**MOSDEX Update**

Dr. Jeremy A. Bloom  
[jeremyblmca@gmail.com](mailto:jeremyblmca@gmail.com)  
February 20, 2019

1. **The Tableau Model**

The usual *matrix* representation of a linear optimization problem takes the *Tableau* form

*Minimize* **cTx**

*Subject to* **Ax ≥ b**

**l ≤ x ≤ u**

The decision variables **x** may be continuous or they may be further restricted to integer or binary domains; although that distinction will become important as this discussion proceeds, we will ignore it for now.

In *component-wise* form, the tableau looks like

*Minimize sum* {*j in* Columns} *cj xj*

*Subject to sum* {*j in* Columns} *aij**xj* ≥ *bi for all i in* Rows

*lj* ≤ *xj* ≤ *uj for all j in* Columns

Two other equivalent forms are also commonly used. The *row-wise* form of the tableau looks like

*Minimize* **cTx**

*Subject to* **aTi∙ x** ≥ *bi for all i in* Rows

**l ≤ x ≤ u**

The *column-wise* form of the tableau looks like

*Minimize sum* {*j in* Columns} *cj xj*

*Subject to sum* {*j in* Columns}***a****∙j* *xj* **≥****b**

*lj ≤ xj ≤ uj for all j in* Columns

Different optimization problems may be most naturally expressed in component-wise, row-wise, or column-wise form. It is important therefore, that a data format standard be able to accommodate all of them.

Most, if not all, mathematical optimization solvers work with the tableau form internally, and thus it is the most natural way of presenting the data for a particular instance to a solver. However, the tableau form masks a significant aspect of most optimization problems: the data, particularly the coefficient matrix **A**, is usually very sparse in practical applications. Solvers, in fact, take advantage of that sparsity in their algorithms to significantly reduce computational effort and speed up solving time. Then too, data handling ahead of solving can also take advantage of sparsity to reduce the volume of data exchanged with the solver. We need not be concerned with the exact representation of sparse matrices used by a solver (there are several commonly used alternatives), since we don’t want to be tied to any particular solver, but abstractly, a sparse matrix is represented by a collection of triples of the form

< *i, j, aij >*

where, of course, only the elements with *aij* ≠ 0 need be present in the data set. We note that sparsity has other consequences as well. Frequently, the coefficient matrix **A** has a lot of structure, with non-zero elements appearing in recognizable patterns, which can be exploited by the solver to further reduce computational effort. That leads to the possibility that the indices *i* and *j* might not be best represented as non-negative integers but rather as more general *tuple* objects.

Impact of sparsity and structure on data handling up-stream of the solver should not be underestimated by the designers of mathematical optimization models. Sometimes the designer has a lot of control over the format of the source data and can structure it to conform to the requirements of the tableau. However, more often, the source data resides in some kind of data store, such as a corporate database, that is outside of her control. In that case, the data must be reshaped for input to the solver. Such transformations can be performed by a custom data pipeline, say as part of the *extraction*, *validation*, *transformation*, and *load* (*EVTL*) process. Domain-specific languages for optimization modeling can also play an important role. For realistic problems encountered in practice, the amount of computational effort required to reshape the instance data into tableau format is non-trivial but often unrecognized. It is sometimes said, for example, that modeling languages add a lot of “overhead” in forming an instance for submission to a solver; however, that “overhead” may simply be the unrecognized data transformation effort, which nevertheless must occur, whether in the modeling layer or as part of the EVTL pipeline. Another source of misperception arises because “textbook” optimization examples are often so small that the transformations can be done manually, so that the reader is unaware that they have taken place at all. It is thus important for optimization application developers to recognize and account for the data restructuring effort wherever it occurs, rather than simply focusing on solver effort as a benchmark.

In developing a data exchange format standard, we distinguish between two types of data exchange that can occur – *instance exchange* and *recipe exchange*. In instance exchange, the data are directly embedded in the model; this type of exchange is accomplished by MPS, among other formats. In recipe exchange, the model and data are separated; as we will argue below, there are many advantages to model/data separation. However, in this kind of exchange, it is necessary to provide a “recipe” for aligning the data elements with the corresponding model elements. As we will discuss below, the recipe typically can be expressed as an SQL SELECT statement. Using recipe exchange, therefore, obligates the optimization designer to provide a means to implement the recipe, such as a relational database. Depending on the architecture of the optimization application, this capability may be easy or hard to accomplish; however, since the data exchange standard cannot enforce a particular architecture, our proposed MOSDEX standard should be able to support both types of data exchange. Bear in mind however, that no matter which type is used, the effort required to reshape the data into the tableau will still need to be done.

1. **The Net1 Example**

This example was discussed previously as net1-v2, which is in recipe form. Extensions to instance form are shown in net1-v3 a, b, and c, respectively component-wise, row-wise, and column-wise. The component-wise form, net1-v3 a, is the closest to the recipe form; the Variables and Constraints have been expanded according to their indexes, and the Coefficients now has an individual entry for each variable and constraint. Although process is straightforward and mechanical, although the example was generated manually. In the row-wise form, net1-v3 b, the Coefficients objects have been moved inside their corresponding Constraints and Decision Expressions. In the column-wise form, net1-v3 c, the Coefficients objects have been moved inside their corresponding Variables, and the Constraints and Decision Expressions and been moved ahead of the Variables, so that they are defined before they are referenced in the variable Coefficients. In all of these examples, we have used the same structure for the coefficients objects; however, the files could be condensed by making some of the fields optional and removing them when they are not needed. For instance, the Name field in the Coefficients object is mostly redundant in the instance form. In the row-wise form, the Constraints field in the Coefficients object is redundant, and similarly for the Variables field in the column-wise form.