Univerzita Pavla Jozefa Šafárika v Košiciach

Prírodovedecká fakulta

ODPORÚČANIE S DOZOROM V INTELIGENTNOM SYSTÉME NA VÝUČBU PROGRAMOVANIA

ŠVK PRÁCA

Študijný odbor:	Informatika
Školiace pracovisko:	Ústav informatiky
Vedúci záverečnej práce:	RNDr. Tomáš Horváth, PhD.
Konzultant:	RNDr. Štefan Pero

Košice 2015

Bc. Tomáš Nguyen

Poďakovanie

Rád by som poďakoval vedúcemu diplomovej práce RNDr. Tomášovi Horváthovi, PhD. a aj konzultantovi RNDr. Štefan Perovi za cenné pripomienky a za obetavosť počas tvorby mojej diplomovej práce.

Abstrakt

V práci prezentujeme webový edukačný systém na výučbu programovania. Rozoberáme základnú myšlienku systému ako aj jeho štruktúru a technickú stránku. Analyzujeme rôzne spôsoby na získavanie informácií o znalostiach študentov z ich riešení. V ďalšej časti analyzujeme viaceré spôsoby odporúčania. Nakoniec rozoberáme experimenty vykonané v našom systéme ako aj ich výsledky.

Abstract

In our work we present web-based education system for teaching programming. We analyse the main idea of the system as well as it's structure and technical side. We then focus on multiple methods of extraction of information about student knowledge. In next part we analyse different methods of recommendation. In the end we discuss experiments preformed in our system as well as their results.

Obsah

Úvod

Štandardný spôsob výučby programovania v dnešnej dobe na školách je taký, že si študenti preštudujú teoretickú časť a následne si tieto poznatky precvičujú prakticky programovaním úloh. Na to aby študent vedel ako má postupovať a či jeho výsledky sú správne, potrebuje pomoc učiteľa. V ideálnom prípade učiteľ študentovi úlohy skontroluje a usúdi či je jeho riešenie správne. Na základe jeho znalostí mu potom vyberá úlohy, ktoré pre neho nie sú veľmi ľahké ale ani veľmi ťažké. Pri príliš ľahkých úlohách sa študentove programovacie zručnosti sa už veľmi nevyvíjajú a pri ťažkých sa zasekne a prakticky sa nič nové nenaučí. Ideálne by mal byť výber úloh osobitný/personalizovaný pre každého študenta, keďže študenti sú rôzni. Z tohto všetkého môžeme usúdiť, že úloha učiteľa je veľmi dôležitá. Nastáva tu však jeden problém, učiteľov je zvyčajne menej ako študentov a často je ten pomer niekoľkonásobný. Tým pádom učiteľ musí vynaložiť zväčšené úsilie, aby sa mohol venovať každému študentovi dostatočne, pričom to aj tak nemusí stíhať.

Webové edukačné systémy sa postupne stávajú populárnejšími než tradičné papierové učebnice [1]. Intelligent tutoring systemy (ITS) ako aj educational recommender systemy (ERS) sa zameriavajú na vedenie študentov pomocou rôznych úloh [2][3]. ITS sa ukázali byť efektívnejšie v zvyšovaní schopností študentov v porovnaní s tradičnými edukačnými metódami. Dôležitý kľúčový prvok, ktorým sa ITS odlišuje od Computer Assisted Instruction (CAI) systému je schopnosť dynamicky uchovávať model znalostí študenta počas štúdia [4].

Cielom tejto práce je vytvorenie edukačného systému na výučbu programovania. Názov nášho ITS systému je Intelligent Programming tutor (IPT). Systém má za úlohu detegovať úroveň študentových znalostí, správne tieto údaje reprezentovať a na základe toho odporučiť študentovi úlohu. Študent bude teda vedený štúdiom pomocou odporúčania. Odporúčanie je personalizované, čiže sa berú do úvahy študentove schopnosti a zručnosti. Takýto systém dokáže výrazne ušetriť čas ako študentom tak aj učiteľom.

Kapitola 1

Intelligent programming tutor

1.1 Idea systému IPT

Základná myšlienka systému je, že sa programovanie bude učiť pomocou riešenia úloh. Každá úloha precvičuje určitú množinu zručností. Študenti po prihlásení do systému môžu riešiť úlohy. Je dôležité podotknúť, že systém samotný neposkytuje študijne materiáli pre študentov, aspoň nie v terajšej podobe. Predpokladáme základné teoretické znalosti študentov z oblasti, ktorú si chcú prakticky precvičiť.

Systém je teda praktickým „cvičiteľom“ v bežnom štúdiu programovania, kde študenti dostanú teoretické znalosti od učiteľov, napríklad formou prednášok, a praktické zručnosti si precvičia v systéme IPT.

Študenti si vyberú úlohu, vyriešia ju a jej riešenie zašlú na server. Učiteľ následne riešenie opraví a pridelí mu hodnotenie vyjadrené počtom bodov a komentárom. Systém na základe hodnotení úloh, ktoré študent dostal od učiteľa, posúdi či študent ovláda konkrétne zručnosti alebo nie. Podľa toho mu potom odporučí ďalšie úlohy, tak aby jeho štúdium prebiehalo čo najhladšie. Riešenia samotné sú odovzdávané formou zdrojových kódov.

Môžeme pozorovať, že systém nie je plne automatizovaný a je závislý od hodnotenia učiteľa. Spočiatku sme mali návrh v ktorom figuroval aj automat – evaluátor, ktorý by automaticky hodnotil riešenia študentov. Od tohto konceptu sme však upustili a to hlavne kvôli tomu, že evaluátory nie sú dostatočne presné, keďže väčšina z nich pracuje spôsobom kontroly výstupov z kódu. Nekontrolujú ako je kód napísaný a taktiež neberú do úvahy okrajové prípady či menšie chyby študentov, ktoré sú v často zanedbateľné. Pre tento účel je dobrý učiteľ nenahraditeľný.

Najdôležitejším problémom, ktorý sme museli riešiť je odporúčanie, keďže na ňom

je založený celý systém. Úlohy sa odporúčajú na základe zručností, ktoré študent ovláda. Túto informáciu systém získa z hodnotení úloh. Ak určité zručnosti robia študentovi problémy, tak mu systém odporučí podobné úlohy. V prípade že ovláda zručnosti potrebné pre zložitejšie zručnosti, tak mu začne odporúčať nové úlohy. Podrobnejšie sa budeme odporúčaniu venovať v neskoršej kapitole.

1.2 Funkcionalita IPT

Začneme s definovaním užívateľov. V IPT sú tieto typy užívateľov:

- študent
- učiteľ
- manažér
- administrátor

Študenti sa môžu v systéme prihlásiť na predmet, prezerať a riešiť úlohy. Následne si môžu prezrieť hodnotenia, ktoré dostali od učiteľom.

Učitelia sú tí, ktorí môžu hodnotiť riešenia študentov v danom predmete. Učiteľov pre konkrétny predmet vyberá manažér predmetu.

Manažér je vedúci predmetu, ktorý môže v upravovať aspekty predmetu, pridávať a odstraňovať úlohy, ale môže tiež aj hodnotiť riešenia študentov ako učitelia. V IPT rozlišujeme medzi pojmami manažér a užívateľ s manažérskymi právami. Užívateľ s manažérskymi právami, môže v systéme vytvárať nové predmety a byť ich manažérom. Je dôležité dodať, že užívateľ je manažérom len tých predmetov, ktoré reálne vytvoril.

Administrátor je osoba s administrátorskými právami. Môže odstraňovať alebo zablokovať ostatných užívateľov a prideľuje manažérske práva.

Predtým ako môže študent riešiť úlohy alebo si ich iba pozrieť, musí sa do systému zaregistrovať. Registruje sa pomocou emailu a hesla, pričom sa zadávajú aj iné osobné údaje – meno, škola, atď. Po registrácii študentovi dostane email s pomocou ktorého aktivuje svoje konto v systéme.

Predmety v IPT predstavujú väčší celok, ktorý sa vyučuje. Môže ísť napríklad o predmety v rôznych programovacích jazykoch (C, C++, Java, C# ...) alebo rozdelenie výučby do menších celkov podľa zložitosti (programovanie v C pre začiatočníkov, programovanie v C pre pokročilých ...).

Každá úloha pozostáva z názvu, popisu, maximálneho počtu bodov a zručností, ktoré precvičuje. Tieto úlohy môže študent riešiť hneď po prihlásení do systému.

Riešenia študent zasiela formou zdrojových kódov, napríklad súbory .java, .cc a podobne. Riešenie môže obsahovať aj viacero súborov. Jednu úlohu môže študent vyriešiť aj viackrát, tj. zašle riešenie viackrát, ale pri hodnotení sa bude zohľadňovať len riešenie zaslané ako posledné. Študentovi ako aj učiteľovi sa pri prezeraní riešenia zobrazí zdrojoví kód na stránke vo forme textu, aby nemuseli súbory sťahovať.

Hodnotenia prideľujú učitelia a manažéri jednotlivým riešeniam osobitne. Zadáva sa počet bodov a slovný komentár. Po pridelení hodnotenia sa študentovi zobrazí na hlavnej stránke oznam s linkou k tomuto hodnoteniu.

1.3 Technická stránka IPT

Po základnom návrhu systému sme ho museli začať implementovať. Ako programovací jazyk sme si zvolili PHP a to hlavne kvôli jeho rozšírenosti na internete. Na implementáciu samotnej aplikácie sme použili PHP framework Yii (skratka pre Yes it is). Tento framework sme si vybrali z dôvodu jeho jednoduchosti, rýchlosti, množstva ponúkaných nástrojov a funkcii, ale hlavne rýchlou a jednoduchou prácou s databázami. Využíva MVC architektúru – model, view, controler. Yii ďalej podporuje viacjazyčnosť, čiže náš systém môže fungovať vo viacerých jazykoch. Momentálne to je slovenčina a angličtina.

Pre IPT sme sa rozhodli použiť relačnú databázu MySQL, ktorá je jednou z najpopulárnejších databáz. V samotných stránkach využívame HTML5 prvky, ktoré sa stávajú novým štandardom na internete. Na dizajn sme použili CSS3 a využívame aj JavaScript avšak iba tam kde je to potrebné – aby sme nemali stránky zaplavené mnohými skriptami.

Kapitola 2

Model študenta

Predtým než sme mohli začať s odporúčaním, sme museli získať informácie o študentovi. Tieto informácie budú tvoriť našu doménu študenta. Konkrétne sme sa zameriavali na jeho získané vedomosti, čiže zručnosti. Museli sme brať do úvahy zručnosti študenta a odporúčať tak, aby ho to viedlo správnym smerom v štúdiu.

V procese získavania informácii o študentoch sme riešili tieto základné problémy:

1. Ako zručnosti v systéme reprezentovať
2. Ako detegovať zručnosti z riešení
3. Ako určiť znalosť zručnosti študenta
4. Ako tieto informácie o zručnostiach použiť pri odporúčaní úloh

2.1 Reprezentácia zručností

Zručnosti v IPT reprezentujeme pomerne jednoducho a to pomocou ID, názvu a typu. Všetky zručnosti rozdeľujeme do troch typov: programovanie, algoritmy a matematika.

Id	Názov zručnosti	Typ
1	Operácie s premennými	1
2	Rekurzia	2
3	Faktorial	3

Tabuľka 2.1: Tabuľka zručností

Ďalej využívame tabuľku potrebných zručností 2.2 na určenie konkrétnych zručností pre každú úlohu. Každá úloha je teda vektor čísel $U = (u_1, u_2, u_3, ..., u_n)$, kde u_i je 0 alebo 1, tj. či úloha precvičuje zručnosť i.

	Podmienky	Vytváranie metód	Volanie metód	Rekurzia
Úloha 1	0	1	1	0
Úloha 2	1	1	0	0
Úloha 3	0	0	0	1

Tabuľka 2.2: Tabuľka potrebných zručností

V reálnej databáze by sme však mali many-to-many reláciu, keďže jedna úloha môže vyžadovať viac zručností a jedna zručnosť môže byť vyžadovaná vo viacerých úlohách, preto využívame prechodovú tabuľku problem-skill (Obr. 4.3).

Ďalšia veľmi dôležitá tabuľka je tabuľka závislostí 2.3, ktorá vyjadruje závislosti medzi jednotlivými zručnosťami. Napríklad: zručnosť rekurzia je výsledkom ovládania zručností vytvárania metód, volanie metód, podmienky.

	Podmienky	Vytváranie metód	Volanie metód	Rekurzia
Podmienky	0	0	0	0
Vytváranie metód	0	0	0	0
Volanie metód	0	0	0	0
Rekurzia	1	1	1	0

Tabuľka 2.3: Tabuľka závislostí

Problémom pri tabuľke závislostí je hlavne určiť správnu topológiu, no v našej práci predpokladáme, že to zabezpečí doménový expert.

Nakoniec samotné informácie o stave znalostí študentov uchovávame v tabuľke zručností študentov 2.4 (ďalej TZS). Každý študent je teda vyjadrený vektorom $S_i = (x_1, x_2, x_3, ..., x_n)$ kde n je počet všetkých zručností a x_i je miera do akej zručnosť i ovláda.

	Podmienky	Vytváranie metód	Volanie metód	Cykly
Študent 1	0.3	0.3	0.5	0.3
Študent 2	0.1	0.0	0.2	0.0
Študent 3	0.2	0.3	1.0	0.0
Študent 4	0.2	0.3	0.2	1.0
Študent 5	0.8	0.5	0.4	0.5

Tabuľka 2.4: Tabuľka znalostí študentov (TZS)

2.2 Detekcia zručností

Prvým problémom pri odporúčaní bola správna detekcia zručností. To znamená, že z riešenia sme museli presne zistiť, ktoré zručnosti použil študent správne (ďalej už len "dobré zručnosti") a ktoré nesprávne (ďalej už len "zlé zručnosti"). Správne určenie študentových znalostí je nevyhnutné pre správne odporúčanie.

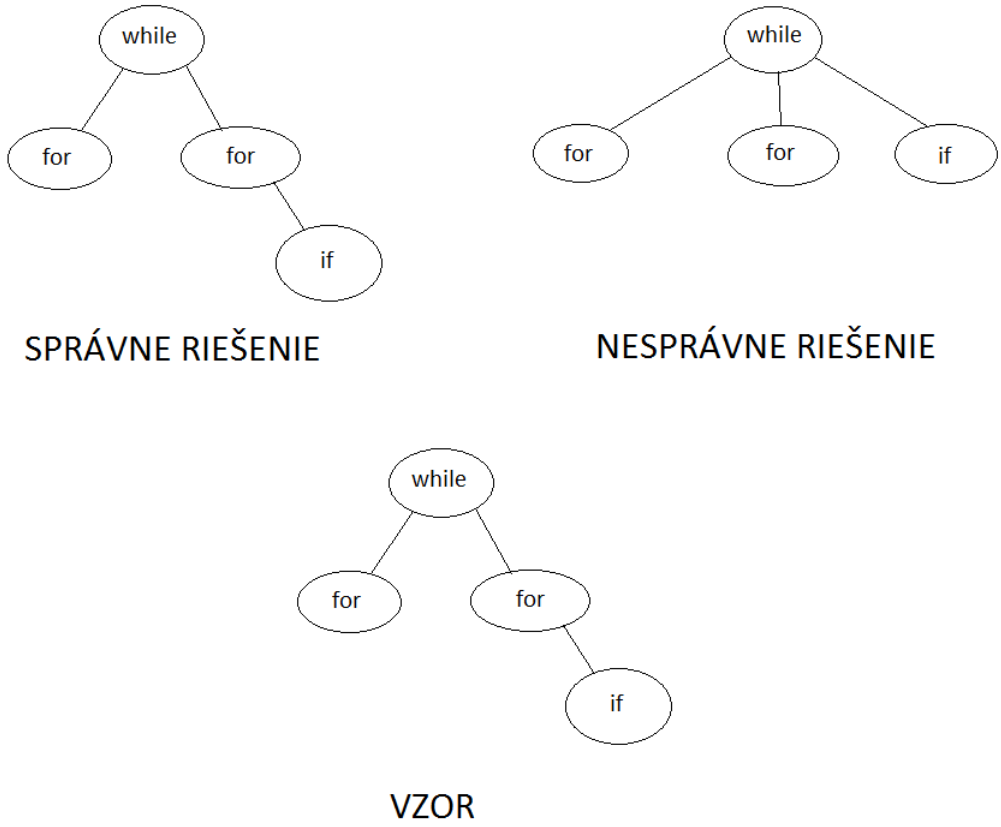
Uvažovali sme tri prístupy:

- detekcia pomocou vzorového riešenia
- detekcia pomocou hodnotenia
- manuálne priradenie
- použiť všetky

2.2.1 Detekcia pomocou vzorového riešenia

Pri detekcii pomocou vzorového riešenia by zadávateľ úlohy zadal aj vzor podľa ktorého by sa posudzovalo či je úloha správna. Algoritmus tejto detekcie bol nasledovný:

1. učiteľ zadá úlohu a aj vzorové riešenie (formou zdrojového kódu)
2. vzorové riešenia sa prevedie do porovnateľného tvaru – napríklad strom
3. každé riešenie študenta by sa po odoslaní takisto previedlo na strom a porovnávalo by sa so vzorom
4. ak sa riešenie zhoduje so vzorom tak je riešenie správne a všetky zručnosti úlohy označíme ako dobré zručnosti



Obr. 2.1: Prístup vzorového riešenia

Výhody tohto prístupu:

- systém je viac automatizovaný, čiže nevyžaduje si obzvlášť veľkú interakciu učiteľa
- spolieha sa len na riešenia študentov, nepotrebuje hodnotenie učiteľov

Nedostatky sú však výraznejšie:

- Prvým je fakt, že väčšina úlohy môže mať viacero správnych riešení, ktoré vyzerajú úplne odlišne. Tento problém by sa dal odstrániť pridávaním dodatočných vzorov. Učiteľ by mohol pri zadávaní úlohy priložiť viacero vzorových riešení ale bolo by možné dodávať vzory dodatočne z riešení študentov. To by však vyžadovalo ďalšiu interakciu učiteľa, ktorý by musel explicitne povedať, ktoré riešenie môže byť použité ako vzor.
- Druhým a aj naším najväčším problémom je fakt, že sa nedá v tomto prístupe

detegovať čiastočná správnosť úlohy a tým pádom aj čiastočné ovládanie zručností

Ako príklad použijeme nasledovnú situáciu: Úloha $U = (1, 1, 0, 0, 1)$ precvičuje zručnosti s_1, s_2, s_5. Študent perfektne ovláda s_1 a s_2 ale má problémy so zručnosťou s_5. Tomuto faktu bude zodpovedať aj jeho riešenie – nebude úplne správne. Systém potom z tohto riešenia usúdi, že riešenie je nesprávne aj keď reálne ovláda dve z troch zručností.

Tento efekt je v našom systéme nežiaduci nakoľko sa snažíme o presnejšie detegovanie zručností. Z tohto dôvodu sme sa rozhodli tento prístup nepoužiť a zamerali sme sa na prístup s väčšou interakciou učiteľa.

2.2.2 Detekcia pomocou hodnotenia

Pri detekcii pomocou hodnotenia je systém závislý od hodnotenia učiteľa a nie od riešenia samotného. Tento prístup uvažuje väčšiu interakciu učiteľa, keďže musí opraviť a ohodnotiť každé jedno riešenie. Pripomíname, že hodnotenie pozostáva z bodového a slovného hodnotenia. V tomto prístupe je dôležité len to bodové. Systém samotný bude vykoná detekciu až potom ako bude riešeniu pridelené hodnotenie.

Prvý algoritmus nad ktorým sme uvažovali je pomerne jednoduchý:

1. Zadáme hranicu *treshold* z intervalu (0,1)
2. Systém vyráta úspešnosť riešenia pomocou
$$\frac{ZiskaneBody}{MaxBodov}$$
3. Zistí, ktoré zručnosti úloha precvičovala
4. Ak je úspešnosť riešenia $\geq$ ako *treshold* tak sa všetky zručností precvičované úlohou označia ako dobré zručnosti.

V tomto algoritme nie je už prítomný problém neuznania rôznych riešení, keďže sme túto úlohu prenechali učiteľovi. No problém s čiastočnou správnosťou riešení a tým pádom aj čiastočnou znalosťou zručnosti pretrváva. Tento problém sme sa snažili vyriešiť v druhom algoritme:

1. Systém vytvorí *bodovú časť* (BC) pre jednu zručnosť zo všetkých, ktoré úloha precvičuje pomocou rovnice

$$BC = \frac{MaxBodov}{PocetZrucnosti}$$

2. Vyráta počet chýbajúcich bodov riešenia CH

$$CH = MaxBodov - ZiskaneBody$$

3. Vyráta počet *zlých zručností* ZZ - tých ktoré neovláda

$$ZZ = \lfloor \frac{CH}{BC} \rfloor$$

4. Ak $ZZ > 0$ tak:

   (a) Utriedime všetky zručnosti úlohy podľa hodnôt v TZS pre konkrétneho študenta od najmenšieho po najväčšie

   (b) Vyberieme prvých ZZ s najmenšou hodnotou a dáme ich do poľa *Neovláda*

   (c) Všetky ostatné dáme do poľa *Ovláda*

Základnou ideou tohto algoritmu je približné určenie počtu zlých zručností a určiť ich z TZS podľa ich hodnoty. Na to používame jednoduchý princíp rozdelenia zručností na rovnomerné bodové úseky. Následne sa z počtu chýbajúcich bodov určí daný počet zlých zručností. Tu však určujeme len počet. Ďalej podľa hodnôt zručností v tabuľke nájdeme x najmenších a tieto zručnosti predstavujú tie, ktoré s najväčšou pravdepodobnosťou neovláda nakoľko je ich hodnota najmenšia (a tým pádom neboli veľmi precvičené).

2.2.3 Manuálne priradenie

Na odporúčanie potrebujeme mať presné informácie o tom, do akej miery študenti ovládajú konkrétne zručnosti. Manuálne priradzovanie je najpresnejší ale aj najprácnejší spôsob. Učiteľ bude pri hodnotení riešenia manuálne určoval, ktoré z precvičovaných zručností študent v riešení správne použil.

Tento prístup sa oproti predchádzajúcim spolieha výlučne na úsudok učiteľa. Predpokladáme, že učiteľ dokáže správne ohodnotiť riešenia a určiť správne použité zručnosti. Tým pádom je presnosť detegovanie zručností oveľa väčšia.

Takýto prístup môže dokonca pomôcť samotným učiteľom pri hodnotení. Pri normálnom hodnotení si totiž musí učiteľ prezrieť riešenie a ohodnotiť ho len pomocou bodov a poprípade nejakého komentára. Pri našom prístupe však má k dispozícii množinu zručností, ktoré úloha precvičuje a tak reálne môže svoje hodnotenie opierať aj o to koľko a aké zručnosti vidí v riešení ako dobre použité.

2.3 Určenie znalostí študenta

Ako sme už spomínali v sekcii 2.2, informácie o znalostiach študentov uchovávame v tabuľke TZS. Lenže spôsobov ako tieto údaje reprezentovať, vytvárať a upravovať je viacero.

2.3.1 Prvý prístup

Ideou tohto prístupu je to, že po získaní dobrých a zlých zručností z riešenia študenta sa hodnoty dobrých zručností v TZS zvýšia o 1 a zlých zručností znížia o 1. Hodnoty v TZS sú z intervalu $(0, \infty)$. Čím je hodnota zručnosti vyššia tým viac ju študent ovláda.

Problémom pri tomto prístupe je fakt, že nevieme z týchto hodnôt spoľahlivo určiť ako veľmi študent tieto zručnosti ovláda, keďže hodnoty sú z hora neohraničené a ich hodnota sa môže aj znižovať. Keby sme iba zvyšovali dobré zručnosti a zle by sme nemenili, tak by hodnoty predstavovali počet správne vyriešených úloh s danou zručnosťou. Toto by však tiež nepomohlo nakoľko rôzne zručnosti môžu byť precvičované rôznym počtom úloh.

2.3.2 Druhý prístup

Hlavnou myšlienkou tohto prístupu je určovanie hodnôt v TZS na základe zložitosti jednotlivých zručností. Majme teda tuto všeobecnú TZS 2.5.

	Podmienky	Vytváranie metód	Volanie metód	Cykly
Študent 1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
Študent 2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
Študent 3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
Študent 4	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$

Tabuľka 2.5: Tabuľka zručností študentov - TZS

Hodnota $x_{i,j} \in \langle 0, 1 \rangle$ určuje do akej miery študent zručnosť ovláda. Pre hodnotu $x_{i,j}$ platí nasledujúca rovnica.

$$x_{i,j} = \frac{v_{i,j}}{u_j} * b_j$$

- i - id študenta
- j - id zručnosti
- $v_{i,j}$ - počet úloh so zručnosťou j, v ktorých študent i použil zručnosť j správne
- u_j - počet všetkých úloh so zručnosťou j
- b_j - bias zručnosti j. Táto hodnota určuje zložitosť zručnosti.

V našom prípade zložitosť b_j bude väčšia pre ťažšie a menšia pre ľahšie zručnosti. Hodnotu b_j určíme pomocou inverse document frequency (IDF) a počítame ju pomocou tejto rovnice.

$$b_j = \frac{N}{|\{u \in U : j \in u\}|}$$

- j - id zručnosti
- u - úloha
- U - množina všetkých úloh
- N - počet všetkých úloh

Hodnota b_j musíme ešte znormovať keďže výsledná hodnota $x_{i,j} \in \langle 0, 1 \rangle$. Všeobecne platí, že ľahké zručnosti sú spojené so základom programovania ako napríklad cykly, podmienky alebo volanie metódy. Takéto zručnosti sa budú vyskytovať vo veľkom počte úloh a ich b_j bude malé. Naopak zložitejšie zručnosti sú často kombináciou

viacerých ľahších zručností a precvičujú sa len v špecifických úlohách a tým pádom ich b_j bude väčšie.

Po pridelení hodnotenia riešeniu študenta i sa v systéme spustí algoritmus na prepočítanie hodnôt $x_{i,j}$ v TZS ktorý prebieha nasledovne:

1. Z = *množina všetkých zručností vybrané učiteľom*
2. $Z = Z \cup$ {*všetky zručnosti od ktorých sú zručnosti v Z závislé*}
3. foreach (Z as j)
   (a) Vypočítaj hodnotu $x_{i,j}$ a zapíš ju do TZS

Prepočítavanie hodnoty $x_{i,j}$ sa nevykonáva len pri hodnotení ale aj pri pridávaní novej úlohy nakoľko hodnota $x_{i,j}$ potrebuje počet úloh so zručnosťou j.

2.3.3 Tretí prístup

V tomto prístupe sa menia hodnoty v TZS na základe úspešnosti riešenia a tiež počte vyriešených úloh s danou zručnosťou. V tomto prípade nerozlišujeme medzi dobrými a zlými zručnosťami ale využívame všetky rovnako.

Reprezentujme všetky zručnosti študenta v TZS ako vektor $S_i = (x_1, x_2, x_3, ..., x_n)$ kde n je počet všetkých zručností. Výpočet hodnoty x_j je nasledovný:

1. Systém vyráta úspešnosť riešenia pomocou rovnice
$$Uspesnost = \frac{ZiskaneBody}{MaxBodov}$$
2. Nech R_j, kde j =(id precvičovaných zručnosti), sú množiny ohodnotených úloh študenta obsahujúce zručnosť j
3. Nech sum_j je súčet úspešností hodnotených úloh z R_j
4. Nech $pocetUloh_j$ je počet úloh v množine R_j
5. Potom hodnotu x_j vyrátame pomocou rovnice
$$x_j = \frac{sum_j}{pocetUloh_j}$$

Príklad: uvažujme tento scenár. Majme tri úlohy, ktoré študent i vyriešil:

- U_1 = (1,1,0,0,0) - 0.7 úspešnosť
- U_2 = (0,1,0,1,0) - 0.3 úspešnosť
- U_3 = (0,0,1,1,1) - 1.0 úspešnosť

Jeho vektor teda bude:

$$S_i = \left(\frac{0.7}{1}, \frac{0.7+0.3}{2}, \frac{1}{1}, \frac{0.3+1}{2}, \frac{1}{1}\right) = (0.7, 0.5, 1, 0.65, 1)$$

Teraz majme úlohu U_4 = (1,0,0,1,1) ktorú študent i práve vyriešil. Učiteľ dal riešeniu celkovo 6 bodov z 8, čiže úspešnosť je 0.75. Jeho vektor sa teraz zmení na:

$$S_i = \left(\frac{\mathbf{0.7+0.75}}{\mathbf{2}}, \frac{0.7+0.3}{2}, \frac{1}{1}, \frac{\mathbf{0.3+1+0.75}}{\mathbf{3}}, \frac{\mathbf{1+0.75}}{\mathbf{2}}\right) = (\mathbf{0.725}, 0.5, 1, \mathbf{0.68}, \mathbf{0.875})$$

2.4 Konečné využitie v systéme

V konečnej verzii systému sme sa rozhodli nedetegovať zručnosti jednotlivo podľa ich spávneho využitia študentom ale využívame všetky precvičované zručnosti danou úlohou. Čiže nevyužívame ani jeden z vyššie uvedených prístupov no namiesto toho používame najjednoduchšie možné riešenie. Pre určovanie znalosti študentov využívame prístup na základe úspešnosti (opísané v 2.3.3).

2.4.1 Ukážkový postup

Majme zručnosti $s_1, s_2, s_3, ..., s_n$ a študenta $S = (x_1, x_2, x_3, ..., x_n)$, kde hodnota x_i je z intervalu $(0, 1)$. Študent S si vyberie úlohu $U = (u_1, u_2, u_3, ..., u_n)$, ktorú aj následne vyrieši. Po vyriešení úlohy zašle zdrojové kódy na server pomocou užívateľského rozhrania v IPT. Server tieto súbory uloží a umožní učiteľovi riešenie ohodnotiť. Pri hodnotení učiteľ zadáva slovný komentár a počet bodov. Po vytvorení hodnotenia systém vyráta úspešnosť riešenia. Následne sa na základe tejto úspešnosti zmenia študentove hodnoty v jeho vektore S.

Kapitola 3

Odporúčanie

Po získaní korektných informácií o vedomostiach študentov sme museli riešiť akým spôsobom budeme úlohy odporúčať. V IPT je každá úloha definovaná vektorom $U = (u_1, u_2, u_3, ..., u_n)$. Proces odporúčania pozostáva z 2 častí. V prvej časti sa realizuje zostavenie množiny odporúčaných úloh. Keďže predpokladáme dostatočne veľký počet úloh, táto množina môže byť rozsiahlejšia. V druhej časti vyberáme konkrétne úlohy, ktoré predložíme študentovi, podľa parametrov odporúčania.

Podobne ako pri detekcii zručností sme aj tu uvažovali viacero prístupov:

- Odporúčanie podľa úspešnosti
- Odporúčanie pomocou tabuľky závislostí
- Odporúčanie pomocou itemsetov

Každý z týchto prístupov vytvorí postupnosť top k úloh na odporúčanie. No všeobecne v didaktike nemôžeme študentovi predložiť k ľahkych alebo k ťažkých úloh. Množina poskytovaných úloh by mala byť vyvážené. To znamená, niekoľko ľahkých, "priemernýchä ťažkých úloh.

Ani v odporúčacích systémoch sa všeobecne neodporúča užívateľovi top k produktov ale odporúčania sú vyvážené, tj. kombinácia produktov, ktoré by ho zaujímali ale aj úplne nové produkty.

3.1 Odporúčanie pomocou závislostí

Každej zručnosti sme priradili hodnotu *skill threshold*. Ak študent bude mať hodnotu zručnosti v TZS väčsiu alebo rovnú ako *skill threshold*, tak hovoríme že zručnosť

ovláda. Keďže úlohy odporúčame na základe ich zručností, museli sme najprv zostaviť množinu odporúčaných zručností pre daného študenta. Následne študentovi odporučíme úlohy obsahujúce aspoň jednu z odporúčaných zručností pričom ostatné zručnosti v úlohu musí študent už ovládať. Úlohy tiež zoradíme podľa počtu neovládaných zručností a odporúčame úlohy s najmenším počtom aby sa študent netrápil s viacerými neznámymi naraz.

Teraz si rozoberieme algoritmus na získanie množiny odporúčaných zručností. Základnou ideou tohto algoritmu je zostavenie množiny zručností, ktoré ešte neovláda ale zároveň je schopný ich zvládnuť, napríklad študent by nezvládol rekurziu ak by nevedel podmienky a volanie metód.

1. Nech Z je množina všetkých zručností, ktorých hodnota v TZS pre daného študenta je menšia ako *skill threshold* - čiže ich neovláda
2. Nech $M = \emptyset$ je výsledná množina
3. foreach (Z as z)
   (a) $D =$ množina zručností od ktorých je zručnosť z závislá
   (b) Ak je $D = \emptyset$ tak $M = M \cup \{z\}$
   (c) Inak $M = M \cup \{z\}$ iba ak $\forall d \in D : d \notin Z$

Po získaní množiny odporúčaných zručností M musíme nájsť úlohy, ktoré obsahujú aspoň jednu zručnosť z M ale žiadnu zručnosť, ktorú zatiaľ nemôže zvládnuť.

1. Nech U je množina všetkých úloh
2. Nech $R = \emptyset$ je výsledná množina úloh
3. foreach (U as u)
   (a) Nech S je množina zručností, ktoré úloha precvičuje
   (b) Ak $\Big((\exists s \in S : s \in M) \& (\forall s \in S : s \notin (Z \setminus M))\Big)$ tak $R = R \cup \{u\}$

Po získaní množiny odporúčaných úloh R predložíme študentovi úlohy z tejto množiny na základe odporúčacích parametrov ako napríklad počet precvičovaných zručností.

3.2 Odporúčanie pomocou itemsetov

Tento prístup odporúčania sa od predchádzajúceho až tak veľmi nelíši. Tiež sa tu využíva TZS a *skill threshold* ale popri tom využíva aj množinu itemsetov. V našom prípade sú to množiny zručností, ktoré sú výstupom algoritmu *eclat*.

Eclat je algoritmus, ktorý sa využíva na itemset mining. Itemset mining vyhľadáva často sa vyskytujúci vzor v dátach, v našom prípade sú to množiny zručností nachádzajúce sa spoločne vo viacerých úlohách.

Celé odporúčanie sa začína výberom zručností, ktoré študent neovláda pomocou TZS a *skill threshold*. Následne sa vytvoria itemsety zručností algoritmom *eclat* pričom sa ponechajú len tie itemsety, ktoré obsahujú aspoň jednu neznámu zručnosť. Od tohto bodu delíme toto odporúčanie na dva typy:

- podľa počtu
- podľa zložitosti

Keď odporúčame podľa počtu tak itemsety zotriedime podľa počtu neznámych zručností. Postupne sa pomocou zotriedených itemsetov vyberajú úlohy zodpovedajúce jednotlivým itemsetom a dávajú sa postupne do poľa úloh, ktorý v našom systéme voláme *problem pool*.

Keď odporúčame podľa zložitosti, tak itemsety triedime podľa celkovej zložitosti všetkých neznámych zručností v itemsete. Zložitosť zručností určujeme pomocou ich hierarchie v tabuľke závislostí. *Problem pool* však vytvoríme rovnakým spôsobom.

Výsledný *problem pool* predstavuje rebríček odporúčaných úloh. Tento rebríček však môže obsahovať priveľa úloh. Preto vyberáme iba určitú podmnožinu.

1. Z = množina odporúčaných zručností, ktorých hodnota v TZS pre daného študenta je menšia ako *skill threshold* - čiže ich neovláda
2. IS = množina itemsetov z výstupu algoritmu *eclat*
3. ISN = podmnožina IS taká, že každú prvok ISN obsahuje aspoň jeden prvok zo Z
4. zotriedime množinu ISN podľa počtu alebo zložitosti
5. PP = rebríček úloh zostavený pomocou ISN

3.3 Odporúčanie podľa úspešnosti

Všetky úlohy sa pre každého študenta delia na vyriešene a nevyriešené. Vyriešené úlohy majú už pridelené hodnotenie a dokážeme vypočítať ich úspešnosť. Tento spôsob odporúčania je založený na predikovaní úspešnosti nevyriešených úloh.

Na predikciu úspešnosti sa využíva vektor konkrétnej úlohy $U = (u_1, u_2, u_3, ..., u_n)$ a vektor študenta $S = (x_1, x_2, x_3, ..., x_n)$ zostavený prístupom 2.3.3. Na začiatok sa zostaví množina nevyriešených úloh študenta. Následne sa predikuje úspešnosť pre každú úlohu. Z tejto množiny predikovaných úloh sa vyberú odporúčané úlohy podľa úspešnosti.

Algoritmus predikcie:

1. Nech $S_i = (x_1, x_2, x_3, ..., x_n)$ je vektor študenta i
2. Nech UN_i je množina nevyriešených úloh študentom S_i
3. $\forall U \in UN_i$
   (a) Nech P_U je predikovaná úspešnosť pre U
   (b) Potom
   $$P_U = \left(\frac{x_1 * u_1 + x_2 * u_2 + ... + x_n * u_n}{\sum_{i \in U} u_i} \right)$$
   (c) Dvojicu (U, P_U) vložíme do množiny UP_i

Algoritmus odporúčania:

1. Nech UP_i je množina úloh s ich predikovanou úspešnosťou
2. Zotriedime množinu UP_i podľa predikcií
3. Vyberáme úlohy z UP_i podľa parametrov odporúčania

Tento spôsob odporúčania sme sa rozhodli používať vo finálnej verzii systému.

Kapitola 4

Experimenty

Na otestovanie odporúčania v našom systéme sme použili reálnu sadu úloh, študentov a zručností z kurzu programovania pre začiatočníkov. Tieto dáta pozostávajú z:

- 79 študentov
- 44 úloh
- 26 riešení pre všetkých 79 študentov - 2054 riešení
- 2054 hodnotení
- 55 zručností

V týchto testovacích dátach je 44 úloh, ktoré sú určené pre začínajúcich programátorov. Všetkých 79 študentov vyriešilo 26 úloh. Prvé z týchto úloh precvičovali len základy a postupne sa prechádzalo na nové zručnosti. Zvyšných 18 úloh sú priamo naviazané na pokračovanie kurzu.

Na týchto dátach sme vykonali 2 rôzne experimenty:

1. Výpočet chyby predikcie
2. Odporúčanie pre jedného študenta

4.1 Výpočet chyby

V prvom experimente sme hľadali chybu RMSE nášho algoritmu. RMSE (root-mean-square error) sa využíva na zistenie rozdielov medzi reálnymi a predikovanými hodnotami. Vo všeobecnosti RMSE predstavuje štandardnú odchýlku medzi reálnymi a predikovanými hodnotami.

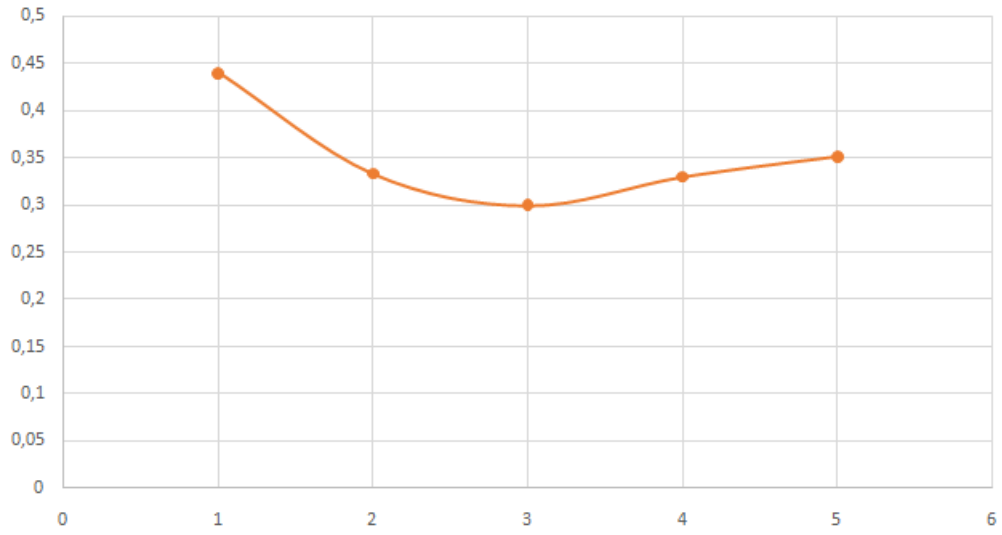
Na výpočet RMSE sme využili techniku k-fold cross validation. Vykonáva sa tak, že na začiatku sa začiatočná vzorka dát rozdelí na k častí rovnakej veľkosti. Z týchto k časti jedna sa použije ako testovacie dáta a zvyšných k-1 častí budú tréningové dáta. Následne sa na testovacích dátach vykoná predikcia pomocou tréningových dát. Z predikovaných a reálnych hodnôt testovacích dát sa vypočíta chyba RMSE. Tento celý proces sa opakuje k-krát. Výsledkom bude k chýb pre rôzne behy.

Pri výpočte chyby využívame 26 vyriešených úloh pričom ostatných 18 nevyriešených sme nepoužívali. Vykonali sme tu k-fold cross validation s k = 5. Dokopy sme teda vykonali päť behov. Úlohy sú zoradené podľa času ich vyriešenia študentmi. Testovacie časti sú teda tiež vyberané podľa času ich vyriešenia. Tým pádom máme úlohy $R = U_1, U_2, ..., U_{26}$ a prvá testovacia sada úloh bude $R_{Te} = U_{22}, U_{23}, U_{24}, U_{25}, U_{26}$, v druhom behu to bude $R_{Te} = U_{17}, U_{18}, U_{19}, U_{20}, U_{21}$, atď. Chybu RMSE sme rátali pomocou vzorca:

$$RMSE = \sqrt{\frac{\sum_{j \in S} \sum_{i \in T} (P_{j,i} - R_{j,i})^2}{n}}$$

Hodnota $P_{j,i}$ je predikovaná úspešnosť úlohy $i \in T$ pre študenta $j \in S$ v danom behu, hodnota $R_{j,i}$ je reálna úspešnosť úlohy $i \in T$ pre študenta $j \in S$ a hodnota $n = |T| * |S|$, Množina T je množina testovacích úloh pre daný beh a množina S je množina všetkých 79 študentov.

Výsledky piatich behov podľa času vyriešenia boli:



Obr. 4.2: Graf RMSE chýb podľa času

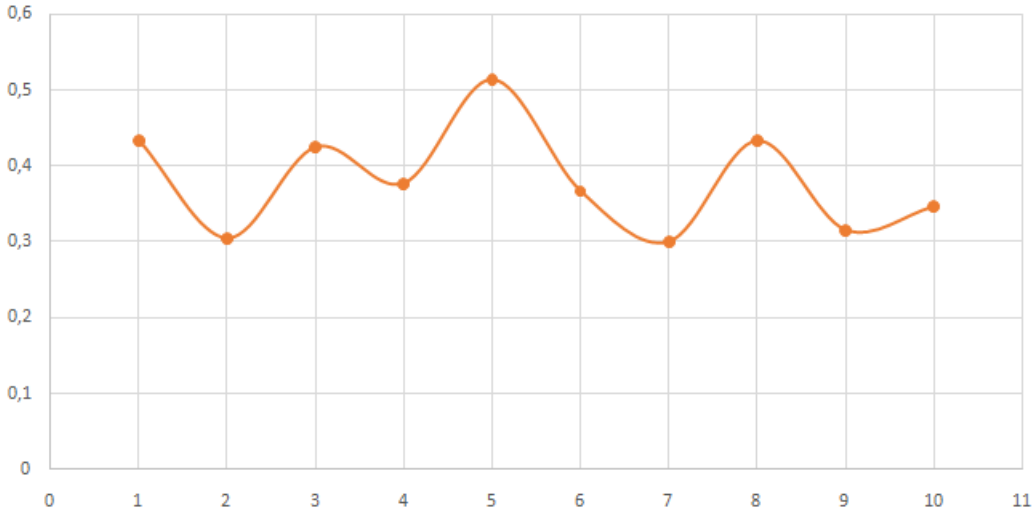
	Beh 1	Beh 2	Beh 3	Beh 4	Beh 5
RMSE	0.440521	0.333293	0.299527	0.329898	0.351567

Tabuľka 4.1: Tabuľka RMSE chýb podľa času

Výsledná chyba po spriemerovaní je teda:

$$RMSE_c = 0.3509612$$

Následne sme vyrátali druhú chybu rovnakou metódou ale z našich pôvodných 26 úloh sme časti nevyberali podľa času ale vyberali sme ich náhodne. Vykonali sme tu aj dva-krát viac behov, čiže k = 10.
Výsledky desiatich behov náhodne vybraných úloh boli:



Obr. 4.3: Graf RMSE chýb pre náhodný vyber

	RMSE
Beh 1	0.434291
Beh 2	0.305141
Beh 3	0.425686
Beh 4	0.377844
Beh 5	0.514907

	RMSE
Beh 6	0.367727
Beh 7	0.300680
Beh 8	0.434116
Beh 9	0.316307
Beh 10	0.346987

Tabuľka 4.2: Tabuľka RMSE chýb pre náhodný vyber

Výsledná chyba po spriemerovaní je teda:

$$RMSE_r = 0.3823686$$

Pre získanie výslednej chyby sme tieto dva hodnoty spriemerovali:

$$\mathbf{RMSE = 0.3666649}$$

Výsledná hodnota $RMSE$ predstavuje chybu predikcie v našom systéme na dátach, ktoré máme k dispozícii.

4.2 Odporúčanie pre študenta

Pre tento experiment sme vybrali náhodného študenta a odporúčali sme mu 3 úlohy. Tomuto študentovi sme predikovali úspešnosť 18 úloh, ktoré nevyriešil. Hodnoty podľa počtu boli:

Úspešnosť	Počet úloh
0.7 - 1.0	0
0.6 - 0.69	1
0.5 - 0.59	2
0.4 - 0.49	3
0.3 - 0.39	2
0.2 - 0.29	2
0.1 - 0.19	0
0.0 - 0.09	8

Tabuľka 4.3: Tabuľka odporúčaných úloh podľa počtu

Pri bližšom pozorovaní konkrétnych odporúčaní sme si všimli, že všetky úlohy s predikovanou úspešnosťou medzi 0.0 až 0.09 boli väčšinou úlohy v neskorších častiach kurzu ku ktorým by sa študenti dostali až neskôr. Zručnosti, ktoré študent v čase odporúčania ovládal, boli v týchto úlohách zastúpené minimálne poprípade vôbec. Boli však aj úlohy, ktorých nízka predikcia bola spôsobená slabou znalosťou niektorých zručností študenta. Tieto úlohy mu preto systém neodporúča nakoľko pravdepodobnosť ich úspešného vyriešenia je veľmi nízka.

V množine úloh s predikciou medzi 0.2 až 0.49 sme sa bližšie pozreli na zručnosti úloh. Analýzou sme v množine našli dva typy úloh. Prvou boli úlohy, ktorých niektoré zručnosti študent ovládal ale niektoré zas boli pre neho úplne nové. Druhý typ boli úlohy, ktorých všetky zručnosti si študent aspoň do určitej miery precvičil. V tejto množine úloh sa teda nachádzajú úlohy, ktorých vyriešením by si študent upevnil znalosti určitých zručností alebo sa aj naučil úplne nové zručnosti. Takéto typy úloh sú veľmi žiadané pre študenta a preto sme mu z tejto množiny odporučili dve úlohy.

Ďalej sme pozorovali, že v množine úloh s vyššou predikovanou úspešnosťou (0.5 až 0.7) sa úlohy čiastočne podobajú na úlohy, ktoré študent úspešne vyriešil. Tým pádom riešením týchto úloh by si študent len upevňoval svoje vedomosti, čo je tiež žiaduci efekt. Preto sme mu z tejto množiny odporučili jednu úlohu.

Takýmto spôsobom sme odporúčali konkrétne úlohy študentovi na našich dátach. Odporúčané úlohy pomôžu študentovi upevniť si vedomosti v zručnostiach s ktorými sa už stretok ako aj získať poznatky o nových zručnostiach a technikách programovania.

Záver

Cieľom tejto práce bolo vytvoriť funkčný edukačný systém na vedenie študenta štúdiom za pomoci odporúčania. Predstavili sme ITS systém s názvom IPT pričom sme sa pozreli aj na existujúce systémy. Rozoberali sme základnú ideu, funkcionalitu a štruktúru.

Ďalej sme navrhli model študenta, v ktorom sme museli riešiť viaceré problémy. Ako prvý sme rozobrali spôsob reprezentácie zručností. Následne sme uvažovali viaceré prístupy detekcie zručností z riešenia študentov. Tieto prístupy boli: detekcia pomocou vzorového riešenie, detekcia pomocou hodnotenia učiteľa, manuálne priradenie a využívanie všetkých zručností. Nasledoval opis niekoľkých spôsobov určovania vedomostí študentov.

V ďalšej časti našej práce sme navrhli rôzne spôsoby odporúčania. Odporúčanie pomocou závislostí sa opieralo o ideu, kde študentovi odporúčame úlohy, ktorých zručnosti by zvládol vzhľadom na ich prerikvizity/závislosti. Odporúčanie pomocou itemsetov využívalo itemset mining, ktorý vyhľadával často sa vyskytujúce zručnosti v úlohách. Na základe toho sa následne vybrali úlohy pre študenta podľa počtu zručností alebo zložitosti. Posledné bolo odporúčanie podľa úspešnosti. Toto odporúčanie pracovalo na základe predikcie. Študentom sa predikovali úspešnosti všetkých nevyriešených úloh, z ktorých sa následne vybrali úlohy na odporúčanie.

Na záver sme vykonali zopár experimentov na našej dátovej sade. Rátali sme chybu predikcie úspešnosti úloh pomocou k-fold cross validation. Následne sme rozobrali celkový výsledok odporúčania pre konkrétneho študenta.

Zoznam použitej literatúry

[1] C.J. Butz, S. Hua, R.B. Maguire, A Web-based Bayesian Intelligent Tutoring System for Computer Programming, Journal Web Intelligence and Agent Systems archive, 2006.

[2] G. Paviotti, P.G. Rossi, D. Zarka, I-TUTOR, Intelligent Tutoring System: An Overview, 2012.

[3] O.C. Santos, J.G. Boticario, Educational Recommender Systems and Technologies: Practices and Challenges, IGI Global, 2012.

[4] R. Santhi, B.Priya, J.M.Nandhini, Review of Intelligent Tutoring Systems using Bayesian approach, National Conference on Computational Linguistics and Integrated Classical Knowledge, 2013.

[5] E. R. Sykes, F. Franek, A prototype for an intelligent tutoring system for students learning to program in Java: The 3rd Ieee International Conference On Advanced Learning Technologies: 9-11 July 2003, Athens, Greece. Los Alamitos: IEEE Computer Society, 2003.

[6] P. Brusilovsky, Adaptive and intelligent technologies for Web-based education, Special Issue on Intelligent Systems and Teleteaching, 4 (1999) 19-25.

[7] V.J. Shute and J. Psotka, Intelligent tutoring systems: past, present, and future, Handbook of Research on Educational Communications and Technology, Macmillan, New York, (1996) 570-600.

[8] Nkambou, R., Mizoguchi, R., Bourdeau, J. (2010). Advances in intelligent tutoring systems. Heidelberg: Springer.

[9] Yujian Zhou, Martha W. Evens, A Practical Student Model in an Intelligent Tutoring System: IEEE Computer Society. (1999) 11th IEEE International Conference

on Tools With Artificial Intelligence: Proceedings, November 9-11, 1999 Chicago, Illinois, : Institute of Electrical & Electronics Enginee.

[10] Gertner, A., VanLehn, K. (2000). "Andes: A coached problem solving environment for physics" in Proc. 5th International Conference. Intelligent Tutoring Systems, Berlin, 2000, pp. 133-142

[11] Mitrovic, A. (2000). "An intelligent SQL tutor on the web". International Journal of Artificial Intelligence in Education, 13 (2-4), 2003, pp. 171-195.