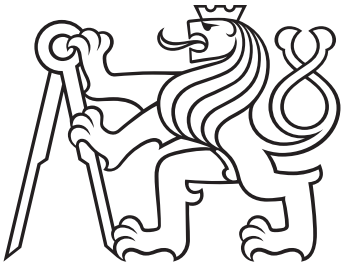
České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství
Obor: Aplikace softwarového inženýrství



Webový systém pro testování znalostí studentů

VÝZKUMNÝ ÚKOL

Vypracoval: Bc. František Navrkal
Vedoucí práce: Ing. Martin Plajner
Rok: 2017

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství Akademický rok 2016/2017

ZADÁNÍ VÝZKUMNÉHO ÚKOLU

Pro: **Bc. František Navrkal**

Obor: **Aplikace softwarového inženýrství**

Název práce: **Webový systém pro testovaní znalostí studentů**

Pokyny pro vypracování:

1. Prostudovat základní teorii adaptivního testování znalostí a bayesovských sítí.
2. Navrhnout a realizovat webový systém pro adaptivní testování znalostí studentů středních škol v oblasti matematiky.
3. Výsledný systém bude obsahovat i uživatelské rozhraní umožnující realizaci adaptivního testu a nástroje pro základní analýzu výsledků a sběr dat pro další využití při aktualizaci modelu.

Doporučená literatura:

1. Almond, R.G., Mislevy, R.J., Steinberg, L., Yan, D., Williamson, D. *Bayesian Networks in Educational Assessment.* Springer, 2015.
2. Vomlel, J. *Bayesian networks in educational testing.* In: International Journal of Uncertainty, Fuzziness and Knowledge Based Systems. Vol. 12, Supplementary Issue 1, 2004, pp. 83-100.
3. Vomlel, J. *Building Adaptive Tests using Bayesian networks.* In: Kybernetika. Volume 40, Number 3, 2004, pp. 333 - 348.
4. Plajner, M. and J. Vomlel. *Student Skill Models in Adaptive Testing.* In: Proceedings of the Eighth International Conference on Probabilistic Graphical Models, pp. 403–414, 2016.

Jméno a pracoviště vedoucího práce:

Ing. Martin Plajner
Ústav teorie informace a automatizace, AV ČR
Pod Vodárenskou věží 4
182 08 Praha 8

Datum zadání práce: 17. 10. 2016

Termín odevzdání práce:

V Praze dne 17.10.2016

....
vedoucí práce vedoucí katedry

Prohlášení

Prohlašuji, že jsem svůj výzkumný úkol vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne ....

Bc. František Navrkal

Poděkování

Děkuji Ing. Martinu Plajnerovi za jeho pomoc a ochotu.

Bc. František Navrkal

Obsah

Úvod

V první kapitole tohoto textu je krátce představen koncept počítačového adaptivního testování a jedna z cest jeho implementace, konkrétně pomocí bayesovských sítí. Toto krátké představení by mělo poskytnout dostatečný znalostní základ pro aspoň obecné pochopení fungování konkrétního systému popsaného v kapitole druhé.

Druhá kapitola zároveň slouží jako obecná dokumentace webové aplikace pro adaptivní testování v rámci výzkumného úkolu vyvíjené. V této kapitole jsou také popsány možnosti dalšího vývoje aplikace.

Na samotném konci tohoto textu jsou jako jediná příloha uvedeny instrukce pro instalaci a nasazení aplikace.

1 Adaptivní počítačové testování

Adaptivní počítačové testování je způsob získávání informací o znalostech studenta, při kterém se průběh testování mění na základě informací získaných v jeho průběhu (nebo i před jeho zahájením). V anglicky psané literatuře se pro toto používá označení computerized adaptive testing (CAT). Využití adaptivního testování umožňuje kombinovat výhody individuálního testování jako zvýšená přesnost a úspora času i námahy testovaného s možností hromadného testování. [1]

1.0.1 Motivace

V [6, strana 10] je uveden následující argument. Hromadně aplikovaný test musí, aby fungoval dobře pro širokou škálu úrovní jednotlivých měřených dovedností, obsahovat otázky různé obtížnosti. Pokud by na příklad neobsahoval lehké otázky, nemohl by test přesně měřit nízkou úroveň příslušných dovedností.

Tento fakt ovšem znamená, že zdatný student ve statickém testu musí nejdříve odpovědět na mnoho pro něj jednoduchých otázek, než se dostane k otázkám, které mají nějakou vypovídací hodnotu o jeho dovednostech. Toto vede k mrhání času a námahy studenta, přináší to též do procesu testování nechtěné faktory jako chyby z nedbalosti způsobené nudou.

V případě, že je úroveň měřených dovedností studenta naopak nízká, hrozí zase zmatek, frustrace a nekvalitní data, pokud slabý student správnou odpověď jen náhodou uhodne.

1.0.2 Průběh adaptivního testování

Samotný průběh adaptivního testování sestává z opakování těchto kroků: [4]

1. Výběr otázky na základě modelu znalostí studenta dle zvoleného výběrového kritéria.
2. Zobrazení otázky studentovi, získání odpovědi a její vyhodnocení.
3. Vložení získaných informací do modelu znalostí studenta a jeho aktualizace.
4. Kontrola ukončovacích kritérií a případné ukončení testu při jejich dosažení.

Celý jeden cyklus těchto kroků zde bude uváděn jako krok celého testu.

Tomuto ovšem musí předcházet tvorba celého testu a příslušného modelu znalostí. Po sběru dat o dostatečném vzorku průběhů testování lze na jejich základě zvýšit

přesnost (resp. prakticky prediktivní schopnost) modelu. Příklad obojího je uveden v sekci 1.2.

1.1 Znalostní model (bayesovské sítě)

Pro modelování znalostí studenta v rámci přístupu k adaptivnímu testování použitému v této práci (navrženému v [1]) se používá bayesovských sítí. Jedná se o pravděpodobnostní grafický model popisující vztahy mezi náhodnými veličinami pomocí rozdělení podmíněné pravděpodobnosti.

1.1.1 Formální definice

Bayesovskou síť pro naše účely můžeme definovat formálně jako uspořádanou dvojici orientovaného acyklického grafu $B_S = (V, E)$ (tzv. struktura sítě), kde každému prvku $i \in V$ odpovídá náhodná veličina X_i, a množiny B_P funkcí $P(X_i \,|\, (X_j)_{j \in pa(i)})$, které udávají podmíněné pravděpodobnostní rozdělení X_i v závislosti na hodnotě veličin odpovídajících (přímým) předchůdcům (neboli rodičům) i v B_S značeným jako $pa(i)$. (Toto značení je možná trochu zavádějící v případě uzlů bez předchůdců, tedy u nichž $pa(i) = \emptyset$, tam totiž vlastně nejde o podmíněné pravděpodobnosti, ale pravděpodobnosti nepodmíněné.) Množinu všech náhodných veličin budu značit $\mathcal{X} = \{X_i \,|\, i \in V\}$.

V dále uvažovaných případech budou X_i nabývat jen určitých diskrétních hodnot (stavů veličiny) z množin $\mathbb{X}_i$, takže si prvky B_P můžeme také představovat jako funkce zobrazující vektor stavů předchůdců i v B_S na vektor pravděpodobností stavů X_i. Prakticky lze takovouto funkci zapsat jako tak zvanou tabulku podmíněných pravděpodobností. (Opět nemusí jít nutně o pravděpodobnosti podmíněné.)

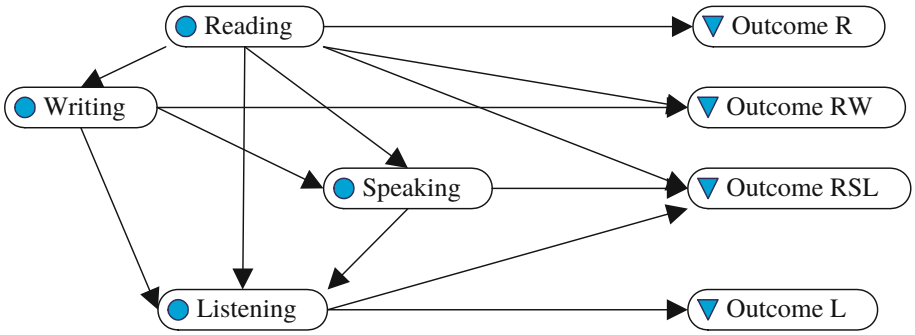
1.1.2 Neformální popis

Bayesovské sítě popisují kauzální vztahy mezi pozorovanými náhodnými veličinami. V našem případě půjde kauzální vztah mezi odpověďmi studenta na otázky v testu a jeho měřenými dovednostmi.

Teoreticky by bylo možné uvažovat všechny možné konjunkce stavů uvažovaných náhodných veličin $\mathbb{X}$ a popsat jejich pravděpodobnostní rozdělení, ale počet možných konjunkcí je dán jako součin počtu stavů jednotlivých veličin:

$$|\mathbb{X}| = \prod_{\mathbb{X}_i \in \{\mathbb{X}_i \,|\, i \in V\}} |\mathbb{X}_i| \,, \tag{1.1}$$

což je v praktických případech (stačí již s řádově desítkami veličin) často astronomické číslo, protože minimální smysluplný počet možných stavů náhodné veličiny jsou 2, což znamená, že $\mathbb{X}$ bude vždy aspoň $2^{|\mathcal{X}|}$.



Obrázek 1.1: Příklad bayesovské síťě z [1].

Toto je důvod, proč se uvažují jen určité možné kauzální vztahy mezi pozorovanými veličinami, konkrétně ty, jejichž přítomnost odhadl buď učící algoritmus na základě sesbíraných dat nebo expert na danou problematiku.

Tyto vztahy jsou pak kvantifikovány opět buďto expertem nebo algoritmicky na základě dat. Konkrétně jsou určeny funkce v B_P, tedy speciálně příslušné tabulky podmíněných pravděpodobností. Metody učení tabulek podmíněných pravděpodobností jsou popsány na příklad v [1, kapitola 9].

Třeba v případě popisovaném v sekci 1.2 autoři použili smíšený přístup: struktura sítě (tedy kvalitativní popis kauzálních vztahů) byla odhadnuta experty (samotnými autory) a tabulky podmíněných pravděpodobností (kvantitativní popis) pak byly naučeny algoritmicky na základě sesbíraných dat.

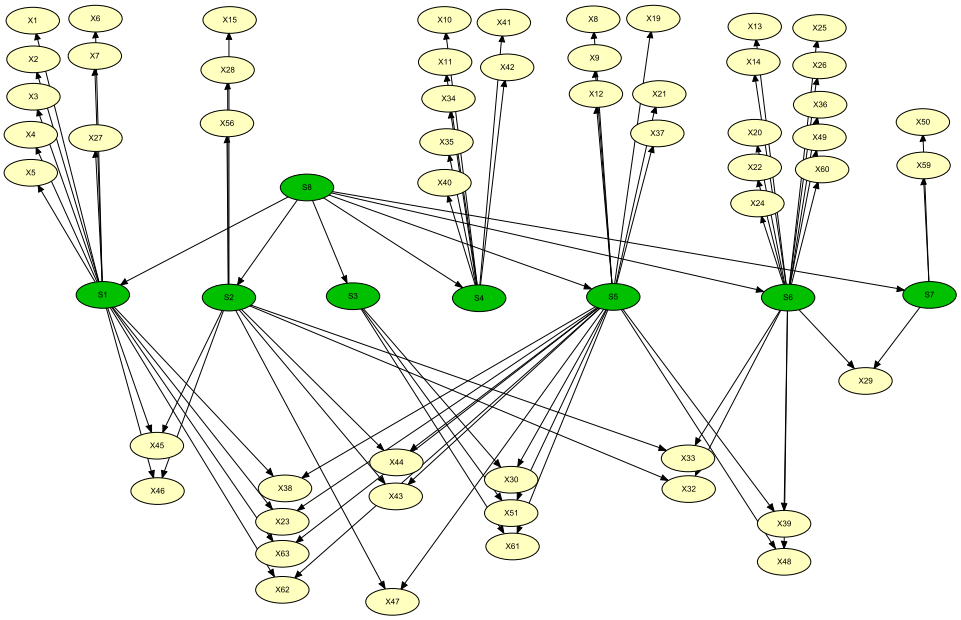
1.1.3 Příklady

V [1] je hned v úvodu jako příklad uvedena síť na obrázku 1.1. Uzly s kroužky udávají jednotlivé dovednosti (čtení, psaní, poslech, mluvení) a uzly s trojúhelníky udávají stavy vyhodnocení úkolů vyžadujících jednotlivé kombinace těchto dovedností.

Další příklad jsem vybral z [4]. Na obrázku 1.2 zelené uzly reprezentují jednotlivé dovednosti a žluté uzly otázky. Uzel *S8* (celková dovednost, konkrétně znalost některých druhů funkcí a jejich vlastností) je předchůdcem všech uzlů, následují ostatní zelené uzly (jednotlivé prvky celkové dovednosti: znalost polynomů, exponenciálních funkcí a tak dále) a na konec žluté uzly, které reprezentují jednotlivé úlohy v testu.

1.1.4 Dovednosti a otázky

Množina všech náhodných veličin $\mathcal{X}$ se ve zde blíže probíraných případech bude typicky dělit na 2 disjunktní podmnožiny $\mathcal{S} \cup \mathcal{Q} = \mathcal{X}$, kde $\mathcal{S}$ je množina veličin představujících dovednosti a $\mathcal{Q}$ otázky (je tomu tak v obou zmíněných příkladech a předpkládám to i ve své webové aplikaci). Dále ještě mohou být v bayesovské síti zastoupeny jako náhodné veličiny další informace o studentovi jako třeba věk, pohlaví nebo výsledky jiného testování.



Obrázek 1.2: Příklad bayesovské sítě z [4].

Vyhodnocení adaptivního testu bude v následujícím textu znamenat na základě získaných dat vypočítat pravděpodobností stavů veličin z $\mathcal{S}$. (Toto lze studentovi prezentovat jako pravděpodobnosti dosažené úrovně jeho znalostí, případně možná přístupněji třeba pomocí určitého odvozeného skóre a podobně.)

Množinu $\mathcal{Q}$ ještě budeme někdy dělit na další 2 disjunktní podmnožiny $Q = \widehat{\mathcal{Q}} \cup \overline{\mathcal{Q}}$, kde $\widehat{\mathcal{Q}}$ je množina dosud nezodpovězených otázek a $\overline{\mathcal{Q}}$ již zodpovězených.

1.1.5 Aktualizace modelu v průběhu testování

Informace získané v průběhu testování se do modelu vkládají tak, že se rozdělení veličiny z $\mathcal{Q}$ na základě odpovědi na příslušnou otázku testu nahradí rozdělením, kde je jeden ze stavů již jistý (tedy vektor pravděpodobnostní stavů je pak vektor obsahující jeden prvek s hodnotou 1 a všechny ostatní s hodnotou 0). Toto umožní aktualizaci rozdělení ostatních veličin jak z $\mathcal{S}$, tak z $\widehat{\mathcal{Q}}$.

Formálně budu psát, že nějaká $X_i \in \overline{\mathcal{Q}}$ nabývá hodnoty $x_i \in \mathbb{X}_i$ jako $X_i = x_i$. Případně k rozlišení nahradím X_i za S_i nebo $\mathbb{X}_i$ za $\mathbb{Q}_i$ a podobně.

1.2 Návrh adaptivního testu

Možností, jak navrhnout test je nespočet. Pro představu krátce shrnu jako příklad článek podrobně popisující návrhu testu, konkrétně [7].

1.2.1 Návrh obsahu a sběr dat

Nejdříve autoři navrhli statický test, ten předložili malé skupině studentů. Na základě zpětné vazby studentů pak vytvořili další, pro studenty srozumitelnější návrh, ze kterého byly také odebrány otázky, které se ukázaly neefektivní pro zisk informací.

Test nakonec obsahoval 29 úloh, které se dále rozdělily celkem na 53 podúloh. Úlohy byly alternativně vyhodnocovány pomocí bodového ohodnocení, nebo jen jako správně či špatně vypracované.

Sběru dat se celkem zúčastnilo 281 studentů. O těchto studentech si autoři sesbírali také některá jejich osobní data jako věk, pohlaví, jméno a jejich známky z matematiky, chemie a fyziky za poslední 3 pololetí.

1.2.2 Srovnání znalostních modelů

V článku autoři porovnali celkem 14 různých modelů, které se lišily mj. počtem uvažovaných stavů náhodných veličin, počtem měřených dovedností a zahrnutím osobních dat studentů.

Struktura sítě byla vždy pro daný srovnávaný model pevná (navržená na základě předpokladů autorů článku o příslušných kauzálních vztazích). Tabulky podmíněných pravděpodobností se vždy v rámci desetinásobné křížové validace odhadly (naučily) na základě $^9/_{10}$ náhodně vybraných případů (učící množiny). Jednotlivé modely pak byly porovnávány na základě chování modelu v simulacích na zbývající $^1/_{10}$ případů (testovací množině).

Modely autoři konkrétně porovnávali na základě míry úspěšnosti odhadu hodnocení zatím v simulaci neřešených úloh na základě úloh již v dané simulaci vyřešených a vyhodnocených, to celé v závislosti na počtu vyřešených úloh (počtu odsimulovaných kroků testování). Za odhad se bral nejpravděpodobnější stav dané zatím neřešené úlohy v danému kroku testu.

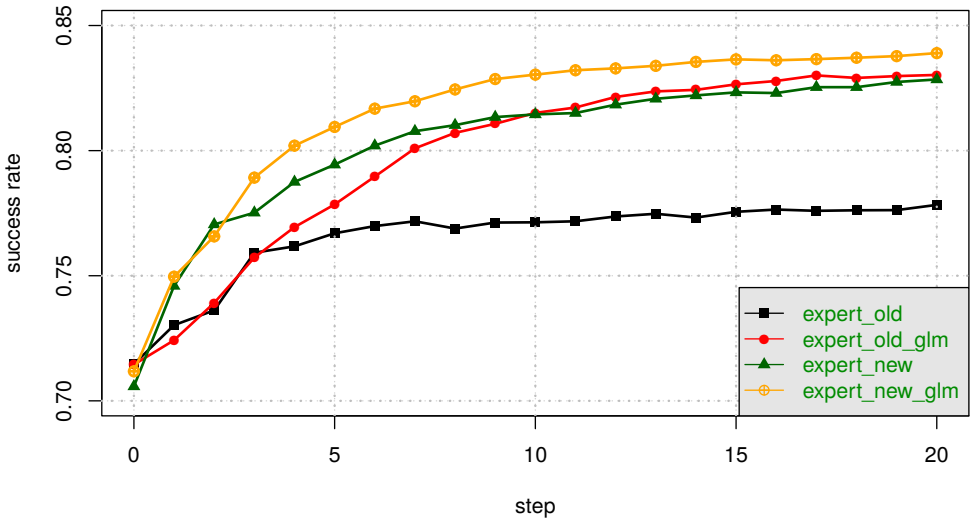
Jako kritérium výběru úlohy v simulacích autoři použili maximalizaci středního informačního zisku (neboli poklesu entropie) stavu dovedností po vyhodnocení kandidátní úlohy. (Podrobně rozebráno v podsekci 1.3.1.)

1.2.3 Vyhodnocení

Autoři došli k tomu, že příliš jednoduché modely si nevedly dobře – ukázalo se jako výhodné uvažovat raději 3 než pouze 2 stavy u dovednostních veličin.

Zahrnutí osobních údajů o studentovi do modelu vylepšilo úspěšnost předpovědí jen v počátečních krocích testování, poté měly pouze malý pozitivní, nebo dokonce negativní efekt.

Složitější z modelů pravděpodobně postihl tak zvaný over-fitting, to jest naučené tabulky podmíněných pravděpodobností nezobecňovaly dobře vztahy z učící množiny pro testovací množinu, takže si nakonec nevedly dobře.



Obrázek 1.3: Srovnání úspěšnosti znalostních modelů z [4]. Modely `expert_old(_glm)` jsou modely se strukturou bez uzlu celkové dovednosti, `expert_new(_glm)` jsou modely s ní. Přípona `_glm` značí jinou metodu učení tabulek podmíněných pravděpodobností.

V jejich novějším článku [4] používají autoři mj. síť na obrázku 1.2, která se liší od té z [7] hlavně přidáním souhrnné dovednostní veličiny jako předchůdce všech ostatních veličin. Tato síť měla hlavně ve spojení s jejich novým algoritmem pro učení z testovací množiny mnohem lepší úspěšnost předpovědí.

1.3 Strategie výběru otázek

Na základě znalostního modelu se mohou v průběhu adaptivního testování vybírat otázky k zobrazení studentovi dle různých kritérií. Uvedu zde příklady jednoduchých „hladových“ strategií srovnávaných v [8] určených pro bayesovské modely znalostí a poté nastíním i možné jiné, komplikovanější, strategie.

Nechť již zodpovězené otázky nabývají hodnot $e = \{Q_i = q_i | Q_i \in \overline{\mathcal{Q}}\}$, pak lze vybrat otázku na příklad následujícími způsoby.

1.3.1 Maximalizace středního informačního zisku

Toto kritérium je motivováno tím, že jako výsledek testu chceme co nejvíce snížit nejistotu ve stavech $S_i \in S$. Hledáme tedy otázku $\widehat{Q}_i^* \in \widehat{\mathcal{Q}}$, jejíž vyhodnocení tuto nejistotu co nejvíce sníží ve smyslu střední hodnoty dle pravděpodobnosti odpovědí na ní. Nejistotu zde měříme pomocí Shannonovy entropie, konkrétně:

$$H(e) = -\sum_{S_i \in \mathcal{S}} \sum_{s_i \in \mathbb{S}_i} P(S_i = s_i|e) \log P(S_i = s_i|e), \tag{1.2}$$

po zodpovězení uvažované otázky $\widehat{Q}_i \in \widehat{\mathcal{Q}}$ a jejím vyhodnocení na stav $\widehat{Q}_i = \widehat{q}_i$ tedy:

$$H(e, \widehat{Q}_i = \widehat{q}_i) = -\sum_{S_i \in \mathcal{S}} \sum_{s_i \in \mathbb{S}_i} P(S_i = s_i | e, \widehat{Q}_i = \widehat{q}_i) \log P(S_i = s_i | e, \widehat{Q}_i = \widehat{q}_i). \quad (1.3)$$

Střední hodnota entropie na dovednostech po nějaké odpovědi na $\widehat{Q}_i$ bude tedy:

$$EH(\widehat{Q}_i, e) = \sum_{\widehat{q}_i \in \widehat{\mathcal{Q}}_i} P(\widehat{Q}_i = \widehat{q}_i | e) H(e, \widehat{Q}_i = \widehat{q}_i). \quad (1.4)$$

Toto kritérium vybírá otázku $\widehat{Q}_i^*$ jako:

$$\widehat{Q}_i^* = \arg\max_{\widehat{Q}_i \in \widehat{\mathcal{Q}}} H(e) - EH(\widehat{Q}_i, e). \quad (1.5)$$

1.3.2 Maximalizace středního rozptylu stavů dovedností

Použití tohoto kritéria ve zde rozebrané podobě předpokládá, že veličiny dovedností S_i jsou binární, tedy mohou nabývat pouze dvou hodnot. Konkrétně zde budou tyto hodnoty značeny jako 0 v případě absence dovednosti a 1 v případě, že student danou dovednost má.

Toto kritérium se snaží o dobrou separaci studentů na základě jejich dovedností. Konkrétně jde o tuto hodnotu:

$$var(\mathcal{S} | e, \widehat{Q}_i) = \sum_{S_i \in \mathcal{S}} \sum_{\widehat{q}_i \in \widehat{\mathbb{Q}}_i} \Big(P(S_i = 1 | e) - P(S_i = 1 | e, \widehat{Q}_i = \widehat{q}_i) \Big)^2 P(\widehat{Q}_i = \widehat{q}_i | e). \quad (1.6)$$

Nakonec se analogicky jako u předchozího kritéria vybere $\widehat{Q}_i^*$ jako:

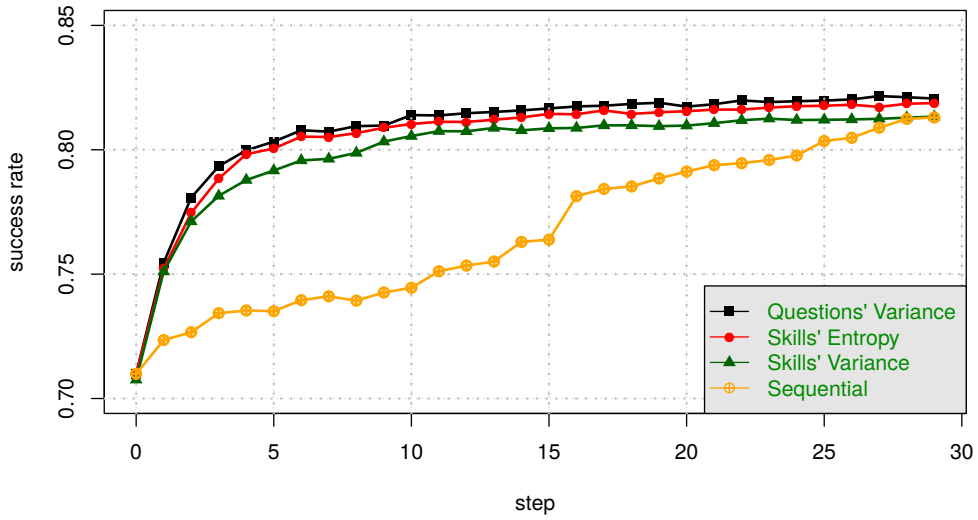
$$\widehat{Q}_i^* = \arg\max_{\widehat{Q}_i \in \widehat{\mathcal{Q}}} var(\mathcal{S} | e, \widehat{Q}_i). \quad (1.7)$$

1.3.3 Maximalizace středního rozptylu stavů otázek

Uvedená podoba tohoto kritéria vyžaduje, aby byly binární tentokrát veličiny otázek Q_i. Zcela analogicky:

$$var(\widehat{Q}_i | e) = \sum_{s \in \mathbb{S}_{pa(i)}} \Big(P(\widehat{Q}_i = 1 | e) - P(\widehat{Q}_i = 1 | e, s) \Big)^2 P(s | e), \quad (1.8)$$

kde $\mathbb{S}_{pa(i)} = \{S_j = s_j \,|\, j \in pa(i),\ s_j \in \mathbb{S}_j\}$ je množina všech možných konjunkcí stavů těch dovednostních veličin v $\mathcal{S}$, jejichž uzly jsou předchůdci uzlu i dané otázky v síti. Stavy ostatních dovednostních veličin není třeba uvažovat, neboť jsou tak zvaně d-separované (koncept vysvětlen názorně třeba v [5, strana 26 až 32]).



Obrázek 1.4: Srovnání úspěšnosti kritérií výběru otázky z [8].

$\widehat{Q}_i^*$ se vybere předvídatelně jako:

$$\widehat{Q}_i^* = \arg\max_{\widehat{Q}_i \in \widehat{\mathcal{Q}}} var(\widehat{Q}_i|e). \tag{1.9}$$

Pokud by k výpočtu $var(\widehat{Q}_i|e)$ bylo nutné uvažovat příliš mnoho konjunkcí stavů dovedností (jejichž počet by udával vzorec 1.1, kde by množina $\mathbb{X}$ byla nahrazena $\mathbb{S}_{pa(i)}$), bylo by možné použít následující aproximaci:

$$var^*(\widehat{Q}_i|e) = \sum_{S_i \in \mathcal{S}_{pa(i)}} \sum_{s_i \in \mathbb{S}_i} \Big(P(\widehat{Q}_i = 1|e) - P(\widehat{Q}_i = 1|e, S_i = s_i)\Big)^2 P(S_i = s_i|e). \tag{1.10}$$

Tato aproximace byla použita i pro dosažení výsledků na obrázku 1.4.

1.3.4 Srovnání a komplexnější strategie

Případ porovnání zmíněných metod výběru je na obrázku 1.4. Srovnávané strategie jsou „hladové“, konkrétně uvažují vždy jen jeden krok testu dopředu.

Šlo by ovšem též simulovat všechny možné scénáře vývoje testu několik kroků dopředu, vyčíslit hodnoty maximalizovaného kritéria na konci těchto simulací a poté přiřadit každé kandidátní otázce $\widehat{Q}_i \in \widehat{\mathcal{Q}}$ příslušnou střední hodnotu kritéria napříč scénáři, které by mohly následovat zodpovězení uvažované otázky. Tento proces je podrobně rozebrán v [3] včetně praktických heuristik pro zrychlení výpočtů.

2 Vyvíjený systém

V této kapitole je popsána samotná aplikace, kterou jsem v rámci výzkumného úkolu vyvíjel. Moje aplikace tvoří uživatelské rozhraní pro knihovnu vyvíjenou na Ústavu teorie informace a automatizace AV ČR, která zajišťuje ve zde rozebírané webové aplikaci samotné výpočty na příslušné bayesovské síti, zvané *Computerized Adaptive Testing Generic Framework* (dále jen `catest`).

Všechny cesty v souborovém systému v této kapitole jsou uvedeny relativně vůči kořenu složky `CArisTotle`. Doporučuji během čtení této kapitoly paralelně procházet kód v repozitáři na `https://github.com/FNj/CArisTotle`.

2.1 Výběr nástrojů

Vedoucí práce mě upozornil na systém *R Shiny* [11]. Jedná se o knihovnu psanou v jazyce *R* [9], ve kterém je psaná i samotná knihovna `catest`. Já zase přišel s nápadem použít jako jádro aplikace „microframework“ (aplikační rámec) *Flask* [12] psaný v *Pythonu* [10].

Systém *R Shiny* je ovšem cílen na různé datově analytické výstupy spíše než na obecnější webové aplikace, také mám opravdu jen malé zkušenosti s programovacím jazykem *R*, ve kterém je tento systém napsaný. Raději jsem tedy nakonec zůstal u *Pythonu* a *Flasku*, se kterým mám jak ze školního, tak profesionálního prostředí asi největší zkušenosti ze všech programovacích jazyků. *Python* je také mnohem více uzpůsoben multiparadigmatickému programování (porovnáváno třeba v [23]).

Protože takto používám kombinaci dvou různých programovacích jazyků, je nutné mít k dispozici nějaké rozhraní mezi nimi. Původně jsem uvažoval o úplně vlastním řešení, ale nakonec jsem našel knihovnu *RPy2* [13], která mi tvorbu rozhraní velmi usnadnila.

Aplikace musí ukládat dlouhodobě informace o uživatelích, definicích adaptivního testu a průběhu jednotlivých případů testování. Tyto jsou ukládány do relační databáze pomocí knihovny *SQLAlchemy* [14], konkrétně jsem při vývoji používal pro jeho jednoduchost nasazení databázový systém *SQLite* [15], ten je ovšem asi nejsnáze nahraditelnou částí celého systému – stačilo by mírně změnit nastavení v souboru `config.py`.

Pro lokalizaci (překlad) tam jsem použil knihovnu *Babel* [16]. Překlad je použit jen tam, kde to bylo vhodné kvůli použití nějaké externí knihovny, co to předpokládala.

2.2 Struktura systému

Celý systém sestává ze 4 hlavních částí:

- aplikace samotné, napsané v *Pythonu* – v podstatě mojí vlastní knihovny `CArisTotle` –,
- SQL databáze, přistupovaná přes *SQLAlchemy*, – nyní pro vývoj *SQLite* –,
- knihovny `catest` v *R*, propojené přes *RPy2* se zbytkem systému a
- *Jinja* šablon pro tvorbu výstupního HTML kódu.

V následující sekci jsou tyto části krátce popsány.

2.2.1 Datový model

V této sekci je krátce popsán použitý datový model. Graficky je znázorněn na obrázku 2.1. Jednotlivé uvedené entity jsou vlastně zároveň tabulky databázi a třídy objektů v samotném prostředí *Pythonu* ve smyslu objektově relačního zobrazení (konceptem shrnutým třeba v [22]). Datový model je deklarován v modulu `datamodel`, konkrétně v souboru `/datamodel/model.py`.

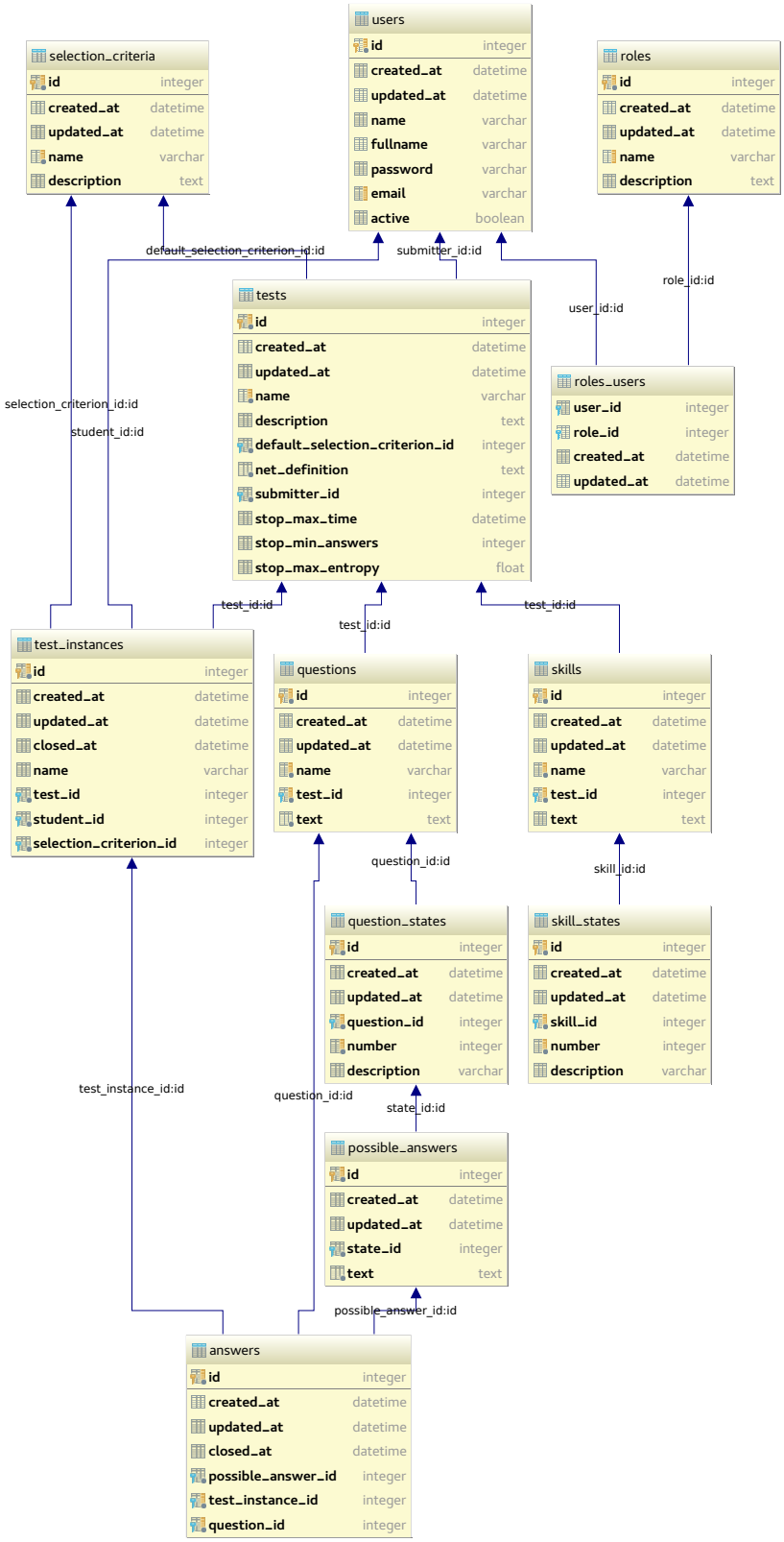
Obecný význam entit (tabulek) a většiny jejich vlastností (sloupců) na obrázku 2.1 je asi zřejmý, ale raději zde dále alespoň zasadím do kontextu všechny tabulky.

Tabulky `users`, `roles` a `roles_users` zaznamenávají identity uživatelů. Existenci tabulky pro role předpokládá knihovna *Flask-Security*, která zajišťuje v aplikaci správu identit. Kdyby tomu tak nebylo, tabulka `roles` ani `roles_users`, by se v modelu nevyskytla. [17]

Tabulka `tests` obsahuje informace o testu jako celku. Sloupec `net_definition` obsahuje definici struktury sítě i tabulek podmíněných pravděpodobností ve formátu, který dokáže zpracovat knihovna `catest`. Konkrétně jde o textový formát, který pro ukládání používá aplikace *Hugin Researcher* [21]. Sloupec `stop_max_entropy` udává požadovanou úroveň (Shannonovy) entropie stavů dovedností v bitech, konkrétně počítané jako suma napříč dovednostmi:

$$H(e) = -\sum_{S_i \in \mathcal{S}} \sum_{s_i \in \mathbb{S}_i} P(S_i = s_i|e) \log_2 P(S_i = s_i|e). \tag{2.1}$$

Tabulky `questions` a `question_states` slouží k záznamu otázek daného testu a jejich možných stavů, analogicky lze popsat tabulky `skills` a `skill_states` (měřené dovednosti a jejich stavy). Sloupec `name` u otázek a dovedností slouží k identifikaci příslušného uzlu ve struktuře sítě, sloupec `number` zase k identifikaci stavu v tabulkách podmíněných pravděpodobností. Tyto tabulky lze částečně generovat na základě `net_definition`. Sloupec `net_definition` ovšem neobsahuje samotný text otázek ani popis jejich stavů – ten musí doplnit zadavatel zvlášť. Hodnoty ve sloupcích `name` a `number` musí odpovídat pojmenování, resp. číslu stavu, daného uzlu v `net_definition`.



Obrázek 2.1: Diagram datového modelu použitého v aplikaci. Jednotlivé rámečky reprezentují příslušné tabulky (resp. entity, případně třídy) a šipky vztahy mezi nimi (reference pomocí cizích klíčů).

V tabulce `possible_answers` se nacházejí možné odpovědi na otázky (které pak vedou na určítý stav otázky). Je tedy nutné, aby test sestával z uzavřených otázek (tedy otázek s předem danou sadou možných odpovědí). Tabulka `test_instances` uchovává informace o jednotlivých případech testů a tabulka `answers` samotné odpovědi studenta na položené otázky.

Všechny tabulky obsahují sloupce `created_at` a `updated_at`, které uchovávají datum a čas vytvoření daného řádku v tabulce, resp. jeho poslední změny. Tabulky `test_instances` a `answers` navíc obsahují sloupec `closed_at`, který udává časovou značku uzavření daného případu testu nebo odpovědi. Tyto záznamy jsou užitečné pro kopírování záznamů do jiných databází: vždy stačí kopírovat jen záznamy, které vznikly nebo byly změněny po času posledního kopírování. Tyto časové údaje mohou také být užitečné pro analýzu dat, třeba pro odhad doby řešení otázek apod.

Při deklaraci datového modelu jsem navíc i nastavil indexování některých sloupců, resp. jejich kombinací (pro zrychlení vyhledávání), a integritní omezení, na příklad to, že v tabulce `skills` musí být kombinace `test_id` a `name` unikátní.

Složitější operace nad datovým modelem jsou naprogramovány jako funkce v souboru `/datamodel/procedures.py`.

```
1  class TimeStampMixin:
2      created_at: datetime = Column(DateTime, default=_get_date)
3      updated_at: datetime = Column(DateTime, onupdate=_get_date)
4
5
6  class Skill(ModelBase, TimeStampMixin):
7      __tablename__ = 'skills'
8
9      id: int = Column(Integer, primary_key=True, index=True)
10     name: str = Column(String, nullable=False)
11     test_id: int = Column(Integer, ForeignKey('tests.id'), nullable=False)
12     text: str = Column(Text)
13
14     test: Test = relationship("Test", back_populates="skills")
15     states: List[SkillState] = relationship("SkillState", back_populates="skill")
16
17     __table_args__ = (UniqueConstraint('name', 'test_id', name='_test_skill_name_ux'),)
```

Ukázka kódu 1: Definice „mixinu“ pro časovou značku vytvoření a poslední změny a deklarace třídy pro dovednost z `datamodel/model.py`.

2.2.2 Rozhraní s *R*

V této sekci krátce popíšu svoji konkrétní implementaci rozhraní mezi *Pythonem* a *R*. Celé toto rozhraní je naprogramováno v modulu `inteRface`.

Potřebné funkce z `catest` jsou volány v souboru `/inteRface/procedures.py`. Bylo nutné, abych si doprogramoval v *R* i dvě pomocné funkce – jsou uvedeny přímo v tomto souboru.

Od těchto implementačních detailů je ovšem zbytek aplikace odstíněn pomocí abstrakce v podobě třídy `BayesNet` v souboru `/inteRface/classes.py`. Pomocí operací na objektech této třídy lze provádět všechny potřebné interakce s knihovnou `catest`, konkrétně jde o zpracování textových definic bayesovských sítí, vkládání informací do modelu, výběr otázek a získávání pravděpodobností stavů dovedností (což jsou v podstatě výsledky testu).

```
ro.r('''get.numbers.of.states <- function(model, node_vector) {
                nl <- model@nodes[node_vector]
                vv <- 1:length(nl@nodes)
                for(node in 1:length(nl@nodes)){
                    vv[node] <- nl@nodes[[node]]@number.of.states
                }
                vv
            }''')
r_get_numbers_of_states = ro.r['get.numbers.of.states']
```

Ukázka kódu 2: Funkce v *R* zahrnuté v kódu *Pythonu* z `inteRface/procedures.py`.

Vzhledem k tomu, že šlo o asi nejkomplikovanější část aplikace na implementaci, napsal jsem k modulu `inteRface`, konkrétně ke třídě `BayesNet`, jednotkové testy do souboru `/dev/test_bayesNet.py`.

2.2.3 Webové rozhraní

Samotnou interakci s uživatelem zajišťuje webové rozhraní. HTTP požadavky od uživatelů jsou obslouženy funkcemi definovanými v souboru `routing.py`. Pro názornost zde jednu uvedu a popíšu.

```
@app.route('/test/<int:test_id>')
@login_required
def test_overview(test_id):
    test: Test = get_entity_by_type_and_id(Test, test_id)
    check_test_existence_and_redirect_if_not_exists(test)
    test_instances = list_test_instances_by_test_and_student(test, current_user)
    possible_criteria = list_selection_criteria()
    test_instance_options_form = \
      TestInstanceOptionsForm(possible_criteria=[(sc.id, sc.name) for sc
                                                 in possible_criteria],
                              default_criterion=test.default_selection_criterion.id)
    return render_template("test_overview.html", test=test,
                           test_instances=test_instances,
                           test_instance_options_form=test_instance_options_form,
                           possible_criteria=possible_criteria)
```

Ukázka kódu 3: Funkce pro zobrazení přehledu testu z `routing.py`.

Na prvních dvou řádcích v ukázce kódu 3 jsou tak zvané dekorátory. První z nich zajišťuje propojení s příslušnou URI požadavku – argument dekorátoru je URI s tím, že pomocí podřetězce `<int:test_id>` je určeno, že na konci URI má být číslo, které

se má pak dekorované funkci předat jako argument. Druhý dekorátor zajišťuje, že pro zobrazení stránky je nutné, aby byl uživatel přihlášen. Provedením řádku 4 se do proměnné `test` nahraje objekt reprezentující test (třída je anotována dle [24]). Následuje kontrola, že test vůbec v databázi existuje (pokud ne, je uživatel přesměrován na seznam testů). Poté jsou do příslušných proměnných nahrány případy daného testu, výběrová kritéria a formulář pro zahájení nového případu testu. Nakonec funkce vrátí „vykreslenou“ šablonu `"test_overview.html"`.

Ostatní funkce definované v `routing.py` mají podobnou strukturu. Nejdříve se načtou do proměnných data z databáze, proběhne kontrola vstupů (fragmenty URI jako `test_id` výše a případně data z formulářů), jsou provedeny změny v databázi, je-li to třeba, nakonec je buď „vykreslena“ příslušná šablona a odeslána uživateli, nebo je uživatel někam přesměrován.

Šablony se vykreslují pomocí systému *Jinja* [19]. Všechny šablony jsou ve složce `templates`. Základní šablona (všechny ostatní jsou od ní odvozeny) tvoří soubor `common.html`, který také slouží jako šablona pro úvodní stránku.

Celá struktura používaných URI je navržena, jako kdyby šlo o REST API. Dokonce platí, že požadavky, které způsobují tvorbu nových záznamů v databázi musejí používat HTTP metodu `POST` (všude jinde se předpokládá metoda `GET`).

Tvorbu a validaci formulářů zajišťuje knihovna *WTForms* [18]. Také asi stojí za zmínku, že samotný web (tedy příslušný HTML a CSS kód) je psán pomocí CSS šablony *Skeleton* [20], díky které je „responzivní“, tedy měl by být použitelný i na mobilních zařízeních (telefony, tablety apod.).

Funkce pro vyřizování HTTP požadavků používají k přístupu k datovému modelu různé pomocné funkce a třídu `BayesNetDataModelWrapper` z modulu `common`. Třída `BayesNetDataModelWrapper` zapouzdřuje je jakýmsi pomyslným těžištěm aplikace, protože integruje operace v rámci `catest` s operacemi na datech v databázi. Pomocí metod této třídy protékají data mezi databází a *R*, případně nakonec i k uživateli jako hypertext.

2.3 Pracovní postup studenta

Student může začít na úvodní stránce, nebo může být odkázán přímo na požadovaný test (třeba `http://cat.utia.cas.cz:1080/test/1`). Poté by měl zahájit nový případ testu.

Po zahájení případu testu začne běžet čas testu (což je relevantní, pokud je nastavena nějaká maximální doba řešení testu). Student si může otázku vybrat sám ze seznamu, nebo využít možnosti automatického výběru dle kritéria.

Jakmile je dosaženo nějakého z ukončovacích kritérií (entropie na dovednostech klesne pod zadanou mez, je zodpovězen dostatečný počet otázek nebo vypší časový limit), je případ testu uzavřen. Výsledky v podobě pravděpodobností stavů dovednostních veličin jsou průběžně zobrazovány v přehledu případu testu.

2.4 Analýza časové náročnosti

Analyzoval jsem zběžně časovou náročnost operací nutných pro chod aplikace. Nejvíce uživatel vždy pocítí, pokud je k vyřízení jeho požadavku nutné použít `catest`. Nejvíce času v tomto případě aplikace stráví výpočty v rámci této knihovny. Sám vedoucí práce uvedl, že se při vývoji autoři `catest` nesnažili příliš optimalizovat časovou náročnost. Problémem je i fakt, že v *R* se zpravidla argumenty funkcí předávají kopírováním, což samozřejmě značně zvyšuje paměťové i výpočetní nároky.

Tyto problémy by šlo řešit třeba tak, že by se po zodpovězení otázky asynchronně (než student zodpoví další otázky) uvažovaly všechny možné scénáře vývoje testu několik kroků dopředu. Tyto scénáře i s příslušnými otázkami by se zaznamenávaly do databáze a výběr další otázky by pak znamenal jen vyhledání příslušného záznamu v databázi. Také by samozřejmě bylo možné provést alternativní implementaci funkcionality `catest`, která by byla výrazně rychlejší třeba v *C++* nebo *Julia* [26]. Další možností vývoje uvedu v následující sekci.

2.5 Možnosti budoucího vývoje

Největším problémem aplikace je, že uživatelské rozhraní nemá žádné nástroje pro analýzu výsledků testu (aspoň nic komplexnějšího než jen zobrazení pravděpodobností stavů dovedností). Nicméně dat o průběhu testů systém shromažďuje, dle mého názoru, dostatek, takže využít aplikaci již teď pro sběr dat bez větších problémů lze.

Celkově by asi bylo na místě vyvinou nějaké vlastní webové rozhraní pro zadávání a administraci testů. Zatím by bylo nejjednodušší pro zadávání testů do systému navrhnout nějaký strojově čitelný formát založený třeba na XML, nebo i prostý CSV soubor či JSON.

Co se týče hlubší analýzy výsledků, mělo by asi smysl uvažovat mimo jiné o analýze častých sledů otázek, korelace stavů otázek, rychlost poklesu entropie v modelu (nebo v jeho částech) nebo třeba o nějakém souhrnu výsledků daného testu (na příklad histogram pravděpodobností stavů).

Závěr

V první kapitole jsem krátce uvedl koncept adaptivního počítačového testování a popsal jsem krátce i možnosti jeho realizace pomocí bayesovských sítí. Na příkladu jsem popsal návrh adaptivního testu a popsal jsem několik strategií pro výběr otázek.

V druhé kapitole jsem popsal aplikaci, kterou jsem v rámci výzkumného úkolu vyvíjel a její vztah k systému pro adaptivní testování, vyvíjeného na Ústavu teorie informace a automatizace AV ČR, do kterého zapadá. Vysvětlil jsem výběr nástrojů, které jsem zvolil pro implementaci a popsal strukturu aplikace.

Aplikace nyní neposkytuje, byť to požadovalo zadání, žádné komplexní analytické výstupy. Na druhou stranu zadání specifikovalo, že by aplikace měla sloužit k testování znalostí studentů středních škol v oblasti matematiky, vyvinul jsem ovšem systém, který nepředpokládá žádnou konkrétní doménu znalostí k testování, pokud je možné otázky koncipovat výhradně jako uzavřené otázky.

Za největším přínos samotného textu považuji poměrně podrobnou dokumentaci systému. V textu je také uvedeno několik návrhů pro další vývoj aplikace třeba v rámci navazující diplomové práce.

Literatura

[1] ALMOND, R.G., MISLEVY, R.J., STEINBERG, L., YAN, D., WILLIAMSON, D. *Bayesian Networks in Educational Assessment.* Springer, 2015.

[2] VOMLEL, Jiří *Bayesian networks in educational testing.* International Journal of Uncertainty, Fuzziness and Knowledge Based Systems. Vol. 12, Supplementary Issue 1, 2004, pp. 83-100.

[3] VOMLEL, Jiří *Building Adaptive Tests using Bayesian networks.* Kybernetika. Volume 40, Number 3, 2004, pp. 333 - 348.

[4] PLAJNER, Martin, Jiří VOMLEL. *Student Skill Models in Adaptive Testing.* Proceedings of the Eighth International Conference on Probabilistic Graphical Models, pp. 403–414, 2016.

[5] JENSEN, Finn V. a Thomas D. NIELSEN. *Bayesian networks and decision graphs*. 2nd ed. New York: Springer, c2007. ISBN 978-0-387-68282-2.

[6] WAINER, Howard. a Neil J. DORANS. *Computerized adaptive testing: a primer*. 2nd ed. Mahwah, N.J.: Lawrence Erlbaum Associates, 2000. ISBN 08-058-3511-3.

[7] PLAJNER, Martin, Jiří VOMLEL. *Bayesian network models for adaptive testing.* Proceedings of the Twelfth UAI Conference on Bayesian Modeling Applications Workshop-Volume 1565. CEUR-WS. org, 2015. p. 24-33.

[8] PLAJNER, Martin, Amal MAGAUINA a Jiří VOMLEL. *Question Selection Methods for Adaptive Testing with Bayesian Networks*. Ústav teorie informace a automatizace, Akademie věd České republiky, 2017.

[9] *R: The R Project for Statistical Computing* [online]. Vídeň: The R Foundation for Statistical Computing, 2017 [cit. 2017-08-31]. Dostupné z: https://www.r-project.org/

[10] *Python* [online]. Beaverton: Python Software Foundation, 2017 [cit. 2017-08-31]. Dostupné z: https://www.python.org/

[11] *R Shiny* [online]. Boston: RStudio, 2017 [cit. 2017-08-30]. Dostupné z: https://shiny.rstudio.com/

[12] *Flask: A python microframework* [online]. Pocoo Team, 2017 [cit. 2017-08-31]. Dostupné z: http://flask.pocoo.org/

[13] *RPy2* [online]. Laurent Gautier et al., 2017 [cit. 2017-08-31]. Dostupné z: https://rpy2.readthedocs.io

[14] *SQLAlchemy: The database toolkit for Python* [online]. 2017 [cit. 2017-08-31].

Dostupné z: http://www.sqlalchemy.org/

[15] *SQLite* [online]. 2017 [cit. 2017-08-31]. Dostupné z: https://www.sqlite.org

[16] *Babel* [online]. Babel Team, c2017 [cit. 2017-08-30]. Dostupné z: http://babel.pocoo.org/en/latest/

[17] *Flask-Security* [online]. Matt Wright, c2012 [cit. 2017-08-31]. Dostupné z: https://pythonhosted.org/Flask-Security/

[18] *WTForms* [online]. WTForms Team, c2010 [cit. 2017-08-31]. Dostupné z: http://wtforms.readthedocs.io/

[19] *Jinja2* [online]. Armin Ronacher, c2014 [cit. 2017-08-31]. Dostupné z: http://jinja.pocoo.org/

[20] *Skeleton: Responsive CSS boilerplate* [online]. Dave Gamache et al., 2014 [cit. 2017-08-31]. Dostupné z: http://getskeleton.com/

[21] *HUGIN* [online]. Aalborg: HUGIN EXPERT, 2017 [cit. 2017-08-31]. Dostupné z: http://www.hugin.com/

[22] Object-relational mapping. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-08-30]. Dostupné z: https://en.wikipedia.org/wiki/Object-relational_mapping

[23] Comparison of multi-paradigm programming languages. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2017 [cit. 2017-08-30]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Comparison_of_multi-paradigm_programming_languages&oldid=797311914

[24] PEP 484: Type Hints. *Python* [online]. Python Foundation, 2014 [cit. 2017-08-31]. Dostupné z: https://www.python.org/dev/peps/pep-0484/

[25] CPython. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2017-08-31]. Dostupné z: https://en.wikipedia.org/w/index.php?title=CPython&oldid=797221729

[26] *Julia Language* [online]. Austin: NumFocus, 2017 [cit. 2017-08-31]. Dostupné z: https://julialang.org/

Instalace a rozhraní přes příkazový řádek

V této sekci krátce popíšu možný postup instalace systému pro vyzkoušení. Budu zde předpokládat instalaci na nějakou moderní distribuci *GNU/Linux* (třeba *Fedora*, *CentOS*, *ArchLinux*, *Antergos*). Dále uvedu několik příkazů, které aplikace podporuje (ze souboru `cli.py`).

Nejdříve je nutné zajistit, že jsou nainstalovány všechny závislosti. Vzhledem k tomu, že, jak jsem zmínil výše, používám typové anotace [24], je nutné mít nainstalovaný *Python* verze aspoň `3.5`. Pro chod knihovny *RPy2* je naneštěstí nutné používat interpreter *CPython* [25]. Dále je potřeba mít nainstalovanou dostatečně novou verzi interpreteru *R* (v moment psaní mám nainstalovanou verzi `3.4.1`). Také je potřeba mít k dispozici nějaký relační databázový systém. Bez zásahu do `CArisTotle/config` lze použít zmiňovaný *SQLite*.

Nejjednodušší asi je si naklonovat repozitář na `https://github.com/FNj/CArisTotle` a přesunout se do příslušné složky pomocí:

```
git clone https://github.com/FNj/CArisTotle
cd CArisTotle
```

poté, byť to není nezbytně nutné, je vhodné vytvořit si pro aplikaci virtuální prostředí *Pythonu* Poté je potřeba nainstalovat knihovnu `catest` a další knihovny *R*, na kterých je závislá. Vše potřebné je ve složce `dependencies/R`. Stačí si spustit interpreter (příkaz `R`) a zadat příkaz:

```
source("dependencies/R/install_deps.R")
```

Budete vyzvání k výberu repozitáře, ze kterého se mají příslušné balíčky nainstalovat. Taky se pak nelekněte spousty varování compileru *C++*.

Poté, byť to není nezbytně nutné, je vhodné vytvořit si pro aplikaci virtuální prostředí *Pythonu* třeba jako:

```
python3 -m venv VEnv
```

a aktivovat si ho:

```
source VEnv/bin/activate
```

Následně si nainstalujte potřebné knihovny *Pythonu*:

```
pip install -r requirements.txt
```

Poté už lze začít používat samotnou aplikaci. Nejdříve je potřeba nastavit proměnnou ukazující na modul s aplikací:

```
export FLASK_APP=CArisTotle/__init__.py
```

dále je potřeba inicializovat databázi (vytvoření schéma, naplnění číselníkových tabulek):

```
flask initdb
```

pak je pro testování nutné naplnit databázi i testovacími daty:

`flask testdata`

a teď už lze aplikaci spustit:

`flask run`

Až skončíte práci s aplikací, můžete databázi snadno promazat:

`flask dropall`

Pro nahrávání testů lze použít třeba skript jako `CArisTotle/dev/test_data.py`.