Data Mining from Process Models: Visualizations or Induction for Better Comprehension?

Tim Menzies
Department of Computer Science,
Portland State University,
P.O. Box 751
Portland, Oregon 97207-0751
tim@menzies.us

Siri-on Setamanit, David Raffo
Portland State University
School of Business
PO Box 751
Portland OR 97207 USA
david@sba.pdx.edu ssetaman@yahoo.com

An extended abstract for ProSim'04 - May 24-25, 2004
http://www.prosim.pdx.edu/prosim2004

A standard methodology in the process simulation community is to build simulations using high-end *visual programming (VP) system* with powerful graphical front-ends. Many such VP tools exist including Vensim (see Figure 1), the Statemate state-based simulation model by i-Logix [1], the Extend discrete event simulation tool [2], just to name a few. Various benefits come from these VP tools including *acceptance*, *debugging support* and increased *comprehension* of the models.

Acceptance: The visual appeal of these VP tools makes it easier to get user buy-in for the models. This can be especially important early in a fledging simulation project when developers need to achieve early results to gain the most interest from business users.

Debugging: The tracing capabilities of these VP tools means that developers can fix erroneous parts of the models, sooner.

Comprehension: Once the parts are fixed, the whole model is engaged and emergent trends of the entire system can be identified. These trends can be analyzed to make business decisions about (e.g.) better resource allocations at various points in times of the process being simulated.

We take no issue with the first benefit since we have seen it all too often in our work with users. However, it is insightful to carefully review claims two and three.

[1] Statemate and I-Logix are registered trademarks ® of I-Logix Inc. (3 Riverside Drive; Andover, Massachusetts 01810 USA).

[2] Extend and Imagine That are registered trademarks ® of Imagine That, Inc. (6830 Via Del Oro, Suite 230, San Jose, California, 95119 USA).

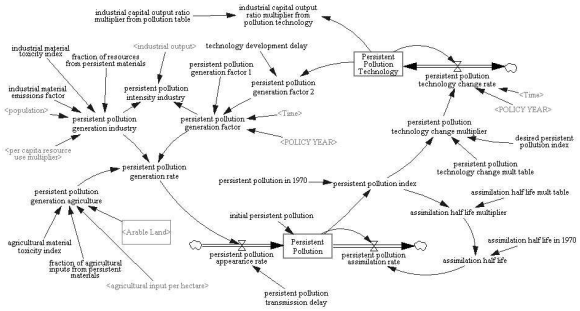

Figure 1. A model displayed in the Vensim environment.

An intuition that might follow from the *debugging* and *comprehension* benefits shown above is that the VP tools used to increased understanding *parts* of the model during debugging would also be useful for better comprehending *all* of the model. We argue that this might not be so; that while visualizations are useful for *local effects* they may not assist in understanding emergent trends of the *entire system*.

Our argument will be in two parts. First, we review the known benefits of visualization. Localization will feature predominately in that review. Second, we compare visualization-based vs textual-based comprehension tools for the emergent properties. At this stage, global effects will be more important that local ones and non-visual tools such as *data miners* can perform better than visual ones.

Visualization: The Theory

Many authors argue that VP tools are a better method for users to interact with a program. For example:

> *Pictures are superior to texts in a sense that they are abstract, instantly comprehensible, and universal.* [2]

In order to analyze these claims, we need a core theory. Larkin and Simon [3] oblige. They distinguish between:

- *Sentential representations* whose contents are stored in a fixed sequence; e.g. propositions in a text.
- *Diagrammatic representations* whose contents are indexed by their position on a 2-D plane.

While these two representations may contain the same information, their computational efficiency may be different. Larkin and Simon present a range of problems modeled in a diagrammatic and sentential representation using production rules. Several effects were noted:

- *Perceptual ease:* Certain features are more easily extracted from diagrams than from sentential representations. For example, adjacent triangles are easy to find visually, but require a potentially elaborate search through a sentential representation.
- *Locality aids search:* Diagrams can group together related concepts. Diagrammatic inference can use the information in the near area of the current focus to solve current problems. Sentential representations may store related items in separate areas, thus requiring extensive search to link concepts.

A common internal representation for a VP systems is one that preserves physical spatial relationships. For example, Narayanan et.al. [6] use Glasgow's array representation [1] to reason about device behaviors. In an array representation, physical objects are mapped into a 2-D grid. Adjacency and containment of objects can be inferred directly from such a representation. Inference engines can then be augmented with diagrammatic reasoning operators which execute over the array (e.g. boundary following, rotation).

Elsewhere, we have been perhaps too critical of the claims for the superiority of visual programming [4]. Our own work with process simulations suggests that adjacent visual entities are typically causally connected. Hence, *debugging local interactions* is made easier by visualizations since a human studying those effects visually gains much from the perceptual ease and locality search of VP.

Non-Visual Analysis

Our introduction took care to distinguish between *debugging* local interactions and *comprehending* global properties that emerge when the whole system runs. Our thesis is that local analysis can miss important interactions amongst the global space.

To demonstrate this, we compared our comprehension of process models using visual and non-visual tools. More specifically: we developed a set of process models using VPs. After debugging we had some sense that we *understood* those models. We called the things we thought we understood our *preconceptions*.

Next, we *sampled* the global interactions of the entire model as follows:

1. Defined a utility function that summarized model output;
2. Conducted thousands of Monte Carlo simulations where the inputs were selected at random from known ranges;

Finally, we *summarized* the sample using data miners. The data miners generated textual output (sentential representations to use Larkin and Simon's terminology) which we studied. When the data miners' summary contradicted our preconceptions, we experimented with the model to confirm or deny the data miners' summary.

A Quick Tutorial on Data Mining

Various learners were used in this study, as described in this section. Learners for continuous classes (e.g. linear regression, regression tree learners) are well understood by the process modeling community. Here, we offer some notes on another class of learners- those that predict for discrete classes.

To generate discrete classes from continuous output, the usual method is to define some *utility function* that summarizes important simulation values into a single simulation value. This distribution of this value is then studied and divided into (e.g.) five equal percentile bands. Each band is then tagged with some symbolic name; e.g. lolo, lo, med, hi, hihi.

Classifiers try to find combinations of *non-class* attributes that predict for a discrete class value. One classifier is J48part [9]. For example:

```
if   wage-increase-first-year > 2.5 AND
     longterm-disability-assistance = yes AND
     statutory-holidays > 10
then class=good

if   wage-increase-first-year <= 4 AND
     working-hours > 36
then class=bad
```

Another classifier is the C45 decision tree learner where each branch is a conjunction and alternate branches are disjunctions [7]. C45 prints its decision trees as follows.

```
wage-increase-first-year <= 2.5: class=bad
wage-increase-first-year > 2.5
|   statutory-holidays <= 10: class=bad
|   statutory-holidays > 10: class=good
```

Association rule learners grant no special meaning to some class attribute. These learners try to find sets of attribute values that occur other. APRIORI is one such associational rule learner [8]. It only executes on discrete attributes. Here's what that learner generates on a data set with 19 classes. Note that no class attribute appears in the output and more than one attribute value can appear after the "==>" symbol (e.g. see association 5).

```
 1. int-discolor=none              ==> sclerotia=absent
 2. mycelium=absent int-discolor=none ==> sclerotia=absent
 3. leaves=abnorm sclerotia=absent ==> mycelium=absent
 4. sclerotia=absent               ==> mycelium=absent
 5. int-discolor=none              ==> mycelium=absent
                                       sclerotia=absent
 6. int-discolor=none sclerotia=absent ==> mycelium=absent
 7. int-discolor=none              ==> mycelium=absent
 8. leaf-malf=absent               ==> mycelium=absent
 9. mycelium=absent                ==> sclerotia=absent
10. leaves=abnorm mycelium=absent  ==> sclerotia=absent
```

Association rule learners can run slower than classifiers since the latter have a very small target concept (the single class attribute) while learners like APRIORI struggle to find associations that predict for any number of attributes.

Lift learners seek combinations of non-class attributes that most improve (or "lift") the weighted distribution of the classes over some baseline distribution. Calculating such a weighted distribution requires users attaching weights to each class; e.g. this class is better than that class. This is easy for data mining from class data tagged using the process described above: each class has a weight equal to the average utility value that falls into that class.

TAR3 is a lift learner that seeks the smallest number of attributes that most lift the baseline [5]. For example, consider a data set describing houses containing 506 examples of great houses (29%), good houses (29%), poor houses (21%) and bad houses (21%). If we assume that $great > good > poor > bad$ then TAR3 learns the rule:

```
6.7 <= rooms < 9.8 AND
12.6 <= parent teacher ratio < 15.9
```

This rule, when applied as a constraint to the housing data, removes all but 39 houses which contain 97% great houses and 3% good houses, and no poor or bad houses. This is called the *control rule* since it offer a policy of what to do to make things better.

Conversely, if we assume that $bad > poor > good > great$ then TAR3 learns the rule:

```
0.6 <= nitrous oxide level < 1.9 AND
17.16 <= living standard < 39.0
```

This rule, when applied as a constraint to the housing data, removes all but 81 houses which contain 98% bad houses and 1% poor houses and 1% good houses and no great houses. This is called the monitor rule since it describes what to watch for to detect the worst situation.

TAR3 can be particularly powerful for summarizing complex learners. The experience with that tool has been that it generally produces much smaller output that the above learners. Our experience is that TAR3's succinct rules are more acceptable to business users.

Results

This work is proceeding and full results will be give an PROSIM. In the meantime, we can say that our data miners *are* surprising us and we have examples were we have learnt more from the sentential outputs of the data miners than from the local effects explored during debugging. For example, in one model, TAR3 found a single attribute that tripled the average output of the utility function- a very strong effect (to say the least) and one that had not be seen before even after months of working with a VP version of that process model.

References

[1] J. Glasgow, H. Narayanan, and B. C. (eds). *Diagrammatic Reasoning : Cognitive and Computational Perspectives*. MIT Press, 1995.

[2] M. Hirakawa and T. Ichikawa. Visual language studies - a perspective. *Software- Concepts and Tools*, pages 61–67, 1994.

[3] J. Larkin and H. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, pages 65–99, 1987.

[4] T. Menzies. Evaluation issues for visual programming languages. In S. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering, Volume II*. World-Scientific, 2002. Available from `http://menzies.us/pdf/00vp.pdf`.

[5] T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from `http://menzies.us/pdf/03tar2.pdf`.

[6] N. H. Narayanan, M. Suwa, and H. Motoda. Behaviour hypothesis from schematic diagrams. In J. Glasgow and B. C. N.H. Narayanan, editors, *Diagrammatic Reasoning*, pages 501–534. The AAAI Press, 1995.

[7] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[8] R.Agrawal, T.Imeilinski, and A.Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD Conference, Washington DC, USA*, 1993. Available from `http://citeseer.nj.nec.com/agrawal93mining.html`.

[9] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.