Modeling Skill Combination Patterns for Deeper Knowledge Tracing

Yun Huang
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA, USA
yuh43@pitt.edu

Julio D. Guerra-Hollstein
School of Information Sciences
University of Pittsburgh
Pittsburgh, PA, USA
Instituto de Informática,
Universidad Austral de Chile
Valdivia, Chile
jdg60@pitt.edu

Peter Brusilovsky
School of Information Sciences & Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA, USA
peterb@pitt.edu

ABSTRACT

This paper explores the problem of modeling student knowledge in complex learning activities where multiple skills are required at the same time, such as in the programming domain. In such cases, it is not clear how the evidence of student performance translates to individual skills. As a result, traditional approaches to knowledge modeling, such as Knowledge Tracing (KT), which traces students' knowledge of each decomposed individual skill, might fall short. We argue that skill combinations might carry extra specific knowledge, and mastery should be asserted only when a student can fluently apply skills in combination with other skills in different contexts. We propose a data-driven framework to model skill combination patterns for tracing students' deeper knowledge. We automatically identify significant skill combinations from data and construct a conjunctive knowledge model with a hierarchical skill representation based on a Bayesian Network. We also propose a novel evaluation framework primarily focuses on the knowledge inference quality, since we argue that traditional prediction metrics no longer suffice to differentiate between shallow and deep knowledge modeling. Our experiments on datasets collected from two programming learning systems show that proposed model significantly increases mastery inference accuracy and tends to more reasonably distribute students' efforts comparing with traditional KT models and its non-hierarchical counterparts. Our work serves as a first step towards building skill application context sensitive model for modeling students' deep, robust learning.

Keywords

complex skill, multiple skill, composition effect, Knowledge Tracing, Bayesian Network, robust learning, deep learning

1. INTRODUCTION

Knowledge Tracing (KT) [6] established itself as an efficient approach to model student skill acquisition in intelligent tutoring systems. The essence of this approach is to decompose domain knowledge into elementary skills and map student problem-solving performance into student knowledge level for each of the skills. Knowledge Tracing demonstrated its ability to track student knowledge for different domains and could be now considered as the most popular learner modeling approach. However, a known limitation of Knowledge Tracing is the assumption of skill independence in problems that involve multiple (complex) skills. Recent research, however, challenged this assumption demonstrating that there is additional knowledge related to specific skill combinations, in other words, the knowledge about a set of skills is more than the "sum" of the knowledge of individual skills [13], some skill must be integrated (or connected) with other skills to produce behavior [20]. For example, students were found significantly worse at translating two-step algebra story problems into expressions (e.g., 800-40x) than they were at translating two closely matched one-step problems (with answers 800-y and 40x) [13].

This points out that at least in some domains, we need to pay specific attention to modeling student knowledge considering skill combinations. One of these domains is arguably computer programming. Research on computer science education and pedagogy has long argued that knowledge of a programming language can't be reduced to a sum of knowledge about different programming constructs since there are many stable combinations or patterns (also known as schemas, plans) that have to be taught and practiced [8, 27]. To generalize to other domains, we argue that skill combination patterns can also potentially represent "chunks", production rules, or general problem solving patterns that are critical for defining expertise in a domain.

In this paper we use datasets from SQL and Java programming tutors to demonstrate the feasibility and the value of modeling skill combination patterns. We present a data-driven framework for modeling skill combinations including a novel student model and new evaluation metrics.

2. BACKGROUND

2.1 Patterns in Programming Expertise

Experts in the area of psychology of programming have long argued that programming plans and other kinds of patterns form an important part of programming expertise [8]. Most actively, all kinds of programming patterns: plans, techniques, templates, and "cliches" were used by researchers in the are of intelligent tutoring systems (ITS) for programming to support intelligent analysis of student programs [18, 31]. While intelligent debuggers recognized and diagnosed pattern errors, they do not maintain a model of student

knowledge on pattern level. First student models on the level of patterns were introduced by Brusilovsky [1] who used expert-suggested construct pairs as knowledge components for problem sequencing and Weber [30] who applied larger programming "episodes" as knowledge components for adaptive recommendation of program examples [31]. The more advanced episodic model has never been expanded or ported to another languages due to its complexity and high knowledge engineering demands. In contrast, the simper pair-based approach has been used in a few follow-up projects [23]. The work presented in this paper continues this stream of research focusing on lower-level skill combinations and their automatic (rather than manual) discovery.

2.2 Complex Skill Student Modeling and its Evaluation

Complex skill student knowledge modeling has been a challenge and attracts increasing attention in student modeling community. Starting from Gong [9] constructing variants based on traditional Knowledge Tracing using multiplication or minimization among skills, more advanced models have been put forward to address the multiple skill credit and blame assignment issue [22, 32, 11, 33], and many of them resemble cognitive diagnosis models such as DINA or NIDA [19]. However, these student models only use a "flat" knowledge structure which might overlook the actual important dependency or interaction among skills. Among the works that consider hierarchies or relations among skills, most of them focus on prerequisite relations [5, 2] or granularity hierarchy [25, 12], which differ from skill combination relation in nature and cannot be readily applied. Also, most of these works rely heavily on expert laboring, and automatic methods to discover skill relations is still a recent endeavor [4].

Regarding the data-driven evaluations of complex skill student models, prior works mostly uses prediction performance [32, 11, 33]. There is a growing concern of only using prediction performance for evaluating student models. [10] have shown that highly predictive model may be useless for adaptive tutoring, and they can have low parameter plausibility or consistency [17]. While some attempts have been made to evaluate in terms of the effect for tutoring [22], a recent learner outcome-effort paradigm [10] offers a promising way to empirically evaluate student models for adaptive tutoring. In this work, we extend both of our previous frameworks [10, 17] for evaluating complex skill student models.

3. PROPOSED FRAMEWORK

This section introduces our framework of incorporating skill combination patterns for deeper knowledge tracing. It consists of model construction and model evaluation.

3.1 Model Construction

We construct a Bayesian Network (BN), *Conjunctive Knowledge Modeling with Hierarchical Skill Combination (CKM-HSC)*, to model skill combinations. Figure 1 shows the structure of our model. It supports three functionalities: 1) performance prediction, 2) knowledge estimation, and 3) mastery decision. The O nodes (shaded) represent observed binary student performance, K nodes represent binary latent skill knowledge level, and M nodes represent the aggregated binary latent skill knowledge level that we call *Mastery*. Particularly, we introduce *Mastery* nodes, in order to reflect the idea of granting skill mastery for each skill based on all the skill combinations' knowledge levels. Edges denote causal relation among the variables. One major challenge of applying BN for complex skill modeling is the time and space complexity. We made simplifications in the model to balance complexity and accuracy, which we further explain.

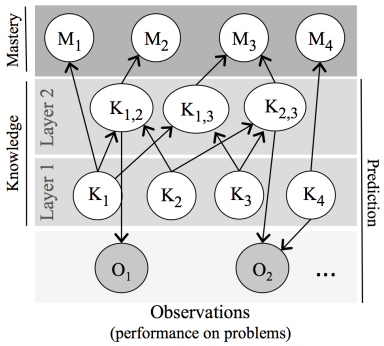


Figure 1: Bayesian Network Structure of Conjunctive Knowledge Modeling with Hierarchical Skill Combinations

3.1.1 Skill Combination Pattern Representation

We represent skill combination patterns in a hierarchical way in the Knowledge and Mastery parts (see Figure 1):

I The first layer consists of basic individual skills. It captures the basic understanding and application of a skill. For example, in Java, we can have K_1 representing the basic understanding and application for *ForStatement*, and similarly, K_2 for *ArrayElement*.

II The intermediate layers consist of skill combinations which can be derived from smaller skill units. These layers capture a deeper understanding of each individual skill considering when and how to apply it with other skills in more complex situations. For example, $K_{1,2}$ can represent the joint skill of *ForStatement* and *ArrayElement* requiring iteratating through an array. As a first step to simplify the problem, we only consider skill combinations from two basic individual skills.

III The last layer represents the *Mastery* in each of the individual skills, where the nodes are fed from skill combinations or single skills. We introduce this layer in order to reflect the idea of granting skill mastery for each skill based on all the skill combinations' knowledge levels. To avoid repeated computation, combined skills only connect to one Mastery node, the one representing the later basic skill in the temporal order in which the skills appear in the course.

Knowing skills from lower layers serves as prerequisites for knowing skills in higher layers. We argue that such a dependency (hierarchy) is crucial. For example, for an item requiring skill combinations, if a student fails, the model can differentiate whether the problem is in the basic skill or due to a lack of experience of applying skills together; if a student succeeds, the model can increase the belief that the student already knows the basic individual skill through the prerequisite link.

To model the relation between basic individual skills and combined skills, we use multinomial distributions, since different basic individual skills might have different importance in affecting the knowledge of combined skills. However, this can impose exponential complexity. In this framework we limit skill combinations to be composed from an upper bound of N lower level skill units, typically choosing N as a

small number. In the case where high order combinations are involved, we can consider using causal independence models to reduce to linear complexity.

3.1.2 Learning Network Structure and Parameters

Since the network involves latent variables, we use Expectation Maximization algorithm to conduct network learning. The final structure of the network depends on which skill combinations are incorporated. If we don't limit the search space of skill combinations, it will scale exponentially. So we employ some heuristics to select skill combinations. Algorithm 1 outlines a greedy search algorithm. It requires a pre-ordering of the skill combination candidates. During each iteration, it compares the cost functions (e.g., data log likelihood) of the network with a skill combination incorporated with the optimal one from previous iterations. However, in each iteration, the posteriors of latent variables have to be computed which can be very time-consuming.

Algorithm 1 Learn CKM-HSC

Input: potential skill combination list C, original item to individual skill adjacency matrix Q (qmatrix), initial skill to skill adjacency matrix H (all zero), student performance data matrix O, initial parameters Θ
Output: new Bayesian Network B with final structure, fitted parameters, and posteriors of latent skills

1: B = ConstructBN(Q, H, Θ)
2: B, $Cost$ = LearnBN(B, O)
3: C' = SelectAndRankSkillCombinations(C, O)
4: **for** each skill combination pattern $c \in C'$ **do**
5: Q' = UpdateQmatrix(Q, c)
6: H' = UpdateHmatrix(H, c)
7: B' = UpdateBN(B, Q', H')
8: B', $Cost'$ = LearnBN(B', O)
9: **if** $Cost' < Cost$ **then**
10: $B = B'$
11: $Cost = Cost'$
12: **end if**
13: **end for**
14: **return** B

To increase the run time efficiency, we construct a simplified version replacing the search procedural with empirical thresholding (pruning), as shown in Algorithm 2 which we use in the current work. Preliminary analysis with a subset of the data (Java dataset with 158 students on 45 items) showed that we can achieve comparable results with the simplified version of the algorithm and reduce computation time significantly (from 9 hour to 30 minutes to finish structure and parameter learning) based on the popular Bayes Net Toolbox [24]. We leave for the future to explore faster implementation tools and alternative techniques (e.g., approximate inference) to address this issue.

Now we introduce how we select skill combinations and thus learn the BN structure in Algorithm 2. Firstly, we get all the possible pairwise combinations of skills by each pair's co-occurrence in each item, which results in a big list of skill combination patterns. Then, we apply following criteria (with higher importance criterion ordered before) through the algorithm to select the skill combinations (note that in order to get ranked skill combinations for an item (Line 14: function *GetRankedSkillCombinationsForItem)*, we also use criterion I-II).

Algorithm 2 Learn CKM-HSC (simplified)

Input: C, Q, H, O, Θ (same as in Algorithm 1); item to possible skill combination matrix P, thresholds $\boldsymbol{\alpha_{1:4}}$
Output: new Bayesian Network B with final structure, fitted parameters, and posteriors of latent skills

1: C'=[ ]
2: **for** each skill combination pattern $c \in C$ **do**
3: β_c = EstimateDifficulty(c, P, O)
4: S = GetIndividualSkills(c)
5: $\boldsymbol{\beta_S}$ = EstimateDifficulties(S, Q, O)
6: **if** (β_c - Max($\boldsymbol{\beta_S}$)) $> \alpha_1$ and $\beta_c > \alpha_2$ **then**
7: C' = Add(C', c)
8: **end if**
9: **end for**
10: M = GetItemList(Q)
11: **for** each item $m \in M$ **do**
12: β_m = EstimateDifficulty(m, O)
13: **if** $\beta_m > \alpha_3$ **then**
14: C_m = GetRankedSkillCombinationsForItem(m, P, O)
15: C'_m = [ ]
16: **for** each combined skill $c \in C_m$ **do**
17: **if** $c \in C'$ and Length(C'_m) $\leq \alpha_4$ **then**
18: C'_m = Add(C'_m, c)
19: **else**
20: break
21: **end if**
22: **end for**
23: Q' = UpdateQmatrix(Q, m, C'_m); $Q = Q'$
24: H' = UpdateHmatrix(H, C'_m); $H = H'$
25: **end if**
26: **end for**
27: B' = ConstructBN(Q', H', Θ)
28: B' = LearnBN(B', O)
29: $B = B'$
30: **return** B

I The difference of difficulties between the combined skill and its hardest individual skill should be large and non-zero (Line 6: β_c - Max($\boldsymbol{\beta_S}$)). Otherwise, the original individual skills should be able to capture the difficulty of an item already and extra skill is not needed.

II The difficulty of the combined skill should be high (Line 6: $\beta_c > \alpha_2$).

III An item with higher difficulty is more needed to be refined, in this case, to be indexed with combined skills (Line 13: $\beta_m > \alpha_3$).

IV Each item is indexed with a limited number of skill combinations (Line 17: Length(C'_m) $\leq \alpha_4$). Our preliminary study shows that the number of skill combination candidates increases quickly even with a small relaxation of this criterion.

In order to estimate difficulty of skills or items (Algorithm 2 the function *EstimateDifficulty* or *EstimateDifficulties*), one possible way is to apply some existed models such as Item Response Theory [29] or Additive Factors Model [3] to extract related parameters. However, any such pre-estimations can't avoid imposing assumptions of learning or skill relations. For example, IRT assumes no learning while our datasets are collected from learning activities; AFM assumes compensatory relation among skills while on our datasets a conjunctive relation is more suitable. We use a simple heuristic method which also imposes assumptions

but we argue it should be no worse than the above alternatives given its simplicity of computation. We estimate item m's difficulty β_m by computing the average error rate across students on this item. For estimating skill s's difficulty β_s, we apply a Min gate (get the minimum value) to the difficulty estimations (β_m) of all items (M) containing this skill, assuming each skill's basic difficulty level is determined by the easiest item in which the skill is required, i.e., $\beta_s = \underset{s \in m, 1 \leq m \leq M}{\text{Min}}(1 - \beta_m)$. We use this formula for both estimating an individual skill's difficulty and a skill combination's difficulty.

Learning BN with hierarchical structures among latent skills based on temporal learning data is non-trivial. To simplify the modeling process, we ignore the temporal learning effect during the model learning process[1], while maintaining the dynamic knowledge estimation power during the application phase (see 3.1.3). Such simplifications have been used in many prior works [5, 28], and we argue that it can be reasonably compensated by the dynamic updating process (see subsection 3.1.3). We leave for future work to include the temporal learning effect in our model.

3.1.3 Dynamic Knowledge Update

At each practice opportunity, the learned network provides the inference of a latent knowledge node. For each student's first practice, the network uses the same priors for latent knowledge nodes to perform inference, and only when we update the knowledge by different students' performance, the network starts to differentiate among students by maintaining different up-to-date knowledge estimates. In order to achieve this, CKM-HSC follows the same dynamic BN roll-up mechanism in [5]: it uses posterior knowledge probabilities conditioned on historical observations as the priors for next time step. Note that we only need to update the basic individual skills in the first layer. The states of other skill variables will be computed based on fitted conditional probabilities. The following formulas outline the update of the knowledge state of a basic individual skill i at time step t after observing an evidence:

$$\text{P}(K_i^{t+1}\text{=known})_{prior} = \text{P}(K_i^t\text{=known})_{posterior} \quad (1)$$

$$= \text{P}(K_i^t\text{=known}|O^t\text{=evidence}) \quad (2)$$

This updating procedure differs from KT [6]: it ignores the transition probabilities between time steps. However, similar to [5], we argue that the change in knowledge estimates is mainly determined by the new evidence. We leave for the future incorporating learning parameters.

3.1.4 Performance Prediction

CKM-HSC applies a Noisy-AND gate for each item's conditional probability distribution, assuming that students need to know all the underlying skills in order to succeed given our datasets' nature on which only one answer is accepted. Noisy-AND gate and other causal independence models has been used in many prior works [5], particularly in the popular psychometric model DINA [19]. The major benefit is that it only requires two parameters (*guess* and *slip*) for each item, regardless of the number of skills required by the item. This significantly reduces the complexity of the model. Note that for items having skill combinations, the probability of getting this item correct only depends on the combined skills and is independent of the basic individual skills. Equations 3 and 4 show the CPT of item i consisting of its guess and slip probabilities g_i and s_i:

$$\text{P(O=correct|all skills=known)} = 1 - s_i \quad (3)$$

$$\text{P(O=correct|at least one skills=unknown)} = g_i \quad (4)$$

3.1.5 Mastery Decision

To access the mastery level of each skill, CKM-HSC aggregates knowledge estimates from each skill combination assigned to current skill (if no skill combinations are present, then the basic skill), and gives a final knowledge level of a skill, based on which, skill mastery is decided. It means that to reach mastery, students need to know both, skill's basic meaning and how to correctly apply it with other skills. We aggregate by computing the joint probability of all required skill units being in known state as the probability of a mastery node as shown in Equation 5. Since skill combinations share parents, computing this joint probability considers the dependencies among skills:

$$\text{P}(M_i\text{=known})=\text{P}(K_{i,1}\text{=known},\ldots\ K_{i,j}\text{=known}) \quad (5)$$

where $K_{i,1}$ to $K_{i,j}$ denotes the skill combinations assigned to M_i. If a skill has no assigned skill combination, then

$$P(M_i = known) = P(K_i = known). \quad (6)$$

3.2 Model Evaluation

We argue that modeling "deeper" knowledge of complex skills requires also "deeper" evaluation, and only examining prediction performance is not enough, as explained in Section 2.2. We propose a new evaluation framework extending a recent Learner Effort-Outcome Paradigm (LEOPARD) [10], and a multifaceted evaluation framework [17]. We think that in a real world tutoring system relying on a student model's knowledge inference to provide support, knowledge inference quality should be of primary importance, and also parameter plausibility shouldn't be overlooked.

Mastery Accuracy (knowledge inference quality). The basic idea is that once a student model asserts a student's mastery for an item's required skills, the student should be very unlikely to fail. In the original metric, each single skill's knowledge state is examined and accuracy is computed on data after reaching mastery threshold. This is not applicable to multiple skill case since the responsibility of each skill for the actual performance is not clear. To address this, we examine the multiple skill knowledge states jointly as shown in Algorithm 3.

Mastery Effort (knowledge inference quality). This metric empirically quantifies the number of practices students needed to reach a level of mastery of the set of skills in a domain inferred by a student model. It is computed based on each individual skill's expected effort as shown in Algorithm 4 (which is similar to the original metric in [10]). To apply LEOPARD to multiple skill models, we make a simplification: each practice count can be treated as a count for each individual skill involved. We argue that it won't change the relative effort comparing among models because all models will be computed under the same simplification. In a preliminary study on our data set, we find out that only

[1]Correspondingly, we compress the data for training by using average success rate across attempts as the performance of a student on an item with discretization (assign 1 if success rate ≥ 0.5 and 0 otherwise)

one out of 20 students has reached mastery for all attempted skills. This suggests that computing effort based on per skill perspective is better since it allows to have sufficient data. We leave for the future to further improve this metric.

Algorithm 3 Get mastery accuracy (multiple skill)

Input: vector Y of actual correctness performance (ordered by time step within each student), matrix K of a student model's inferred knowledge levels (probability of known) for required skills per observation (with the same order as Y), item to skill matrix Q, mastery threshold p
Output: mastery accuracy metric

NbObsWithSkillsInferredMastery = 0
NbObsCorrectAndSkillsInferredMastery= 0
for each observation $y_t \in Y$ **do**
  $\boldsymbol{s}_t$ = GetRequiredAggregatedIndividualSkills(y_t, Q)
  $\boldsymbol{k}_t$ = GetInferredKnowledge(t, s_t, K)
  AllSkillsInferredMastery= TRUE
  for each skill's inferred knowledge $k_{q,t} \in \boldsymbol{k}_t$ **do**
    ▷ *Judge whether current required skill is inferred mastery; if any one skill is not inferred mastery, change the flag and stop checking:*
    if $k_{q,t} < p$ **then**
      AllSkillsInferredMastery = FALSE; break
    end if
  end for
  if AllSkillsInferredMastery **then**
    NbObsWithSkillsInferredMastery += 1
  end if
  if y_t==1 **then**
    NbObsCorrectAndSkillsInferredMastery += 1
  end if
end for
if NbObsWithSkillsInferredMastery>0 **then**
  ▷ *Check among observations with required skills all inferred mastery, how many of them are with actual correct responses*:
  MasteryAccuracy = $\frac{NbObsCorrectAndSkillsInferredMastery}{NbObsWithSkillsInferredMastery}$
else
  return "No sufficient data."
end if
return MasteryAccuracy

Two important considerations for computing the metrics described above are explained as follows.

- *The balance between Mastery Accuracy and Effort.* As our later experiments shown, it seems that a model can achieve higher mastery accuracy by letting students practice for longer time. However, we argue that having an acceptable level of Mastery Accuracy (e.g.,> 0.85) is necessary for a real world tutoring system, and thus it is needed and worthy that more practice effort is required before reaching this level of accuracy. Otherwise, the system will risk granting mastery in few practices when students haven't reached it, which we think is more problematic than delaying the granting of mastery.
- *Mastery Threshold.* Different mastery thresholds have been used in prior works [6, 10]. Typically 0.95 is used, yet the rationality behind it is also not clear. We present evaluations on an extensive range of mastery thresholds [0.5, 0.99], but primarily focus on higher threshold regions (e.g., ≥ 0.7).
- *Amount of data to compute mastery metrics.* As shown in Algorithm 3 and 4, to compute mastery accuracy or effort, we need to have data with corresponding knowledge states inferred as mastery state by student models.When mastery is not reached (by inference), LEOPARD uses imputation, which we think is not suitable on thresholds with few students inferred to reach mastery. This might distort the original distribution. So, We remove imputation, and only focus on thresholds with sufficient data with at least 25% of the complete data available to compute the mastery metrics.

Algorithm 4 Get mastery effort (multiple skill)

Input: K, Q, p (same as in Algorithm 3)
Output: mastery effort metric

MasteryEffort=0
for each skill q in the complete skill list **do**
  MasteryEffortPerStu$_q$=[ ]
  for each student u attempted items requiring q **do**
    MasteryEffort$_{u,q}$ = 0
    ▷ *Select inferred skill q's knowledge level sequence corresponding to student u's response sequence*:
    $\boldsymbol{k}_{u,q}$ = Filter(K)
    for each practice $t \in$[0, Length($\boldsymbol{k}_{u,q}$)] **do**
      ▷ *Judging whether at the current practice a student is inferred reaching mastery of a skill*:
      if $k_{u,q,t} < p$ **then**
        ▷ *If a student isn't inferred reaching mastery of a skill, keep counting*:
        MasteryEffort$_{u,q}$ += 1
      else
        ▷ *Otherwise, stop counting:*
        break
      end if
    end for
    MasteryEffortPerStu$_q$ =
     Add(MasteryEffortPerStu$_q$, MasteryEffort$_{u,q}$)
  end for
  ▷ *Compute on average how many practices are required to reach mastery for skill q*:
  MasteryEffort$_q$ = Average(MasteryEffortPerStu$_q$)
  ▷ *Accumulating number of practices to get the total number of practices to reach mastery of all skills for a student on average*:
  MasteryEffort += MasteryEffort$_q$
end for
return MasteryEffort

IDI (parameter plausibility). To examine parameter plausibility, we utilize the Item Discriminative Index (IDI) used in psychometric models for evaluating item qualities and refining q-matrix [7]:

$$\text{IDI}_i = 1 - g_i - s_i \tag{7}$$

where g_i and s_i denotes item i's guess and slip probabilities. A higher IDI is preferable which have both low guessing and slipping rates. We utilize this metric to evaluate models with item level guess and slip probabilities.

RMSE, AUC (performance prediction accuracy). We report two popular prediction metrics used in evaluating student models, Root mean squared error (RMSE) and Area Under the Receiver Operating Characteristic curve (AUC) based on a recent paper [26]'s suggestion.

Table 1: Dataset descriptive statistics.

Dataset	#obs.	#items	#skills	avg #skills/item	#users	%correct
SQL	17,197	45	34	5 (from 1 to 10)	366	58%
Java	25,988	45	56	5 (from 1 to 11)	347	67%

4. STUDIES

In this section we describe studies that demonstrate CKM-HSC's advantage over traditional and alternative models, give a closer examination of extracted skill combinations, and briefly explore the effect of adding external knowledge.

4.1 Datasets and Experimental Setup

We used two datasets, SQL and Java programming, collected through classroom studies between Fall 2013 and Fall 2015 at the University of Pittsburgh. The SQL dataset is from an online system SQL-KNOT, and the Java dataset is from the system JavaGuide [15]. Both systems contain problems requiring students to apply multiple skills at the same time, and they allow students to have multiple attempts where each attempt corresponds to a new instantiation of the same template. In SQL-KNOT, students are asked to write complete SQL statements to solve problems, and only one solution is accepted; in JavaGuide, students are requested to predict output of different Java programs and only one answer is accepted. Both systems assess correctness (0/1). Problems are grouped by topics and indexed by a set of fine-grained concepts by experts assisted with ontologies and an automatic Java parser [14]. For computational efficiency, on Java dataset, we remove two most complex topics (Interface and Inheritance), and randomly selected 45 items. Table 1 shows the descriptive statistics of the final datasets used (with multiple attempts).

One both datasets we conducted a 10-fold student stratified cross-validation. In each fold we trained on 90% of students , and predicted the probability of getting a problem correct and inferred the probability of knowing a skill for each observation of the remaining 10% of new students. On each fold, we evaluate each model from following multi-faceted metrics introduced in Section 3.2:

- **Knowledge inference quality:** Mastery Accuracy, Mastery Effort.
- **Parameter plausibility:** IDI
- **Performance prediction accuracy:** RMSE, AUC

For each metric, we compute the average value across 10 folds and 95% confidence interval based on t-distribution. Note that during predicting on test set, we perform dynamic update: each model gradually gather information of the new students. We used Bayes Net Toolkit (BNT) [24] to construct all the models. We initialize each models' skill parameters by setting them as the estimated difficulties introduced in Section 3.1.2. We initialize the learn rate for KT based models as 0.15, and initialize all models' guess and slip as 0.3. We set convergence criterion as 10^{-4} and maximum EM iteration as 500. For applying Algorithm 2, we set α_1 as the 75^{th} percentile's value of the positive range of the difference (β_c-Max($\boldsymbol{\beta_S}$)) computed on all skill combinations based on the performance data (ordered from small to large), and similarly, we set α_2, α_3 as the 75^{th} percentile of β_c and β_m values computed from the data. We set α_4=2. These values of α are set based on our preliminary studies consulting experts. On average (across 10 folds), our automatic method extracts 14 and 30 skill combinations on SQL and Java datasets.

Table 2: Comparison among CKM-HSC, CKM and traditional KT models (average across 10 folds with 95% CI).

Datasets	SQL			Java		
Models	IDI	RMSE	AUC	IDI	RMSE	AUC
KT-Single	/	.46(.01)	.70(.01)	/	.44(.01)	.69(.01)
WKT	/	.47(.00)	.69(.01)	/	.44(.00)	.72(.01)
CKM	.33(.01)	.47(.01)	.70(.01)	.39(.01)	.45(.01)	.73(.01)
CKM-HSC	.35(.01)	.47(.01)	.70(.01)	.46(.01)	.45(.01)	.73(.01)

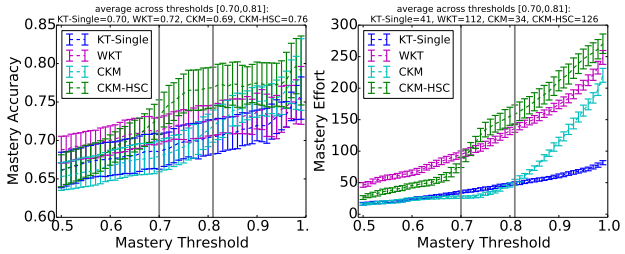
4.2 Is proposed skill combination incorporated model better than traditional KT models?

In the first study, we compare following models:

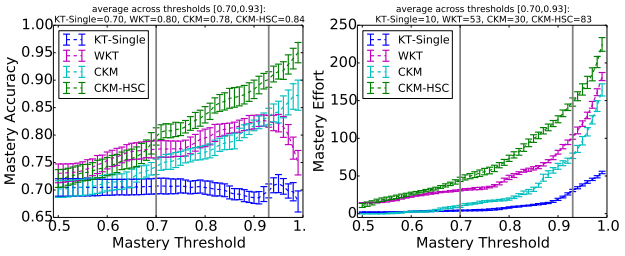
- **KT-Single:** Each item is mapped to one coarse-grained skill (topic). This is an implementation of the classic Knowledge Tracing [6] which uses a Hidden Markov Model for each skill to model the latent knowledge and predict the performance of students. The hidden variable represents knowledge level (learned or unlearned) and the observable represents performance (correct or incorrect). There are four parameters for each skill: the probability of a student initially learned a skill before practicing, the probability of a student transitioning from unlearned to learned state, the probability of a student guessing correctly given the student is in unlearned state and the probability of a student slipping given the student is in learned state. We fit and predict each skill independently.
- **WKT:** Each item is mapped to a set of skills (concepts). We fit each skill independently as classic Knowledge Tracing [6], and then take the minimum of each skill's predicted probability of success as the final prediction. We only update the knowledge of this weakest skill when the evidence is an incorrect response, while we update all skills when the evidence is a correct response. This is a model used in many prior works [9].
- **CKM:** Our proposed conjunctive knowledge modeling without incorporating skill combinations. It fits, predicts and updates skills jointly by Bayesian Network.
- **CKM-HSC:** Our proposed model. Each item is mapped to a set of skills, which can be either individual skill or skill combination with hierarchy among them. It fits, predicts and updates skills jointly using a BN.

On both datasets, CKM-HSC has comparable predictive performance as other models (Table 2). However, CKM-HSC has significantly better mastery accuracy than other models (on thresholds with enough data points to compute mastery metrics and high enough values to be considered as proper mastery thresholds), and it also estimates that more effort is needed to reach mastery (Figure 2). As mentioned in Section 3.2, we think that having such more practices in CKM-HSC is necessary in order to reach an acceptable mastery accuracy.

We are still interested in comparing how CKM-HSC and WKT distribute students' effort, so we further conduct a drill-down effort analysis on SQL dataset (Figure 3). For skills that potentially have skill combinations, WKT indifferently updates the skill whether an item requires skill combinations or not, and blindly distributes students' effort among different application context, risking students with shallow learning reaching mastery, and also it directs students to spend more effort on skills without combinations. CKM-HSC clearly distributes students' efforts as follows: it



(a) Sql



(b) Java

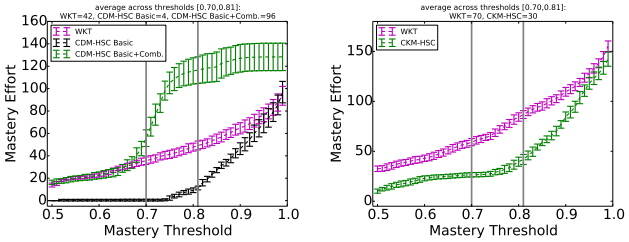
Figure 2: Comparison among CKM-HSC, CKM and traditional KT models. Grey lines denote regions with enough data points to compute mastery metrics and high enough values to be considered as proper mastery thresholds.

tends to make students spend little effort (≤ 10) on the basic individual skill, spend less effort on skills without skill combinations, and spend much more effort on mastering skill combinations by its estimation.

Does the improvement in mastery accuracy of CKM-HSC mainly come from the Bayesian Network responsibility assignment mechanism rather than incorporating skill combinations? Comparing the simple CKM with KT-Single and WKT, we can see there is no clear advantage of simple CKM, which is not the case for CKM-HSC. So the advantage starts to appear after we incorporate skill combinations. We also can see CKM-HSC has better IDI than CKM (Table 2).

Will the difference in mastery accuracy and effort just an artifact of setting different mastery thresholds? No. For example, on SQL dataset, for reaching similar mastery accuracy by selecting different thresholds, WKT usually tend to require much more effort to reach mastery than CKM-HSC. Similar phenomenon can be observed on Java dataset.

An interesting observation is that on Java dataset, although KT-Single has quite comparable RMSE with other models, it shows very low mastery accuracy and asserts very low mastery effort. This further shows the problem of using predictive performance to evaluate student models.



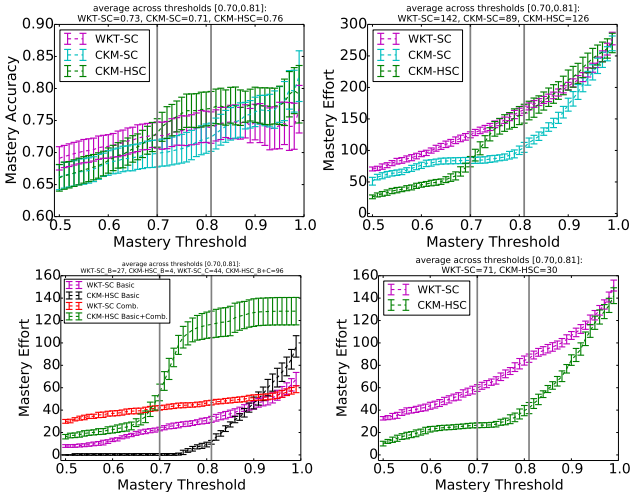
(a) Skills requiring knowing combined skills (b) Skills not requiring knowing combined skills

Figure 3: A drill down-comparison CKM-HSC vs. WKT on mastery effort on SQL dataset.

4.3 Is using hierarchy better than independence for incorporating skill combinations?

In this study, we investigate the effect of using hierarchy for incorporating skill combinations. We compare following three models:

- **WKT-SC:** Using WKT's framework, adding skill combinations as new independent individual skills.
- **CKM-SC:** Using proposed model's framework, adding skill combinations as new independent individual skills.
- **CKM-HSC:** Our proposed model, adding skill combinations in a hierarchical way.



(a) Skills requiring knowing combined skills (b) Skills not requiring knowing combined skills

Figure 4: CKM-HSC vs. alternatives to incorporate skill combinations on SQL. Grey lines denote regions with enough data points to compute mastery metrics and high enough values to be considered as proper mastery thresholds.

We mainly report results on SQL dataset. Three models have similar predictive performance (RMSE=0.47±0.01, and AUC=0.7±0.01). However, comparing other metrics, we see important difference among the models (Figure 4) . Comparing WKT-SC with CKM-HSC, the latter has better mastery accuracy and similar effort on higher mastery thresholds.

A drill-down analysis (Figure 4) shows that WKT-SC tends to require students to spend much more effort on basic individual skills' understanding for skills with combinations and also tends to require more mastery effort on skills without combinations; while CKM-HSC tends to require students to put major effort in skill combinations. Comparing CKM-HSC with CKM-SC, we can see that including hierarchy significantly improves mastery accuracy and requires more mastery effort, and slightly improves parameter plausibility from an IDI value of 0.34±0.01 to a value of 0.35±0.01. Similarly, we observe that the different behaviors among three models are not simply the effect of setting different mastery thresholds.

On Java dataset, we reach similar conclusion: all models have similar predictive performance, but CKM-HSC improves mastery accuracy and requires similar mastery effort, and it also has higher IDI than CKM-SC.

Table 3: Comparison among CKM-HSC and alternatives adding external knowledge on Java (average across 10 folds with 95% CI; mastery metrics computed on [0.7, 0.93]).

Models	M.Acc	M.Effort	IDI	RMSE	AUC
CKM-HSC	.84	83	.46(.01)	.45(.01)	.73(.01)
CKM-HSC-P	.82	70	.45(.01)	.45(.01)	.73 (.01)
CKM-HSC-P-E	.82	59	.42(.01)	.45(.01)	.73 (.01)

4.4 Can we improve modeling by adding external knowledge for skill combination extraction?

Our impression from manual inspection of the automatically extracted skill combinations from data was mostly positive. For example, on Java dataset, many extracted skill combinations contain *Loop* related concepts (such as *ForStatement*) combined with *IfStatement*, *AddAssignmentExpression*, or *Array* related concepts. However, we also observed extracted pairs which are not apparently meaningful. For example, *java.lang.String.substring* combined with *java.lang.System.out.print*. We conduct further analysis to investigate whether using external information to increase skill combinations' interpretability can improve student modeling. We compare three models on the Java dataset:

- **CKM-HSC:** Our proposed model only using performance data to extract skill combinations. This results in 41 skill combinations from 10 folds.
- **CKM-HSC-Proximity:** Extracted pairs are further constrained by *proximity*, i.e we only consider pairs appearing in the same line or in the same block (for example a skill inside a *for-loop* and the *for-loop* are in the same block). This results in 31 skill combinations.
- **CKM-HSC-Proximity-Expert:** Experts mark pedagogically meaningful pairs from the skill pairs extracted by *CKM-HSC-Proximity*. This results in 19 skill combinations.

We summarize the results in Table 3. Interestingly, adding extra knowledge sacrifices mastery accuracy slightly, and also decrease models' parameter plausibility, but it saves much more effort. This shows that purely data-driven approach, despite some human interpretability issues, may capture some latent, implicit relationship among skills to enable it has the highest mastery accuracy and parameter plausibility, but it may require much more mastery effort. A promising direction is to find a balance between data-driven approach and human interpretability.

5. CONCLUSIONS

In this paper, we examined the importance of skill combinations in domains of multiple-skill problems where multiple skills can be integrated (combined) together to require extra knowledge or produce extra difficulty not captured by contributing individual skills. Our work is the first attempt to model this skill combination effect using data-driven techniques. We constructed a Conjunctive Knowledge Model with Hierarchical Skill Combinations (CKM-HSC) based on Bayesian Networks. In our new model, mastery of a skill can only be granted when a student demonstrates ability to apply this skill with other skills in varied contexts. We demonstrated that incorporating skill combinations can significantly increase mastery assertion accuracy and more reasonably direct students' practice effort comparing with traditional knowledge tracing models and its non hierarchical counterpart. We propose an evaluation framework for student models built for multiple skill knowledge modeling. Under our new evaluation framework, we effectively quantify the accuracy and expected mastery effort of a student model's knowledge inference, which can not be addressed by traditional performance prediction metrics.

We also addressed the problem of computational complexity by using suitable network representation and heuristic data-driven methods. In the future, we will explore more efficient implementation tool and new techniques.

We are aware that it is very challenging to learn the structure of a complex hierarchical, multi-layer Bayesian Network, so we will consider collecting larger datasets, and datasets with more sparse connections among nodes. We will also consider consulting experts for determining (refining) the structures of the network, and giving structure priors of the network.

Our current study only considers skill combination in pairs, and it will be to interesting to consider more complex skill combinations, and even consider difficulty factors from cognitive task analysis [21]. In the longer term, we expect that such a framework can be extended to model skill application context, "chunks", schemas or patterns for defining expertise or deep learning of a domain [16].

We expect that our new skill combination model can provide more significant benefits when deployed in real-world tutoring systems. It can potentially enable more powerful visualization for open student models and better remediation. It can encourage students to practice in more contexts on the way to mastery, and guide content authors in developing content that addresses skill combinations. Not only serving as an attempt to bridge data mining with pedagogical and learning theory, our work also raises attention in the community to build student models for deeper, robust student learning.

Acknowledgement

This research is supported by the Advanced Distributed Learning Initiative (http://www.adlnet.gov/) and the National Science Foundation Cyber-Human Systems (CHS) Program under Grant IIS-1525186.

6. REFERENCES

[1] P. Brusilovsky. Intelligent tutor, environment and manual for introductory programming. *Educational and Training Technology International*, 29(1):26–34, 1992.

[2] C. Carmona, E. Millán, J.-L. Pérez-de-la Cruz, M. Trella, and R. Conejo. Introducing prerequisite relations in a multi-layered bayesian student model. In *User Modeling 2005*, pages 347–356. Springer, 2005.

[3] H. Cen, K. Koedinger, and B. Junker. Learning factors analysis: A general method for cognitive model evaluation and improvement. In M. Ikeda, K. Ashley, and T.-W. Chan, editors, *Intelligent Tutoring Systems*, volume 4053 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin / Heidelberg, 2006.

[4] Y. Chen, P.-H. Wuillemin, and J.-M. Labat. Discovering prerequisite structure of skills through probabilistic association rules mining. In *The 8th International Conference on Educational Data Mining*, pages 117–124, 2015.

[5] C. Conati, A. Gertner, and K. Vanlehn. Using bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction*, 12(4):371–417, 2002.

[6] A. Corbett and J. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278, 1995.

[7] J. De La Torre. An empirically based method of q-matrix validation for the dina model: Development and applications. *Journal of educational measurement*, 45(4):343–362, 2008.

[8] D. J. Gilmore and T. R. G. Green. Programming plans and programming expertise. *The Quarterly Journal of Experimental Psychology Section A*, 40(3):423–442, Aug. 1988.

[9] Y. Gong, J. E. Beck, and N. T. Heffernan. Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In *Intelligent Tutoring Systems*, pages 35–44. Springer, 2010.

[10] J. P. González-Brenes and Y. Huang. Your model is predictive – but is it useful? theoretical and empirical considerations of a new paradigm for adaptive tutoring evaluation. In *Proc. 8th Intl. Conf. Educational Data Mining*, pages 187–194, 2015.

[11] J. P. González-Brenes, Y. Huang, and P. Brusilovsky. General features in knowledge tracing: Applications to multiple subskills, temporal item response theory, and expert knowledge. In *Proc. 7th Int. Conf. Educational Data Mining*, pages 84–91, 2014.

[12] J. E. Greer and G. I. McCalla. A computational framework for granularity and its application to educational diagnosis. In *IJCAI*, pages 477–482, 1989.

[13] N. T. Heffernan and K. R. Koedinger. The composition effect in symbolizing: The role of symbol production vs. text comprehension. In *the Nineteenth Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates.

[14] R. Hosseini and P. Brusilovsky. Javaparser: A fine-grained concept indexing tool for java problems. In *AIEDCS workshop*, 2013.

[15] I.-H. Hsiao, S. Sosnovsky, and P. Brusilovsky. Guiding students to the right questions: adaptive navigation support in an e-learning system for java programming. *Journal of Computer Assisted Learning*, 26(4):270–283, 2010.

[16] Y. Huang. Deeper knowledge tracing by modeling skill application context for better personalized learning. In *The 24nd Conference on User Modeling, Adaptation and Personalization Doctoral Consortium*, 2016.

[17] Y. Huang, J. P. González-Brenes, R. Kumar, and P. Brusilovsky. A framework for multifaceted evaluation of student models. In *Proc. 8th Int. Conf. Educational Data Mining*, pages 203–210, 2015.

[18] W. L. Johnson and E. Soloway. Proust: Knowledge-based program understanding. *IEEE Transactions on Software Engineering*, 11(3):267–275, 1985.

[19] B. W. Junker and K. Sijtsma. Cognitive assessment models with few assumptions, and connections with nonparametric item response theory. *Applied Psychological Measurement*, 25(3):258–272, 2001.

[20] K. R. Koedinger, A. T. Corbett, and C. Perfetti. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science*, 36(5):757–798, 2012.

[21] K. R. Koedinger, E. A. McLaughlin, and J. C. Stamper. Automated student model improvement. *Proc. 8th Intl. Conf. on Educational Data Mining*, pages 17–24, 2012.

[22] K. R. Koedinger, P. I. Pavlik Jr, J. C. Stamper, T. Nixon, and S. Ritter. Avoiding problem selection thrashing with conjunctive knowledge tracing. In *EDM*, pages 91–100, 2011.

[23] A. N. Kumar. A scalable solution for adaptive problem sequencing and its evaluation. In *4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'2006)*, pages 161–171. Springer Verlag.

[24] K. Murphy et al. The bayes net toolbox for mabtlab. *Computing science and statistics*, 33(2):1024–1034, 2001.

[25] Z. A. Pardos, N. T. Heffernan, B. Anderson, and C. L. Heffernan. The effect of model granularity on student performance prediction using bayesian networks. In *User modeling 2007*, pages 435–439. Springer, 2007.

[26] R. Pelanek. Metrics for evaluation of student models. *Journal of Educational Data Mining*, 7(2):1–19, 2015.

[27] E. Soloway and K. Ehrlich. Empirical studies of programming knowledge. *IEEE Trans. Software Engineering*, SE-10(5):595–609, 1984.

[28] N. Thai-Nghe, L. Drumond, A. Krohn-Grimberghe, and L. Schmidt-Thieme. Recommender system for predicting student performance. *Procedia Computer Science*, 1(2):2811–2819, 2010.

[29] W. J. van der Linden and R. K. Hambleton. *Handbook of modern item response theory*. Springer Science & Business Media, 2013.

[30] G. Weber. Episodic learner modeling. *Cognitive Science*, 20(2):195–236, 1996.

[31] G. Weber. Individual selection of examples in an intelligent learning environment. *Journal of Artificial Intelligence in Education*, 7(1):3–31, 1996.

[32] Y. Xu and J. Mostow. Comparison of methods to trace multiple subskills: Is LR-DBN best? In *Proc. 5th Intl. Conf. Educational Data Mining*, pages 41–48, Chania, Greece, 2012.

[33] Y. Xu and J. Mostow. A unified 5-dimensional framework for student models. In *Workshop on Approaching Twenty Years of Knowledge Tracing at the 7th Intl. Conf. on Educational Data Mining*, pages 122–129. Citeseer, 2014.