Incentive Contract Design Dataset

Louis-Philippe Kerkhove, Mario Vanhoucke

October 11, 2014

This document gives an overview of the files contained in this archive, as well as the way to use this data for use in your research efforts. For more information on the interpretation of the various elements in this dataset we refer to the paper which introduced the modelling approach (CITATION TO BE ADDED HERE). To gain a full understanding of the data contained in this dataset it is advised to keep this paper close at hand, as this document is not indented to be a full tutorial on the properties of the models and their components.

1 Datasets

1.1 Uni-dimensional experiments

The folder *"1-Unidimensional-Experiments"* contains the datasets used to carry out the uni-dimensional experiments. This dataset is comprised of:

- Contract datasets:
  - 41,601 cost incentive components
  - 1,195,488 duration incentive components
  - 74,790 scope contract components
- 25 evaluation model instances
- 380 trade-off model instances

1.2 Multi-dimensional experiments

The folder *"2-Multidimensional-Experiments"* contains the datasets used for the multi-dimensional experiments. Due to the full factorial nature with respect to the contract combinations the number of elements in this dataset is substantially lower.

- Contract datasets:
  - 189 cost incentive components
  - 4,536 duration incentive components
  - 225 scope contract components
- 11 evaluation model instances
- 27 trade-off model instances

2 Data encoding

2.1 Contract Models

The contract data is organised based on the incentivised dimension: project cost, duration or time. For each of these dimensions a summary file which contains meaningful summary information and headers and a data file which contains the data in the format which is easiest to read. The data files always start with a code indicating the type of the file, as a check to prevent incorrect information being used.

2.1.1 Cost Contracts

The following information is included in the cost contract files:

CC_ID Simple identification number for the cost contract component.

ContractType The type of the contract, which can be linked to more descriptive names using the *"unidim-contractNames.txt"* file.

type_is_piecewise Boolean indicating if the contract is a linear or piecewise linear contract.

type_is_nonlinear Boolean indicating if the contract is nonlinear.

target The cost target set by the contract.

nbRegions The number of regions used in case the contract is (piecewise) linear.

bounds The bounds of the regions in case the contract is (piecewise) linear.

SharingRatios The sharing rations used in the regions if the contract is (piecewise) linear.

maxIncentive The maximal incentive amount in case the contract is nonlinear.

maxDisIncentive The greatest disincentive amount which can be allocated in case the contract is nonlinear.

lowerBound The lower bound in case the contract is nonlinear.

upperBound The upper bound in case the contract is nonlinear.

Values which are needed for a nonlinear contract but not for a (piecewise) linear contract get the value "NA", the inverse also being true for the nonlinear contract parameters. The titles above are only included in the summary file, the data file only includes the data itself. In this dataset a single line corresponds to a single instance, the C++ code shows how this information can be loaded for use, the code itself is self-explanatory.

```
void CostContract::importCostContractCompact(string filename){
    ifstream MyInputFile;
    MyInputFile.open(filename.c_str());

    //Test if the file has the correct format
    long int temp;
    MyInputFile >> temp;
     if (temp != FILECODE_COST_CONTRACT_COMPACT)  {
        cout << "**************** ERROR MESSAGE ****************************" << endl;
        cout << "Vaidation of cost contract file integrity failed" << endl;
        cout << "Filename and path causing error:\t" << filename << endl;
        cout << "Error:\t" << "Incorrect file identification code at start of file" << endl;
        cout << "Observed filecode:\t" << temp << endl;
```

```
        cout << "Expected filecode:\t" << FILECODE_COST_CONTRACT_COMPACT << endl;
        assert(temp == FILECODE_COST_CONTRACT_COMPACT);
    }

    MyInputFile >> type_is_piecewise;
    MyInputFile >> type_is_nonlinear;
    MyInputFile >> target;

    //Check that contract is not piecewise and nonlinear and the same time
        if(type_is_piecewise && type_is_nonlinear)  {
        cout << "***************** ERROR MESSAGE **********************" << endl;
        cout << "Vaidation of cost contract file integrity failed" << endl;
        cout << "Filename and path causing error:\t" << filename << endl;
        cout << "Error:\t" << "Type is both piecewise linear and nonlinear" << endl;
        cout << "***************** ERROR MESSAGE **********************" << endl;
        assert(!(type_is_piecewise && type_is_nonlinear));
    }

    if(type_is_piecewise){
        //Get the number of regions
        MyInputFile >> nbRegions;

        //Get the bounds
        Bound.clear();
        float tempFloat;
        for (int r = 0; r < nbRegions; r++) {
            MyInputFile >>tempFloat;
            Bound.push_back(tempFloat);
        }

        //Get the sharing ratios
        SharingRatio.clear();
        for (int r = 0; r < nbRegions; r++) {
            MyInputFile >>tempFloat;
            SharingRatio.push_back(tempFloat);
        }
    }else if(type_is_nonlinear){
        MyInputFile >> maxIncentive;
        MyInputFile >> maxDisincentive;
        MyInputFile >> LowerBound;
        MyInputFile >> UpperBound;
    }//END if-else checking which contract type is used

    MyInputFile >> contractType;

    MyInputFile.close();

}//END import cost contract method
```

2.1.2 Duration Contracts

The encoding of the duration contracts is similar to that of the cost contracts, but also includes the following values giving more information on the possible inclusion of lump-sum incentive amounts:

lump_sum_is_used Boolean indicating if the contract includes a lump sum.

lump_sum_target_time The target duration associated with the lump sum, "NA" in case no lump sum is included.

lump_sum_amount The amount of the lump sum incentive, "NA" in case no lump sum is included.

The duration contracts can be imported using C++ code similar to the following:

```
void DurationContract::importDurationContractCompact(string filename){
    ifstream MyInputFile;
    MyInputFile.open(filename.c_str());

    //Test if the file has the correct format
    long int temp;
    MyInputFile >> temp;
    if (temp != FILECODE_DURATION_CONTRACT_COMPACT) {
        //Display information on the error
        cout << "**************** ERROR MESSAGE ******************************" << endl;
        cout << "Vaidation of duration contract file integrity failed" << endl;
        cout << "Filename and path causing error:\t" << filename << endl;
        cout << "Error:\t" << "Incorrect file identification code at start of file" << endl;
        cout << "Observed filecode:\t" << temp << endl;
        cout << "Expected filecode:\t" << FILECODE_DURATION_CONTRACT_COMPACT << endl;
        assert(temp == FILECODE_DURATION_CONTRACT_COMPACT);
    }


    MyInputFile >> type_is_piecewise;
    MyInputFile >> type_is_nonlinear;
    MyInputFile >> target;

    //Check that contract is not piecewise and nonlinear and the same time
    if(type_is_piecewise && type_is_nonlinear) {
        cout << "**************** ERROR MESSAGE ************************" << endl;
        cout << "Vaidation of duration contract file integrity failed" << endl;
        cout << "Filename and path causing error:\t" << filename << endl;
        cout << "Error:\t" << "Type is both piecewise linear and nonlinear" << endl;
        cout << "**************** ERROR MESSAGE ************************" << endl;
        assert(!(type_is_piecewise && type_is_nonlinear));
    }

    if(type_is_piecewise){
        //Get the number of regions
        MyInputFile >> nbRegions;

        //Get the bounds
        Bound.clear();
        float tempFloat;
        for (int r = 0; r < nbRegions; r++) {
            MyInputFile >>tempFloat;
            Bound.push_back(tempFloat);
        }

        //Get the sharing ratios
        RegionValuation.clear();
        for (int r = 0; r < nbRegions; r++) {
            MyInputFile >>tempFloat;
            RegionValuation.push_back(tempFloat);
        }
    }else if(type_is_nonlinear){
        MyInputFile >> maxIncentive;
        MyInputFile >> maxDisincentive;
        MyInputFile >> LowerBound;
        MyInputFile >> UpperBound;
    }//END if-else checking which contract type is used

    MyInputFile >> lump_sum_is_used;
    if (lump_sum_is_used) {
```

```
        MyInputFile >> lumpSumTargetTime;
        MyInputFile >> lumpSumIncentiveAmount;
    }

    MyInputFile >> contractType;

    MyInputFile.close();

}//END import duration contract
```

2.1.3 Scope Contracts

The encoding of the scope contracts is identical to the encoding of the cost contracts and will not be repeated here.

2.2 Evaluation (Payoff) Models

The instances of the evaluation (payoff) models are all included in a single file "gen4-payoffs.txt". This file includes the following information:

PO_ID A unique ID for the instance.

owner_has_deadline A boolean indicating if there is a certain deadline which is relevant to the project owner.

DL In case a deadline is included (previous value is TRUE), the specific date which is relevant.

LB_I_tot_ratio The lower bound for the maximal net contractor gain. Note that this value is a ratio and still has to be multiplied with the average cost of a project to get the real lower bound.

R_I_tot_ratio The minimal size for the range of contractor outcomes, i.e. the minimal difference between the maximal and minimal owner incentive. Again this value still has to be multiplied with the average cost of the project to get the correct value.

E_cost_m The maximal cost of effort.

ROI_E The average return on investment the contractor gets from effort investments.

timeval_alpha The owner's financial valuation of a single time unit.

timeval_beta The owner's valuation of the project deadline. If this value is zero, the owner does not care for a specific deadline.

scopeval_gamma The owner's valuation of a single unit of scope.

The way in which to import this data is trivial and will not be detailed here.

2.3 Trade-off Models

The data points of the trade-off model instances are stored in separate files *"ProjectTradeoff*.txt"*. The properties of these instances are summarised in the file *"TradeoffFilesLog.txt"*. The following properties of the trade-off instances are summarised in the logfile:

TO_ID Unique identification code for the trade-off instance.

variedParam The parameter type which is different from its default value, a property of the generation procedure.

slope_D The slope of the linear approximation of the relationship between the duration and the cost of the project, assuming other dimensions are set at their lowest cost mode.

slope_S The slope of the linear approximation of the relationship between the scope and the cost of the project, assuming other dimensions are set at their lowest cost mode.

slope_E The slope of the linear approximation of the relationship between the effort and the cost of the project, assuming other dimensions are set at their lowest cost mode.

M_D_S (and variants) The value for the multiplicators, this specific name corresponds with the m_D^S multiplicator.

Cmin The minimal cost of the project, i.e. the cost when all independent dimensions are set to their lowest cost option.

CM_D The convexity magnitude of the relationship between the project duration and the project cost.

CM_S The convexity magnitude of the relationship between the project scope and the project cost.

CM_E The convexity magnitude of the relationship between the project effort and the project cost.

The tradeoff files themselves do not contain any additional information to reduce the file size, the information in these files can be imported using the code below. The code itself is self-explanatory.

```
ProjectTradeoffs::ProjectTradeoffs(string filename){
    ifstream MyInputFile;
    MyInputFile.open(filename.c_str());

    //Test if the file has the correct format
    long int temp;
    MyInputFile >> temp;
    assert(temp == COMPACT_TRADEOFF_EXPORT_FILE_CODE);

    MyInputFile >> this->nD;
    MyInputFile >> this->nS;
    MyInputFile >> this->nE;
    this->nL = (nD + 1) * (nS + 1) * (nE + 1);

    MyInputFile >> this->D_0;
    MyInputFile >> this->D_n;
    MyInputFile >> this->S_0;
    MyInputFile >> this->S_n;
    MyInputFile >> this->E_0;
    MyInputFile >> this->E_n;

    MyInputFile >> this->slope_D;
    MyInputFile >> this->slope_S;
    MyInputFile >> this->slope_E;

    MyInputFile >> this->M_D_S;
    MyInputFile >> this->M_D_E;
    MyInputFile >> this->M_S_D;
    MyInputFile >> this->M_S_E;
    MyInputFile >> this->M_E_D;
    MyInputFile >> this->M_E_S;

    MyInputFile >> this->Cmin;

    MyInputFile >> this->CM_D;
    MyInputFile >> this->CM_S;
```

```
    MyInputFile >> this->CM_E;

    //Tests if input values are sensible

    //The number of options has to be strictly positive
    assert(nD > 0);
    assert(nS > 0);
    assert(nE > 0);

    //The index _0 should be the lowest cost option
    assert(D_0 > D_n);
    assert(S_0 < S_n);
    assert(E_0 > E_n);

    //The slopes should all be inputted as positive values
    assert(slope_D >= 0.0);
    assert(slope_S >= 0.0);
    assert(slope_E >= 0.0);

    //The multipliators should be in [0, +inf[
    assert(M_D_E >= 0.0);
    assert(M_D_E >= 0.0);
    assert(M_S_D >= 0.0);
    assert(M_S_E >= 0.0);
    assert(M_E_D >= 0.0);
    assert(M_E_S >= 0.0);

    //The minimal cost should be positive
    assert(Cmin >= 0.0);

    //The convexity magnitude should be within the range [0, (n-1)/n[
    assert(CM_D >= 0.0 && CM_D < (float) (nD - 1) / nD);
    assert(CM_S >= 0.0 && CM_S < (float) (nS - 1) / nS);
    assert(CM_E >= 0.0 && CM_E < (float) (nE - 1) / nE);

    //Fill the appropriate vectors
    for(int i = 0; i <= nD; i++){
        D.push_back(D_0 - (float) i * (D_0 - D_n) / nD);
        dD.push_back((float) i * (D_0 - D_n) / nD);
    }

    for(int j = 0; j <= nS; j++){
        S.push_back(S_0 + (float) j * (S_n - S_0) / nS);
        dS.push_back((float) j * (S_n - S_0) / nS);
    }

    for(int k = 0; k <= nE; k++){
        E.push_back(E_0 - (float) k * (E_0 - E_n) / nE);
        dE.push_back((float) k * (E_0 - E_n) / nE);
    }


    //Initialise the vectors holding the cost and delta cost values, read numbers from file
    deltaCost.resize(nD+1);
    cost.resize(nD+1);
    for(int i = 0; i <= nD; i++){
        deltaCost[i].resize(nS + 1);
        cost[i].resize(nS + 1);
        for(int j = 0; j <= nS; j++){
            deltaCost[i][j].resize(nE + 1);
            cost[i][j].resize(nE + 1);
```

```
            for(int k = 0; k <= nE; k++){
                MyInputFile >> deltaCost[i][j][k];
                cost[i][j][k] = Cmin + deltaCost[i][j][k];
            }//k
        }//j
    }//i

    MyInputFile.close();

    //Fill the array used to convert l values to i,j,k values
    fillConversionArray();

    //Calculate the average cost
    calculateAverageCost();
}//END constructor: load from file
```