**Technical Document**

**Table of Contents**

**Section 1: Initial Learning Map 1**

**Section 2: The Student Data 2**

**Section 3: Merge Operation 3**

**Section 4: Modeling Technique: How we create a Bayesian Network for the Learning Progression 8**

**Section 5: Random Graph Generation 11**

**Section 6: Evaluation Procedure 16**

**Section 7: Experiment 1 18**

**Section 8: Experiment 2 19**

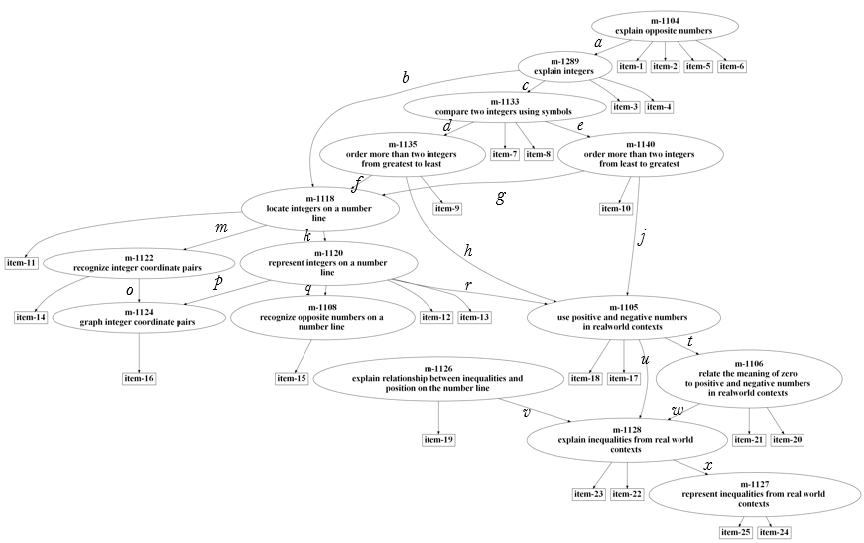
**Section 9: Experiment 3 24**

**Appendix 30**

## Section 1: Initial Learning Map

We started with a prerequisite skill graph that was received from a team led by Neal Kingston, director of the Dynamic Learning Map Consortium. The DLM Consortium is building a computer adaptive system to try to represent 3,000 skills for English and Math for Kindergarten to 12th grade. Their team of math education specialists has created a mathematics learning progression. They shared with us a small subsample of it that has 15 skills (Figure 1.1). The graph contains 15 skills and 25 items (i.e., questions). Each skill is labeled with an identifier like M-1104. All of the questions were multiple choice questions with four possible solutions. Each skill has at least one item/question attached to it. The items attached to the skill are items that are expected to represent the skill which they are attached to. When a student answers those questions correctly, we are assuming that the student has mastered the skill. It is expected that the skills and items lower in the skill graph hierarchy are more difficult than their predecessors higher up in the hierarchy.

**Figure 1.1: Initial Learning Map**



*Figure 1.1: The initial learning map that scientists at UK created. Each circle represents a “skill” and each rectangle represents a question.*

### Section 2: The Student Data

This study examined a section of the learning map containing 15 concepts and skills related to understanding integers. The map was developed using mathematics educational literature describing how students learn to understand and operate with integers. The set of integers includes the whole numbers and their opposites, presenting many students their first exposure to negative numbers. Although many students have prior knowledge of negative values within contexts such as debt or temperatures below freezing, they often struggle when first learning to work with negative numbers. Proficiency with integers includes understanding opposite numbers, comparing integers, representing integers on number lines and graphs, and using integers in real world problem contexts. The learning map shown in Figure 1.1 illustrates the component concepts and skills that comprise such understanding. This map suggests that students should learn to identify opposite numbers (M-1104) and integers (M-1289) in preparation for comparing and ordering integers (M-1133, M-1135, M-1140) as well as representing integers on number lines (M-1118, M-1120, M-1108, M-1126) and coordinate planes (M-1122, M-1124). Because integers challenge the initial counting strategies students learned for positive numbers, it is beneficial for students to work with integers in real-world contexts (M-1106, M-1105, M-1127, M-1128).

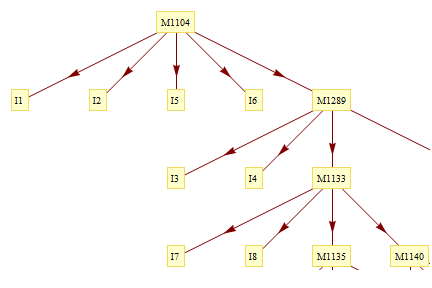
The data for this study was gathered from student responses to 25 test items aligned to the 15 skills shown in the learning map in Figure 1.1. All of the test items were multiple choice questions, with four answer options per question. Each skill was assessed by at least one test item, in many cases two items. As part of the test development process, subject matter experts confirmed the alignment of each item to its associated skill, meaning that the item was judged by experts to evoke the intended skill. Therefore, when a student answered a test item correctly, we assumed in this study that the student had mastered the skill associated with that test item. Furthermore, due to the hierarchical structure of the learning map, items associated with skills lower in the learning map were assumed to be more difficult, i.e., require more skills, than items associated with skills higher in the learning map.

In addition to the graph, we utilized a data-set containing the responses of 2,846 students answering the same sequence of 25 items in the learning map. All the students were chosen from middle schools in a Midwestern state from grades 6 (8%), 7 (49%), 8 (39%) and 10 (4%). The students’ responses were dichotomous, ‘1’ for correct and ‘0’ otherwise.

### Section 3: Merge Operation

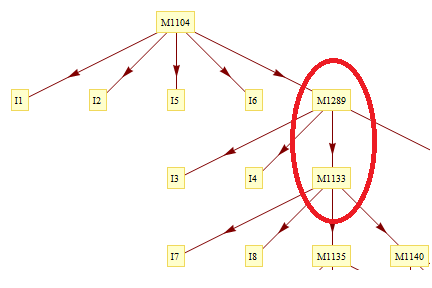
### In all of our experiments we will be merging or combining skills. This is what we will call the “merge” operation. A merge operation is when two skills adjacent to each other in the skill graph are combined into one skill. Items in both skills that are merged are attached to only the one merged skill. The prerequisites of the constituent skills become prerequisites of the merged skill and the same applies to the post-requisites. An example merge operation is shown below on a section of the skill graph. How the skill graph looks before, during, and after the merge operation are shown in figures 2, 3, and 4 respectively. M--1289 and M-1133 are the skills being merged into a single skill, which is called “M1289XM1133”. Note that the names of the skill have no meaning in of itself.

**Figure 3.1: Before Merging M-1289 and M-1133**



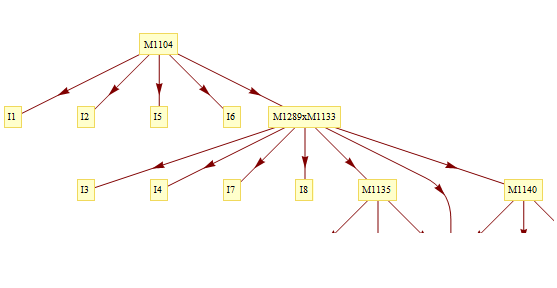
*Figure 3.1: Before the merge. What part of the learning progression looked like before the merge.*

## Figure 3.2: During the Merge of M-1289 and M-1133



*Figure 3.2: During the merge. Showing the two skills that are going to get merged together. Note that I3 & I4 and I7 and I8 will all be grouped together.*

**Figure 3.3: After Merging M-1289 and M-1133**



*Figure 3.3: After the merge. Note that M-1289 and M-1133 now have all the items of each of them mapped to the same merged skill.*

The code that does the merging functions at the item level and not the skill level. The result is conceptually the same. Below is the same merge operation shown in figure 3.3, but shown as how the merge is done programmatically instead of conceptually. In the merge example skills M-1289 and M-1133 are being merged. Cells highlighted in blue show all items in skills that are parent skills of the items to be merged. In this example skill m-1104 is a parent skill of M-1289 and contains the items 1, 2, 5, and 6. Therefore those rows that contain elements that have a ‘1’ are highlighted in blue. This represents that those items are parent items of items 3 and 4. Skill M-1289 is also a parent as well as one of the skills being merged. Since skill M-1289 is a parent skill of skill M-1133, those cells are also highlighted in blue. Skills M-1135, M-1140, and M-1118 are child skills of one of the skills that are going to be merged (M-1289 and M-1133). Items in these skills are items 9, 10, and 11. Cells highlighted in red show the rows of the merged skills that have the items of the child skill.

**Table 3.1: Merge Example: Item x Item Matrix before Merging**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **2** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **3** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **1** |
| **4** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **1** |
| **5** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **6** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** |
| **8** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** |
| **9** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **10** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **11** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

In order to merge skills M-1289 and M-1133, the items in both of those skills must have the same parent items and child items. Every item that is in a parent skill of skill M-1289 must be a parent item of every item in skill M-1133. The same is true for items that are in a parent skill of M-1133. The matrix shown below shows the updated parent cells in green. Since M-1104 is now a parent of the merged skill, items 1, 2, 5, and 6 are parents of items 7 and 8 now. In addition since skill M-1289 is merged into the same skill as M-1133, the items in skill M-1289 are no longer parents of the items in skill M-1133.

**Table 3.2: Merge Example: Item x Item Matrix Parents merged**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **2** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **3** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **4** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **5** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **6** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** |
| **8** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** |
| **9** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **10** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **11** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

The next step in the merge is to merge the children of the merged skills. This is similar to merging the parents of the merged skills. All items in children of M-1289 and M-1133 must be children of items of all items in both M-1289 and M-1133. The pink cells in the table show the changed after merging the children. Items 3, 4, 7, and 8 now all have the same rows.

**Table 3.3: Merge Example: Item x Item Matrix Children merged**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **2** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **3** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **4** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **5** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **6** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **8** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **9** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **10** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **11** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

The final matrix shown below shows that all items merged skill M1289XM1133 have the same row entries. All items in the merged skill have the same children, which was a combination of the children from each skill. All items in the merged skill also have the same parents, which was a combination of the parent skills from each skill. Parent items are highlighted in blue, and child items are highlighted in red.

**Table 3.4: Merge Example: Item x Item Matrix after Merge**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **2** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **3** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **4** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **5** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **6** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **8** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **9** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **10** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **11** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**Section 4: Modeling Technique: How we create a Bayesian Network for the Learning Progression**

Given Figure 1.1, there are many ways you could attempt to fit a Bayesian network to such a model. In this section we lay out a few of the issues and the choices we made. To apply a Bayesian modeling approach, we treat each of the skill nodes as latent Boolean variables. Each question is an observable node because for each student we know which questions they got correct and incorrect. For skills without parent skills such as M-1104 the Bayesian network will learn the best fitting estimate of that skills knowledge across all the student data. It will learn this and all other parameter using Expectation Maximization, which is the standard way of fitting Bayesian networks parameters. We use the Kevin Murphy Bayes Net Toolkit in Matlab to fit all models. To fit skill nodes that have parents, we model the conditional probability tables as latent variables that EM will attempt to fit. For example, to model skill M-1118, we need to learn 4 values for the conditional probabilities. The first row in the table below shows the probability that the student know skills M-1118 given they do not know either of the parent skills (M-1135 and M-1140). The value of 0.2 is listed here, but is in fact something that EM will learn. The second row represents the probability that the student knows M-1118 given they do not know skill M-1135 but do know skill M-1140. The values in the third column are values that EM will learn.

**Table 4.1: Conditional Probability Table for Skill M1118**

|  |  |  |
| --- | --- | --- |
| **M1135** | **M1140** | **Prob(M1118)** |
| **0** | **0** | **.2** |
| **0** | **1** | **.4** |
| **1** | **0** | **.5** |
| **1** | **1** | **.8** |

*Table 4.1: The conditional probability that models the probability of knowing the probability of M1118 given the probability of knowing M1135 and M1140.*

Although we do not know the real values in the third column, we are assuming that the probability of the last row should be higher than the probability of any of the other rows; otherwise we are modeling something very unusual. In this unusual case we would be assuming you will do better if you do not know the prerequisite skills. If this is true, it will probably be the case that if we deleted the arcs between skill M-1118 and its parents, we would get a better fitting model. In the paper we do not consider deleting arcs and only look at merges, but there is no reason this cannot be considered.

In modeling it is often handy to think about the number of parameters in your model. Therefore we examined the parameters for the model created for Figure 1.1. Given that there are 15 circles/skills, we have to separate them into cases in order to count the number of parameters for each skill. Skills that have different numbers of parents will require a different number of parameters. As a general rule, skills that have more parents will require more parameters. Skills with only one parent will only require only a single skill parameter. Skills that have only one parent skill will have two parameters. One parameter represents the probability that a student knows the post-requisite skill given that the student knows the prerequisite skill. The second parameter represents the probability that a student knows the post-requisite skill given that the student does not know the prerequisite skill. Table 4.1 showed that a skill that has two parents requires four parameters. If a skill has 3 parents it would require 8 parameters. Figure 1.1 has 2 skills without parents (M-1104 and M-1126), 9 skills with 1 parent, 1 skill with 2 parents (M-1124) and 3 skills with 3 parents (M-1118, M-1105, M-1128). This results in 2\*1 + 9\*2 + 1\*4 + 3\*8 = 48 parameters. Additionally we modeled each of the 25 questions as a Boolean variable, where we learned a probability for each question based on the student correctness for each question.

**Table 4.2: Item Probability Given the Skill**

|  |  |  |
| --- | --- | --- |
| **M-1104** | **Probability the Student Gets I1 Correct** | **Comment** |
| 0 | .25 | The probability the student will get question I1 correct given they don’t know the skills (aka the chance they can guess the question.) |
| 1 | .7 | The probability the student will get question I1 correct given they know skill M1104 |

*Table 4.2: The conditional probability table for questions that models the probability of getting question one(i.e., “I1”) correct given the student knows skills M-1104 . The values in the second column are learned by EM.*

From the model of Figure 1.1, we learned 48 parameters in the relationships between skills and 25\*2=50 parameters in predicting performance of the questions based on estimates of the knowledge of the relevant skill. There are other alternative modeling techniques we could have considered that we have not yet. For instance, we could set all guess values to so some figure, say 0.25. This might be reasonable since all the questions were multiple choice questions with four choices per question. We also could have reduced another 24 parameters by assuming the probability you get an item correct given you know its parent skills is fixed across all 25 questions. If the different questions difficulty is only dependent upon its parent skill that might make sense. In this paper we will consider “merges” that make the number of skill less, so the 48 parameters will drop as the number of skills drop, but the number of question parameters will not change. One extreme model would be to merge all the skills together so there would be only one skill, which would result in a single skill parameter instead of 48.

**Section 5: Random Graph Generation**

The code used to randomly generate possible graphs from the original skill graph is in LearnGraph.java in the appendix. The algorithm starts by taking two CSV files as input. One CSV file contains an item x item matrix for all the items and the other CSV file contains a single row that maps all the items to a skill. The matrix below is the item x item matrix for all the 25 items in the original skill graph.

**Table 5.1: Item x Item Matrix of Original Skill Graph**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** |
| **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **2** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **3** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **4** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **5** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **6** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **8** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **9** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **10** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **11** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **12** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **13** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **14** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **15** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **16** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **17** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **0** |
| **18** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **0** |
| **19** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** |
| **20** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** |
| **21** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **0** |
| **22** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** |
| **23** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** |
| **24** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **25** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

This matrix represents the skills of the items at the item level. Whenever there is an arrow pointing from one skill to another, the matrix will contain a 1 for all items in that skill for all the columns of the items in the other skill. For example the first skill in the original skill graph (M-1104) contains item 1, item 2, item 5, and item 6. Therefore in the item x item matrix, numbers 1, 2, 5, and 6 have the exact same rows. Since skill M-1104 has a link to skill M-1289 and skill M-1289 contains items 3 and 4, the matrix entries for rows 1, 2, 5, and 6 all have 1’s for columns 3 and 4.

In addition to the item x item matrix, a simple table is also requires to map each item to a skill and the name of the skill. The second CSV file for the original skill graph is shown transposed below. It is shown transposed so that it will display better. Each item is assigned an integer value representing what skill it belongs to. The integer value used is the same value as the first item in the group starting at 0. So since the first skill (M-1104) contains the first item, all items in that skill will be assigned to skill 0. Another example is skill M-1120, where the first item in that skill is item 12. Therefore all the items for that skill will be assigned a value of 11.

**Table 5.2: Item to Skill Mapping**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Skill Name** | **Skill Number** |  | **Skill Name** | **Skill Number** |  | **Skill Name** | **Skill Number** |
| M1104 | 0 |  | M1135 | 8 |  | M1105 | 16 |
| M1104 | 0 |  | M1140 | 9 |  | M1105 | 16 |
| M1289 | 2 |  | M1118 | 10 |  | M1126 | 18 |
| M1289 | 2 |  | M1120 | 11 |  | M1106 | 19 |
| M1104 | 0 |  | M1120 | 11 |  | M1106 | 19 |
| M1104 | 0 |  | M1122 | 13 |  | M1128 | 21 |
| M1133 | 6 |  | M1108 | 14 |  | M1128 | 21 |
| M1133 | 6 |  | M1124 | 15 |  | M1127 | 23 |
|  |  |  |  |  |  | M1127 | 23 |

To generate random skill graphs from possible merges, the following steps are run for several iterations. We used 10 million iterations; however the number of iterations does not need to be exact. The number just needs to be large enough to generate all the distinct random graphs or enough of them. Since the algorithm generates graphs randomly, several graphs generated will not be distinct graphs; they will have been previously generated and saved. Those graphs will not be saved again; therefore the algorithm will generate less distinct graphs the longer it runs until it randomly finds all possible graphs. This means the number of distinct random graphs found will be much less than the number of iterations.

1. Make a copy of the original graph to work with

2. Pick two adjacent skills randomly

3. Merge the items in the skills to a single skill

4. Save the graph if it is acyclic and has not already been saved

5. Generate output code for Matlab and Mathematica

The data structures used in our program are shown at the top of the LearnGraph.java file. We briefly describe them here for convenience. The variable “Vector <int[]> groupVector” is a vector object that contains all the item to skill mappings for each random graph generated. This basically corresponds to the table shown above for mapping items to skills. The variable “Vector <int[][]> structureVector” is a vector object that contains all the item x item matrices for all the random skill graphs generated. Both the groupVector and the structureVector are in sync with each other. This means that the elements in both vectors will be holding the respective information for the same graph. Element 1 in both vectors will correspond to the first graph and element 2 in both vectors will correspond to the second graph.

The other data structre the program uses are arrays holding the original information on the skill graph, and information on the current skill graph. “private int structure[][];” holds the item x item matrix of the original skill graph described above and “groupArray[]” holds the item to skill mapping table shown above. Likewise structureCopy and groupArrayCopy hold working copies of these matrices that will change as the random graph is generated.

Each iteration of the algorithm to generate a random graph starts by restoring the original skill graph to the current variables structureCopy, and groupArrayCopy. This working copy is then used to be transformed into a random graph. The “runCrushing()” function is what transforms the graph into a new random graph by randomly applying merges.

Two factors are considered when deciding when to stop merging . The first factor is if there are no more possible merges. This situation occurs when all the skills have been merged into a single skill and cannot be merged anymore. The merge process must stop at this point and return the single skill graph. Another more interesting factor is how to randomly choose when to stop merging skills. In the code this is represented as the “stop” variable inside the “runCrushing” function. This variable is a random number between zero and the number of items - the number of merges done. When this variable equals zero, the merge process will stop and the random graph will be returned. as the number of merges increases the random range decreases making it more likely for the merge process to stop. This will give the effect that the merge process will not likely be stopped too early and will likely end before there are no more merges left. The result allows all possible graphs of varying number of skills merged to be generated.

For each merge in the random merge process, the first skill to merge is chosen somewhat randomly. Technically the skills with more items have a large chance to be merged since the items are what are really being chosen and the skills are being chosen by what skill the item maps to, however running the entire process 10 million times should make things more fair because duplicate graphs are not stored. The second skill to merge is also chosen randomly. A random skill is selected and if that skill is different from the first skill and adjacent to the first skill, then those two skills will be merged. If the two skills are the same or non-adjacent then a new skill will be randomly chosen until it can be merged.

Once two skills are chosen to be merged the list of items is iterated over. If the item belongs to one of the skills the “crush()” function is called. The crush function is only called once, but is done in a way so that the merged skill will keep the smaller skill number. Inside the function each item in one of the two skills is assigned the number of the lower numbered skill. Once this is complete all the items in the merged skill are added to a vector called groupNumbers. To complete the merge every item in this vector should have the same item x item matrix. In other words, when two skills are merged, all the parents of one skill should be parents of the other skill and all the children of one skill should be children of the other skill. This is done at the item level. So all items of a parent skill (of the merged skill) will have their matrices changed to have links to all the items in the merged skill. Basically this simplifies to having all items in the merged skill having the same columns in the item x item matrix. This process is described in the merge operation section both conceptually and programmatically.

Once the merge process is complete the structureArray may have the identity matrix containing ones. This is a byproduct of the algorithm and is fixed by simply setting the identity to 0, since no item can have a link to itself. After that the result is a new random skill graph. This graph is then compared to all the other previously generated random graphs. If the new graph is different from all the other graphs then it is saved and added to the groupVector and structrureVector vector objects. When the maximum number of iterations is complete each matrix in these vectors is written to a file to be used as input graphs to create and evaluate a bayesian network in matlab. We also output files that generated mathematica code to generate each of the graphs in a viewable way.

For several of our experiments we had to generate all possible merges (a single iteration) for a given graph. The code for random graph generation described above was modified to accomplish this. The only change we had to make was to stop the random graphs after a single iteration. This will give us graphs that only used a single merge operation for two random skills. Running that process enough times will result in randomly finding all the possible merges for a given graph. The original skill graph had 16 possible merges, therefore making the maximum number of iterations really big basically guaranteed that the program would randomly find all possible merges for the current skill graph.

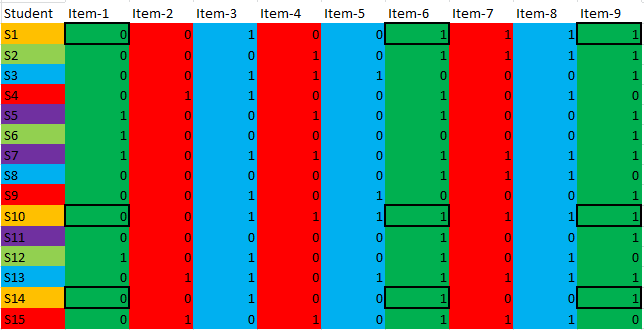
To make things even easier, instead of generating new input files for each graph, we just started with the original skill graph input files. We fixed the number of merges to be exactly one more than the current iteration and fixed the previous merges to be the merges that actually took place. This allowed us to start from a graph generated from the original skill graph and apply one more merge to it. Running the process several times results in finding all possible merges for any given iteration of our iterative search experiment.

**Section 6: Evaluation Procedure**

For evaluating the models we used per student per item cross validation with 5 student folds and 3 item folds. Our student and item folds were chosen randomly for our evaluation. For the student folds, each student was randomly assigned a value from 1 to 5 to indicate which fold the student belongs to. For the item folds, each item was randomly assigned a value from 1 to 3 to indicate which fold the item belongs to. Doing the fold assignment this way means that each student will have the same random item folds. The item folds were assigned randomly but kept consistent for each student.

An example of how our cross-validation works is shown below in the color-coded table. The table is a Q-matrix of student data. A Q-matrix is a table where each column represents a question or item and each row represents a student. Each cell contains a 1 or a 0 to indicate if the student got the item correct or not. The item folds in the Q-matrix are colored with background colors of green, red, and blue to indicate which item fold the data belongs to. Similarly the student numbers are also colored to indicate which student fold a student belongs to.

**Table 6.1: Cross-validation Example**



*Table 6.1: Cross Validation Example. This table shows the student and item folds color coded and shows what a single student + item training fold will look like with a black border around the cells containing the training data*

An example test fold is shown in the table where a black border around the cells indicates data used in the example test fold. The student rows in the example test fold are the rows highlighted in orange. All other student rows are used as training data for this example fold. In addition to a student row, an item column is also used in the test fold. The items with the green background are the items used in the example test fold. When the student and item test folds are combined, the cells where they intersect are given a black border. These cells are what one test fold consists of. All the non-bordered cells are used for training in that fold.

The cross-fold validation works by looping through each student fold and then each item fold given the current student fold. The test set predictions are then saved to be evaluated later.

## Section 7: Experiment 1

**7.1: Purpose:**

The purpose of experiment 1 was to explore the search space (the possible graphs that could exist) by using the merge operation described in section 3. Our goal was to see if any trends existed among the best and worst graphs. We also wanted to use this opportunity to make sure our best models were significantly different than our worst models.

**7.2: Method:**

In this experiment we randomly generated over 26,000 possible graphs by using our merge operation on random skills for a random number of times starting with the original skill graph. Out of the 26,000 possible graphs we chose to evaluate 3,000 of them because that is as many as we could evaluate in a reasonable amount of time (2 weeks). Our evaluation procedure slightly differed from the evaluation procedure described in section 6. Instead of training on 80% of the data we trained on 20% of the data. This can be shown by taking Table 6.1 and inverting the selected cells. This means all cells that have a border around them would be the training set instead of the test set. The reason why we chose to only use 20% of the data to train on is because we typically do not have much data on students. We wanted to see if we can make good predictions for new students where we do not have much data on.

**7.3: Results:**

Table 7.1 shows a set of our “best” and “worst” models chosen by RMSE. We obtained our significance value by first taking the predicted values from our model and calculating the absolute difference between the actual and predicted values for each predicted value. This gives us the absolute error for each prediction. After calculating the error for both the “best” and “worst” models we used a two-tailed paired T-test to see if the “best” and”worst” models were significantly different from each other. Our paired T-test showed that there were no significant differences between these models; therefore we did not do anymore analysis on this table. Not having a significant difference between our best and worst models made us rethink our evaluation procedure and lead to experiment 2, which trained on 80% of the data and tested on 20% of the data instead of training on 20% and testing on 80%.

**Table 7.1: Results of Best and Worst Models from random search**

**Top Models Worst Models**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model ID** | **# of Skills** | **AUC** | **RMSE** | **Accuracy** |  | **Model ID** | **# of Skills** | **AUC** | **RMSE** | **Accuracy** |
| **1997** | **10** | **0.8045** | **0.3807** | **0.7912** |  | **18** | **11** | **0.7762** | **0.3907** | **0.7829** |
| **1788** | **8** | **0.8042** | **0.3807** | **0.7920** |  | **1052** | **7** | **0.7755** | **0.3907** | **0.7834** |
| **2904** | **10** | **0.8039** | **0.3808** | **0.7917** |  | **1330** | **6** | **0.7763** | **0.3904** | **0.7843** |
| **731** | **6** | **0.8042** | **0.3808** | **0.7912** |  | **1366** | **11** | **0.7792** | **0.3903** | **0.7824** |
| **2837** | **6** | **0.8042** | **0.3809** | **0.7913** |  | **2749** | **9** | **0.7777** | **0.3900** | **0.7829** |
| **1584** | **7** | **0.8042** | **0.3809** | **0.7910** |  | **1929** | **8** | **0.7794** | **0.3900** | **0.7830** |
| **1274** | **10** | **0.8040** | **0.3809** | **0.7916** |  | **632** | **7** | **0.7786** | **0.39** | **0.7839** |
| **646** | **6** | **0.8037** | **0.3809** | **0.7914** |  | **1527** | **8** | **0.7784** | **0.3899** | **0.7835** |
| **39** | **9** | **0.8041** | **0.3809** | **0.7916** |  | **273** | **7** | **0.7788** | **0.3898** | **0.7834** |
| **1317** | **6** | **0.8042** | **0.3810** | **0.7908** |  | **1391** | **9** | **0.7792** | **0.3898** | **0.7827** |

**Section 8: Experiment 2**

**8.1: Purpose:**

The purpose of this experiment was to use more data to see if we could predict student response better and have reliable results between the best and worst models. This follows up from the previous experiment where the results between the best and worst models were not reliably different.

**8.2: Method:**

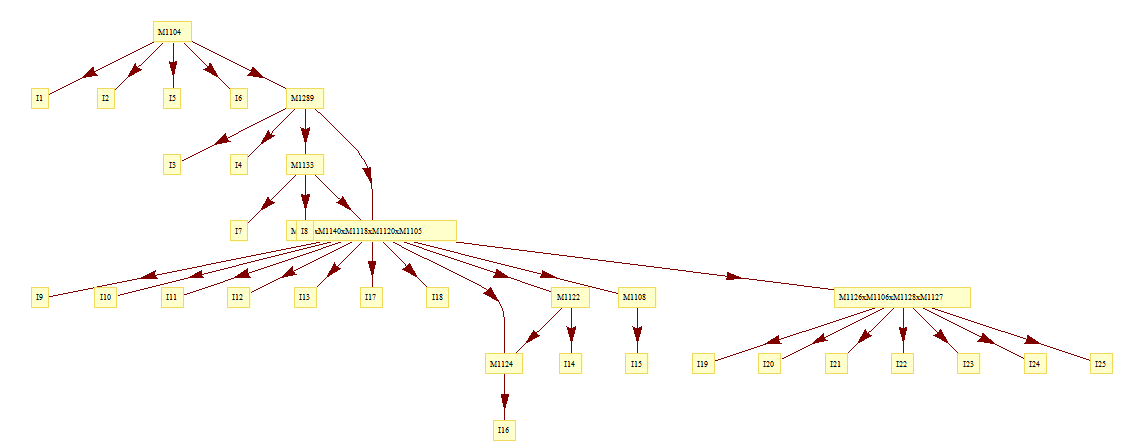
To be sure that our evaluation procedure was not the cause of the statistical unreliability of the differences between our best and worst models, we decided to use 80% of the data to train on and 20% to test on as shown in Table 2.3. We choose the 10 best and 10 worst models from the previous experiment. Additionally we randomly chose two models from the groups of *n*-skill models where *n* is between 2 and 15 skills. This resulted in about 40 different models in total.

## 8.3: Results

**Table 8.1: Experiment 2 Results**

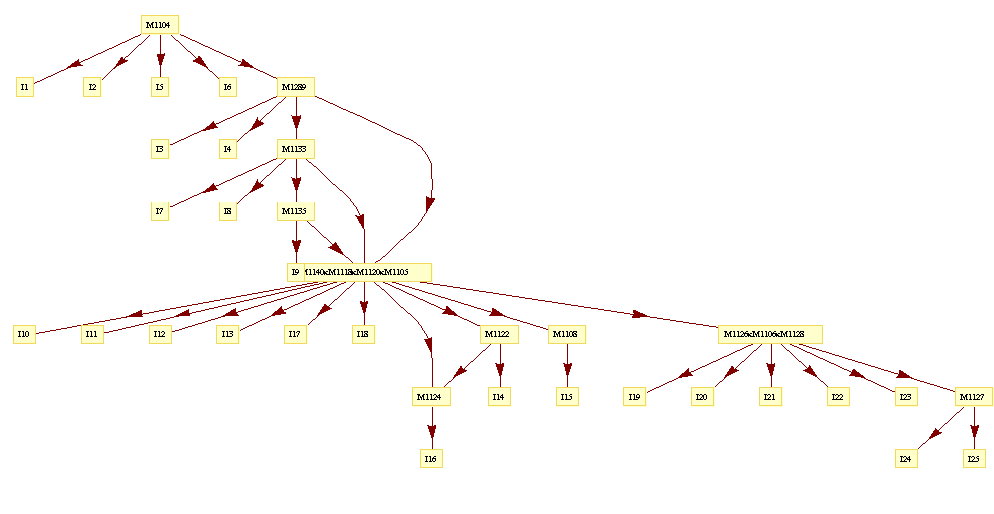
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model ID** | **# of Skills** | **AUC** | **RMSE** | **Accuracy** |  | **Model ID** | **# of Skills** | **AUC** | **RMSE** | **Accuracy** |
| 2508 | 13 | 0.8181 | 0.3732 | 0.8036 |  | 1330 | 6 | 0.8155 | 0.3772 | 0.7927 |
| 1170 | 6 | 0.8205 | 0.3727 | 0.8021 |  | 1366 | 11 | 0.8013 | 0.3810 | 0.7916 |
| 1232 | 14 | 0.8217 | 0.3722 | 0.8016 |  | 550 | 5 | 0.8060 | 0.38 | 0.7910 |
| 1595 | 14 | 0.8212 | 0.3725 | 0.8006 |  | 1311 | 5 | 0.8055 | 0.3802 | 0.7910 |
| 2837 | 6 | 0.8215 | 0.3731 | 0.7995 |  | 1929 | 8 | 0.8024 | 0.3810 | 0.7905 |
| 2288 | 12 | 0.8148 | 0.3751 | 0.7989 |  | 391 | 6 | 0.7996 | 0.38142 | 0.7904 |
| 1943 | 8 | 0.8118 | 0.3769 | 0.7980 |  | 2709 | 11 | 0.8081 | 0.3797 | 0.7904 |
| 1162 | 9 | 0.8220 | 0.3734 | 0.7979 |  | 2749 | 9 | 0.7968 | 0.3825 | 0.7904 |
| 2753 | 9 | 0.8183 | 0.3748 | 0.7975 |  | 1527 | 8 | 0.8056 | 0.3804 | 0.7903 |
| 39 | 9 | 0.8217 | 0.3738 | 0.7975 |  | 273 | 7 | 0.7981 | 0.3821 | 0.7903 |
| 2532 | 11 | 0.8072 | 0.3776 | 0.7974 |  | 761 | 2 | 0.8136 | 0.3782 | 0.7902 |
| 731 | 6 | 0.8189 | 0.3742 | 0.7973 |  | 1052 | 7 | 0.8066 | 0.3802 | 0.7894 |
| 215 | 7 | 0.8204 | 0.3741 | 0.7972 |  | 1589 | 3 | 0.805 | 0.3810 | 0.7891 |
| 1788 | 8 | 0.8214 | 0.3729 | 0.7967 |  | 18 | 11 | 0.8017 | 0.3820 | 0.7886 |
| 1997 | 10 | 0.8176 | 0.3751 | 0.7965 |  | 1342 | 6 | 0.8003 | 0.3825 | 0.7886 |
| 1274 | 10 | 0.8197 | 0.3750 | 0.7960 |  | 632 | 7 | 0.8090 | 0.3803 | 0.7874 |
| 637 | 7 | 0.8179 | 0.3754 | 0.7960 |  | 883 | 4 | 0.8075 | 0.3811 | 0.7874 |
| 1786 | 12 | 0.8182 | 0.3748 | 0.7960 |  | 1426 | 2 | 0.8078 | 0.3810 | 0.7873 |
| 1317 | 6 | 0.8205 | 0.3746 | 0.7957 |  | 445 | 10 | 0.7995 | 0.3826 | 0.7867 |
| 1584 | 7 | 0.8174 | 0.3758 | 0.7950 |  | 715 | 6 | 0.7945 | 0.3852 | 0.7852 |
| 1533 | 8 | 0.8120 | 0.3771 | 0.7944 |  | 1957 | 10 | 0.7974 | 0.3843 | 0.7848 |
| 1058 | 7 | 0.8178 | 0.3757 | 0.7941 |  | 2717 | 3 | 0.8018 | 0.3829 | 0.7822 |
| 1394 | 9 | 0.8078 | 0.3786 | 0.7935 |  |  |  |  |  |  |

The results from changing our training and test set size indicated significant differences in the models, with p-values below 2.0 x 10^-9. We analyzed the structures of our best models to see if there are any similarities in these models. We also chose to look at the worst model to compare to the best two models. What we observed is that the best models merged most of the skills in the lower level of the graph without merging a lot of the upper level skills.

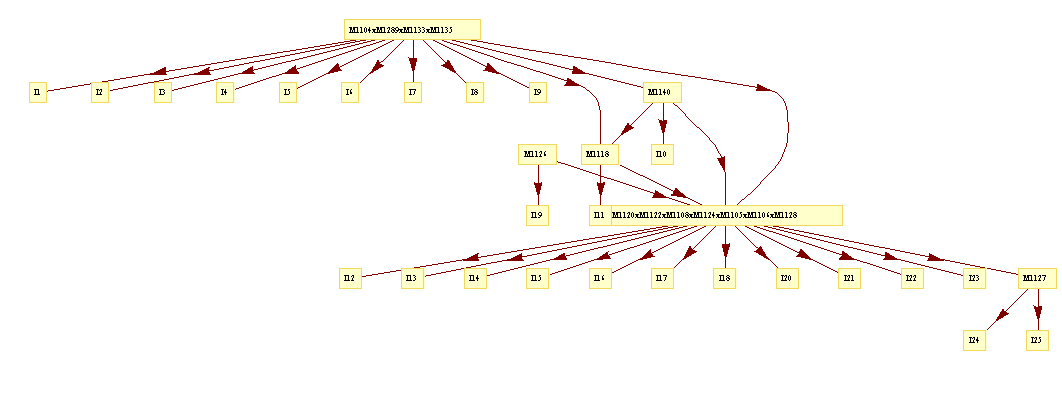
**Figure 8.1: Best model: ID = 1788**

*Figure 8.1: Best model - Experiment 2*

**Figure 8.2: Second best model: ID = 1274**



*Figure 8.2: Second best model - Experiment 2*

**Figure 8.3: Worst model: ID = 715**

*Figure 8.3: Worst Model - Experiment 2*

It can be noted from the worst model that the major difference between the best and worst models is the level of the hierarchy at which merging occurred. The worst models have a lot of skills merged at the top of the graph hierarchy. This suggests that the skills at the top of the hierarchy are really different skills and should not be merged. It may even be that some of these skills should be split. This is another research question for future work.

The results of the second experiment showed that there was a significant difference between our best and worst models. This reassured us that there are models that can perform better than others. It also leads us to believe that using more training data helps make a distinction between models. As a result of this we were able to make some observations on trends among the best models. We found that the best models merge skills at the bottom level of the graph and do not merge skills near the upper levels of the graph. The worst model merged several skills at the top level of the graph suggesting to us that these skills are different and should not be merged.

Whilst this experiment yields reliable differences in the models, the experiment still lacked generalizability in the way the models were selected. The models that were evaluated were chosen randomly from the 20,000 different models that were generated. There was still no systematic and repeatable method that was used to do the model selection and simplification. This experiment was merely for exploration and to make sure our methods were sound before we continued with our analysis.

**Section 9: Experiment 3**

**9.1: Purpose:**

The purpose of this experiment is to take the original skill graph generated by Neal Kingston’s team and to create and run a search algorithm to find a better skill graph that can model student performance. Our goals are to try to simplify the original skill graph and improve its accuracy at predicting student performance. We are also looking to find other useful properties that can be obtained from the search.

**9.2: Search Algorithm:**

In this experiment we start with the initial skill graph shown in figure 1.1. Using the Bayesian network described in section 4, we create a model for the skill graph. This graph is the starting point of our search algorithm. Starting with the original graph we programmatically find all possible skill pairs that can be merged. We consider merging adjacent skills as defined in our merge operation section 3. Each possible merge is evaluated with the evaluation procedure described in section 6 and the best merge is chosen. We apply the best possible merge to the skill graph to create a new skill graph, which will have one less skill. The new skill graph is used as the input to the next iteration of the algorithm. This technique is iteratively applied until all the skills are merged into a single skill. The basic steps are listed below.

**Iterative Search Steps**

Step 1: Start with an initial graph / model

Step 2: Find all possible skill pairs that can be merged

Step 3: Evaluate all models from the possible merges

Step 4: Choose the best graph and use this graph as the input to the next iteration

Step 5: Repeat steps 1-4 until there is only one graph with a single skill

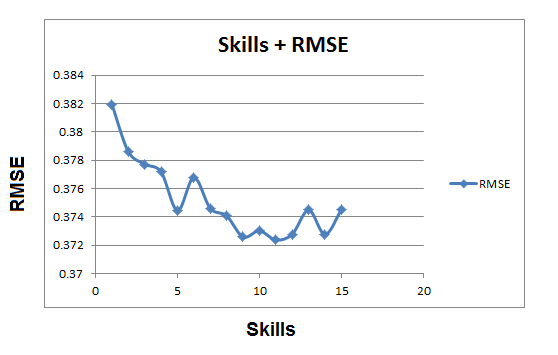
**9.3: Results and Analysis**

Below are a table and a graph of the results from the iterative search. The search starts at iteration 0, which is the initial skill graph consisting of 15 skills before any merges are applied to it. The search ends at iteration 14, which is a graph consisting of just one skill with all the items attached to that one skill. The best models from each iteration are shown below. We recorded AUC, RMSE, accuracy, AIC, and BIC metrics although we only use RMSE to choose the best models at each iteration and guide our search. We chose RMSE as the deciding metric because we believe it to be the most accurate metric, although all the metrics tend to agree on the best models.

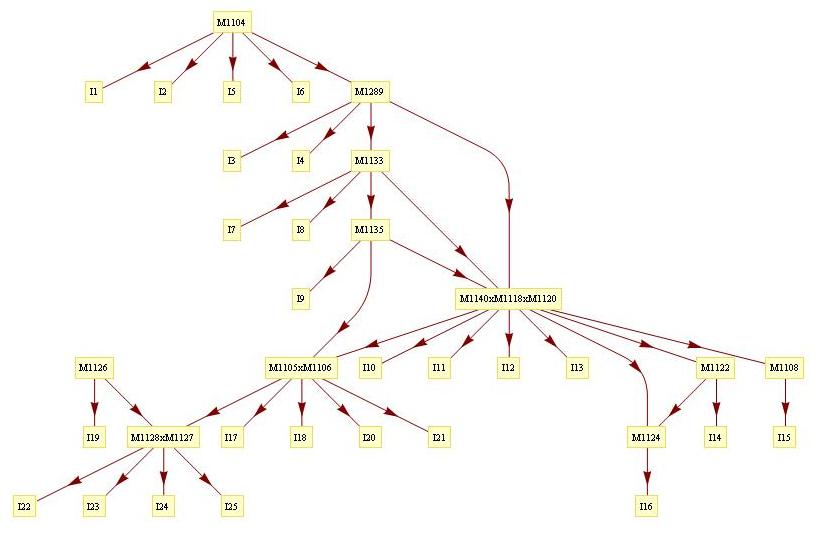
**Table 9.1: Results for Experiment 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Best Models** |  |  |  |
| **Iteration Number** | **Skills** | **AUC** | **RMSE** | **Accuracy** | **AIC** | **BIC** |
| 0 | 15 | 0.81843 | 0.37451 | 0.79813 | 539988.22937 | 540884.62767 |
| 1 | 14 | 0.82075 | 0.37277 | 0.80087 | 533428.45840 | 534306.56286 |
| 2 | 13 | 0.8192 | 0.37449 | 0.79769 | 538538.26208 | 539398.07270 |
| 3 | 12 | 0.82086 | 0.37279 | 0.80127 | 536138.56542 | 536943.49450 |
| 4 | 11 | 0.82137 | 0.37238 | 0.80185 | 533921.04780 | 534671.09536 |
| 5 | 10 | 0.82185 | 0.37303 | 0.79951 | 536019.94810 | 536733.40797 |
| 6 | 9 | 0.82126 | 0.37259 | 0.80193 | 535269.73527 | 535946.60746 |
| 7 | 8 | 0.81906 | 0.37408 | 0.79854 | 539371.24687 | 539993.23753 |
| 8 | 7 | 0.81767 | 0.37461 | 0.79787 | 538287.05902 | 538872.46199 |
| 9 | 6 | 0.81526 | 0.37679 | 0.79341 | 547561.78928 | 548128.89840 |
| 10 | 5 | 0.81912 | 0.37445 | 0.79758 | 539138.56159 | 539678.22996 |
| 11 | 4 | 0.81557 | 0.37718 | 0.79317 | 547274.26056 | 547795.63508 |
| 12 | 3 | 0.81471 | 0.37773 | 0.79038 | 549777.55035 | 550280.63103 |
| 13 | 2 | 0.81241 | 0.37861 | 0.78983 | 552058.20873 | 552542.99557 |
| 14 | 1 | 0.80576 | 0.38194 | 0.78756 | 558689.91568 | 559156.40867 |

**Figure 9.1: Skill Graph Results**



**Figure 9.2: Best Model Skill Graph (11 skills at iteration 4)**



The results show that the best RMSE obtained was from the 11-skill graph at iteration 4 with an RMSE of 0.37238. This is slightly better compared to the original skill graph RMSE of 0.37451. The 11-skill graph has a small but significant improvement (*p* = 6.22624E-68) from the original skill graph. The graph above also shows that models consisting between 9-12 skills have similar RMSE values and are alternative choices for a best model depending on the level of skill granularity desired. Those models are also significantly better than the original model. A significance table is shown below for skill graph consisting between 15-8 skills inclusive.

**Table 9.2: P-values (Skill models 15-8)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Skill model** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **15** |  | 5.16E-79 | 7.74E-12 | 4.81E-22 | 6.22E-68 | 7.37E-59 | 1.59E-30 | 0.23 |
| **14** | 5.16E-79 | 0 | 1.60E-44 | 2.64E-23 | 0.05 | 1.76E-22 | 1.31E-09 | 2.84E-51 |
| **13** | 7.74E-12 | 1.60E-44 | 0 | 5.29E-09 | 5.36E-39 | 4.34E-21 | 6.88E-15 | 0.05 |
| **12** | 4.81E-22 | 2.64E-23 | 5.29E-09 | 0 | 1.14E-33 | 0.77 | 0 | 6.10E-25 |
| **11** | 6.22E-68 | 0.05 | 5.36E-39 | 1.14E-33 | 0 | 2.96E-13 | 1.35E-06 | 1.32E-56 |
| **10** | 7.37E-59 | 1.76E-22 | 4.34E-21 | 0.77 | 2.96E-13 | 0 | 0.02 | 1.04E-15 |
| **9** | 1.59E-30 | 1.31E-09 | 6.88E-15 | 0 | 1.35E-06 | 0.02 | 0 | 1.80E-47 |
| **8** | 0.237 | 2.84E-51 | 0.05 | 6.10E-25 | 1.32E-56 | 1.04E-15 | 1.80E-47 | 0 |

**P-value table from the paired T-Test of the absolute errors from the models**

**P-values truncated to 2 digits after the decimal point**

In addition to looking at which model predicted the best, we also looked at which skills were being merged throughout our iterative search to see if we could find any general trends. A list of the merges is shown below. The individual skills are represented as their original numbering and a merged skill is represented by the number of each skill concatenated with an ‘x’. The numbering is in topological order, meaning that the skill highest up on the skill graph will be listed first for a merged skill. The first merge is between skill 1128 and 1127. Since skill 1128 is a parent of skill 1127, it is listed first in the combined skill name 1128x1127.

Merge 1: 1128x1127

Merge 2: 1140x1118

Merge 3: 1140x1118x1120

Merge 4: 1105x1106

Merge 5: 1122x1124

Merge 6: 1140x1118x1120x1105x1106

Merge 7: 1135x1140x1118x1120x1105x1106

Merge 8: 1289x1233

Merge 9: 1135x1140x1118x1120x1105x1106x1108

Merge 10: 1126x1128x1127

Merge 11: 1135x1140x1118x1120x1105x1106x1108x1122x1124

Merge 12: 1104x1289x1233

Merge 13: 1135x1140x1118x1120x1105x1106x1108x1122x1124x1126x1128x1127

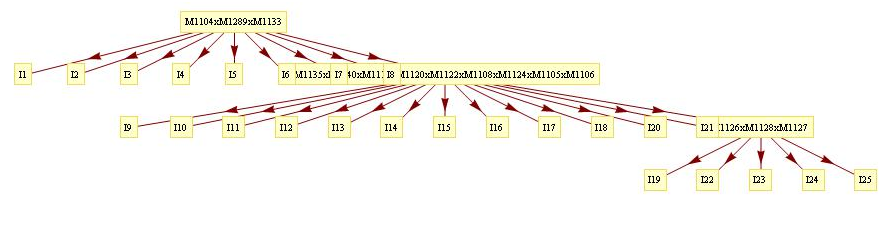
Merge 14: 1104x1289x1233x1135x1140x1118x1120x1105x1106x1108x1122x1124x1126x1128x1127

One observation is that the skills that are chosen to be merged in the first few iterations tend to be near the bottom of the skill graph. This suggests that the skills near the top of the skill graph are really separate skills compared to the skills near the bottom of the skill graph, which is the same observation made in experiment 2. Since the skills near the bottom are merged first based on best RMSE, those skills are better predicted with one skill parameter. Therefore those skills are not really that distinct since they are better modeled with one skill parameter compared to skills near the top of the skill graph.

The last skills to be merged are more likely to be distinct skills from the other skills. Our last merge was the merge between the skill group of 1104x1289x1233 and the rest of the skills in the skill graph. Since this was the last merge chosen resulting in the worst RMSE, it is likely that the group of skills 1104x1289x1233 is a separate group from the rest of the skills.

We believe the structure of the initial skill graph also has influence on the distinct skill groups. Since our merge operation only merges adjacent skills, it takes several merges for a skill at the top of the graph to merge with a skill at the bottom of the graph. A separation will naturally occur between the different levels of the graph. This implies that the original skill graph would need to be somewhat correct in terms of network topology. If non-adjacent skills were in fact the same or similar skills, there would be no easy way for our iterative algorithm to merge them, because our merge operation can only merge adjacent skills. Three distinct skill groups can be seen after iteration 12 in a 3-skill graph. These skills show the influence of the original network topology. The skills tend to group by their locations in the original network. The topmost skill group consists of the skills at the top of the graph, the middle group of skills consists of the skills in the middle of the graph, and the bottommost group of skills consists of the skills at the bottom of the original skill graph.

**Figure 9.3: Skill Graph after Merge 12 (3 skills)**



An additional observation is that some of the skills tend to merge by pairing up with one and only one adjacent skill before RMSE starts to decline. Before merge 5, the merges are all pairwise with the exception of merge 3. After merge 5, the skills tend to keep merging into the same skill. The graph generated after merge 4 corresponds to the best skill graph. This suggests that the adjacent skills tend to be similar skills. It also suggests that skills 1140 and 1120 are similar although they are not adjacent. This is a stronger relationship for several reasons. Firstly, the merges that culminated in the merger of M1140, M1118, and M1120 all took place before the best skill graph was reached. This indicates that those three skills give better predictive performance when represented as one skill. Secondly, this was the first 3-skill group to be merged and the only 3-skill group in the best model before RMSE declines. Lastly the three skills took two iterations of the search algorithm to merge together because skills 1140 and 1120 were not adjacent skills. Despite the initial graph topology our search decided to merge these three skills. The combination of all these factors give stronger reasoning that the three skills 1140, 1118, and 1120 are not really distinct skills.

**Appendix**

**Questions**

**1.** Which statement about 3+ (-2) and 3 2 best describes the meaning of the symbols "+" and " "?

A) 3 + (-2) and 3 2 both mean subtracting 2 from 3

B) 3 + (-2) and 3 2 both mean adding 3 to the opposite of 2

C) 3 + (-2) means subtracting 2 from 3, and 3 2 means adding 3 to the opposite of 2

D) X 3 + (-2) means adding 3 to the opposite of 2, and 3 2 means subtracting 2 from 3

QuestionId: 50049, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**2.** Which statement about 6 (-5) and 6 + 5 best describes the meaning of the symbols "+" and " "?

A) 6 (-5) and 6 + 5 both mean adding 6 and 5

B) 6 (-5) and 6 + 5 both mean subtracting the opposite of 5 from 6

C) X 6 (-5) means subtracting the opposite of 5 from 6, and 6 + 5 means adding 6 and 5

D) 6 (-5) means adding 6 and 5, and 6 + 5 means subtracting the opposite of 5 from 6

QuestionId: 50050, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**3.** Which statement about integers is true?

A) X -3 is an integer because 3 is an integer

B) is an integer because both 1 and 2 are integers

C) 0.75 is an integer because 75 is an integer

D) -1.26 is an integer because both -1 and 26 are integers

QuestionId: 50051, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**4.** Which is always an integer?

A) positive fraction

B) negative decimal

C) improper fraction

D) X zero

QuestionId: 50052, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**5.** Which correctly explains why -5 is the opposite of 5?

A) X -5 added to 5 equals 0

B) 0 is to the left of 5 on a number line

C) -5 is to the left of 5 on a number line

D) they are located on different sides of 0

QuestionId: 50053, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**6.** Which choice contains a pair of opposite numbers?

A) -5,

B) -5,

C) -5, -5

D) X -5, 5

QuestionId: 50054, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**7.** Which statement comparing two integers is correct?

A) -1,537 > 1,576

B) -1,537 < -1,576

C) 1,537 < -1,576

D) X -1,537 > -1,576

QuestionId: 50055, Standard 6 "NS", Benchmark 7 "7", Indicator "a", Sub Indicator ""

**8.** Which expression comparing two integers is correct?

A) -2,789 > 2,980

B) X -2,789 > -2,980

C) 2,789 < -2,980

D) -2,789 < -2,980

QuestionId: 50056, Standard 6 "NS", Benchmark 7 "7", Indicator "a", Sub Indicator ""

**9.** Which list of temperatures is in order from **highest** to **lowest**?

A) X 45 degrees F, 2 degrees F, -32 degrees F, -38 degrees F, -40 degrees F

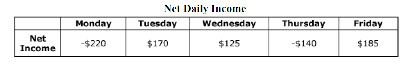
B) 45 degrees F, -40 degrees F, -38 degrees F, -32 degrees F, 2 degrees F

C) 2 degrees F, -32 degrees F, -38 degrees F, -40 degrees F, 45 degrees F

D) -40 degrees F, -38 degrees F, -32 degrees F, 2 degrees F, 45 degrees F

QuestionId: 50057, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

**10.** A business's net daily income for five days is shown in the table below.



Which list of incomes is in order from **least** to **greatest**?

A) -$220, $185, $170, -$140, $125

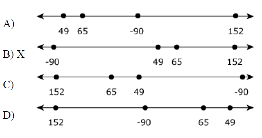
B) $125, -$140, $170, $185, -$220

C) X -$220, -$140, $125, $170, $185

D) $185, $170, $125, -$140, -$220

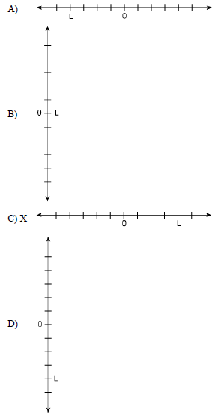
QuestionId: 50058, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

**11.** The school's band members are selling cookies to raise money. The band members need to pay for unsold boxes. The table below shows the total sales, including fees for unsold boxes of each member.



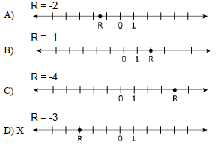
QuestionId: 50059, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

**12.** Each graph displays points L and 0 on a number line. On which number line is point L positioned at the value of



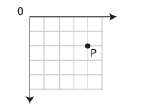
QuestionId: 50060, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

**13.** Which graph correctly represents the integer shown?



QuestionId: 50061, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

**14.** Point P is plotted in the graph shown below. Which coordinate pair lists the correct values for P?



A) (-2, 4)

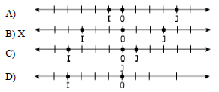
B) (-5, 4)

C) X (4, -2)

D) (4, -5)

QuestionId: 50062, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

**15.** Which number line shows that I is the opposite of J?



QuestionId: 50063, Standard 6 "NS", Benchmark 6 "6", Indicator "a", Sub Indicator ""

**16.** Which procedure describes correctly how to graph the ordered pair (-2, 3) on the coordinate plane?

A) X Starting from the origin, move 2 units to the left, and move 3 units up.

B) Starting from the origin, move 2 units to the right, and move 3 units up.

C) Starting from the origin, move 2 units to the left, and move 3 units down.

D) Starting from the origin, move 2 units to the right, and move 3 units down.

QuestionId: 50064, Standard 6 "NS", Benchmark 6 "6", Indicator "c", Sub Indicator ""

**17.** The freezing point of water is 0 degrees C. Which temperature is 10 degrees below the freezing point of water?

A) 10 degrees C

B) 100 degrees C

C) X -10 degrees C

D) -100 degrees C

QuestionId: 50065, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**18.** Antonio's account had a $5 balance before he went to the bank to make a deposit. What is his balance after putting $25 into this account?

A) $0

B) -$20

C) $25

D) X $30

QuestionId: 50066, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**19.** Points R and T are plotted on the number line shown below.



Which sentence correctly explains the relationship between R and T?

A) X R is less than T, because R is to the left of T on the number line

B) T is less than R, because T is to the right of R on the number line

C) R is greater than T, because the number line extends to the left and right of R

D) T is greater than R, because the number line extends to the left and right of T

QuestionId: 50067, Standard 6 "NS", Benchmark 7 "7", Indicator "a", Sub Indicator ""

**20.** Bob owed $50 to a business. Then Bob paid $50 to the business. Which statement is true?

A) X Bob now owes the business $0.

B) Bob now owes the business $100.

C) The business now owes Bob $50.

D) The business now owes Bob $100.

QuestionId: 50068, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**21.** The elevation of Village A is 5 meters below sea level. Village B is 4 meters above Village A. What is Village B's elevation, in relation to sea level?

A) 1 meter above sea level

B) X 1 meter below sea level

C) 9 meters above sea level

D) 9 meters below sea level

QuestionId: 50069, Standard 6 "NS", Benchmark 5 "5", Indicator "", Sub Indicator ""

**22.** Which sentence correctly explains why a temperature of 3 degrees below 0 is warmer than a temperature of 7 degrees below 0?

A) X -3 is greater than -7

B) -3 is not equal to -7

C) -3 is equal to -7

D) -3 is less than -7

QuestionId: 50070, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

**23.** Which sentence below correctly explains why 6 meters below sea level is lower than 2 meters below sea level?

A) -6 is greater than -2

B) -6 is not equal to -2

C) X -6 is less than -2

D) -6 is equal to -2

QuestionId: 50071, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

**24.** Family J spent 3 1/2 hours traveling to a campsite. Departing from the same place, FamilyK spent 3 5/6 hours traveling to the same campsite. Which statement correctly comparesthe two families' traveling time?

A) Family J spent more time traveling because 3 1/2 > 3 5/6.

B) X Family J spent less time traveling because 3 1/2 < 3 5/6.

C) Family K spent more time traveling because 3 1/2 > 3 5/6.

D) Family K spent less time traveling because 3 1/2 < 3 5/6.

QuestionId: 50072, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

**25.** Two gas pipelines are being placed in a town. Pipeline A is placed at a depth of 23/7 meters below the ground and pipeline B is placed at a depth of 3 1/4 meters below theground. Which statement is correct?

A) Pipeline A is deeper than pipeline B because -2 3/7 < -3 1/4.

B) X Pipeline B is deeper than pipeline A because -2 3/7 > -3 1/4.

C) Pipeline A is deeper than pipeline B because -2 3/7 > -3 1/4.

D) Pipeline B is deeper than pipeline A because -2 3/7 < -3 1/4.

QuestionId: 50073, Standard 6 "NS", Benchmark 7 "7", Indicator "b", Sub Indicator ""

**Example Mathematica code to view the graphs**

**Code generated by the code in LearnGraph.java**

This is example code that can be used in mathematica to generate a nice image of the graphs. This mathematica code was generated by the code in the LearnGraph.java code. The output shown below is a single file of output. The file is an example file of the first iteration of possible merges done in the iterative search experiment. Code for all the 16 possible new graphs are in this file.

(\*1\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133xM1140, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133xM1140 -> M1135, M1133xM1140 -> M1118, M1133xM1140 -> M1105, M1133xM1140-> I7, M1133xM1140-> I8, M1133xM1140-> I10, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1118 -> M1120, M1118 -> M1122, M1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*2\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120, M1118 -> M1122, M1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1126xM1128, M1105 -> M1106, M1105-> I17, M1105-> I18, M1126xM1128 -> M1127, M1126xM1128-> I19, M1126xM1128-> I22, M1126xM1128-> I23, M1106 -> M1126xM1128, M1106-> I20, M1106-> I21, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*3\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118xM1122, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118xM1122, M1135 -> M1105, M1135-> I9, M1140 -> M1118xM1122, M1140 -> M1105, M1140-> I10, M1118xM1122 -> M1120, M1118xM1122 -> M1124, M1118xM1122-> I11, M1118xM1122-> I14, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*4\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1120xM1105, M1135-> I9, M1140 -> M1118, M1140 -> M1120xM1105, M1140-> I10, M1118 -> M1120xM1105, M1118 -> M1122, M1118-> I11, M1120xM1105 -> M1108, M1120xM1105 -> M1124, M1120xM1105 -> M1106, M1120xM1105 -> M1128, M1120xM1105-> I12, M1120xM1105-> I13, M1120xM1105-> I17, M1120xM1105-> I18, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*5\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1105xM1106, M1135-> I9, M1140 -> M1118, M1140 -> M1105xM1106, M1140-> I10, M1118 -> M1120, M1118 -> M1122, M1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105xM1106, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105xM1106 -> M1128, M1105xM1106-> I17, M1105xM1106-> I18, M1105xM1106-> I20, M1105xM1106-> I21, M1126 -> M1128, M1126-> I19, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*6\*)LayeredGraphPlot[{M1104xM1289 -> M1133, M1104xM1289 -> M1118, M1104xM1289-> I1, M1104xM1289-> I2, M1104xM1289-> I3, M1104xM1289-> I4, M1104xM1289-> I5, M1104xM1289-> I6, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120, M1118 -> M1122, M1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*7\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120, M1118 -> M1122, M1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128xM1127, M1105-> I17, M1105-> I18, M1126 -> M1128xM1127, M1126-> I19, M1106 -> M1128xM1127, M1106-> I20, M1106-> I21, M1128xM1127-> I22, M1128xM1127-> I23, M1128xM1127-> I24, M1128xM1127-> I25}, VertexLabeling -> True]

(\*8\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118xM1120, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118xM1120, M1135 -> M1105, M1135-> I9, M1140 -> M1118xM1120, M1140 -> M1105, M1140-> I10, M1118xM1120 -> M1122, M1118xM1120 -> M1108, M1118xM1120 -> M1124, M1118xM1120 -> M1105, M1118xM1120-> I11, M1118xM1120-> I12, M1118xM1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*9\*)LayeredGraphPlot[{M1104 -> M1289xM1133, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289xM1133 -> M1135, M1289xM1133 -> M1140, M1289xM1133 -> M1118, M1289xM1133-> I3, M1289xM1133-> I4, M1289xM1133-> I7, M1289xM1133-> I8, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120, M1118 -> M1122, M1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*10\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120, M1118 -> M1122, M1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106xM1128, M1105-> I17, M1105-> I18, M1126 -> M1106xM1128, M1126-> I19, M1106xM1128 -> M1127, M1106xM1128-> I20, M1106xM1128-> I21, M1106xM1128-> I22, M1106xM1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*11\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1135xM1118, M1289-> I3, M1289-> I4, M1133 -> M1135xM1118, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135xM1118 -> M1120, M1135xM1118 -> M1122, M1135xM1118 -> M1105, M1135xM1118-> I9, M1135xM1118-> I11, M1140 -> M1135xM1118, M1140 -> M1105, M1140-> I10, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*12\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120xM1124, M1118 -> M1122, M1118-> I11, M1120xM1124 -> M1108, M1120xM1124 -> M1105, M1120xM1124-> I12, M1120xM1124-> I13, M1120xM1124-> I16, M1122 -> M1120xM1124, M1122-> I14, M1108-> I15, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*13\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133xM1135, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133xM1135 -> M1140, M1133xM1135 -> M1118, M1133xM1135 -> M1105, M1133xM1135-> I7, M1133xM1135-> I8, M1133xM1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120, M1118 -> M1122, M1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*14\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120xM1108, M1118 -> M1122, M1118-> I11, M1120xM1108 -> M1124, M1120xM1108 -> M1105, M1120xM1108-> I12, M1120xM1108-> I13, M1120xM1108-> I15, M1122 -> M1124, M1122-> I14, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*15\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140, M1133-> I7, M1133-> I8, M1135 -> M1118, M1135 -> M1105, M1135-> I9, M1140 -> M1118, M1140 -> M1105, M1140-> I10, M1118 -> M1120, M1118 -> M1122xM1124, M1118-> I11, M1120 -> M1122xM1124, M1120 -> M1108, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122xM1124-> I14, M1122xM1124-> I16, M1108-> I15, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

(\*16\*)LayeredGraphPlot[{M1104 -> M1289, M1104-> I1, M1104-> I2, M1104-> I5, M1104-> I6, M1289 -> M1133, M1289 -> M1140xM1118, M1289-> I3, M1289-> I4, M1133 -> M1135, M1133 -> M1140xM1118, M1133-> I7, M1133-> I8, M1135 -> M1140xM1118, M1135 -> M1105, M1135-> I9, M1140xM1118 -> M1120, M1140xM1118 -> M1122, M1140xM1118 -> M1105, M1140xM1118-> I10, M1140xM1118-> I11, M1120 -> M1108, M1120 -> M1124, M1120 -> M1105, M1120-> I12, M1120-> I13, M1122 -> M1124, M1122-> I14, M1108-> I15, M1124-> I16, M1105 -> M1106, M1105 -> M1128, M1105-> I17, M1105-> I18, M1126 -> M1128, M1126-> I19, M1106 -> M1128, M1106-> I20, M1106-> I21, M1128 -> M1127, M1128-> I22, M1128-> I23, M1127-> I24, M1127-> I25}, VertexLabeling -> True]

**Matlab code**

**Example graph code**

**Code generated by the code in LearnGraph.java**

function dag = structure\_1

% auto-generated code by Doug based on Zachs code

% Bayesian Prediction Analysis for Dynamic Learning Maps (Kansas)

% in collaboration with Neil Heffernan and colleagues (WPI)

% data property of Angela Broaddus (Kansas) and Neal Kingston (Kansas)

N=39;

M1104=1;

M1289=2;

M1133\_M1140=3;

M1135=4;

M1118=5;

M1120=6;

M1122=7;

M1108=8;

M1124=9;

M1105=10;

M1126=11;

M1106=12;

M1128=13;

M1127=14;

I1=15;

I2=16;

I3=17;

I4=18;

I5=19;

I6=20;

I7=21;

I8=22;

I9=23;

I10=24;

I11=25;

I12=26;

I13=27;

I14=28;

I15=29;

I16=30;

I17=31;

I18=32;

I19=33;

I20=34;

I21=35;

I22=36;

I23=37;

I24=38;

I25=39;

dag=zeros(N,N);

dag(M1104, [M1289 I1 I2 I5 I6]) = 1;

dag(M1289, [M1133\_M1140 M1118 I3 I4]) = 1;

dag(M1133\_M1140, [M1135 M1118 M1105 I7 I8 I10]) = 1;

dag(M1135, [M1118 M1105 I9]) = 1;

dag(M1118, [M1120 M1122 I11]) = 1;

dag(M1120, [M1108 M1124 M1105 I12 I13]) = 1;

dag(M1122, [M1124 I14]) = 1;

dag(M1108, I15) = 1;

dag(M1124, I16) = 1;

dag(M1105, [M1106 M1128 I17 I18]) = 1;

dag(M1126, [M1128 I19]) = 1;

dag(M1106, [M1128 I20 I21]) = 1;

dag(M1128, [M1127 I22 I23]) = 1;

dag(M1127, [I24 I25]) = 1;

**Evaluation code**

clc;clear all;close all;

seed = 10;

s = RandStream('mt19937ar', 'Seed', seed);

RandStream.setGlobalStream(s);

%% load data

load integers\_dlm.mat;

filename = 'eval\_results1.txt';

results = fopen(filename,'w');

i = 1;

FuncName = ['OutputStructure\_14\_',num2str(i)];

%% Evaluate DAG

para = dlm\_basic\_eval(dlm,feval(FuncName),i);

data = load(['dlm\_resultsFinalRun',num2str(i),'.txt']);

%% Record Results

aic = aic(data(:,5),data(:,4), para);

bic = bic(data(:,5),data(:,4), para);

aucR(i) = auc(data(:,4),data(:,5));

rmseR(i) = sqrt(mean((data(:,4)-data(:,5)).^2));

accuracyR(i) = mean(round(data(:,4))==data(:,5));

fprintf(results, '%d %.5f %.5f %.5f %.5f %.5f\n',i,aucR(i),rmseR(i),accuracyR(i), aic, bic)

fclose(results);

function total\_params = dlm\_basic\_eval(dlm,dag,FileNum)

% Zach A. Pardos (zp@csail.mit.edu)

% Bayesian Prediction Analysis for Dynamic Learning Maps (Kansas)

% in collaboration with Neil Heffernan and colleagues (WPI)

% data property of Angela Broaddus (Kansas) and Neal Kingston (Kansas)

dlm.responses = dlm.responses(dlm.complete,:);

dlm.studentfold5 = dlm.studentfold5(dlm.complete);

N=size(dag,1);

hidden\_nodes = 1:N-size(dlm.responses,2);

observed\_nodes = N-size(dlm.responses,2)+1:N;

eclass=1:N;

bnet = mk\_bnet(dag,2\*ones(1,N),'observed',observed\_nodes,'discrete',1:N,'equiv\_class',eclass );

total\_params=0;

for C=1:N

bnet.CPD{C} = tabular\_CPD(bnet, C);

cpt = CPD\_to\_CPT(bnet.CPD{C});

total\_params = total\_params + length(cpt(:));

end

total\_params =total\_params/2;

cases = cell(size(dlm.responses,1),N);

cases(:,observed\_nodes) = num2cell(dlm.responses+1);

engine = jtree\_inf\_engine(bnet);

max\_iter = 300;

filename = ['dlm\_resultsFinalRun',num2str(FileNum),'.txt'];

report = fopen(filename,'w');

for fold=1:5

cases2=cases(dlm.studentfold5 ~= fold,:);

[bnet2, LLtrace, engine2] = learn\_params\_em(engine,cases2',max\_iter);

cases3=cases(dlm.studentfold5 == fold,:);

for c=1:size(cases3,1)

fprintf('%d%% done with fold %d of 5\n',round(c\*100/size(cases3,1)),fold);

case1=cases3(c,:);

for ifold=1:3

scase=cell(1,N);

evitems=dlm.itemfold3 ~= ifold;

scase(observed\_nodes(evitems)) = case1(observed\_nodes(evitems));

[engine3,ll] = enter\_evidence(engine2,scase);

titems=find(dlm.itemfold3 == ifold);

for t=titems'

m = marginal\_nodes(engine3,observed\_nodes(t));

m = m.T(2);

fprintf(report,'%d %d %d %.5f %d\n',fold,c,t,m,case1{observed\_nodes(t)}-1);

end

end

end

end

fclose(report);

**Java Code**

**Learn Graph.java**

The code in this file is described in 3. It takes a graph as an item matrix of 1’s and 0’s as input as well as a list of skills each item maps to. The code randomly chooses merges and outputs all unique randomly derived graphs from the given graph using the merge operation. By slightly modifying this code, the possible merges for one iteration can be obtained and were used in the iterative search experiment for each iteration.

import java.io.\*;

import java.util.Vector;

import java.util.Stack;

import java.util.Random;

//class that will generate random graphs by random merges given an input graph

public class LearnGraph

{

private Vector <int[]> groupVector;

private Vector <int[][]> structureVector;

private int structureCopy[][];

private int groupArrayCopy[];

private int structure[][];

private int groupArray[];

private int INPUT\_ROWS, INPUT\_COLUMNS;

String skillNames[];

public LearnGraph(String structurePath, String groupPath)

{

groupVector = new Vector();

structureVector = new Vector();

readFile(structurePath, groupPath);

outputMerges();

for(int i=0; i<100000; i++)

{

//copy stuff then run off of copy

structureCopy = new int[INPUT\_COLUMNS][INPUT\_COLUMNS];

groupArrayCopy = new int[INPUT\_COLUMNS];

for(int j=0; j<INPUT\_COLUMNS; j++)

{

for(int k=0; k<INPUT\_COLUMNS; k++)

{

structureCopy[j][k] = structure[j][k];

}

groupArrayCopy[j] = groupArray[j];

}

runCrushing();

for(int j=0; j<INPUT\_COLUMNS; j++)

{

structureCopy[j][j] = 0;

}

//save copy if not already exists and acyclic

boolean acyclic = cycleCheck(structureCopy, groupArrayCopy);

//check all in order...

boolean exists = false;

for(int h=0; h<groupVector.size() && !exists; h++)

{

int tempGroup[] = groupVector.get(h);

int tempStructure[][] = structureVector.get(h);

boolean same = true;

for(int j=0; j<INPUT\_COLUMNS && same; j++)

{

if(tempGroup[j] != groupArrayCopy[j])

{

same = false;

}

}

if(same)

{

for(int j=0; j<INPUT\_COLUMNS && same; j++)

{

for(int k=0; k<INPUT\_COLUMNS && same; k++)

{

if(tempStructure[j][k] != structureCopy[j][k])

{

same = false;

}

}

}

}

if(same)

{

exists = true;

}

}

if(!exists && acyclic)

{

groupVector.add(groupArrayCopy);

structureVector.add(structureCopy);

}

System.out.println(i);

}

for(int i=0; i<groupVector.size(); i++)

{

outputStructureFirstRowHeader(i, structureVector.get(i), groupVector.get(i));

outputStructureNoHeaders(i, structureVector.get(i), groupVector.get(i));

outputStructureMatlab(i, structureVector.get(i), groupVector.get(i));

}

outputStructureMathematica();

System.out.println("XX" + groupVector.size());

}

private void outputMerges()

{

for(int i=0; i<INPUT\_COLUMNS; i++)

{

for(int j=i+1; j<INPUT\_COLUMNS; j++)

{

if(structure[i][j] == 1)

{

System.out.println("Possible Merge = Skill " + skillNames[i] + " to skill " + skillNames[j]);

}

}

}

}

private void readFile(String structurePath, String groupPath)

{

//get array sizes

try

{

String line = "";

BufferedReader br = new BufferedReader(new FileReader(structurePath));

line = br.readLine();

String sizeArray[] = line.split(",");

INPUT\_COLUMNS = sizeArray.length;

br.close();

}

catch(Exception e)

{

e.printStackTrace();

System.exit(-1);

}

structure = new int[INPUT\_COLUMNS][INPUT\_COLUMNS];

//initialize everything to -1

for(int i=0; i<INPUT\_COLUMNS; i++)

{

for(int j=0; j<INPUT\_COLUMNS; j++)

{

structure[i][j] = -1;

}

}

try

{

String line = "";

BufferedReader br = new BufferedReader(new FileReader(structurePath));

int currentRow = 0;

while((line = br.readLine()) != null)

{

String row[] = line.split(",");

for(int i=0; i<INPUT\_COLUMNS; i++)

{

structure[currentRow][i] = Integer.parseInt(row[i]);

}

currentRow++;

}

br.close();

}

catch(Exception e)

{

e.printStackTrace();

System.exit(-1);

}

groupArray = new int[INPUT\_COLUMNS];

//initialize everything to -1

for(int i=0; i<INPUT\_COLUMNS; i++)

{

groupArray[i] = -1;

}

try

{

String line = "";

BufferedReader br = new BufferedReader(new FileReader(groupPath));

line = br.readLine();

skillNames = line.split(",");

line = br.readLine();

String groupArray0[] = line.split(",");

br.close();

for(int i=0; i<skillNames.length; i++)

{

groupArray[i] = Integer.parseInt(groupArray0[i]);

}

}

catch(Exception e)

{

e.printStackTrace();

System.exit(-1);

}

//check to make sure everything was loaded okay

for(int i=0; i<INPUT\_COLUMNS; i++)

{

for(int j=0; j<INPUT\_COLUMNS; j++)

{

if(structure[i][j] == -1)

{

System.out.println("Input Structure file not loaded correctly");

System.exit(-1);

}

}

if(groupArray[i] == -1)

{

System.out.println("Input groups file not loaded correctly");

System.exit(-1);

}

}

}

private boolean cycleCheck(int structures[][], int groups[])

{

boolean rv = true;

Vector <Integer> uniqueGroups = new Vector();

for(int i=0; i<groups.length; i++)

{

if(!uniqueGroups.contains(groups[i]))

{

uniqueGroups.add(groups[i]);

}

}

int skillStructure[][] = new int[uniqueGroups.size()][uniqueGroups.size()];

int rowCount = 0;

int columnCount = 0;

for(int i=0; i<INPUT\_COLUMNS; i++)

{

boolean iMember = false;

for(int j=0; j<INPUT\_COLUMNS; j++)

{

boolean jMember = false;

for(int k=0; k<groups.length; k++)

{

if(i == groups[k])

{

iMember = true;

}

if(j == groups[k])

{

jMember = true;

}

}

//i and j are both group numbers

if(iMember && jMember)

{

skillStructure[rowCount][columnCount] = structures[i][j];

columnCount++;

}

}

if(iMember)

{

rowCount++;

columnCount = 0;

}

}

Stack <Integer> removed = new Stack();

boolean change = true;

while(change)

{

change = false;

for(int i=0; i<uniqueGroups.size() && !change; i++)

{

boolean incoming = false;

for(int j=0; j<uniqueGroups.size(); j++)

{

//detect incoming edge

if(skillStructure[j][i] == 1)

{

incoming = true;

}

}

if(!incoming && removed.search(i) < 0)

{

removed.push(i);

for(int j=0; j<uniqueGroups.size(); j++)

{

skillStructure[i][j] = 0;

}

change = true;

}

}

}

if(removed.size() != uniqueGroups.size())

{

rv = false;

}

return rv;

}

private void runCrushing()

{

boolean keepRunning = true;

int iteration = 0;

Random randomNumber = new Random();

while(iteration <= 13 /\*keepRunning\*/)

{

//pick two connected groups somewhat randomly

int group1 = -1;

int group2 = -1;

//question spot, not actually the group, need to use as array spot for group number

group1 = randomNumber.nextInt(groupArrayCopy.length);

group2 = -1;

boolean adjacent = false;

while(!adjacent)

{

group2 = randomNumber.nextInt(groupArrayCopy.length);

if(group2 != group1 && groupArrayCopy[group2] != groupArrayCopy[group1] && (structureCopy[group2][group1] == 1 || structureCopy[group1][group2] == 1))

{

adjacent = true;

}

}

boolean done = false;

for(int i=0; i<groupArrayCopy.length && !done; i++)

{

if(groupArrayCopy[i] == groupArrayCopy[group1])

{

crush(group1, group2);

done = true;

}

if(groupArrayCopy[i] == groupArrayCopy[group2])

{

crush(group2, group1);

done = true;

}

}

//determine keepRunning randomly from iteration

iteration++;

int stop = randomNumber.nextInt(groupArrayCopy.length - iteration);

if(stop == 0)

{

keepRunning = false;

}

//make sure there is more than 1 group

if(keepRunning)

{

keepRunning = false;

for(int i=0; i<groupArrayCopy.length; i++)

{

if(groupArrayCopy[i] != 0)

{

keepRunning = true;

}

}

}

}

private void crush(int question1, int question2)

{

int oldGroup = groupArrayCopy[question2];

//set skill for question 2 to be the skill of question 1

for(int i=0; i<INPUT\_COLUMNS; i++)

{

if(groupArrayCopy[i] == oldGroup)

{

groupArrayCopy[i] = groupArrayCopy[question1];

}

}

Vector <Integer> groupNumbers = new Vector();

for(int i=0; i<groupArrayCopy.length; i++)

{

if(groupArrayCopy[i] == groupArrayCopy[question1])

{

groupNumbers.add(i);

}

}

//for all group members, all same children

//for all group members, columns should be the same

for(int i=0; i<groupNumbers.size(); i++)

{

for(int j=0; j<groupArrayCopy.length; j++)

{

if(structureCopy[groupNumbers.get(i)][j] == 1)

{

for(int k=0; k<groupNumbers.size(); k++)

{

structureCopy[groupNumbers.get(k)][j] = 1;

}

}

if(structureCopy[j][groupNumbers.get(i)] == 1)

{

for(int k=0; k<groupNumbers.size(); k++)

{

structureCopy[j][groupNumbers.get(k)] = 1;

}

}

}

}

}

private void outputStructureFirstRowHeader(int outputNumber, int structures[][], int groups[])

{

StringBuilder outputString = new StringBuilder(INPUT\_COLUMNS \* INPUT\_COLUMNS);

Vector <Integer> uniqueGroups = new Vector();

for(int i=0; i<groups.length; i++)

{

if(!uniqueGroups.contains(groups[i]))

{

uniqueGroups.add(groups[i]);

}

}

for(int i=0; i<uniqueGroups.size(); i++)

{

for(int j=0; j<groups.length; j++)

{

if(groups[j] == uniqueGroups.get(i))

{

outputString = outputString.append(j+1);

outputString = outputString.append(" ; ");

}

}

outputString = outputString.append(",");

}

outputString = outputString.append("\n");

for(int i=0; i<INPUT\_COLUMNS; i++)

{

boolean iMember = false;

for(int j=0; j<INPUT\_COLUMNS; j++)

{

boolean jMember = false;

for(int k=0; k<groups.length; k++)

{

if(i == groups[k])

{

iMember =true;

}

if(j == groups[k])

{

jMember = true;

}

}

//i and j are both group numbers

if(iMember && jMember)

{

outputString = outputString.append(Integer.toString(structures[i][j]));

outputString = outputString.append(",");

}

}

if(iMember)

{

outputString = outputString.append("\n");

}

}

String dataString = outputString.toString();

try

{

FileWriter file = new FileWriter("OutputFirstRowHeader" + File.separator +"OutputStructure" + (outputNumber+1) + ".csv", false);

file.write(dataString);

file.flush();

file.close();

}

catch(Exception e)

{

e.printStackTrace();

System.exit(0);

}

}

//

private void outputStructureNoHeaders(int outputNumber, int structures[][], int groups[])

{

StringBuilder outputString = new StringBuilder(INPUT\_COLUMNS \* INPUT\_COLUMNS);

for(int i=0; i<INPUT\_COLUMNS; i++)

{

boolean iMember = false;

for(int j=0; j<INPUT\_COLUMNS; j++)

{

boolean jMember = false;

for(int k=0; k<groups.length; k++)

{

if(i == groups[k])

{

iMember =true;

}

if(j == groups[k])

{

jMember = true;

}

}

//i and j are both group numbers

if(iMember && jMember)

{

outputString = outputString.append(Integer.toString(structures[i][j]));

outputString = outputString.append(",");

}

}

if(iMember)

{

outputString = outputString.append("\n");

}

}

String dataString = outputString.toString();

try

{

FileWriter file = new FileWriter("OutputNoHeaders" + File.separator +"Structure" + (outputNumber+1) + ".csv", false);

file.write(dataString);

file.flush();

file.close();

}

catch(Exception e)

{

e.printStackTrace();

System.exit(0);

}

}

private void outputStructureMathematica()

{

StringBuilder outputString = new StringBuilder(INPUT\_COLUMNS \* INPUT\_COLUMNS \* structureVector.size());

for(int h=0; h<structureVector.size(); h++)

{

outputString = outputString.append("(\*");

outputString = outputString.append(h+1);

outputString = outputString.append("\*)");

outputString = outputString.append("LayeredGraphPlot[");

outputString = outputString.append("{");

int structures[][] = structureVector.get(h);

int groups[] = groupVector.get(h);

Vector <Integer> uniqueGroups = new Vector();

for(int i=0; i<groups.length; i++)

{

if(!uniqueGroups.contains(groups[i]))

{

uniqueGroups.add(groups[i]);

}

}

//create the group name by combining all skills in the group into one name

String skillGroupNames[] = new String[uniqueGroups.size()];

for(int i=0; i<uniqueGroups.size(); i++)

{

String name = "";

for(int j=0; j<groups.length; j++)

{

String skillName = skillNames[j];

if(groups[j] == uniqueGroups.get(i) && name.indexOf(skillName) == -1)

{

name = name.concat(skillName + "x");

}

}

skillGroupNames[i] = name.substring(0, name.length()-1);

//System.out.println(name);

}

for(int i=0; i<uniqueGroups.size(); i++)

{

int currentGroup = uniqueGroups.get(i);

int firstGroupMember = -1;

boolean found = false;

for(int j=0; j<groups.length && !found; j++)

{

if(groups[j] == currentGroup)

{

firstGroupMember = j;

found = true;

}

}

//get all unique groups that this current group points to

Vector <Integer> uniqueGroups2 = new Vector();

for(int j=0; j<groups.length; j++)

{

if(groups[firstGroupMember] != groups[j] && structures[firstGroupMember][j] == 1)

{

if(!uniqueGroups2.contains(groups[j]))

{

uniqueGroups2.add(groups[j]);

}

}

}

for(int j=0; j<uniqueGroups2.size(); j++)

{

int groupNum = uniqueGroups2.get(j);

//find skill group name that has skill name of the group number

int specialSpot = -1;

for(int k=0; k<skillGroupNames.length && specialSpot == -1; k++)

{

if(skillGroupNames[k].indexOf(skillNames[groupNum]) != -1)

{

specialSpot = k;

}

}

outputString = outputString.append(skillGroupNames[i] + " -> " + skillGroupNames[specialSpot] + ", ");

}

for(int j=0; j<groups.length; j++)

{

if(groups[j] == groups[currentGroup])

{

int specialSpot = -1;

for(int k=0; k<skillGroupNames.length && specialSpot == -1; k++)

{

if(skillGroupNames[k].indexOf(skillNames[currentGroup]) != -1)

{

specialSpot = k;

}

}

outputString = outputString.append(skillGroupNames[specialSpot] + "-> I" + (j+1) + ", ");

}

}

}

outputString = outputString.deleteCharAt(outputString.length()-1);

outputString = outputString.deleteCharAt(outputString.length()-1);

outputString = outputString.append("}, ");

outputString = outputString.append("VertexLabeling -> True]");

outputString = outputString.append("\n");

}

String dataString = outputString.toString();

try

{

FileWriter file = new FileWriter("OutputMathematica" + File.separator +"Structures.txt", false);

file.write(dataString);

file.flush();

file.close();

}

catch(Exception e)

{

e.printStackTrace();

System.exit(0);

}

}

//

private void outputStructureMatlab(int outputNumber, int structures[][], int groups[])

{

StringBuilder outputString = new StringBuilder((INPUT\_COLUMNS \* INPUT\_COLUMNS) + 10000);

outputString = outputString.append("function dag = structure\_" + (outputNumber+1) + "\n");

outputString = outputString.append("% auto-generated code by Doug based on Zachs code" + "\n");

outputString = outputString.append("% Bayesian Prediction Analysis for Dynamic Learning Maps (Kansas)" + "\n");

outputString = outputString.append("% in collaboration with Neil Heffernan and colleagues (WPI)" + "\n");

outputString = outputString.append("% data property of Angela Broaddus (Kansas) and Neal Kingston (Kansas)" + "\n");

Vector <Integer> uniqueGroups = new Vector();

for(int i=0; i<groups.length; i++)

{

if(!uniqueGroups.contains(groups[i]))

{

uniqueGroups.add(groups[i]);

}

}

int nSize = uniqueGroups.size() + INPUT\_COLUMNS;

int nCount = 1;

outputString = outputString.append("N=" + nSize + ";\n" + "\n");

//create the group name by combining all skills in the group into one name

String skillGroupNames[] = new String[uniqueGroups.size()];

for(int i=0; i<uniqueGroups.size(); i++)

{

String name = "";

for(int j=0; j<groups.length; j++)

{

String skillName = skillNames[j];

if(groups[j] == uniqueGroups.get(i) && name.indexOf(skillName) == -1)

{

name = name.concat(skillName + "\_");

}

}

skillGroupNames[i] = name.substring(0, name.length()-1);

}

//output groups and questions

for(int i=0; i<skillGroupNames.length; i++)

{

outputString = outputString.append(skillGroupNames[i]);

outputString = outputString.append("=" + nCount + ";\n");

nCount++;

}

for(int i=0; i<INPUT\_COLUMNS; i++)

{

outputString = outputString.append("I" + (i+1) + "=" + nCount + ";\n");

nCount++;

}

outputString = outputString.append("\ndag=zeros(N,N); \n");

//output links

for(int i=0; i<uniqueGroups.size(); i++)

{

outputString = outputString.append("dag(");

int currentGroup = uniqueGroups.get(i);

int firstGroupMember = -1;

boolean found = false;

for(int j=0; j<groups.length && !found; j++)

{

if(groups[j] == currentGroup)

{

firstGroupMember = j;

found = true;

}

}

//get all unique groups that this current group points to

Vector <Integer> uniqueGroups2 = new Vector();

for(int j=0; j<groups.length; j++)

{

if(groups[firstGroupMember] != groups[j] && structures[firstGroupMember][j] == 1)

{

if(!uniqueGroups2.contains(groups[j]))

{

uniqueGroups2.add(groups[j]);

}

}

}

outputString = outputString.append(skillGroupNames[i] + ", [");

int count = 0;

for(int j=0; j<uniqueGroups2.size(); j++)

{

int groupNum = uniqueGroups2.get(j);

//find skill group name that has skill name of the group number

int specialSpot = -1;

for(int k=0; k<skillGroupNames.length && specialSpot == -1; k++)

{

if(skillGroupNames[k].indexOf(skillNames[groupNum]) != -1)

{

specialSpot = k;

}

}

outputString = outputString.append(skillGroupNames[specialSpot] + " ");

count++;

}

if(uniqueGroups.size() == 1)

{

outputString = outputString.append(skillGroupNames[0] + ", [");

}

//attach questions too

for(int j=0; j<groups.length; j++)

{

if(groups[j] == groups[currentGroup])

{

outputString = outputString.append("I" + (j+1) + " ");

count++;

}

}

outputString = outputString.delete(outputString.length()-1, outputString.length());

outputString = outputString.append("]) = 1;" + "\n");

if(count <= 1)

{

int indexOfOpen = outputString.lastIndexOf("[");

int indexOfClosed = outputString.lastIndexOf("]") - 1;

outputString = outputString.deleteCharAt(indexOfOpen);

outputString = outputString.deleteCharAt(indexOfClosed);

}

}

String dataString = outputString.toString();

try

{

FileWriter file = new FileWriter("OutputMatLab" + File.separator +"OutputStructure" + "\_" + uniqueGroups.size() + "\_" + (outputNumber+1) + ".m", false);

file.write(dataString);

file.flush();

file.close();

}

catch(Exception e)

{

e.printStackTrace();

System.exit(0);

}

}

public static void main(String args[])

{

if(args.length < 2)

{

System.out.println("Must enter an input structure files");

}

LearnGraph lg = new LearnGraph(args[0], args[1]);

}

}