BASIC STRUCTURAL MODELING PROJECT

Joseph J. Simpson

Mary J. Simpson

04-07-2014

DRAFT DETAILED DESIGN REPORT DRAFT

Version 0.37

Table of Contents

Preface

In the preface to Battelle Monograph Number 4, 1974, Structuring Complex Systems, John N. Warfield stated the following:

> *"In developing this monograph, it was useful to think of structural models of two generic types. The first type, the basic structural models, are those whose theory has evolved out of mathematics. They are the graphs and digraphs which carry no empirical or substantive information. Much is known about their properties. Methods exist for performing operations upon them that permit extensive manipulation and structural insight. The second type, the interpretive structural models, are those developed to help organize and understand empirical, substantive knowledge about complex systems or issues. Intent structures, DELTA charts and decision trees, illustrated in earlier monographs, are examples of interpretive structural models. Other examples include interaction graphs, PERT diagrams, signal-flow graphs, organization charts, relevance trees, state diagrams, and preference charts.*
>
> *If the full knowledge of basic structural models could be brought to bear upon the development of interpretive structural models, a significant advance could be made in the rational analysis and synthesis of complex systems. Yet, it seems impractical to expect that those who are engaged in day-to-day interaction with the complexity in human affairs would take the time to learn to apply such abstract concepts as mathematical logic, matrix theory, and the theory of graphs in their work. It also seems unlikely that mathematicians would take the time to become highly knowledgeable of real-world systems and issues. The dilemma of how to wed substantive issues and knowledge of complex systems to the mathematics seems significant. But even if people had all the mathematics and understood the complex system or issue, still another problem would be present. That is the extreme tyranny of working systematically to establish relations among many elements in the form of an interpretive structural model, and the long time period required to do this by manual methods.*
>
> *One approach shows promise of a way out of the mentioned difficulties. This approach is to introduce the digital computer to aid in problem definition. If the necessary mathematical knowledge as well as the logistical tyranny can be transferred to the computer, leaving the developer of the interpretive structural model only the minimum, but critical, core of effort – providing the substantive knowledge of the system or issue – then the developer would not need to learn the associated mathematics, nor would the he have to absorb the tyranny associated with the extensive manipulation of ideas on paper that would otherwise be required. The computer could be a major factor in compressing the time scale for development of an interpretive structural model.*
>
> *This monograph presents a method whereby the computer can carry out the necessary operations for those interpretive structural models that can be put in correspondence with digraphs. Since the monograph is largely limited to such models, it does not encompass all possible kinds of interpretive structural models."*

It is clear that structural modeling has two distinct components: basic structural modeling and interpretive structural modeling. To create an effective, computer-based tool to support various kinds of interpretive structural modeling applications, the relevant elements associated with basic structural modeling must be placed "in proper correspondence" with the interpretive structural model of interest; that is, the specific mathematical relations and constructs must be paired with an appropriate organizing relationship that defines the interpretive structural model being sought. The task of identifying, evaluating and selecting the correct basic structural modeling elements for a given interpretive structural model is a nontrivial task. The authors developed a third component of structural modeling that addresses the selection, organization, and adaptation tasks associated with transforming basic structural modeling components into interpretive structural models. This third component of structural modeling is called 'structural integration modeling'. The primary artifact that is developed using structural integration modeling is called an abstract relation type (ART).

The ART form and its format encode the mathematical structural knowledge consistent with its intended application to one or more specific interpretive structural model types. Each ART has a prose description, a graphic representation, and a set of executable computer code. As ART forms are developed and placed in an ART form library, a type of structural modeling "pattern language" is developed to support a wide range of interpretive structural models. Figure 1 depicts the general contextual arrangement.

Basic Structural Modeling	Abstract Relation Type	Interpretive Structural Modeling
Contains ***mathematical relations and constructs***	Focuses on the ***organizing relationship*** for a given structure	Consists of structures that reflect ***natural language relationships***
BSM Examples	*ART Examples*	*ISM Examples*
Binary Relations Vector Sets Augmented Boolean Algebra Binary Matrices Binary Matrix Models Interconnection Theory Digraphs Digraph Maps Digraph Models Transitive Embedding	Is-north-of Implies Is-assigned-to Is-more-useful-than Must-precede Is-a-function-of Is-implied-by Is-a-member-of Causes Affects Aggravates Weakens	Intent structures DELTA Chart Decision Trees PERT Diagrams Signal-flow Graphs Organization Charts Relevance Trees State Diagrams Preference Charts N-Squared Charts Design Structure Matrices Ranking (Subordination) Charts

Figure 1. Structural Modeling Context

When the structural modeling process was executed by an individual using manual methods, that individual had to have advanced expertise in the domain application area as well as advanced knowledge of a range of mathematical techniques. When all or part of the structural modeling process is transferred to a computer a new set of expertise in computer programming and software systems design is required. While the computer support makes the structural modeling methods less onerous (or feasible to any person with average intelligence and mathematical ability) it also introduces a new set of issues and problems.

The authors address the structural modeling computer support system development in a phased manner. The first phase of computer support system development addresses the mathematical techniques and structures associated structural modeling. The Symbolic Algebra and Graphical Environment (SAGE) mathematics tool developed and maintained by the University of Washington is used to prototype, develop and communicate the mathematical structural components. The SAGE executable code is used in the executable code section of the ART form. The goal is to create a catalog of canonical ART forms that may be used in a wide range of interpretive structural modeling (ISM) computer applications. During the last 40 years, over 100 different types of ISM computer code bases have been developed. These were closed source systems designed to run on computer systems that are now mostly obsolete. An open source code base that is supported by the SAGE tool will be available for an extended period of time and also be computer system independent.

In the second phase of computer support system development, closed source computer based tools may be developed to support structural modeling techniques, however, these closed source systems should be validated and verified by testing know system configurations and outputs that are generated by the canonical ART form set. This type of approach allows interested individuals to learn the mathematical components by obtaining the SAGE based system. When ready, individuals may create closed source systems that compete on usability features and verify that their system conforms to the structural modeling ART standards and patterns. Individuals could, also, choose to create open source computer systems to support structural modeling and allow other interested individuals the opportunity to analyze the program application structure.

The general types of knowledge and expertise needed in the structural modeling process is shown in Figure 2. The open-source SAGE tool, developed by the University of Washington, is used as the standard tool for the computer-based,

mathematics component of the ART form. The prose section is used by application domain experts to encode the specific types of empirical and substantive information used in the ART form. The "proper alignment" of the basic structural modeling features and interpretive structural modeling domain area is the responsibility of the system scientist or engineer that is developing and testing the ART form.

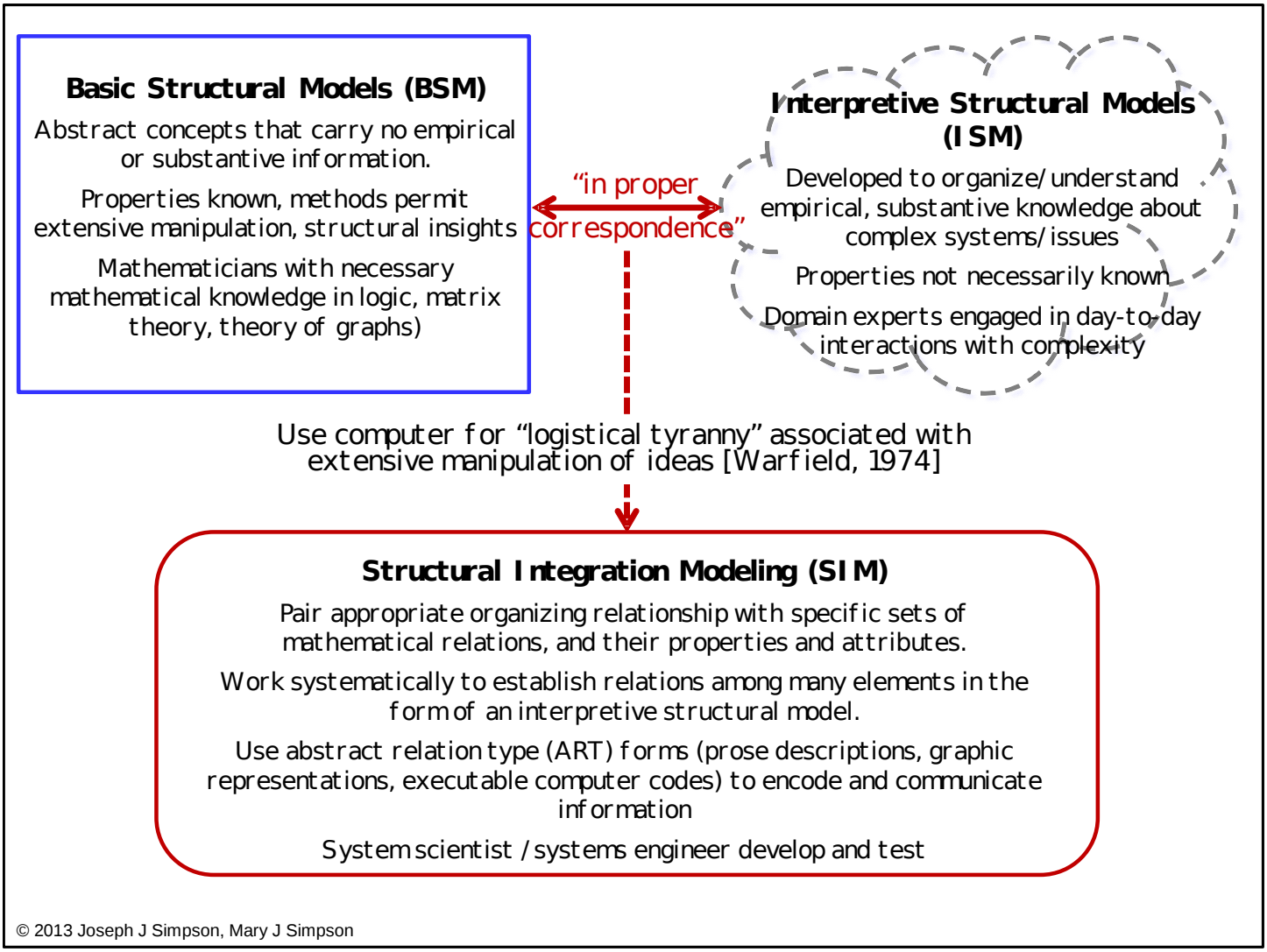

Figure 2. Structural Modeling Components And Activities

Introduction

The Basic Structural Modeling Project (BSMP) is focused on clearly defining the elements of structural modeling, developed by John N. Warfield, and presenting these elements in adaptive, configurable units. Structural Modeling has two components: Basic Structural Modeling (BSM) and Interpretive Structural Modeling (ISM). The authors have added a third component to structural modeling named Structural Integration Modeling (SIM). An abstract relation type (ART), creates unique combinations of the mathematical components found in BSM, and packages these components into uniquely named ART constructs. These ART constructs are then integrated using structural integration modeling techniques to generate a specific ISM approach. Each ART construct has a prose section, a graphics section and a mathematics section designed to present the model-exchange isomorphic versions of the specific natural language relationship of interest. A model-exchange isomorphism is a procedure whereby one type of model can be exchanged for another [Warfield, April, 1974, p2-11].

The ART provides a well-defined, well-documented executable programming construct that can be used with relations and relationships similar to the way in which abstract data types are used with data. An ART construct places the natural language relationship in proper correspondence to the mathematical relation. The ART form directly supports and encodes the structural system relationship methods and techniques developed by Warfield. Sets of ART are being developed and used to create a systems science and/or systems engineering language. The ART of systems engineering and systems science may be combined to produce a type of executable pattern language.

In this document, the organizing, natural-language relationship ‘is-north-of’ is applied to a set of cities. The identity of the individual cities is known, the general structuring relationship is known, but the specific relationship between any two cities is unknown at the beginning of the ISM process. Substantive, empirical data is gathered about the relationship between sets of two cities. Then the inference process is executed to infer additional information about the current state of the system structure. The processes of collecting empirical data, and inferring additional information from the aggregate information set, are repeated until the complete system is properly structured. In this case, the amount of empirical data collected should be minimized to minimize the cost of the total structuring operation.

The mathematical relation used in this case, to represent the ‘is-north-of’ natural language relationship, has a set of properties and attributes. The clear recognition and communication of the mathematical relation’s properties and attributes require a set of constraining rules and definitions. In this specific case, only a small number of rules and definitions are needed. The rules and definitions are:

- A strict subordination matrix form is used, which allows only one city at each level.
- Since only one city is allowed at each level
  if we know that city A is north of city B,
  then we know that city B is south of city A.
- No city is allowed to be north of itself.

These rules create a hierarchy and matrix relation form that is irreflexive, transitive and asymmetric.

It is possible to have an ‘is-north-of’ natural language relationship that allows two cities to be at the same level. This type of ‘is-north-of’ natural language relationship is called a general subordination matrix, and would require a slightly different process and approach. The ART form is designed to clearly document these semantic differences in the natural language relationship, and clearly communicate these differences. Placing natural-language relationships in an ART form establishes a common, well-defined mechanism to present and use structuring system concepts that have similar semantics. Each ART form has two basic types of information: informal information and formal information. These information types are used to create a rich, communications vehicle that may have redundant information and ambiguous information in the informal sections. The formal section refines the informal information, and may contain redundant information, but ambiguity is removed.

Description of Abstract Relation Type (ART) Format

Along with abstract data types, binary matrices and model-exchange isomorphic forms, also impact the design of the ART form and format. The model-exchange approach is used to process a mental model of a system into a binary matrix model of a system, and then into a graphical model of the system. The system mental model may be in the mind of one or more individuals. The three primary system design and description language types are:

1) prose,
2) graphics and
3) mathematics.

The ART form has a dedicated component for each of these three language types. The content and value of language is based on robust methods of definition and communication. As a result, the ART form supports not only the three basic language types, but also four common modes of definition. The modes of definition are:

1) naming,
2) extension,
3) intension, and
4) relationship.

The ART form is designed to fully support all four modes of definition.

A formal naming convention, numbering system, and version-control system are part of the operational context necessary to create, track and use unique ART forms in an unambiguous manner. These contextual rule sets are currently being designed. Typical patterns of language mode application (i.e., prose, graphics, mathematics) are also viewed as a valuable resource, and will be defined along with the systems science and systems engineering ART forms. Appendix C, of this document, contains a description of the ART form and format.

Abstract Relation Type – Prose Section

The prose section of the ART form is divided into two subsections: 1) informal prose and 2) formal prose. An informal prose section presents information in a human readable form that outlines the ART form purpose, application, typical context of application, related ART forms, as well as special cases and frequently asked questions with answers. A formal prose section contains two types of formal prose: 1) formal system patterns and 2) structured executable prose..

This is the ART for the natural language relationship 'is-north-of.' This relationship is in Warfield's spatial category of relationship types.

Prose Section – Informal Prose

Informal name: is-north-of.

Natural-language relationship, purpose, typical application areas. The 'is-north-of' natural language relationship is used to order geographic points on a map. The complete global set of spatial data is not available, and the global arrangement of the geographic points must be generated using data associated with the relationship between and among individual sets of points.

Relationship attributes, properties and application context. The 'is-north-of' relationship is transitive, asymmetric and irreflexive. The 'is-north-of' relationship is paired with the 'is-south-of' natural-language relationship. In this specific case a strict subordination matrix is used, which means that only one city can be located at any given degree of latitude. Given the strict subordination matrix, if we know that city A is north of city B, then we also know that city B is south of city A.

Similar ART forms, typical groups of application ART forms used together. Other similar spatial natural-language relationships are 'is-east-of,' 'is-west-of,' 'is-right-of,' and 'is-left-of.'

Prose Section – Formal Prose

Formal name. Initial-Incomplete-Is-North-Of-Strict-Subordination

Formal version number. Version 0.03 : 03-24-2014

Formal system pattern.

- Pattern Name. Strictly-Is-North-Of
- Problem statement. This general pattern is used to order objects in a strict hierarchy. When the general, natural language ordering relationship is known, but the specific relationship between any two objects is unknown, the problem occurs. This pattern is used to minimize the cost of gathering the information needed to create the ordered hierarchy.
- Forces. The simple relationship between two object creates a situation where the number of possible connections is the square of the number of objects under consideration. Empirical data about the individual connections can be used to support formal inference about other object connection sets.
- Application Context.
- Related patterns.
- Problem Solution.

Abstract Relation Type – Graphics Section

The ART graphics section is divided into two parts: informal and formal. Similar to the prose section, the informal graphics section may contain redundant and ambiguous graphics in an effort to create an 'information-rich' communication channel. The formal graphics section contains formal graphics that follow a well-defined specification and/or are executable in an open-source graphics tool.

Graphics Section – Informal Graphics

Each ART form must have an informal graphics section.

Informal graphics associated with the information in the prose section - and/or created to communicate the system, context, operations and values of interest - are located in this section of the document. These graphics should focus on the binary matrix form, object configuration, or other substantive features of the current problem structure.

Graphics Section – Formal Graphics

Each ART form must have a formal graphics section.

Formal graphics associated with the information in the prose section and the mathematics section are located in this section of the ART form. Each ART form must have a binary matrix or a primary structuring graphics display form that is directly associated with the system relation of interest. Other executable graphic types may be located here as long as the executable code is available in the mathematics section.

Abstract Relation Type – Mathematics Section

The mathematics section is divided into two parts: equations and executable code. The equations section presents the current equations of interest, and the executable code section presents the computer code to implement the equations.

A general set of equations may be developed and added as an appendix to the applicable ART form documents. The executable code that supports the mathematical operations are presented in the open-source, SAGE math format. Other types of executable code may be provided, but the SAGE math code (or some other example executable code) is required for a complete ART form.

The SAGE math code for this ART form is composed of two files. The first file is the SAGE worksheet. The second file is the load_random_city.sage input file.

Two Javascript files used in the bsmp web application are also presented in this section. All the code presented here plus the other files needed for an executable web application are available on GitHub at: https://github.com/jjs0sbw/bsmp

The contents of the first file (Random_City_Generator) are:

```
Random_City_Generator
system:sage

<h2>Notebook to Generate City Test Data </h2>
<p>This notebook supports the development and exploration of structural modeling
techniques by creating a known order of 19 cites. Once the known order is
complete then the city data is randomly scrambled and a sample of the random
city order data is produced. The random set of city data is used with the city
ordering process to structure and order the cities. The output from the city
ordering process is compared with the known city order. In this manner the city
structuring methods and processes can be tested to determine if they perform as
expected. </p>

{{{id=1|
#place the load_twenty.sage file in a directory that can be accessed and change
the line below to reflect the directory
attach("/Users/jjs0sbw/Desktop/Projects/BSMP/sage_docs/sage_scripts/
load_random_generator.sage")
start_process()
///
Start Process Begin ...
Start Process End ...
}}}

{{{id=13|
#this is a random process so it may not complete, if not just press the evaluate
button until it completes.
```

input_list = random_input_creation(19)
input_list
///
The make input list function creates a list of city and number pairs.
The list is then used to create the randomized input to the program.
City number list is:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]

The known ordered set is:
{'A': 9, 'C': 12, 'B': 1, 'E': 16, 'D': 17, 'G': 4, 'F': 0, 'I': 18, 'H': 2,
'K': 3, 'J': 7, 'M': 13, 'L': 5, 'O': 15, 'N': 10, 'Q': 8, 'P': 11, 'S': 14,
'R': 6}

Alphabetical Order is:
A - 9 - 1

B - 1 - 2

C - 12 - 3

D - 17 - 4

E - 16 - 5

F - 0 - 6

G - 4 - 7

H - 2 - 8

I - 18 - 9

J - 7 - 10

K - 3 - 11

L - 5 - 12

M - 13 - 13

N - 10 - 14

O - 15 - 15

P - 11 - 16

Q - 8 - 17

R - 6 - 18

S - 14 - 19

The inverse dict is:
0 - F 1 - 5

1 - B 2 - 1

2 - H 3 - 7

3 - K 4 - 10

4 - G 5 - 6

5 - L 6 - 11

6 - R 7 - 17

7 - J 8 - 9

8 - Q 9 - 16

9 - A 10 - 0

10 - N 11 - 13

11 - P 12 - 15

12 - C 13 - 2

13 - M 14 - 12

14 - S 15 - 18

15 - O 16 - 14

16 - E 17 - 4

17 - D 18 - 3

18 - I 19 - 8

The ramdon sample input list is:
input_list contents are: [[[True, 3, 10, 'K', 'N'], [True, 3, 5, 'K', 'L'],
[False, 3, 12, 'K', 'C'], [False, 3, 0, 'K', 'F'], [False, 3, 9, 'K', 'A']],
[[True, 2, 14, 'H', 'S'], [True, 2, 18, 'H', 'I'], [False, 2, 16, 'H', 'E'],
[True, 2, 3, 'H', 'K'], [True, 2, 5, 'H', 'L']], [[True, 16, 4, 'E', 'G'],
[True, 16, 3, 'E', 'K'], [False, 16, 9, 'E', 'A'], [True, 16, 11, 'E', 'P'],
[True, 16, 6, 'E', 'R']], [[False, 4, 16, 'G', 'E'], [True, 4, 18, 'G', 'I'],
[False, 4, 9, 'G', 'A'], [True, 4, 11, 'G', 'P']], [[False, 14, 3, 'S', 'K'],
[False, 14, 13, 'S', 'M'], [False, 14, 15, 'S', 'O'], [False, 14, 6, 'S', 'R'],
[False, 14, 0, 'S', 'F']], [[True, 0, 8, 'F', 'Q'], [True, 0, 2, 'F', 'H'],
[True, 0, 10, 'F', 'N'], [False, 0, 9, 'F', 'A']], [[False, 8, 1, 'Q', 'B'],
[False, 8, 18, 'Q', 'I'], [False, 8, 11, 'Q', 'P'], [True, 8, 14, 'Q', 'S'],
[False, 8, 5, 'Q', 'L']], [[True, 17, 0, 'D', 'F'], [True, 17, 10, 'D', 'N'],
[False, 17, 12, 'D', 'C'], [True, 17, 13, 'D', 'M'], [False, 17, 1, 'D', 'B']],
[[False, 6, 7, 'R'
, 'J'], [False, 6, 0, 'R', 'F'], [False, 6, 17, 'R', 'D'], [False, 6, 4, 'R',
'G']], [[False, 5, 4, 'L', 'G'], [False, 5, 2, 'L', 'H'], [True, 5, 15, 'L',
'O'], [False, 5, 17, 'L', 'D'], [True, 5, 8, 'L', 'Q']], [[False, 7, 16, 'J',
'E'], [True, 7, 5, 'J', 'L'], [False, 7, 9, 'J', 'A'], [False, 7, 4, 'J', 'G'],
[True, 7, 6, 'J', 'R']], [[False, 18, 1, 'I', 'B'], [True, 18, 14, 'I', 'S'],
[True, 18, 7, 'I', 'J'], [True, 18, 8, 'I', 'Q'], [False, 18, 2, 'I', 'H']],
[[False, 13, 17, 'M', 'D'], [True, 13, 6, 'M', 'R'], [True, 13, 14, 'M', 'S'],
[True, 13, 15, 'M', 'O'], [False, 13, 18, 'M', 'I']], [[True, 9, 8, 'A', 'Q'],
[True, 9, 0, 'A', 'F'], [True, 9, 12, 'A', 'C'], [True, 9, 3, 'A', 'K']],
[[False, 11, 17, 'P', 'D'], [False, 11, 12, 'P', 'C'], [False, 11, 3, 'P', 'K'],
[False, 11, 5, 'P', 'L']], [[True, 1, 3, 'B', 'K'], [True, 1, 13, 'B', 'M'],

[True, 1, 2, 'B', 'H'], [True, 1, 16, 'B', 'E'], [True, 1, 10, 'B', 'N']],
[[False, 10, 5, 'N', 'L'], [True, 10, 6, 'N', 'R'], [False, 10, 17, 'N', 'D'],
[False, 10,
18, 'N', 'I'], [True, 10, 14, 'N', 'S']], [[False, 15, 3, 'O', 'K'], [True, 15,
8, 'O', 'Q'], [True, 15, 6, 'O', 'R'], [False, 15, 5, 'O', 'L'], [False, 15, 16,
'O', 'E']], [[True, 12, 4, 'C', 'G'], [True, 12, 7, 'C', 'J'], [True, 12, 10,
'C', 'N'], [True, 12, 5, 'C', 'L'], [True, 12, 18, 'C', 'I']]]

length_list is: [5, 5, 5, 4, 5, 4, 5, 5, 4, 5, 5, 5, 5, 4, 4, 5, 5, 5, 5]

[[3, 10], [3, 5], [12, 3], [0, 3], [9, 3], [2, 14], [2, 18], [16, 2], [2, 3],
[2, 5], [16, 4], [16, 3], [9, 16], [16, 11], [16, 6], [16, 4], [4, 18], [9, 4],
[4, 11], [3, 14], [13, 14], [15, 14], [6, 14], [0, 14], [0, 8], [0, 2], [0, 10],
[9, 0], [1, 8], [18, 8], [11, 8], [8, 14], [5, 8], [17, 0], [17, 10], [12, 17],
[17, 13], [1, 17], [7, 6], [0, 6], [17, 6], [4, 6], [4, 5], [2, 5], [5, 15],
[17, 5], [5, 8], [16, 7], [7, 5], [9, 7], [4, 7], [7, 6], [1, 18], [18, 14],
[18, 7], [18, 8], [2, 18], [17, 13], [13, 6], [13, 14], [13, 15], [18, 13], [9,
8], [9, 0], [9, 12], [9, 3], [17, 11], [12, 11], [3, 11], [5, 11], [1, 3], [1,
13], [1, 2], [1, 16], [1, 10], [5, 10], [10, 6], [17, 10], [18, 10], [10, 14],
[3, 15], [15, 8], [15, 6], [5, 15], [16, 15], [12, 4], [12, 7], [12, 10], [12,
5], [12, 18]]
}}}

The contents of the second file (load_random_generator.sage) are:

 # Basic Structural Modeling (J. N. Warfield) mathematics.
 # Copyright (C) 2013, 2014 Joseph James Simpson
#
 # This program is free software: you can redistribute it and/or modify it
 # under the terms of the GNU General Public License as published by the
 # Free Software Foundation, either version 3 of the License, or
 # (at your option) any later version.
#
 # This program is distributed in the hope that it will be useful,
 # but WITHOUT ANY WARRANTY; without even the implied warranty of
 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 # See the GNU General Public License for more details.
#
 # You should have received a copy of the GNU General Public License
 # along with this program. If not, see <http://www.gnu.org/licenses/>.

```
import random
import copy

# create global variables for input sage script and out files
# outfile directory structure... place some where your sage program can access
outfile = '/Users/jjs0sbw/Desktop/Projects/BSMP/sage_docs/sage_out/rcl_2.txt'



# empty swap list
swap_element_ab = []

## Display license and open output file
## First function
def start_process():
    """
    Displays the GPL 3 license opens the output file for data
    """
    print "Start Process Begin ... "
    open_file()  ## Open output file and write data OK
    print "Start Process End ..."




## Need to create a function that creates the randomized input data set
## and returns the known data set and the randomized sample input data
def random_input_creation(matrix_size):
    """
    Creates a randomized list of of cities.  This list is used to test
    the structuring process.  This function has three sub-functions:
    -- create input data lists
    -- create input sample data list
    """

    city_names_list_ab, city_number_list_ab, input_sample_data_list_ab = create_input_data_lists(matrix_size)

    input_sample_data_list_in = create_input_sample_data_list(input_sample_data_list_ab)
```

```
    #create return tuple
    input_matrix_sample_data = (input_sample_data_list_in)

    # return tuple
    return input_matrix_sample_data


# random input creation sub function two (ric 1)
def create_input_data_lists(matrix_size):
    # print notice of actions
    print 'The make input list function creates a list of city and number pairs.'
    print 'The list is then used to create the randomized input to the program.'

    # initialize array variables
    input_list_ab = []
    city_names_list_ab = []
    city_number_list_ab = []

    # populate the array variables with data from returned tuple
    city_names_list_ab, city_number_list_ab, input_sample_data_list_ab = make_input_lists(matrix_size)

    # write list to file
    input_list_ab_out(input_sample_data_list_ab) ## file_out OK

    input_data_lists = (city_names_list_ab, city_number_list_ab, input_sample_data_list_ab)

    return input_data_lists

# random input creation sub function three (ric 2)
def create_input_sample_data_list(input_sample_data_list_ab):
    # print notice of action
    #print 'PROCESS INPUT LIST START: Create in_one_list....\n'
    print '\nThe ramdon sample input list is: '

    #create new array variable
    input_list_in = []

    #populate new variable
    input_sample_data_list_in = process_input_list(input_sample_data_list_ab)
```

```
        #write to file
    input_list_in_out(input_sample_data_list_in) ## file_out OK

    return input_sample_data_list_in

#create a known city order for test purposes
#create a randomized set of data to use as unknown data input for test purposes
def make_input_lists(matrix_size):
    """
    This function creates and returns a list of city and number pairs.
    The list is then used to create the randomized input to the program
    """
    #city name tages
    city_tags_ab = 'ABCDEFGHIJKLMNOPQRS'

    #new variable lists
    city_number = []
    city_name = []

        #create initial populated city name and number lists
        #for each list cell number in the range of 0 to matrix size minus one
    for z in range(0, int(matrix_size)):
        #place the number in the city number cell
      city_number.append(z)

      #place the city name in the city name cell
      city_name.append(city_tags_ab[z])

    #print city number to out file
    city_number_out(city_number) ## file_out OK

    #print city name to out file
    city_name_out(city_name) ## file_out OK


    #print city number to screen
    print 'City number list is: '
    print city_number
```

```
        #create new variables populated with random paired city data
name_num_ab = random.sample(city_number,int(matrix_size))
name_tag_ab_new = random.sample(city_name, int(matrix_size))

#create data pairs
name_pair_ab = zip(name_num_ab, name_tag_ab_new)

#create city data pair dictionaries
dict_1_ab = dict(name_pair_ab)

#create known pairs dictionary
known_pairs_ab = dict((v,k) for k,v in dict_1_ab.iteritems())

#print known pairs data to the screen
print '\nThe known ordered set is: '
print known_pairs_ab
print '\n'

#print "The original order (unknown order) is: "
#for k,v in known_pairs_ab.iteritems():
        #print k,v

print "\n Alphabetical Order is: "
p_str("\n Alphabetical Order is: \n")
for k in city_tags_ab:
        p_str_out = str(k) + " - " + str(known_pairs_ab[k]) + " - " + str(city_tags_ab.index(k) + 1) + " \n"
        p_str(p_str_out)
        print p_str_out

inv_ko = {v:k for k,v in known_pairs_ab.items()}

print "The inverse dict is: "
p_str("\n The inverse dict is: \n")
for k,v in inv_ko.iteritems():
        p_str_out_1 = str(k) + " - " + str(v) + "   " + str(k+1) + " - " + str(city_tags_ab.index(v)) + " \n"
    p_str(p_str_out_1)
        print p_str_out_1

        #print the known pairs data to the out file
known_pairs_ab_out(known_pairs_ab)  ## file_out OK
```

```
#create known city order list
known_order_ab = [[x,known_pairs_ab[x]] for x in name_tag_ab_new]

#create random city data set
data_ab = random.sample(city_number,int(matrix_size))

#create random city data set
#data_set_1_ab = [[x,random.sample(data_ab, int(matrix_size/3))] for x in data_ab]  ############
data_set_1_ab = [[x,random.sample(data_ab, int(matrix_size))] for x in data_ab]

        #print random data set to out file
data_set_1_ab_out(data_set_1_ab)  ## file_out OK

        #create total list variable
total_sample_data_list = []

#for x in the range of zero (0) to matrix size
for x in range(0,int(matrix_size)):
        #create a new list variable
    list_sub = []

    #for num in the range of zero (0) to (matrix size)
    for num in range(0,5):
        #list_sub = [] # moved list_sub  up to other loop
        #if the x and y coordinates are not equal to each other
        #create some test code to print out the data elements
        if( dict_1_ab[data_set_1_ab[x][0]] != dict_1_ab[data_set_1_ab[x][1][num]]):
         #if the x element is less that the y element
         #create test code to print out the elements being compared
            flag_1 = dict_1_ab[data_set_1_ab[x][0]] < dict_1_ab[data_set_1_ab[x][1][num]]

            #append elements to the list sup list...
            list_sub.append([flag_1, data_set_1_ab[x][0], data_set_1_ab[x][1][num],dict_1_ab[data_set_1_ab[x]
[0]],dict_1_ab[data_set_1_ab[x][1][num]]])

        #create copy of the list_sub to return..
    total_sample_data_list.append(copy.deepcopy(list_sub))
```

```
        #print the total_list to the out file
    total_list_out(total_sample_data_list)  ## file_out OK

    #create return tuple
    lists_back = (city_name, city_number, total_sample_data_list)

    #return the tuple
    return lists_back



#take the randomized list of city information and process the information to
#structure the cities and recreate the known city order
def process_input_list(input_list):
    print "input_list contents are: " + str(input_list) + " \n"
    length_list = []
    for i in range (0,len(input_list)):
        length_list.append(len(input_list[i]))

    print "length_list is: " + str(length_list) + " \n"
    #create and initialize length variables
    length_one = len(input_list)
    length_two = len(input_list[0])

        #print process input list action header to the out file
    process_input_list_out()  ## file_out OK

        #print length of the input list to out file
    length_inout_list_out(len(input_list))  ## file_out OK

        #print length of the first sub list the out file
    length_inout_list_out_0(len(input_list[0]))  ## file_out OK

        #create new list variable
    in_one_list_ab = []

    #for x in the range of zero (0) to the length of list one
    #create some tests at print results to the screen
    for x in range(0,length_one):
```

```
        #for y in the range of zero (0) to length two
    for y in range(0,len(input_list[x])):
        #append the value of input list x,y to the in_one_list
        in_one_list_ab.append(input_list[x][y])

        #append the first, second and third values from the lists in list one to list two
in_two_list = [in_one_list_ab[x][0:3] for x in range(len(in_one_list_ab))]

        #print in_two_list to the out file
in_two_list_out(in_two_list)  ## file_out OK

        #create the correct variable order for the items in list three
in_three_list = order_entry(in_two_list)

        #print list three to the out file
in_three_list_out(in_three_list)  ## file_out OK

        #return list three
return in_three_list

#creates the proper order for the items in the input list
def order_entry(in_list):
        #create new list variable
    out_list = []

    #for x in the range of zero (0) to the length of the in list minus one
    for x in range(len(in_list)):

        #check to see if the first entry in the the word True
        if(in_list[x][0] == True):
            # if True then entries in proper order
            #append the second and third item in the sub list to the out list
            out_list.append(in_list[x][1:3])

        #if False entries need to be swapped
        if(in_list[x][0] == False):
            #create new list variable
            new_cell = []

            #enter third entry first
```

```
        new_x = in_list[x][2]

        #append new_x to new cell
        new_cell.append(new_x)

        #enter second entry second
        new_y = in_list[x][1]

        #append new y to new cell
        new_cell.append(new_y)

        #append the swapped entry cell to the out list
        out_list.append(new_cell)

    #return the out list
    return out_list

##########
############

### start file section -- functions to print data to out file

def p_str(in_str): ## Open output file and write data
    #Place appropriate file path in next statement
    f_out_ab = open(outfile, 'a')
    f_out_ab.write(in_str)
    f_out_ab.close() ## start_file header complete

#open out file and write start header -- keep this one
def open_file(): ## Open output file and write data
    #Place appropriate file path in next statement
    f_out_ab = open(outfile, 'w')
    f_out_ab.write('##############################################    START PROCESS   ####\n')
    f_out_ab.write('####  START FILE ab: Open output file and write data         ####\n')
    f_out_ab.write('#########################################################################\n\n')
    f_out_ab.close() ## start_file header complete



#open the out file and print in put list to file -- keep this one
def input_list_ab_out(input_list_ab):
```

```
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('########################################    INPUT LIST AB      ####\n')
    f_out_ab.write('#### INPUT LIST AB :  \n' + str(input_list_ab)+    '         ####\n')
    f_out_ab.write('#########################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## list out complete

#open out file and print list input -- keep this one
def input_list_in_out(input_list_in):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('########################################    INPUT LIST IN     ####\n')
    f_out_ab.write('#### INPUT LIST IN :  \n' + str(input_list_in)+    '         ####\n')
    f_out_ab.write('#########################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## list out complete

#open out file and print city number list -- keep this one
def city_number_out(city_number):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('########################################   CITY NUMBER     #####\n')
    f_out_ab.write('#### CITY NUMBER IS :  \n' + str(city_number)+    '         ####\n')
    f_out_ab.write('#########################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## city number out complete

#open out file and print city name list -- keep this one
def city_name_out(city_name):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('########################################    CITY NAME          ####\n')
    f_out_ab.write('#### CITY NAME IS :  \n' + str(city_name)+    '              ####\n')
    f_out_ab.write('#########################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## city name out complete

#open out file and print known pairs  -- keep this one
def known_pairs_ab_out(known_pairs_ab):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('##########################   KNOWN PAIRS  (ORDERED STRUCTURE) ####\n')
    f_out_ab.write('#### KNOWN PAIRS ARE :  \n' + str(known_pairs_ab)+    '       ####\n')
    f_out_ab.write('#########################################################################\n')
```

```
    f_out_ab.write('       \n')
    f_out_ab.close() ## known pairs out complete

#open out file and print data set 1 -- need better description of list -- keep this one
def data_set_1_ab_out(data_set_1_ab):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('#################################       DATA SET AB         ####\n')
    f_out_ab.write('#### DATA SET 1 AB IS :  \n' + str(data_set_1_ab)+    '      ####\n')
    f_out_ab.write('##########################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## data set 1 out complete

#open out file and print total list -- need better description of list -- keep this one
def total_list_out(total_list):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('#################################          TOTAL LIST       ####\n')
    f_out_ab.write('#### TOTAL LIST IS :  \n' + str(total_list)+    '           ####\n')
    f_out_ab.write('##########################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## total list out complete

#open out file and print header for list processing -- keep this one
def process_input_list_out():
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('#################################      PROCESS INPUT LIST    ####\n')
    f_out_ab.write('####  PROCESS INPUT LIST START: Create in_one_list..         ####\n')
    f_out_ab.write('##########################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## process input list out complete

#open out file and print length of input list -- keep this one
def length_inout_list_out(length):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('##########################################################################\n')
    f_out_ab.write('#### LENGTH OF INPUT LIST IS :  \n' + str(length)+'  ####\n')
    f_out_ab.write('##########################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## length input list out complete

#open out file and print length of sub list -- keep this one
```

```
def length_inout_list_out_0(length):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('################################################################################\n')
    f_out_ab.write('#### LENGTH OF INPUT LIST[0] IS :  \n' + str(length)+'  ####\n')
    f_out_ab.write('################################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## length input list [0] list out complete

#open out file and print in two list -- keep this one
def in_two_list_out(in_two_list):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('################################################################################\n')
    f_out_ab.write('#### IN TWO LIST IS :  \n' + str(in_two_list)+'  ####\n')
    f_out_ab.write('################################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## in two list out complete

#open out file and print in three list -- keep this one
def in_three_list_out(in_three_list):
    f_out_ab = open(outfile, 'a')
    f_out_ab.write('################################################################################\n')
    f_out_ab.write('#### IN THREE LIST IS :  \n' + str(in_three_list)+'  ####\n')
    f_out_ab.write('################################################################################\n')
    f_out_ab.write('       \n')
    f_out_ab.close() ## in three list out complete
```

The contents of the first Javascript file bsmp_binary_math.js are:

```
"use strict";

/*
 * Copyright (c) 2013, 2014 Joseph J. Simpson
 * This file is part of the bsmp_wa__1_v0_11 program.
 * This file in bsmp_wa__1_v0_11 is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by the Free Software
 * Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * The bsmp_wa_1 program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE.  See the GNU General Public License for more details.
 *
```

```
 * You should have received a copy of the GNU General Public License along with bsmp_wa_1.
 * If not, see <http://www.gnu.org/licenses/>
 */

var BSMP = {
  limit: 1e-5,
  notice: "Special binary matrix and vector type.",
  reference: "Societal Systems: Planning, Policy, and Complexity, 1976 - J. Warfield."
};

BSMP.VecBin = function() {};

BSMP.VecBin.new_one = function(ele) {
  var Vb = new BSMP.VecBin();
  return Vb.setElements(ele);
};

var $Vb = BSMP.VecBin.new_one;

BSMP.VecBin.O = function(n) {
  var ele = [];
  for(i=0; i<n; i++)
  {
          ele.push(0);
  }
  return BSMP.Vector.new_one(ele);
};

BSMP.VecBin.l = function(n) {
  var ele = [];
  for(i=0; i<n; i++)
  {
          ele.push(1);
  }

  return BSMP.VecBin.new_one(ele);
};

 BSMP.VecBin.prototype = {
  e: function(i) {
```

```
  return (i < 1 || i > this.elements.length) ? null : this.elements[i-1];
 },

 setE: function(x, y) {
  if(x < 1|| x > this.elements.length) {
        alert ("Element out of range, please enter valid element.");
  } else {
        this.elements[x-1] = y;
  }
 },

 indexOf: function(x) {
  var index = null;
  var n = this.elements.length;
  for (var i = 0; i < n; i++) {
   if (index === null && this.elements[i] == x) {
    index = i + 1;
   }
  }
  return index;
 },

 view: function() {
  return '[' + this.elements.join(', ') + ']';
 },


 setElements: function(els) {
  this.elements = (els.elements || els).slice();
  return this;
 }
};

BSMP.MatrixBin = function() {};

BSMP.MatrixBin.new_one = function(els) {
 var MB = new BSMP.MatrixBin();
 return MB.setElements(els);
};
var $Mb = BSMP.MatrixBin.new_one;
```

```
BSMP.MatrixBin.Id = function(n) {
  var els = [];
  var i;
  var j;
  for(i = 0; i < n; i++) {
    els[i] = [];
  for(j = 0; j < n; j++) {
      els[i][j] = (i === j) ? 1 : 0;
    }
  }
  return BSMP.MatrixBin.new_one(els);
};



 BSMP.MatrixBin.O = function(a) {
  var els = [];
  var i;
  var j;
  for(i = 0; i < a; i++) {
    els[i] = [];
  for(j = 0; j < a; j++) {
      els[i][j] = 0;
    }
  }
  return BSMP.MatrixBin.new_one(els);
};



 BSMP.MatrixBin.prototype = {
  e: function(i,j) {
    if (i < 1 || i > this.elements.length || j < 1 || j > this.elements[0].length) {
          alert ("Matrix element, to return, is out of range, please enter valid element cell.");
          return null;
    }
    return this.elements[i-1][j-1];
  },
```

```
setE: function(i,j, v) {
  if (i < 1|| i > this.elements.length || j < 1 || j > this.elements[0].length) {
        alert ("Matrix element, to set, is out of range, please enter valid element cell.");
  } else {
        this.elements[i-1][j-1] = v;
  }

},

row: function(i) {
  if (i < 1 || i > this.elements.length) {
        alert ("Matrix row, to return, is out of range, please enter valid row.");
        return null;
   }
  return BSMP.VecBin.new_one(this.elements[i-1]);
},

setRow: function(i, newRow) {
        if(this.elements.length === 0) {
        alert ("Matrix (row) is empty. Select a matrix with content.");

        } else if ( i < 1 || i > this.elements.length) {
                alert ("Matrix row, to replace, is out of range, please enter valid row.");
                return null;
        } else {
                this.elements[i-1] = newRow.elements ;
        }

},

col: function(j) {
  if (this.elements.length === 0) {
        alert ("Matrix (col) is empty. Select a matrix with content.");
        return null;
  }
  if (j < 1 || j > this.elements[0].length) {
        alert ("Matrix column, to return, is out of range, please enter valid column .");
        return null;
  }
  var col = [], n = this.elements.length;
```

```
  for (var i = 0; i < n; i++) { col.push(this.elements[i][j-1]); }
  return BSMP.VecBin.new_one(col);
},

setCol: function(j, newColumn) {
        var n;
        var j;
        var i;
  if (this.elements.length === 0) {
        alert ("Matrix column, to set, is out of range, matrix is empty. Select a matrix with content.");
        return null;
  }
  if (j < 1 || j > this.elements[0].length) {
        alert ("Matrix column, to set is out of range, please enter valid column .");
        return null;
  }
  n = this.elements.length;
  for (var i = 0; i < n; i++) {
        this.elements[i][j-1] = newColumn.e(i+1);
  }
},

rows: function() {
  return this.elements.length;
},

colus: function() {
  if (this.elements.length === 0) { return 0; }
  return this.elements[0].length;
},

dup: function() {
  return BSMP.MatrixBin.new_one(this.elements);
},

map: function(fn, context) {
  if (this.elements.length === 0) { return BSMP.MatrixBin.new_one([]); }
  var els = [];
  var i = this.elements.length;
  var nj = this.elements[0].length;
```

```
  var j;
  while (i--) {
    j = nj;
    els[i] = [];
    while (j--) {
      els[i][j] = fn.call(context, this.elements[i][j], i + 1, j + 1);
    }
  }
  return BSMP.MatrixBin.new_one(els);
},

isSameSizeAs: function(matrix) {
  var M = matrix.elements || matrix;
  if (typeof(M[0][0]) === 'undefined') { M = BSMP.MatrixBin.new_one(M).elements; }
  if (this.elements.length === 0) { return M.length === 0; }
  return (this.elements.length === M.length &&
      this.elements[0].length === M[0].length);
},

add: function(matrix) {
  if (this.elements.length === 0) return this.map(function(x) { return x });
  var M = matrix.elements || matrix;
  if (typeof(M[0][0]) === 'undefined') { M = BSMP.MatrixBin.new_one(M).elements; }
  if (!this.isSameSizeAs(M)) { return null; }
  return this.map(function(x, i, j) { return x + M[i-1][j-1]; });
},

 subtract: function(matrix) {
  if (this.elements.length === 0) return this.map(function(x) { return x });
  var M = matrix.elements || matrix;
  if (typeof(M[0][0]) === 'undefined') { M = BSMP.MatrixBin.new_one(M).elements; }
  if (!this.isSameSizeAs(M)) { return null; }
  return this.map(function(x, i, j) { return x - M[i-1][j-1]; });
},

leftMultiply: function(binary_matrix) {
  if (this.elements.length === 0) { return false; }
  var M = binary_matrix.elements || binary_matrix;
  if (typeof(M[0][0]) === 'undefined') { M = BSMP.MatrixBin.new_one(M).elements; }
  return (this.elements[0].length === M.length);
```

```
},

boolMultiply: function(binary_matrix) {
  if (this.elements.length === 0) { return null; }
  if (!binary_matrix.elements) {
    return this.map(function(x) { return x * binary_matrix; });
  }
  var returnVector = binary_matrix.modulus ? true : false;
  var M = binary_matrix.elements || binary_matrix;
  if (typeof(M[0][0]) === 'undefined') { M = BSMP.MatrixBin.new_one(M).elements; }
  if (!this.leftMultiply(M)) { return null; }
  var i = this.elements.length;
  var nj = M[0].length;
  var j;
  var cols = this.elements[0].length;
  var c;
  var elements = [];
  var sum;
  while (i--) { j = nj;
    elements[i] = [];
    while (j--) { c = cols;
      sum = 0;
      while (c--) {
        sum += this.elements[i][c] * M[c][j];
      }
      if(sum === 0) { elements[i][j] = 0;}
      if(sum !== 0) { elements[i][j] = 1;}
    }
  }
  var M = BSMP.MatrixBin.new_one(elements);
  return returnVector ? M.col(1) : M;
},

setElements: function(els) {
  var i;
  var j;
  var elements = els.elements || els;
  if (elements[0] && typeof(elements[0][0]) !== 'undefined') {
    i = elements.length;
    this.elements = [];
```

```
    while (i--) {
     j = elements[i].length;
     this.elements[i] = [];
     while (j--) {
      this.elements[i][j] = elements[i][j];
     }
    }
    return this;
   }
   var n = elements.length;
   this.elements = [];
   for (i = 0; i < n; i++) {
    this.elements.push([elements[i]]);
   }
   return this;
  },

  view: function() {
   var matrix_rows = [];
   var n = this.elements.length;
   if (n === 0) return '[]';
   for (var i = 0; i < n; i++) {
    matrix_rows.push(BSMP.VecBin.new_one(this.elements[i]).view());
   }
   return matrix_rows.join('<br>');
  },

};
```

The contents of the second Javascript file bsmp.js are:

```
 "use strict";
/*
 * Copyright (c) 2013, 2014 Joseph J. Simpson
 * This file is part of the bsmp_wa__1_v0_11 program.
 * This file in bsmp_wa__1_v0_11 is free software: you can redistribute ist and/or modify
 * it under the terms of the GNU General Public License as published by the Free Software
 * Foundation, either version 3 of the License, or (at your option) any later version.
 *
```

```
* The bsmp_wa_1 program is distributed in the hope that it will be useful, but WITHOUT
* ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE.  See the GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along with bsmp_wa_1.
* If not, see <http://www.gnu.org/licenses/>
*/


// Add javascript code here to draw the BSMP Grid
// and add the initial color scheme..
// Global variables -- time to refactor these out --
// need a global function and/or library -- use strict
var gridText;
var gridColor;
var vNames;

window.onload = function() {
        document.getElementById("one").value = "N";
            document.getElementById("two").value = "N";
                                document.getElementById("northTrue").style.display="none";
                                document.getElementById("northFalse").style.display="none";
                                document.getElementById("moveData").style.display="none";

                                var initButton = document.getElementById("initButton");
                                initButton.onclick = initHandler;

    };

// initialize the window with a blank canvas
// and data input area..

    function initHandler() {
                    var canvas = document.getElementById("BSMP_1");
                    var context = canvas.getContext("2d");
                    var canvas_1 = document.getElementById("BSMP_2");
                    var context_1 = canvas_1.getContext("2d");
                    var canvas_2 = document.getElementById("BSMP_3");
                    var context_2 = canvas_2.getContext("2d");
```

```
                    gridText = BSMP.MatrixBin.O(19);
                    gridColor = BSMP.MatrixBin.O(19);

                    vNames = BSMP.VecBin.new_one([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]);

                    initGrid(canvas, context, canvas_1, context_1, canvas_2, context_2, gridText, vNames);

                    document.getElementById("one").value = "N";
            document.getElementById("two").value = "N";
                    document.getElementById("moveData").style.display="none";
                    document.getElementById("northTrue").style.display="none";
        document.getElementById("northFalse").style.display="none";

        //write out gridText
        //writeToDocOne(gridText.view());

        function writeToDocOne(message) {
                document.getElementById("test_out").innerHTML = message;
    }



    }


//#############################################################################################
 //#############################################################################################
 //#####
 //#####       Functions to initialize the 19 by 19 grid
 //#####
 //#############################################################################################
 //#############################################################################################

         function drawYellowCell(x, y, canvas, context, cellSize) {
                var x1 = x;
                var y1 = y;
      context.fillStyle = "yellow";
      context.fillRect(x1 * cellSize , y1 * cellSize , cellSize - 1, cellSize - 1);
    }
```

```
function enterYellowColor(x,y) {
        gridColor.setE(x+1,y+1,1);
}

function drawXGridCell(x, canvas_1, context_1, cellSize, vNames) {
        var text;
    context_1.fillStyle = "wheat";
    context_1.fillRect((x-1) * cellSize, 0, cellSize - 1, cellSize - 1);
    context_1.fillStyle = "black";
    context_1.font = "1em tahoma";
    text = vNames.e(x);
    if(x < 10){
        context_1.fillText(text, (x * cellSize)-20, 15);
    }
    if(x >=10) {
        context_1.fillText(text, (x * cellSize)-20, 15);
    }
}

function drawYGridCell(y, canvas_2, context_2, cellSize,vNames) {
    var text;
    context_2.fillStyle = "wheat";
    context_2.fillRect(0, (y-1) * cellSize, cellSize - 1, cellSize - 1);
    context_2.fillStyle = "black";
    context_2.font = "1em tahoma";
    text = vNames.e(y);
    context_2.fillText(text, 0, (y * cellSize) - 5);
}

function drawRedCell(x, y, canvas, context, cellSize) {
    context.fillStyle = "red";
    context.fillRect(x * cellSize , y * cellSize , cellSize - 1, cellSize - 1);
}

function enterRedColor(x,y) {
        gridColor.setE(x+1,y+1,2);
}

function drawGreenCell(x, y, canvas, context, cellSize) {
    context.fillStyle = "green";
```

```
   context.fillRect(x * cellSize , y * cellSize , cellSize - 1, cellSize - 1);
}

function enterGreenColor(x,y) {
        gridColor.setE(x+1,y+1,3);
}

function drawLightBlueCell(x, y, canvas, context, cellSize) {
   context.fillStyle = "lightblue";
   context.fillRect(x * cellSize , y * cellSize , cellSize - 1, cellSize - 1);
}

function enterLightBlueColor(x,y) {
        gridColor.setE(x+1,y+1,4);
}

function drawOrangeCell(x, y, canvas, context, cellSize) {
   context.fillStyle = "orange";
   context.fillRect(x * cellSize , y * cellSize , cellSize - 1, cellSize - 1);
}

function enterOrangeColor(x,y) {
        gridColor.setE(x+1,y+1,5);
}



        function drawText(x, y, canvas, context, cellSize, gridText){
          var text;
          context.fillStyle = "black";
          context.font = "1em tahoma";
          context.textAlign = "center";
          text = gridText.e(x+1, y+1);
          context.fillText(text, (((x + 1) * cellSize) - (cellSize / 2)), (((y + 1) * cellSize) - (cellSize / 4)));

        }

        function drawText_1(x, y, canvas, context, cellSize){
          context.fillStyle = "black";
          context.font = "1em tahoma";
          context.textAlign = "center";
```

```
    context.fillText("1", (((x + 1) * cellSize) - (cellSize / 2)), (((y + 1) * cellSize) - (cellSize / 4)));

  }

  function drawText_0(x, y, canvas, context, cellSize){
    context.fillStyle = "black";
    context.font = "1em tahoma";
    context.textAlign = "center";
    context.fillText("0", (((x + 1) * cellSize) - (cellSize / 2)), (((y + 1) * cellSize) - (cellSize / 4)));

  }


  function initGrid(canvas, context, canvas_1, context_1, canvas_2, context_2) { // remove vNames, gridText
var x;
var y;
var cellSize = 20; // should be in global function
context.fillStyle = "black";
context.fillRect(0,0, canvas.width, canvas.height)
context_1.fillStyle = "black";
context_1.fillRect(0,0, canvas_1.width, canvas_1.height)
context_2.fillStyle = "black";
context_2.fillRect(0,0, canvas_2.width, canvas_2.height)

for (x = 0; x < 19 ; x++) {
  drawXGridCell(x+1, canvas_1, context_1, cellSize, vNames);

 for (y = 0; y < 19; y++) {
        if(x != y){
        drawYellowCell(x, y, canvas, context, cellSize);
        enterYellowColor(x, y);
        drawYGridCell(y+1, canvas_2, context_2, cellSize, vNames);
  }

  if(x == y){

                drawRedCell(x, y, canvas, context, cellSize);
                enterRedColor(x, y);
  }
```

```
            drawText(x, y, canvas, context, cellSize, gridText);
           }
       }
    }

//#########################################################################################################
//#########################################################################################################
//#####
//#####       Functions to enter data into the 19 by 19 grid
//#####
//#########################################################################################################
//#########################################################################################################
function enterData() {
        var rcOne = document.getElementById("one").value;
        var rcTwo = document.getElementById("two").value;
        var canvas = document.getElementById("BSMP_1");
        var context = canvas.getContext("2d");
        var cellSize = 20;
        var color = 0;
        var rcOneIndex;
        var rcTwoIndex;
        var ei;
        var ex;
        var ey;

        // get index numbers for the entered values
        rcOneIndex = vNames.indexOf(rcOne);
        rcTwoIndex = vNames.indexOf(rcTwo);

        if (!((rcOneIndex >= 1) && (rcOneIndex <= 19) && (rcTwoIndex >= 1) && (rcTwoIndex <= 19))) {
                alert ("Please enter a value from 1 to 19 in each box");
        } else if (rcOneIndex === rcTwoIndex){
                alert ("Please enter two different values.. values can not be the same.");
        } else {
                // need to select the correct index of the input values
                gridText.setE(rcOneIndex, rcTwoIndex, 1);
                gridText.setE(rcTwoIndex, rcOneIndex, 0);
                enterRedColor(rcTwoIndex-1, rcOneIndex-1);
                enterGreenColor(rcOneIndex-1, rcTwoIndex-1);
```

```
                // need to redraw the main canvas using text values
                // use the gridColor matrix
                for (ex = 0; ex < 19 ; ex++) { // reduce to 0 and <
        for (ey = 0; ey < 19; ey++) { // reduce to 0 and <
         color = gridColor.e(ex+1,ey+1); // take out the + 1
         switch(color)
         {
                case 1:
                        drawYellowCell(ex, ey, canvas, context, cellSize);
                        break;
                case 2:
                        drawRedCell(ex, ey, canvas, context, cellSize);
                        break;
                case 3:
                        drawGreenCell(ex, ey, canvas, context, cellSize);
                        break;
                case 5:
                        drawLightBlueCell(ex, ey, canvas, context, cellSize);
                        break;
         }

         drawText(ex, ey, canvas, context, cellSize, gridText);
         }
                }
                document.getElementById("northTrue").style.display="inline";
                document.getElementById("northFalse").style.display="inline";
                document.getElementById("entryButton").style.display="none";
     document.getElementById("processButton").style.display="none";
     document.getElementById("inferenceButton").style.display="none";

                //writeToDocOne(gridColor.inspect());
         }

}

//#############################################################################################################
//#############################################################################################################
//
//   Now build a function to handle the order assessment
//
```

```
//#########################################################################################################
//#########################################################################################################

function noSwapRC() {
	document.getElementById("one").value = "N";
	document.getElementById("two").value = "N";
	document.getElementById("northTrue").style.display="none";
  document.getElementById("northFalse").style.display="none";
  document.getElementById("entryButton").style.display="inline";
  document.getElementById("processButton").style.display="inline";
  document.getElementById("inferenceButton").style.display="inline";

}

function processData() {
	document.getElementById("moveData").style.display="inline";

}

//#########################################################################################################
//#########################################################################################################
//
//   Now build a function to infer new information
//   from the existing properly formed matrix (all green points in the lower triangular.)
//
//#########################################################################################################
//#########################################################################################################
function inferenceProcess() {
	var cellSize = 20; // should be in global function variable
	var canvas = document.getElementById("BSMP_1");
	var context = canvas.getContext("2d");
	var canvas_1 = document.getElementById("BSMP_2");
	var context_1 = canvas_1.getContext("2d");
	var canvas_2 = document.getElementById("BSMP_3");
	var context_2 = canvas_2.getContext("2d");

	var tempGridColor;
	var tempGridText;
	var tempId;
	var rMatrix;
```

```
        var rmOne;
        var rmTwo;
        var rmOut_1;
        var rmOut_2;
        var rmDiff_1;
        var rmDiff_2;
        var colorInferred;
        var textInferred;
        var ii;
        var ix;
        var iy;

        tempGridColor = gridColor.dup();
        tempGridText = gridText.dup();
        tempId = BSMP.MatrixBin.Id(19);
        rMatrix = tempGridText.add(tempId);

        reachabilityMatrix(rMatrix);

        //############### draw code here ########
// need to redraw the main canvas using text values
        for (ix = 0; ix < 19 ; ix++) {
                drawXGridCell(ix+1, canvas_1, context_1, cellSize, vNames); //added +1
    for (iy = 0; iy < 19; iy++) {
          var color;
                drawYGridCell(iy+1, canvas_2, context_2, cellSize, vNames); //added +1
        color = gridColor.e(ix+1,iy+1);
        switch(color)
        {
                case 1:
                        drawYellowCell(ix, iy, canvas, context, cellSize);
                        break;
                case 2:
                        drawRedCell(ix, iy, canvas, context, cellSize);
                        break;
                case 3:
                        drawGreenCell(ix, iy, canvas, context, cellSize);
                        break;
                case 5:
                        drawLightBlueCell(ix, iy, canvas, context, cellSize);
```

```
                        break;
            }

            drawText(ix, iy, canvas, context, cellSize, gridText);
        }
            }

            //############## draw code end ###########

}

function reachabilityMatrix(matrixIn) {
            var rmOne;
            var rmTwo;
            var rmOut_1;
            var rmOut_2;
            var rmDiff_1;
            var rmDiff_2;
            var colorInferred;
            var textInferred;
            var tempGridColor;
            var tempGridText;
            var done = new Boolean(0);
            var loopFlag = 25;
            var ri;
            var rx;
            var matrix_sum;

// start utility function here
            rmOne = matrixIn.dup();
            rmTwo = matrixIn.dup();
            rmOut_1 = rmOne.boolMultiply(rmTwo);
            rmOut_2 = rmOut_1.boolMultiply(rmTwo);
            tempGridColor = gridColor.dup();
            tempGridText = gridText.dup();

            // start a loop to calculate the reachability matrix
            while((!done) && (loopFlag > 0)) {
                        rmDiff_1 = rmOut_2.subtract(rmOut_1);
                        //check to see if sum of rmDiff_1 is equal to zero
```

```
                // if not boolMultiply rmOut_2 by rmTwo
                // continue the boolMultiply until sum of rmDiff_1 is zero
                matrix_sum = rmOut_2.subtract(rmOut_1);
                if(matrix_sum === 0) {
                        done = new Boolean(1);
                        loopFlag = 0;
                }else {
                        rmOut_1 = rmOut_2;
                        rmOut_2 = rmOut_2.boolMultiply(rmTwo);
                }

                loopFlag = loopFlag - 1;
        }

        rmDiff_2 = rmOut_2.subtract(rmOne);

        colorInferred = rmDiff_2.map(function(rx){if(rx >= 1){return 4;}else{return 0;}});
  textInferred = rmDiff_2.map(function(rx){if(rx >= 1){return 1;}else{return 0;}});

  tempGridColor = tempGridColor.add(colorInferred);
  tempGridText =tempGridText.add(textInferred);

  gridColor = tempGridColor;
        gridText = tempGridText;
  // end utility function here....
}

//#########################################################################################################
//#########################################################################################################
//
//   Now build a function to swap the selected row and column pairs
//   That are existing data in the grid (move city data )
//
//#########################################################################################################
//#########################################################################################################

function moveRC() {
        var cellSize = 20;
        var moveOne = document.getElementById("moveOne").value;
        var moveTwo = document.getElementById("moveTwo").value;
```

```
var canvas = document.getElementById("BSMP_1");
var context = canvas.getContext("2d");
var canvas_1 = document.getElementById("BSMP_2");
var context_1 = canvas_1.getContext("2d");
var canvas_2 = document.getElementById("BSMP_3");
var context_2 = canvas_2.getContext("2d");

var tempRowOneColor;
var tempRowTwoColor;
var tempRowOneText;
var tempRowTwoText;

var tempColOneColor;
var tempColTwoColor;
var tempColOneText;
var tempColTwoText;

var tempGridColor;
var tempGridText;
var mx;
var my;
var mi;
var moveOneIndex;
var moveTwoIndex;


tempGridColor = gridColor.dup();
tempGridText = gridText.dup();


moveOneIndex = vNames.indexOf(moveOne);
moveTwoIndex = vNames.indexOf(moveTwo);

vNames.setE(moveTwoIndex, moveOne);
vNames.setE(moveOneIndex, moveTwo);

tempRowOneColor = gridColor.row(moveOneIndex);
tempRowTwoColor = gridColor.row(moveTwoIndex);
tempRowOneText = gridText.row(moveOneIndex);
tempRowTwoText = gridText.row(moveTwoIndex);
```

```
        tempGridColor.setRow(moveTwoIndex, tempRowOneColor);
        tempGridColor.setRow(moveOneIndex, tempRowTwoColor);
        tempGridText.setRow(moveTwoIndex, tempRowOneText);
        tempGridText.setRow(moveOneIndex, tempRowTwoText);

        tempColOneColor = tempGridColor.col(moveOneIndex);
        tempColTwoColor = tempGridColor.col(moveTwoIndex);
        tempColOneText  = tempGridText.col(moveOneIndex);
        tempColTwoText  = tempGridText.col(moveTwoIndex);

        tempGridColor.setCol(moveTwoIndex, tempColOneColor);
        tempGridColor.setCol(moveOneIndex, tempColTwoColor);
        tempGridText.setCol(moveTwoIndex, tempColOneText);
        tempGridText.setCol(moveOneIndex, tempColTwoText);

        gridColor = tempGridColor;
        gridText = tempGridText;

        //############### draw code here ########
// need to redraw the main canvas using text values
        // use the gridColor matrix
        for (mx = 0; mx < 19 ; mx++) {
           var color;
                drawXGridCell(mx+1, canvas_1, context_1, cellSize, vNames); //added +1
    for (my = 0; my < 19; my++) {
                drawYGridCell(my+1, canvas_2, context_2, cellSize, vNames); //added +1
        color = gridColor.e(mx+1,my+1);
        switch(color)
        {
                case 1:
                        drawYellowCell(mx, my, canvas, context, cellSize);
                        break;
                case 2:
                        drawRedCell(mx, my, canvas, context, cellSize);
                        //alert ("Drawing red cell in the box swap function");
                        break;
                case 3:
                        drawGreenCell(mx, my, canvas, context, cellSize);
                        break;
```

```
                case 5:
                        drawLightBlueCell(mx, my, canvas, context, cellSize);
                        break;
        }

        drawText(mx, my, canvas, context, cellSize, gridText);
    }
        }

        //############## draw code end ############

        document.getElementById("moveData").style.display="none";
}


//#########################################################################################################
//#########################################################################################################
//
//   Now build a function to swap the selected row and column pairs
//   That are entered for the first time in the grid.. (modify swapRC()
//   based on the approach in moveRC()
//
//#########################################################################################################
//#########################################################################################################
function swapRC() {
        var cellSize = 20; // should be in global function variable
        var rcOne = document.getElementById("one").value;
        var rcTwo = document.getElementById("two").value;
        var canvas = document.getElementById("BSMP_1");
        var context = canvas.getContext("2d");
        var canvas_1 = document.getElementById("BSMP_2");
        var context_1 = canvas_1.getContext("2d");
        var canvas_2 = document.getElementById("BSMP_3");
        var context_2 = canvas_2.getContext("2d");


        var tempRCOneIndex;
        var tempRCTwoIndex;
        var tempGridColorSwap;
        var tempGridTextSwap;
```

```
var tempRowOneColorSwap;
var tempRowTwoColorSwap;
var tempRowOneTextSwap;
var tempRowTwoTextSwap;
var tempColOneColorSwap;
var tempColTwoColorSwap;
var tempColOneTextSwap;
var tempColTwoTextSwap;
var si;
var sx;
var sy;

tempRCOneIndex = vNames.indexOf(rcOne);
tempRCTwoIndex = vNames.indexOf(rcTwo);

tempGridColorSwap = gridColor.dup();
tempGridTextSwap = gridText.dup();

vNames.setE(tempRCTwoIndex, rcOne);
vNames.setE(tempRCOneIndex, rcTwo);

tempRowOneColorSwap = gridColor.row(tempRCOneIndex);
tempRowTwoColorSwap = gridColor.row(tempRCTwoIndex);
tempRowOneTextSwap = gridText.row(tempRCOneIndex);
tempRowTwoTextSwap = gridText.row(tempRCTwoIndex);

tempGridColorSwap.setRow(tempRCTwoIndex, tempRowOneColorSwap);
tempGridColorSwap.setRow(tempRCOneIndex, tempRowTwoColorSwap);
tempGridTextSwap.setRow(tempRCTwoIndex, tempRowOneTextSwap);
tempGridTextSwap.setRow(tempRCOneIndex, tempRowTwoTextSwap);

tempColOneColorSwap = tempGridColorSwap.col(tempRCOneIndex);
tempColTwoColorSwap = tempGridColorSwap.col(tempRCTwoIndex);
tempColOneTextSwap  = tempGridTextSwap.col(tempRCOneIndex);
tempColTwoTextSwap  = tempGridTextSwap.col(tempRCTwoIndex);

tempGridColorSwap.setCol(tempRCTwoIndex, tempColOneColorSwap);
tempGridColorSwap.setCol(tempRCOneIndex, tempColTwoColorSwap);
tempGridTextSwap.setCol(tempRCTwoIndex, tempColOneTextSwap);
tempGridTextSwap.setCol(tempRCOneIndex, tempColTwoTextSwap);
```

```
        gridColor = tempGridColorSwap;
        gridText = tempGridTextSwap;

        //############### draw code here ########
// need to redraw the main canvas using text values
        for (sx = 0; sx < 19 ; sx++) {
           var color;
                drawXGridCell(sx+1, canvas_1, context_1, cellSize, vNames); // added +1
    for (sy = 0; sy < 19; sy++) {
                drawYGridCell(sy+1, canvas_2, context_2, cellSize, vNames); // added +1
        color = gridColor.e(sx+1,sy+1);
        switch(color)
        {
                case 1:
                        drawYellowCell(sx, sy, canvas, context, cellSize);
                        break;
                case 2:
                        drawRedCell(sx, sy, canvas, context, cellSize);
                        //alert ("Drawing red cell in the box swap function");
                        break;
                case 3:
                        drawGreenCell(sx, sy, canvas, context, cellSize);
                        break;
                case 5:
                        drawLightBlueCell(sx, sy, canvas, context, cellSize);
                        break;

        }

        drawText(sx, sy, canvas, context, cellSize, gridText);
    }
        }



        //############# draw code end ###########

        document.getElementById("one").value = "N";
        document.getElementById("two").value = "N";
```

```
        document.getElementById("northTrue").style.display="none";
  document.getElementById("northFalse").style.display="none";
  document.getElementById("entryButton").style.display="inline";
  document.getElementById("processButton").style.display="inline";
  document.getElementById("inferenceButton").style.display="inline";
}

//#############################################################################################################
//#############################################################################################################
//
//   A function to detect the presence of a one (1) in the upper triangular area
//   -- If there are no ones (1) in the upper triangular area,
//   -------- then the inference button is displayed for use
//   -- If there are ones (1) in the upper triangular area ,
//   -------- then the process button is displayed.
//
//   Adds tighter process logic to the GUI interface
//
//#############################################################################################################
//#############################################################################################################

// scan_upper_for_ones returns true if a 1 is found, false otherwise
// all the matrix diagonal entries are zero (0), so the matrix may be correctly
// scanned by starting with entry 0,0 and adding all the values in the first row (0).
// Then proceed with scanning row two (2) at 1,1 and add all elements in column 1 to 19.
// Then scan row three (3) starting at 2,2 and add all elements in  columns 2 to 19.
// Next scan row four (4) starting at 3,3 and add all elements in columns 3 to 19.
// Continue scanning each row and adding the elements in the selected columns.
// If the sum of the adding operation is greater than zero (0) return false.
// If the sum of the adding operation is equal to zero (0) return true

function scan_upper_for_ones() {
   var num = 19;
   var sum = 0;
   var value = 0;
   for (var i = 0; i < num; i++) {
        for(var j = 0; j <= i; j++) {
                value = gridText.elements[i,j];
                sum = parseInt(sum) + parseInt(value);
        }
```

```
    }

    return sum;

}
```

Abstract Relation Type – Notes Section

The notes section provides an area to record relevant information and ideas that are not part of the Art form.

Appendix A – Detailed Design Issues

Clearly define ART types.

Need to define a common naming convention.

Appendix B - NOTES

NOTE 1-- November 9th, 2013, started to create this document.

NOTE 2 – February 21st, 2014, removed current Sage math code and added new Javascript code.

Appendix C – Abstract Relation Type Form and Format

The ART form has a dedicated component for each of the three primary language types: prose, graphics and mathematics. The content and value of language is based on robust methods of definition and communication. Therefore, the ART form supports not only the three basic language types, but also four common modes of definition. The modes of definition are: 1) naming, 2) extension, 3) intension, and 4) relationship. The ART form is designed to fully support all four modes of definition.

A formal naming convention, numbering system, and version-control system are part of the operational context necessary to create, track and use unique ART forms in an unambiguous manner. These contextual rule sets are currently being designed. Each ART form has a prose section, a graphics section, a mathematics section, and a notes section. The content of each section is detailed next.

Abstract Relation Type – Prose Section

The prose section of the ART form is divided into two subsections: 1) informal prose and 2) formal prose. The informal prose section presents information in a human readable form that outlines the ART form purpose, application, typical context of application, related ART forms (typically used with), special cases and frequently asked questions with answers. The formal prose section contains two types of formal prose: 1) formal system patterns and 2) structured executable prose (XML, JASON or other executable formats).

Prose Section – Informal Prose

Each ART form must have an informal prose section.

Informal name of the ART form.

Narrative describing the natural-language relationship, purpose and typical application areas.

Narrative description of relationship attributes, properties and application context.

Description of similar ART forms and typical groups of application ART forms that are used together.

Narrative of special cases and issues of note.

Frequently asked questions.

Prose Section – Formal Prose

Each ART form must have a formal prose section.

Formal name of the ART form.

Formal version control history

Formal system pattern

- Evocative Name – formal name
- Problem statement that presents difficulty, uncertainty in the situation

- Forces or tensions that influence pattern application
- Context within which the pattern will be applied
- Related patterns that are connected in some way to the current pattern
- Solution that resolves the problem within the given context.

Formal concept analysis formal context and concept lattices

JASON system and/or relationship representation

Abstract Relation Type – Graphics Section

The ART graphics section is divided into two parts: informal and formal. Similar to the prose section, the informal graphics section may contain redundant and ambiguous graphics in an effort to create a, 'information-rich' communication channel. The formal graphics section contains formal graphics that follow a well-defined specification and/or are executable in an open source graphics tool.

Graphics Section – Informal Graphics

Each ART form must have an informal graphics section.

Informal graphics associated with the information in the prose section and/or created to communicate the system, context, operations and values of interest are located in this section of the document. These graphics should focus on the binary matrix form, object configuration or other substantive features of the current problem structure.

Graphics Section – Formal Graphics

Each ART form must have a formal graphics section.

Formal graphics associated with information in the prose and the mathematics sections are located in this section of the ART form. Each ART form must have a binary matrix or a primary structuring graphics display form that is directly associated with the system relation of interest. Other executable graphic types may be located here as long as the executable code is available in the mathematics section.

Abstract Relation Type – Mathematics Section

The mathematics section is divided into two parts: equations and executable code. The equations section presents the current equations of interest and the executable code section presents the computer code to implement the equations.

A general set of equations may be developed and added as an appendix to the applicable ART form documents. The executable code that supports the mathematical operations are presented in the SAGE math format. Other types of executable code may be provided but the SAGE math code is required for a complete ART form.

Abstract Relation Type – Notes Section

The notes section provides an area to record relevant information and ideas that are not part of the Art form.