Learning Objects Based Adaptive Textbooks with Dynamic Traversal for Quantum Cryptography

V. Bommanapally, M. Subramaniam, A. Parakh, P. Chundi, and V. Puppala

University of Nebraska, Omaha NE 68182, USA
{vbommanapally, msubramaniam, aparakh, pchundi, vpuppala}@unomaha.edu

Abstract. The theory of learning objects allows one to create educational entities that follow an object-oriented paradigm and can be adapted efficiently for rapid creation of courses, training modules as well as textbooks. In this paper, we introduce one such repository of learning objects for the field of quantum cryptography. These learning objects are leveraged into automatic generation of personalized and adaptive textbooks built on user preferences, learning styles, and desired learning outcomes. The textbooks are generated using the highly customizable platform of Jupyter notebooks using several digital assets, coding environments, interactive visualizations, and self-graded quizzes and tests. The presented approach is general and can be easily adopted for the development of adaptive textbooks for other disciplines.

Keywords: Learning object repositories · adaptive textbooks · quantum cryptography · cybersecurity

1 Introduction

With the potential arrival of quantum computers that can compromise cybersecurity infrastructure, interest in quantum cryptography education is rapidly increasing across institutions. The interdisciplinary nature of this area coupled with the lack of adequate course textbooks and time constraints make it challenging for instructors to design courses that meet the learning outcomes required by the various cybersecurity certifying agencies. Consequently, instructors usually resort to manually synthesizing heterogeneous, brittle, lesson plans from a variety of textbooks, which are often too rigid to accommodate students with diverse learning preferences and backgrounds and not amenable to effectively track the required learning outcomes.

Serious games and cybersecurity learning object repositories (LORs) [2, 16, 21, 7, 15] have been recently developed to address some of the above limitations of traditional pedagogical models for quantum cryptography education. Serious game based learning approaches gamify only a narrow set of topics compared to the high cost of building these games spanning an entire subject area. While the LORs [2] cover a vast number of cybersecurity concepts in their learning objects and relate them to the learning outcomes specified by several certifying agencies, they provide limited support to automatically generate textbooks of learning objects for a particular course. Often, instructors have to search/browse these repositories to manually collate the materials and organize them into a textbook based on associated outcome and concept dependencies.

This paper presents a quantum cryptography learning object repository (QCL) that we have developed and deployed in an LOR called Clark [2] and describe an automated approach to automatically generate adaptive quantum cryptography textbooks of learning objects (QCTs) from this repository. The quantum cryptography learning objects are designed using Python Jupyter notebooks and include several digital assets, coding environments, interactive visualizations, and self-graded quizzes and tests. Our approach for generating QCTs is inspired by the earlier rule-based frameworks [14, 23]. Given set of learning outcomes, student preferences and backgrounds, a QCT consisting of a set of instantiated learning objects is automatically generated such that the successful completion QCT satisfies the input (desired) learning outcomes. To the best of our knowledge, our work is the first to apply rule-based approaches to learning objects to generate adaptive textbooks for quantum cryptography. Further, the generated QCTs are versatile in providing a wide range of coverage similar to LORs as well as provide dynamic learner interaction similar to serious games due to the underlying Jupyter notebook features.

Achieving the desired learning outcomes using a large QCT without additional assistance can be challenging. Since QCTs are outcome-driven, they also include an evaluation engine to assess student performance with respect to a given outcome. In order to guide a student through the generated QCT in an efficient and challenging manner, we present a novel QCT traversal algorithm that uses student performance at each step to choose the next set of learning objects so that the desired learning outcomes can be achieved. The proposed approach has been used to synthesize various QCTs ranging from basic quantum computing fundamentals about tensor products to the entanglement based quantum key exchange protocol, E91.

The rest of the paper is organized as follows. We briefly describe related works, next. Section 2 describes the quantum cryptography learning object repository, QCL. The QCT synthesis framework is described in section 3. Section 4 describes the QCT traversal algorithm and its application to a QCT covering a non-trivial case study of tensor products used for multi-qubit quantum systems. Section 6 concludes the paper along with possible future work.

1.1 Related Work

Adaptive web-based textbook generation has a long and rich history in e-learning. Rule-based frameworks have been extensively investigated by several earlier works to automatically synthesize course books [14, 23] for a given set of learning concepts, scenarios, and pedagogical goals. Large learning object repositories parameterized by concepts and learner preferences and backgrounds have been developed by several works [1, 2, 6, 3, 17] for computer science education. Informally, a generative learning object denotes a family of related LO instances that can be automatically generated on demand by instantiating the associated metadata parameters. While the learning object review instrument [22] to evaluate these repositories includes adaptation as one of its nine criteria, they provide limited support ([6] is an exception) for adaptivity as compared to the rule-based frameworks.

Our learning repository QCL is a part of a recently developed LOR [2] where learning outcomes are cross-indexed with those of several well-known cybersecurity education certifying programs. While our QCT generation framework is largely inspired from earlier rule-based approaches, our proposed approach, uses learning outcomes (in addition to preferences, concepts, and backgrounds) to generate QCTs. Using learning outcomes to synthesize textbooks requires QCL to be designed to include measurable assessment instruments for every learning object in QCL. We also develop an outcome-driven algorithm to traverse a QCT in an efficient manner. To the best of our knowledge this is the first learning object repository for quantum cryptography. QCL is also unique in CLARK in using Python Jupyter notebooks to support interactive code, visualizations, and self-graded quizzes and exams.

2 Quantum Cryptography Learning Object Repository: QCL

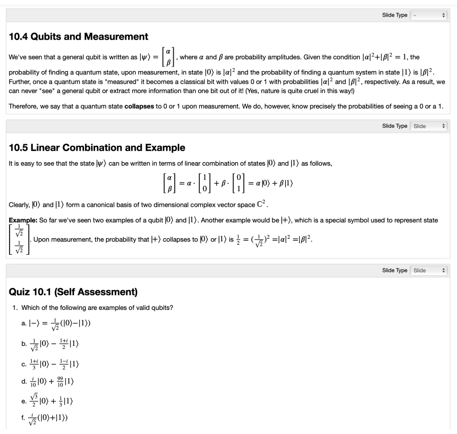
2.1 Clark

Clark is a digital library of learning objects developed mainly through funding from the National Security Agency and the National Science Foundation. It consists of learning objects that have been contributed from over fifty universities across the United States on various topics in cybersecurity. A collection of learning objects, developed by some of the authors of this paper, on Quantum Computing and Cryptography is available on Clark for free download.

Clark organizes the learning objects into five classes, based on completion times, called nanomodules, micromodules, modules, units and courses; units, being over 10 hours in length, often consist of collection of smaller modules and a course consists of multiple units. Each class of learning object is accompanied with learning outcomes that tell the user what they will achieve upon the successful completion of the modules and exercises therein. These learning outcomes are mapped to various cognitive levels of Bloom's taxonomy thereby providing a learner an idea of the organization of the learning material and its complexity and emphasis. Clark also maps various modules to different frameworks such as CAE-CD and CAE-CO [5], NICE cybersecurity workforce framework [4], etc. delineating the relationship of modules to knowledge units in those frameworks.

The Quantum Computing and Cryptography learning objects, in particular, are written in highly customizable Jupyter notebooks. Jupyter notebooks have become the defacto standard for quantum programming as they allow for an easy integration of mathematics, code as well as visual objects that can then interface to external APIs (such as the IBM simulator or quantum computer) with minimal effort. Python was the chosen language for the development of these notebooks since Python is a popular language for quantum programming and well as the cybersecurity domain. While each Jupyter notebook themselves supports two types of cells - *markdown and* code, we further logically organize these cells into various categories. Therefore, each Jupyter notebook cell contains one of the following categories: textual explanations of topics, examples, sample code, (interactive) simulations, (interactive, self-graded) exercises, quizzes, etc. The notebooks are accompanied with final quizzes (F-Q) that are intended to be used by instructors in a graded setting. In total, there are 28 notebooks at present that cover the basic review of linear algebra, basic quantum concepts and quantum cryptography protocols forming three units.

Figure 1 shows various categories of cells available in Jupyter notebooks in QCL on Clark.


Slide Type -

10.4 Qubits and Measurement

We've seen that a general qubit is written as $|\psi\rangle = \begin{bmatrix}\alpha \\ \beta\end{bmatrix}$, where α and β are probability amplitudes. Given the condition $|\alpha|^2+|\beta|^2 = 1$, the probability of finding a quantum state, upon measurement, in state $|0\rangle$ is $|\alpha|^2$ and the probability of finding a quantum system in state $|1\rangle$ is $|\beta|^2$. Further, once a quantum state is "measured" it becomes a classical bit with values 0 or 1 with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. As a result, we can never "see" a general qubit or extract more information than one bit out of it! (Yes, nature is quite cruel in this way!)

Therefore, we say that a quantum state **collapses** to 0 or 1 upon measurement. We do, however, know precisely the probabilities of seeing a 0 or a 1.

Slide Type Slide

10.5 Linear Combination and Example

It is easy to see that the state $|\psi\rangle$ can be written in terms of linear combination of states $|0\rangle$ and $|1\rangle$ as follows,

$$\begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \alpha \cdot \begin{bmatrix}1 \\ 0\end{bmatrix} + \beta \cdot \begin{bmatrix}0 \\ 1\end{bmatrix} = \alpha|0\rangle + \beta|1\rangle$$

Clearly, $|0\rangle$ and $|1\rangle$ form a canonical basis of two dimensional complex vector space $\mathbb{C}^2$.

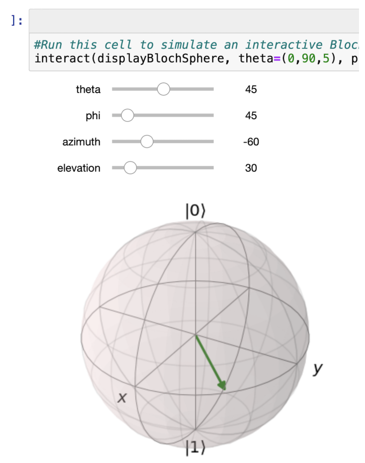
Example: So far we've seen two examples of a qubit $|0\rangle$ and $|1\rangle$. Another example would be $|+\rangle$, which is a special symbol used to represent state $\begin{bmatrix}\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}}\end{bmatrix}$. Upon measurement, the probability that $|+\rangle$ collapses to $|0\rangle$ or $|1\rangle$ is $\frac{1}{2} = (\frac{1}{\sqrt{2}})^2 = |\alpha|^2 = |\beta|^2$.

Slide Type Slide

Quiz 10.1 (Self Assessment)

1. Which of the following are examples of valid qubits?
   a. $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
   b. $\frac{1}{\sqrt{2}}|0\rangle - \frac{1+i}{2}|1\rangle$
   c. $\frac{1+i}{3}|0\rangle - \frac{1-i}{2}|1\rangle$
   d. $\frac{i}{10}|0\rangle + \frac{99}{10}|1\rangle$
   e. $\frac{\sqrt{3}}{2}|0\rangle + \frac{1}{3}|1\rangle$
   f. $\frac{i}{\sqrt{2}}(|0\rangle + |1\rangle)$

(a) Static cells with definitions, examples and self-assessment quizzes.


```
]:
#Run this cell to simulate an interactive Bloc
interact(displayBlochSphere, theta=(0,90,5), p
```

theta 45
phi 45
azimuth -60
elevation 30

(b) Interactive cell with simulation of Bloch sphere.

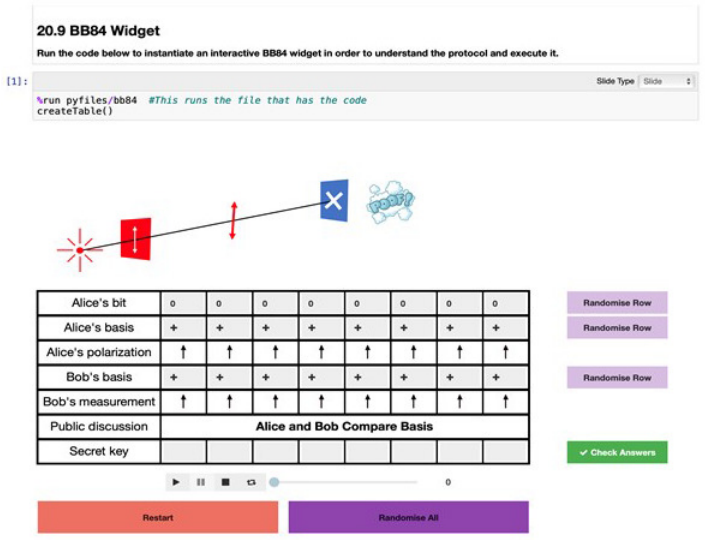
Fig. 1: Illustration of various categories of cells available in the Jupyter notebooks for QCL.

3 Generating QCTs Guided by Outcomes, Preferences

This section describes our procedure for generating quantum cryptography textbooks, QCTs, using the learning object repository QCL for a given set of learning outcomes, student preferences, and backgrounds.

3.1 Instrumenting QCL

The modules of repository QCL are available in some random order on the Clark [2] website. In order to use QCL, students usually have to perform a keyword search to identify modules of interest and go through them in some chosen order. The QCT generation algorithm, described here, aids the user in automatically collecting relevant materials from the QCL and present them in a coherent manner aimed at achieving the desired learning outcomes. To generate QCTs, all the cells in the QCL are instrumented with data obtained from domain experts describing the relations of each cell to other objects, concepts, and outcomes along with the digital assets included in the cell.


20.9 BB84 Widget

Run the code below to instantiate an interactive BB84 widget in order to understand the protocol and execute it.

```
[1]:
%run pyfiles/bb84  #This runs the file that has the code
createTable()
```

Alice's bit	0	0	0	0	0	0	0	0
Alice's basis	+	+	+	+	+	+	+	+
Alice's polarization	↑	↑	↑	↑	↑	↑	↑	↑
Bob's basis	+	+	+	+	+	+	+	+
Bob's measurement	↑	↑	↑	↑	↑	↑	↑	↑
Public discussion	Alice and Bob Compare Basis							
Secret key								

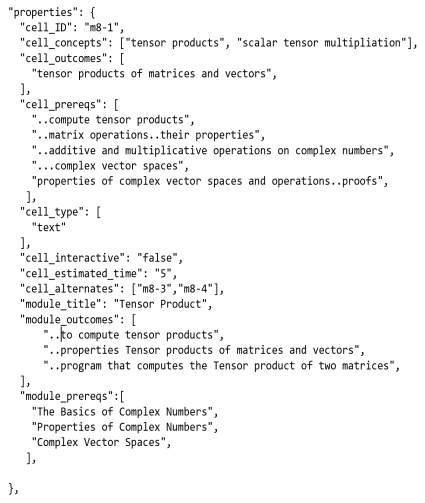
Fig. 2: Interactive simulation of BB84 protocol.

The metadata in each notebook cell of the QCL consists of*local* and *global* data members related to that cell and the module containing that cell, respectively. The local data members include information such as the `cell-id`, `cell-concepts`, `cell-outcomes`, and `cell-prerequisites`, i.e., outcomes that must be met in order to use this cell. The local data `cell-type`, can be one of – *text, code, image, symbolic example, numeric example, video, widgets, quiz, auto-graded-simulation, auto-graded-quiz, code-IDE, code-IDE-wtests*. Users can write Python programs and test them using a given test suite in cells of type *code-IDE*, and *code-IDE-wtests*. The *numeric example, auto-graded-quiz, widget, code-IDE, code-IDE-wtests* cells allow users to interact with a cell by modifying the cell contents whereas *text, code, symbolic example, image, quiz*, and *video* cells allow more limited forms of user interaction.

For example, figure (1b) depicts a Bloch sphere *widget* cell where users can modify values for parameters `theta, phi, azimuth,` and `elevation` using sliding scales to generate different qubit states visualized in the sphere. Figure 2 depicts an *auto-graded-simulation* cell of the BB84 quantum key exchange protocol where users can modify qubit values and orthonormal bases to orient, measure qubits and enter the resulting secret key answer to be checked for correctness.

The Boolean valued local data `cell-interactive` denotes whether a cell is interactive or not. A cell has to be interactive to be used to assess a learning outcome. For instance, cell depicted in 2 with the auto-grader is used in QCL to determine the outcome about users having a basic understanding of the BB84 protocol. The *auto-graded quizzes, numeric-examples, and code-IDE-wtests* cells in the QCL are used to determine the learning outcomes of the modules containing these cells.

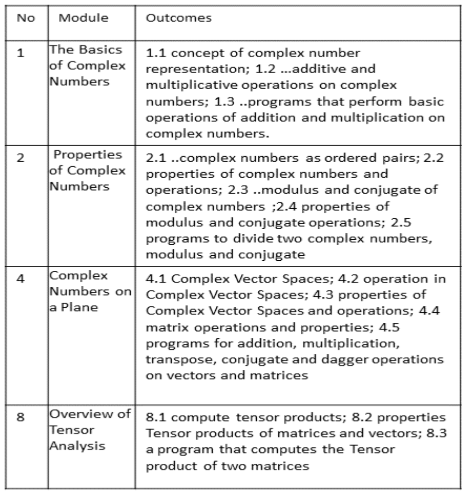
The local data `cell-alternates` provides a list of alternate cells that are semantically equivalent to the current cell but with cell types different than the current cell. Alternate cells are used to adapt QCTs to student preferences and also for suggesting next cells based on student performance as described in the next section.



```
"properties": {
  "cell_ID": "m8-1",
  "cell_concepts": ["tensor products", "scalar tensor multipliation"],
  "cell_outcomes": [
    "tensor products of matrices and vectors",
  ],
  "cell_prereqs": [
    "..compute tensor products",
    "..matrix operations..their properties",
    "..additive and multiplicative operations on complex numbers",
    "...complex vector spaces",
    "properties of complex vector spaces and operations..proofs",
   ],
  "cell_type": [
    "text"
  ],
  "cell_interactive": "false",
  "cell_estimated_time": "5",
  "cell_alternates": ["m8-3","m8-4"],
  "module_title": "Tensor Product",
  "module_outcomes": [
      "..to compute tensor products",
      "..properties Tensor products of matrices and vectors",
      "..program that computes the Tensor product of two matrices",
  ],
  "module_prereqs":[
    "The Basics of Complex Numbers",
    "Properties of Complex Numbers",
    "Complex Vector Spaces",
   ],

},
```

(a)



No	Module	Outcomes
1	The Basics of Complex Numbers	1.1 concept of complex number representation; 1.2 ...additive and multiplicative operations on complex numbers; 1.3 ..programs that perform basic operations of addition and multiplication on complex numbers.
2	Properties of Complex Numbers	2.1 ..complex numbers as ordered pairs; 2.2 properties of complex numbers and operations; 2.3 ..modulus and conjugate of complex numbers ;2.4 properties of modulus and conjugate operations; 2.5 programs to divide two complex numbers, modulus and conjugate
4	Complex Numbers on a Plane	4.1 Complex Vector Spaces; 4.2 operation in Complex Vector Spaces; 4.3 properties of Complex Vector Spaces and operations; 4.4 matrix operations and properties; 4.5 programs for addition, multiplication, transpose, conjugate and dagger operations on vectors and matrices
8	Overview of Tensor Analysis	8.1 compute tensor products; 8.2 properties Tensor products of matrices and vectors; 8.3 a program that computes the Tensor product of two matrices

(b)

Fig. 3: (a) Snapshot of a module cell metadata. (b) Modules and corresponding outcomes.

The global data members of a cell include the containing `module-title, module-outcomes`, and `module-prerequisites` i.e., other modules whose outcomes must be achieved in order to achieve the outcomes of the current module.

As an example, the figure (3a) depicts the data members with the first cell in the module 8 of QCL. The figure (3b) shows other QCL modules, viz. 1, 2, and 4, that are related to the module 8 along with the outcomes of the modules.

3.2 Generating QCT from Instrumented QCL

The main steps to generate a QCT from the instrumented QCL are the following.

- **QCL Instrumentation Pre-processing**: The integrity of the QCL instrumentation is verified using a variety of lightweight properties such as: a) all global data in all cells in a module are identical, b) outcomes of all the cells of a module belong to the outcomes of that module, c) type checking of cells along with their alternates. Next, module and outcome dependency graphs are built from the cell data. Lastly, these graphs are verified using properties such as: a) absence of dangling edges and cycles, b) module dependencies are a projection of outcome dependencies and that c) there exist one or more auto-graded quiz cells (**FQ**) in each module that interactively evaluate all the outcomes that module. The semantically verified cells and the graphs are used to generate QCTs subsequently. Note that the verification is a one time activity that needs to be performed only when QCL repository is updated.
- **Identify QCL Learning Objects**: Given a set of outcomes, the QCT mode, and a ranked list (of student preferences whose elements are cell types), a forest of relevant QCL learning objects are identified and used to generate the corresponding QCT. The building blocks of a QCT can either be QCL modules or QCL cells and this is specified by the QCT mode input. To build a QCT based on QCL modules, modules whose outcomes include a given input outcome are identified. A reverse topological sort of the module dependency graph starting with each of these modules (with ties among the modules being broken arbitrarily) produces a linear chain of modules which are then merged while obeying module dependencies to create a single chain of QCL modules.
- **Generating a QCT**: The linear chain of modules obtained from the previous step are then organized into a hierarchy of nano-, micro- modules, units, and courses based on the estimated times to output a QCT whose successful completion is guaranteed to satisfy the input outcomes. In order to incorporate the student preferences in a QCT, for each cell in the linear chain, the cell is retained if it matches the highest ranked input student preference; otherwise we replace it by a better matching alternate cell, if one exists. Certain modules are pruned from the QCT based on student background.

  A QCT can also be built from QCL using individual cells instead of entire modules using a similar procedure. In this case, the outcome dependency graph is used instead of the module dependency graph. This often, leads to more focused QCTs with lesser cells taking lesser instruction time. Student background is used to prune cells as well as modules in this case as well.

3.3 An Example QCT: Tensor Products

Primarily, tensor products are used to develop a vector representation for quantum systems that consist of multiple quantum bits and gates. They are also useful tools in determining whether two or more quantum systems are in an entangled state or not. Since any computing system that does useful work typically works on multiple qubits at any given time, tensor products are central to the study of quantum theory. Because of their importance, we have chosen module 8, Overview of Tensor Analysis, as the running example. Consider two outcomes from this module - "8.2 Compute Tensor products" and "8.3 Implement program that computes Tensor product of two matrices". These outcomes have dependencies in QCL modules 1, 2, and 4 (see figure 3b). These four modules are depicted in the figure (4a) along with their outcomes. To generate QCTs involving these modules, first they are instrumented and sanity checks mentioned above were performed. Next, the module and outcome dependency graphs, depicted in figure (4a) and figure (4b) were generated and statically verified for the properties described above to obtain well-formed QCLs and graphs.

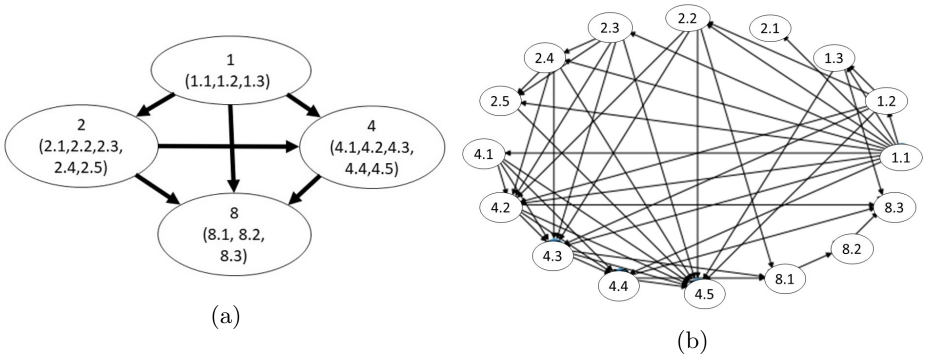
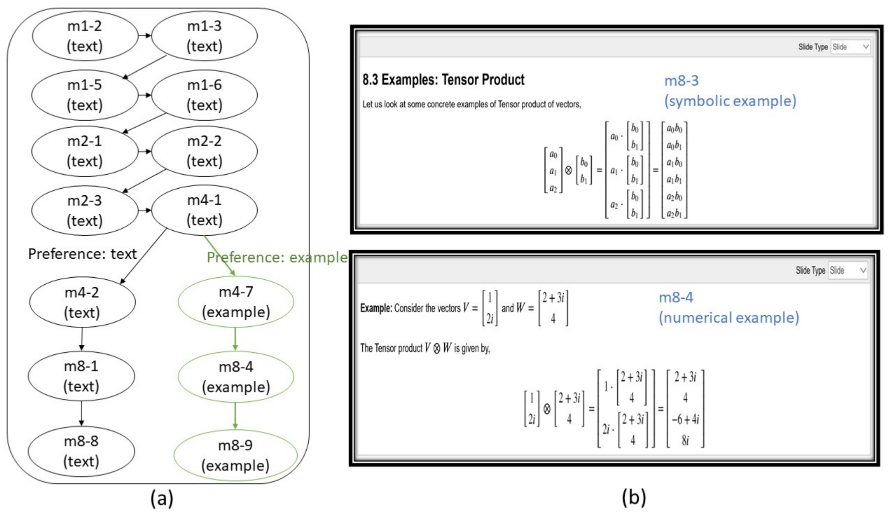


Fig. 4: (a) Module dependency graph, (b) Outcome dependency graph for outcome 8.2. at cell level detail. (Note that all the dependencies including transitive dependencies are shown in the graph for clarity.)

Inputs consisting of outcome 8.2, QCT mode specifying modules as building blocks, *text* cell-type as the highest preference and a novice user background are used to generate a QCT that includes all of the cells of QCL module 8 along with those of the QCL modules 1, 2, and 4 on which module 8 depends. The reverse topological sorting of the module dependency creates a QCT with a linear chain of modules beginning with 12 cells of module 1, followed by 10 cells in module 2, followed by 18 cells of module 4, and ending with 11 cells of module 8. Since it is the case that either all these cells satisfy the highest preference or there exist no alternate cells that do, none of the cells in the QCT is replaced. Finally, the generated QCT is partitioned into two nanomodules (1 and 2), and 2 micromodules (4 and 8) based on estimated times.



8.3 Examples: Tensor Product

Let us look at some concrete examples of Tensor product of vectors,

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \otimes \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} a_0 \cdot \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \\ a_1 \cdot \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \\ a_2 \cdot \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \\ a_2 b_0 \\ a_2 b_1 \end{bmatrix}$$

Example: Consider the vectors $V = \begin{bmatrix} 1 \\ 2i \end{bmatrix}$ and $W = \begin{bmatrix} 2+3i \\ 4 \end{bmatrix}$

The Tensor product $V \otimes W$ is given by,

$$\begin{bmatrix} 1 \\ 2i \end{bmatrix} \otimes \begin{bmatrix} 2+3i \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot \begin{bmatrix} 2+3i \\ 4 \end{bmatrix} \\ 2i \cdot \begin{bmatrix} 2+3i \\ 4 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 2+3i \\ 4 \\ -6+4i \\ 8i \end{bmatrix}$$

Fig. 5: (a) Cell path with learner preference “text” and outcome “8.2” arrows showing possible alternate cells for cell “m8-1”, (b) Cell path for learner preference “example”

Next, consider generating a QCT using the same inputs as above, but changing the QCT mode to use QCL cells instead of QCL modules as building blocks of QCT. The outcome dependency graph in figure (4b) is used in this case to generate the QCT consisting of only 11 QCL cells across the four QCL modules 1, 2, 4, and 8, which is shown in figure (5a). We can also generate a QCT by changing the input to specify highest preference cell-type to be a *numerical-example* instead of *text* and this will lead to the replacement cell “m8-1” by cell “m8-4” shown in figure (5b) which are the alternate cells of “m8-1”. Note, here m8-1 is cell number 1 of module 8. If we were to provide the preference [numerical-example > example > text], then several alternate cells will be used to generate the final QCT. The resulting linear chain in the QCT is shown in the figure (5a) with right branch showing preference as example. These cells are organized based on their estimated time into a single nanomodule.

Note that building a QCT for outcome 8.2 using QCL modules leads to QCT with 50 cells whereas we need only 11 cells if we use QCL cells to generate the QCT, which is a saving of 39 cells.

4 Dynamic QCT Traversal Using Student Performance

Achieving the desired learning outcomes by serially following a QCT that is constructed solely based on dependencies can be challenging. For example, it is conceivable that students may want to study a concept in different contexts to achieve the required outcome; this may also involve attempting to achieve a specified outcome in conjunction with others. For example, an outcome “understand the basics of a quantum protocol" such as E91 may be achieved in conjunction with an outcome that requires to create a functional prototype of this protocol. Consequently, in this section, we present a novel QCT traversal algorithm that uses student performance at each outcome assessment step to choose the next set of learning objects such that the desired outcomes can be achieved in an efficient manner.

The proposed traversal algorithm works on a *quiz dependency graph*, Q, whose nodes are QCL auto-graded quiz cells, (recall from section 3.2 that these cells (**FQ**), taken together assess all the outcomes of the module containing them). The information at each node c_i of Q, $R(c_i)$, is the set of outcomes that are assessed by that node. Each node c_i, in Q, is assigned a node weight, v_i as follows: $v_i = \langle \frac{1}{|C(o_j)|}, o_j \; \epsilon \; R(c_i) \rangle$. Here $|C(o_j)|$ is the number of nodes in Q that contain an *unmet* outcome o_j for all o_j that are in node c_i.

There is a labeled edge $(c_i,\ c_j,\ w_{i,j})$ in Q from node c_i to c_j with weight $w_{i,j}$ if some outcome in $R(c_j)$ depends on an outcome in $R(c_i)$. Let $NR_{j,i}$ be the number of outcomes common to c_i and c_j and $N_{j,i}$ be the number of outcomes in c_j that depend on some outcome in c_i. The *dependency overlap* between node c_i and c_j is, $D(c_i, c_j) = N_{ji} - \delta * NR_{ji}$; where $\delta = 10^{-ceil(\log(N))}$, is the redundancy factor and N is the number of outcomes in Q.

Given node weights and the dependency overlap, the edge weight is given by, $w_{i,j} = \left(\frac{D(c_i,c_j)}{|R_{cj}|} * v_j \right)$.

A quiz cell dependency graph involving outcomes from the QCL modules 1, 2, 4, and 8 discussed earlier is depicted in figure 6.

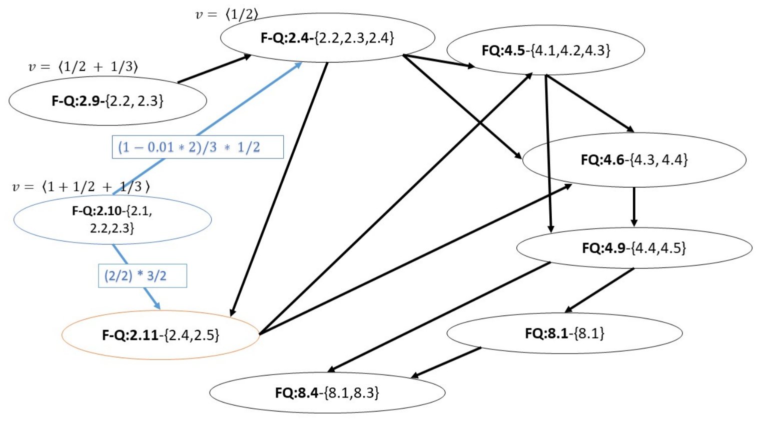


Fig. 6: Quiz cell dependency graph

4.1 Adaptive QCT Traversal Algorithm

The adaptive traversal through quiz cell dependency graph is achieved using the traversal algorithm given below. The proposed algorithm is presented for cell level navigation. However, it can be easily adapted to the case where student prefers module level traversal of the QCT.

Algorithm
Inputs: Textbook QCT, Quiz Dependency Graph (Q)
Steps:

1. Pick next node c_i. If there are one or more unvisited source nodes, then choose one with the maximum node weight. Otherwise, pick the next node that has the maximum edge weight from a visited node.
2. Generate a new QCT with target outcomes $R(c_i)$ from the input QCT.
3. Assess outcomes $R(c_i)$ using the quiz cell corresponding to c_i.
4. If some of outcomes in $R(c_i)$ are achieved, update Q by re-calculating the node and edge weights. Node weights are modified as the number of unmet outcomes change and consequently edge weights change as well. Remove nodes with 0 node weight, edges among nodes with independent or disjoint outcomes. Go to step 1 and repeat if Q is not empty.
5. If one or more outcomes $R(c_i)$ are not met, go to step 2.

The above algorithm traverses a given QCT by starting with quizzes which cover unmet outcomes that do not have any prerequisites. If there is a choice among such quizzes then those covering the largest number of unmet outcomes is chosen. The student is given a new QCT with cells related to the outcomes in the chosen quiz and their performance is evaluated on the quiz after going through the new QCT. If the student succeeds in the quiz, the graph is updated to reflect the outcomes that are already met. Then, the next quiz is chosen based on maximum edge weight, which picks a quiz with least redundancy in terms of satisfied outcomes among all candidate quizzes and has the most number of prerequisites satisfied among all the candidate quizzes. Note that all prerequisites must be met for every candidate quiz.

Example: Consider the quiz dependency graph Q in figure 6 involving the auto-graded quizzes from the QCL modules 1, 2, 4, and 8. Each node in the graph represents a quiz and its associated outcomes. The node FQ: 2.10 represents question number 10 from the final quiz for module 2 and so on. The node weights and edge weights are calculated. The initial node weights of two base nodes FQ: 2.9 and FQ: 2.10 are 5/6 and 11/6 respectively. Hence FQ: 2.10 is chosen initially being node with maximum edge weight. After the student gains knowledge in the QCT generated, the outcome is measured by providing the quiz FQ: 2.10. Once the quiz is solved successfully, all the node weights are re-calculated and the outcomes are marked met in the other available quiz cell. Hence the value of node FQ: 2.9 is 0 once the student achieves the outcomes from FQ: 2.10 similarly, value of node FQ: 2.4 is 1/2. Next, the edge weights of the adjacent nodes are calculated as shown on the edges from FQ: 2.10 in the above figure.

The edge with maximum weight i.e, FQ: 2.11 is chosen as next best quiz for the student. Accordingly, the related QCT is generated from QCL based on the outcomes and student preferences. The procedure is repeated until the student successfully achieves all the target outcomes.

5 Conclusion

In this paper, we presented a new method to generate adaptive quantum cryptography textbooks (QCT) from a repository of quantum cryptography learning objects. The textbook generation is customized based on the learning outcomes, student preferences and the background of the student. Each learning object is instrumented with metadata that is used to generate new textbooks. A case study shows that the outcome level dependency reduces the overhead of cells required to achieve a learning outcome compared to module level dependency. We also introduce an adaptive QCT traversal algorithm to further guide the students through the generated QCTs based on their performance on quizzes. This helps the student achieve the desired outcomes efficiently.

In future work, we will extend our work with reinforcement learning techniques and automate the generation of QCTs. Further, QCTs generated will be supplemented with a query engine for each lookup of learning objects for a just-in-time learning process.

5.1 Acknowledgment

This project was partly funded through National Science Foundation (NSF) award 1623380.

References

1. About Learning Objects, https://www.wisc-online.com/about-learning-objects
2. Cybersecurity library, https://www.clark.center/home
3. MERLOT, https://www.merlot.org/merlot/
4. Nice cybersecurity workforce framework, https://www.nist.gov/itl/applied-cybersecurity/nice/nice-cybersecurity-workforce-framework-resource-center
5. NSA Center for Academic Excellence - Cyber Defence and Cyber Operations Criteria, https://www.nsa.gov/resources/students-educators/centers-academic-excellence/
6. Model-driven processes and tools to design robot-based generative learning objects for computer science education. Science of Computer Programming **129**, 48 – 71 (2016), special issue on eLearning Software Architectures.
7. Abeyrathna, D., Vadla, S., Bommanapally, V., Subramaniam, M., Chundi, P., Parakh, A.: Analyzing and predicting player performance in a quantum cryptography serious game. In: International Conference on Games and Learning Alliance. pp. 267–276. Springer (2018)
8. Al-Chalabi, H., HUSSEIN, A.: Ontologies and personalization parameters in adaptive e-learning systems: Review. Journal of Applied Computer Science Mathematics **14**, 14–19 (01 2020)

9. Bommanapally, V., Subramaniam, M., Chundi, P., Parakh, A.: Navigation hints in serious games. iLRN 2018 Montana p. 115 (2018)
10. Brondani, S.C.P., Santos, T.O., Witt, R.R., Silveira, D.T., Ferla, A.A.: Nursing care: use of digital learning objects with chronic illnesses. NI 2012 : 11th International Congress on Nursing Informatics, June 23-27, 2012 **2012**, 47 (2012)
11. Burbaite, R., Bespalova, K., Damasevicius, R., Stuikys, V.: Context-aware generative learning objects for teaching computer science. International Journal of Engineering Education **30**, 929–936 (01 2014)
12. Chen, P., Lu, Y., Zheng, V.W., Chen, X., Yang, B.: Knowedu: A system to construct knowledge graph for education. IEEE Access **6**, 31553–31563 (2018)
13. Costea, F., Chirila, C., Creţu, V.: Designing e-learning content using aglos. In: 2019 23rd International Conference on System Theory, Control and Computing (ICSTCC). pp. 685–690 (2019)
14. Melis, E., Andres, E., Budenbender, J., Frischauf, A., Goduadze, G., Libbrecht, P., Pollet, M., Ullrich, C.: ActiveMath: A Generic and Adaptive Web-Based Learning Environment. International Journal of Artificial Intelligence in Education (IJAIED) **12**, 385–407 (2001)
15. Parakh, A., Chundi, P., Subramaniam, M.: An approach towards designing problem networks in serious games. In: IEEE Conference on Games (CoG). pp. 1–8 (2019)
16. Parakh, A., Subramaniam, M., Ostler, E.: Quasim: A virtual quantum cryptography educator. In: 2017 IEEE International Conference on Electro Information Technology (EIT). pp. 600–605 (May 2017)
17. Rossano, V., Joy, M., Roselli, T., Sutinen, E.: A taxonomy for definitions and applications of los: A meta-analysis of icalt papers. Educational Technology Society **8**, 148–160 (01 2005)
18. Salhi, I., El Fazazi, H., Qbadou, M., Mansouri, K.: Towards a semantic annotation of pedagogical unstructured documents. In: Embedded Systems and Artificial Intelligence. pp. 623–633. Springer Singapore, Singapore (2020)
19. Stuikys, V., Burbaite, R., Bespalova, K., Ziberkas, G.: Model-driven processes and tools to design robot-based generative learning objects for computer science education. Sci. Comput. Program. **129**, 48–71 (2016)
20. Štuikys, V., Damaševičius, R.: Development of generative learning objects using feature diagrams and generative techniques. Informatics in education **7**, 277–288 (2008)
21. Vadla, S., Parakh, A., Chundi, P., Surbamaniam, M.: Quasim: A multi-dimensional quantum cryptography game for cyber security. In: Journal of The Colloquium for Information System Security Education. vol. 6, pp. 19–19 (2019)
22. Vargo, J., Nesbit, J., Belfer, K., Archambault, A.: Learning object evaluation: Computer-mediated collaboration and inter-rater reliability. International Journal of Computers and Applications **25**(3), 198–205 (2003)
23. Weber, G., Brusilovsky, P.: Elm-art: An adaptive versatile system for web-based instruction. International Journal of Artificial Intelligence in Education **12** (01 2001)