TIMES Migration Feasibility Study

A. Lehtilla (VTT), T. Sharma (UCC), O. Balyk (DTU),
G. Giannakidis (ETSAP), M. Gargiulo (E4SMA)

ETSAP Workshop
Gothenburg 17 June 2018

Motivation

Cost of the tools is one of the main issues for developing countries to use ETSAP's tools.
An open source language will reduce the cost that has to be paid by new users in developing countries for using TIMES.

Objective

Aim: to develop a feasibility study on the possibility to migrate the TIMES code in a new language. The focus of the project will be on the comparison between GAMS and alternative open source options (e.g. Python, Julia/JuMP), to identify the best alternative.

Overview of TIMES Model Generator Tasks

- Extensive pre-processing of input data, including:
  - Interpolation / extrapolation of time-series attribute
  - Levelisation of time slice-specific attributes
  - Creation of various control sets to speed up model generation
  - Model reduction (optional; useful but not necessary)
- Calculation of final model constraint matrix coefficients:
  - Capacity transfer coefficients
  - Coefficients for capacity-related flows
  - Process transformation coefficients
  - Attribute shaping coefficients
  - Objective function coefficients
- Model generation: Generation of equations and variables for the solver
- Processing of results returned by the solver:
  - Calculation of numerous reporting attributes

Assessing the Difficulties in Migrating TIMES

- Pre-processing of data in TIMES:
  - Thousands of lines of very compact GAMS code
  - Not easily translated to other types of languages (e.g. Julia)
  - Amount of code lines would probably increase substantially
- Calculation of final model coefficients:
  - About 1000 lines of very compact and efficient GAMS code
  - Not easily translated to other types of languages (e.g. Julia)
- Model generation:
  - Large number of model equations, but manageable
  - Can be relatively straightforward due to similarity in types of constructs
  - Translating user constraint generation might be a complex case
- Processing of results returned by the solver
  - About 2000 lines of very compact GAMS code
  - Not easily translated to other types of languages (e.g. Julia)

Possible Two-step option for phased migration strategy considered initially.

- GAMS Demo mode (without license) is officially stated to be free for:
  - Academic users (unlimited free use of GAMS for data processing)
  - Any other “instructional and evaluation purposes”

    Note: For GAMS versions before v24.7 no such restrictions existed

- GAMS could be used for TIMES pre-processing and reporting, while model generation and solution could be done on an open-source platform.

Free Demo and Unrestricted Time Limited License

1. The FREE DEMO may be used to evaluate THE SOFTWARE, or to use THE SOFTWARE for instructional purposes.

However it seems that the current GAMS End User License Agreement makes it very clear that using the unlicensed (Demo) version as proposed is most definitely a violation of the EULA.

Alternatives analysed at this stage:

GNU MathProg

GNU MathProg is a high-level language for creating mathematical programming models. GNU MathProg is part of the GLPK solver (and a subset of the AMPL modeling language). MathProg can also be referred to as GMPL (GNU Mathematical Programming Language).

Alternatives analysed at this stage:

Julia / Jump

Julia is a high-level, high-performance dynamic programming language for numerical computing. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. **JuMP** is a domain-specific modeling language for mathematical optimization embedded in Julia. It currently supports a number of open-source and commercial solvers for a variety of problem classes, including linear programming, mixed-integer programming, second-order conic programming, semidefinite programming, and nonlinear programming.

Alternatives analysed at this stage:

Python / Pyomo

Pyomo is a Python-based open-source software package that supports a diverse set of optimization capabilities for formulating, solving, and analyzing optimization models.

A core capability of Pyomo is modeling structured optimization applications. Pyomo can be used to define general symbolic problems, create specific problem instances, and solve these instances using commercial and open-source solvers.

Pyomo's modeling objects are embedded within a full-featured high-level programming language providing a rich set of supporting libraries.

TIMES Migration to GNU MathProg

- GNU MathProg is a widely used open source modelling tool
  - E.g. the OSeMOSYS model generator is based on GNU MathProg
- Language is declarative and has very limited support for data manipulation
  - Not a feasible option for TIMES unless pre-processing and reporting can be done mostly with other tools
  - GAMS could be used for TIMES pre-processing and reporting
- Advantages of this option:
  - Possibly the smallest effort in terms of TIMES translation
  - Would be feasible to support all TIMES basic model variants
- Disadvantages of this option:
  - Not a serious option for a complete migration strategy
  - Data transfers and some processing is required (notable overheads)
  - Performance of model generation is not good for large models

TIMES Migration to Julia / Jump

- Julia is a widely used open source modelling tool
- The Julia language has rich declarative and procedural features and is thus well suitable for data manipulation
  - Feasible option for TIMES under a complete migration strategy
  - For first phase, GAMS could be used for pre-processing and reporting
- Advantages of this option:
  - Reasonably small effort for a first-phase TIMES translation (equations)
  - Would make complete migration feasible in a step-wise manner
  - Would be feasible to support at least all TIMES basic model variants
- Disadvantages of this option:
  - Data transfers and some processing is required (reasonable overheads)
  - Performance of model generation does not seem as good as in GAMS, but it is indeed much better than in GNU MathProg

TIMES Migration to Python / Pyomo

- Python is a widely used open source software
- Pyomo is a Python module designed for formulating optimisation models
- The Python language has rich declarative and procedural features, large standard library, as well as numerous specialised software packages and is thus well suitable for data manipulation:
  - Feasible option for TIMES under a complete migration strategy
  - For first phase, GAMS could be used for pre-processing and reporting
- Advantages of this option:
  - Reasonably small effort for a first-phase TIMES translation (equations)
  - Would make complete migration feasible in a step-wise manner
  - Would be feasible to support all TIMES basic model variants
- Disadvantages of this option:
  - Data transfers and some processing is required (reasonable overheads)
  - Performance of model generation of the prototype falls behind GNU MathProg

Testing Prototypes for TIMES

- For GNU Mathprog, a prototype with a sufficient basic set of model equations was implemented, in order to be able to run a full test model (“Utopia”).

- The GNU MathProg prototype was then reimplemented in Python/Pyomo
  - The constructs needed in Python / Pyomo appeared reasonably similar to the GNU MathProg implementation.

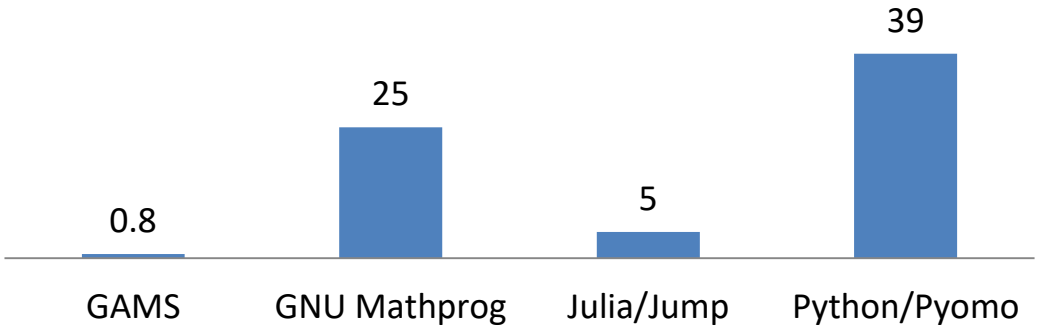
  - Compared to GNU MathProg, no additional non-python software was needed (i.e. sqlite driver).

Testing Prototypes for TIMES

- For Julia / Jump, test implementation was made for a single TIMES equation
  - The constructs needed in Julia appeared reasonably similar to the GNU Mathprog implementation (test case: EQ_ACTFLO).
  - Implementing a basic set of model equations should be reasonably straightforward in Julia / Jump as well.

Testing Prototypes for TIMES

- Some initial experience with performance (~300,000 EQ_ACFLO equations):



Time for generating
the test equations in seconds

- Performance of Julia / Jump appears acceptable, but falls well behind GAMS.
- Performance of Python / Pyomo can probably be optimised to be between Julia / Jump and GNU MathProg.
- Overhead caused by data transfer and data type conversions estimated roughly ~1 min for a model with ~1 million equations (for Julia / Jump).

Cost related issues to be considered

- Cost of GAMS academic licenses is low – **but** in developing countries the prospective users are usually Governmental institutions so they cannot benefit from the low academic licenses.
- GAMS Application single license at $6400 with CPLEX - Other platforms will probably not be able to offer the CPLEX **discount** now available through GAMS for TIMES. GAMS Application single license at $1600 with free solvers.
- Can the alternative modelling languages handle generating/reporting of large production models reliably and within a reasonable timeframe? It seems that Julia has such possibilities but an **IT specialist** should be part of the development team in a possible migration exercise.
- The cost of **redeveloping** and then the cost of **maintaining both GAMS & the alternative platform** needs to be quantified.

As a first conclusion

- From the technical point of view it seems that the performance of Julia/Jump could be acceptable for the equations generation.
- Julia/Jump and Python/Pyomo seem plausible alternative options for a full scale migration (Since the option of using GAMS for processing initially is not possible (due to the EULA) the two step approach is not possible).
- Computer scientists **should** be part of the development team to harness all the possibilities of Julia/Python.
- Solvers: it seems that open source alternatives close to CPLEX/Gurobi speed-wise for LP models exist. Both Pyomo and Jump allow interfacing with a number of open source solvers (as well as commercial).
- The development costs and the annual maintenance cost of an alternative platform needs to be quantified.

Comparison of Formulations (ex: EQ_ACTFLO)

GAMS

```
EQ_ACTFLO(RTP_VINTYR(R,V,T,P),S)$(PRC_TS(R,P,S)$PRC_ACT(R,P))..
   VAR_ACT(R,V,T,P,S)$RTP_VARA(R,T,P)  =E=
   SUM(RPC_PG(R,P,C)$RTPCS_VARF(R,T,P,C,S),(VAR_FLO(R,V,T,P,C,S)$RP_FLO(R,P) +
   SUM(RPC_IRE(R,P,C,IE)$RP_AIRE(R,P,IE), VAR_IRE(R,V,T,P,C,S,IE))$RP_IRE(R,P)) /
     PRC_ACTFLO(R,V,P,C));
```

GNU MathProg:

```
s.t. EQ_ACTFLO{(r,v,t,p) in RTP_VINTYR, s in PRC_TS[r,p]:PRC_ACT[r,p]}:
   RTP_VARA[r,p,t] * PrcAct[r,v,t,p,s] =
   sum{c in RP_PGC[r,p]} (if RP_FLO[r,p] then PrcFlo[r,v,t,p,c,s] else
   sum{ie in IMPEXP:RPC_IRE[r,p,c,ie] && RP_AIRE[r,p,ie]} IreFlo[r,v,t,p,c,s,ie]) /
     PRC_ACTFLO[r,v,p,c];
```

Julia / Jump:

```
@constraint(m,EQ_ACTFLO[(r,t,p,v) in RTP_VINTYR, s in PRC_TS[r,p]; PRC_ACT[r,p]],
  (RTP_VARA[r,t,p] ? PrcAct[r,t,p,v,s] : 0) ==
  sum((RP_FLO[r,p] ? PrcFlo[r,t,p,v,c,s] :
  sum(IreFlo[r,t,p,v,c,s,ie] for ie in IMPEXP if RPC_IRE[r,p,c,ie] && RP_AIRE[r,p,ie]))/
    PRC_ACTFLO[r,v,p,c] for c in RP_PGC[r,p]))
```

Python / Pyomo:

```
def EQ_ACTFLO(model,r,v,t,p,s):
  if model.PRC_ACT[r,p] and s in model.RP_TS[r,p]:
     return (model.RTP_VARA[r,p,t] * model.PrcAct[r,v,t,p,s] == sum((sum(
        model.IreFlo[r,v,t,p,c,s,ie] for ie in model.IMPEXP if model.RPC_ISIRE[r,p,c,ie]
        and model.RP_AIRE[r,p,ie]) if model.RP_ISIRE[r,p] else
        model.PrcFlo[r,v,t,p,c,s]/model.PRC_ACTFLO[r,v,p,c]) for c in model.RP_PGC[r,p]))
  else:
     return Constraint.Skip
model.EQ_ACTFLO = Constraint(model.RTP_VINTYR, model.TSLICE, rule=EQ_ACTFLO)
```