);
    x[i][i].setBounds(0, 0);
    for (j = 0; j < numNodes; ++j) {
        sprintf(varName, "x.%d.%d", (int) i, (int) j);
        x[i][j].setName(varName);
    }
    mod.add(x[i]);
    // ajout au modèle de la matrice de variable x, vecteur par vecteur
}

// Create objective function: minimize sum((i,j) in A) c(i,j) * x(i,j)

IloExpr obj(env);
for (i = 0; i < numNodes; ++i) {
    arcCost[i][i] = 0;
    obj += IloScalProd(x[i], arcCost[i]);
}
mod.add(IloMinimize(env, obj));
obj.end();

// Add the out degree constraints.
// forall i in V: sum((i,j) in delta+(i)) x(i,j) = 1

for (i = 0; i < numNodes; ++i) {
    IloExpr expr(env);
    for (j = 0; j < i; ++j)
        expr += x[i][j];
    for (j = i+1; j < numNodes; ++j)
        expr += x[i][j];
    mod.add(expr == 1);
    expr.end();
}

// Add the in degree constraints.
// forall i in V: sum((j,i) in delta-(i)) x(j,i) = 1

for (i = 0; i < numNodes; i++) {
    IloExpr expr(env);
    for (j = 0; j < i; j++)
        expr += x[j][i];
    for (j = i+1; j < numNodes; j++)
        expr += x[j][i];
    mod.add(expr == 1);
    expr.end();
}
}

} // END createMasterILP

// This routine set up the IloCplex algorithm to solve the worker LP, and

```

```

// creates the worker LP (i.e., the dual of flow constraints and
// capacity constraints of the flow MILP)
//
// Modeling variables:
// forall k in V0, i in V:
//   u(k,i) = dual variable associated with flow constraint (k,i)
//
// forall k in V0, forall (i,j) in A:
//   v(k,i,j) = dual variable associated with capacity constraint (k,i,j)
//
// Objective:
// minimize sum(k in V0) sum((i,j) in A) x(i,j) * v(k,i,j)
//           - sum(k in V0) u(k,0) + sum(k in V0) u(k,k)
//
// Constraints:
// forall k in V0, forall (i,j) in A: u(k,i) - u(k,j) <= v(k,i,j)
//
// Nonnegativity on variables v(k,i,j)
// forall k in V0, forall (i,j) in A: v(k,i,j) >= 0
//
void createWorkerLP(IloCplex cplex, IloNumVarArray v, IloNumVarArray u,
                  IloObjective obj, IloInt numNodes)
{
    IloInt i, j, k;
    IloEnv env = cplex.getEnv();
    IloModel mod(env, "atstp_worker");

    // Set up IloCplex algorithm to solve the worker LP

    cplex.extract(mod);
    cplex.setOut(env.getNullStream()); // pas d'output de CPLEX

    // Turn off the presolve reductions and set the CPLEX optimizer
    // to solve the worker LP with primal simplex method.

    cplex.setParam(IloCplex::Param::Preprocessing::Reduce, 0);
    cplex.setParam(IloCplex::Param::RootAlgorithm, IloCplex::Primal);

    // Create variables v(k,i,j) forall k in V0, (i,j) in A
    // For simplicity, also dummy variables v(k,i,i) are created.
    // Those variables are fixed to 0 and do not participate to
    // the constraints.

    IloInt numArcs = numNodes * numNodes;
    IloInt vNumVars = (numNodes-1) * numArcs;
    IloNumVarArray vTemp(env, vNumVars, 0, IloInfinity);
    for (k = 1; k < numNodes; ++k) {
        for (i = 0; i < numNodes; ++i) {
            vTemp[(k-1)*numArcs + i * numNodes + i].setBounds(0, 0);
        }
    }
    v.clear();
    v.add(vTemp); // copie du vecteur de variables temporaires
                 // dans vecteur permanent

    vTemp.end();
    mod.add(v);
}

```

```

// Set names for variables v(k,i,j)

for (k = 1; k < numNodes; ++k) {
    for(i = 0; i < numNodes; ++i) {
        for(j = 0; j < numNodes; ++j) {
            char varName[100];
            sprintf(varName, "v.%d.%d.%d", (int) k, (int) i, (int) j);
            v[(k-1)*numArcs + i*numNodes + j].setName(varName);
        }
    }
}

// Associate indices to variables v(k,i,j)

IloIntArray vIndex(env, vNumVars);
for (j = 0; j < vNumVars; ++j)
{
    vIndex[j] = j;
    v[j].setObject(&vIndex[j]); // assigne un index (ptr) à la variable
}

// Create variables u(k,i) forall k in V0, i in V

IloInt uNumVars = (numNodes-1) * numNodes;
IloNumVarArray uTemp(env, uNumVars, -IloInfinity, IloInfinity);
u.clear();
u.add(uTemp); // copie du vecteur de variables temporaires
              // dans vecteur permanent

uTemp.end();
mod.add(u);

// Set names for variables u(k,i)

for (k = 1; k < numNodes; ++k) {
    for(i = 0; i < numNodes; ++i) {
        char varName[100];
        sprintf(varName, "u.%d.%d", (int) k, (int) i);
        u[(k-1)*numNodes + i].setName(varName);
    }
}

// Associate indices to variables u(k,i)

IloIntArray uIndex(env, uNumVars);
for (j = 0; j < uNumVars; ++j)
{
    uIndex[j] = vNumVars + j;
    u[j].setObject(&uIndex[j]); // assigne un index (ptr) à la variable
}

// Initial objective function is empty

obj.setSense(IloObjective::Minimize);
mod.add(obj);

// Add constraints:

```

```

// forall k in V0, forall (i,j) in A: u(k,i) - u(k,j) <= v(k,i,j)
for (k = 1; k < numNodes; ++k) {
  for(i = 0; i < numNodes; ++i) {
    for(j = 0; j < numNodes; ++j) {
      if ( i != j ) {
        IloExpr expr(env);
        expr -= v[(k-1)*numArcs + i*(numNodes) + j];
        expr += u[(k-1)*numNodes + i];
        expr -= u[(k-1)*numNodes + j];
        mod.add(expr <= 0);
        expr.end();
      }
    }
  }
}
} // END createWorkerLP

// This routine separates Benders' cuts violated by the current x solution.
// Violated cuts are found by solving the worker LP
//
IloBool separate(const Arcs x, const IloNumArray2 xSol, IloCplex cplex,
                const IloNumVarArray v, const IloNumVarArray u,
                IloObjective obj, IloExpr cutLhs, IloNum& cutRhs)
{
  IloBool violatedCutFound = IloFalse;

  IloEnv env = cplex.getEnv();
  IloModel mod = cplex.getModel();

  IloInt numNodes = xSol.getSize();
  IloInt numArcs = numNodes * numNodes;
  IloInt i, j, k, h;

  // Update the objective function in the worker LP:
  // minimize sum(k in V0) sum((i,j) in A) x(i,j) * v(k,i,j)
  // - sum(k in V0) u(k,0) + sum(k in V0) u(k,k)

  mod.remove(obj); // enlève l'objectif précédent
  IloExpr objExpr = obj.getExpr();
  objExpr.clear();
  for (k = 1; k < numNodes; ++k) {
    for (i = 0; i < numNodes; ++i) {
      for (j = 0; j < numNodes; ++j) {
        objExpr += xSol[i][j] * v[(k-1)*numArcs + i*numNodes + j];
      }
    }
  }
  for (k = 1; k < numNodes; ++k) {
    objExpr += u[(k-1)*numNodes + k];
    objExpr -= u[(k-1)*numNodes];
  }
  obj.setExpr(objExpr);
  mod.add(obj); //ajoute un nouvel objectif
  objExpr.end();

```

```

// Solve the worker LP
cplex.solve();

// A violated cut is available iff the solution status is Unbounded
if ( cplex.getCplexStatus() == IloCplex::Unbounded ) {
  IloInt vNumVars = (numNodes-1) * numArcs;
  IloNumVarArray var(env);
  IloNumArray val(env);

  // Get the violated cut as an unbounded ray of the worker LP
  cplex.getRay(val, var);

  // Compute the cut from the unbounded ray. The cut is:
  // sum((i,j) in A) (sum(k in V0) v(k,i,j)) * x(i,j) >=
  // sum(k in V0) u(k,0) - u(k,k)

  cutLhs.clear();
  cutRhs = 0.;

  for (h = 0; h < val.getSize(); ++h) {
    IloInt *index_p = (IloInt*) var[h].getObject();
    // récupère l'index (ptr) de la variable
    IloInt index = *index_p;

    if ( index >= vNumVars ) {
      index -= vNumVars;
      k = index / numNodes + 1;
      i = index - (k-1)*numNodes;
      if ( i == 0 )
        cutRhs += val[h]; // met à jour la valeur du RHS
      else if ( i == k )
        cutRhs -= val[h];
    }
    else {
      k = index / numArcs + 1;
      i = (index - (k-1)*numArcs) / numNodes;
      j = index - (k-1)*numArcs - i*numNodes;
      cutLhs += val[h] * x[i][j];
      // met à jour l'expression de la contrainte
    }
  }

  var.end();
  val.end();

  violatedCutFound = IloTrue;
}

return violatedCutFound;
} // END separate

```