P-Res Tutor: An Intelligent Tutoring System for Propositional Resolution

Alexandre Lopes Poeira
Instituto Superior Técnico
Lisbon, Portugal
alexandre.poeira@tecnico.ulisboa.pt

ABSTRACT

Intelligent Tutoring Systems (ITSs) are computer-based instructional systems that specify what to teach and how to teach it[10]. Given a specific knowledge base, an ITS presents activities related to this base for the user so they can "learn by doing" in realistic and meaningful contexts, thus providing a personalised learning experience dependent on the user's experience. However, constructing an ITS is a challenge in itself[11], being a multidisciplinary task involving multiple research fields, from which artificial intelligence and education stand out. To simplify, the "intelligent" part of an ITS can be independent of the knowledge base, being instead based on the empirical estimation of learning progress of the user given the correctness of their answer[3].

This dissertation focuses on the construction of an online ITS, P-res Tutor, with the purpose of teaching Propositional Resolution[14], a rule of inference of Propositional Logic, and its applications in solving theorems. Since learning this rule requires learning other subjects, the system also teaches the basics of Propositional Logic and the Conjunctive Normal Form (CNF) with different activities based on Logic exercises and other rules of inference. For this system, we apply multi-armed bandit methods [3] and develop a unique error diagnosis process [6] for exercise selection and student modelling. The prototype of the system was implemented and a user-study was conducted from which we got results validating our system's teaching potential.

Author Keywords

Intelligent Tutoring Systems; Propositional Logic; Propositional Resolution; Multi-Armed Bandits;

INTRODUCTION

As a teaching supplement, computer courses have been researched and developed intensively. Inevitably, Artificial Intelligence methods were introduced to the area of online teaching, from which the area of *Intelligent Tutoring Systems* was born. These systems, in a general manner, are computer courses based on AI programming techniques, which change according to the student's input[1], providing an individualised and personalised learning experience. For a student to learn a skill, a sequence of exercises or activities is followed. For an expert/teacher, the challenge lies on creating a sequence that maximizes learning for every student. An ITS serves as an artificial expert that determines the optimal sequence of activities for the acquisition of the skill, while being customized to fit each student's unique characteristics.

The design and development of ITSs require resources from multiple research fields, including artificial intelligence, cognitive sciences, education, human-computer interaction, and software engineering, all of this without taking into account knowledge about the ITS's main subject of teaching. As such, building this kind of system presents a thoroughly challenging task, given the massive multidisciplinary requirements[11].

There are two main motivating factors for researchers to build an ITS[12]:

- *Pure Research Needs*: Since ITS research lies at the intersection of multiple domains, it provides an excellent testbed for various theories from cognitive psychologists, educational theorists and/or AI scientists. Different teaching scenarios can be simulated by changing parameters of the system, such as presenting different activities or changing the overall difficulty of the course midway.

- *Practical Needs*: Since generally there are more students than teachers, most educational systems become geared towards group teaching methods, while losing most advantages that are found in one-on-one tutoring. One of the primary advantages of an ITS is the possibility to provide one-on-one tutoring without necessarily losing the advantages of the group teaching environment, for example by providing each student their own copy of the ITS.

The aim of this thesis is, thus, to create a practical system which can serve as a research tool for ITS researchers and as a teaching tool for anyone who wishes to teach or learn Propositional Resolution. This system must include exercises and activities that test every skill needed to learn Resolution. Subsequently, it must also estimate the knowledge of the student given their answers on the exercises shown and select exercises accordingly using this student model.

It must also provide feedback to the learner, so they can operate independently, and to the teacher, so the system can be improved if needed and to study the importance of different adjustable parameters in the system. These can range from

having a limited set of fixed exercises to adjusting the learning speed.

RELATED WORK

In this section we discuss previous work done in the area, starting with ITS construction and common components, knowledge estimation, Multi-armed bandit methods for ITSs, error diagnosis processes and recommended metrics for evaluating student steps and, finally, a theoretical introduction to our system's domain, Propositional Resolution.

ITS Construction and Components

Due to their complexity, ITSs are normally not easy to construct [11]. Since the resources needed come from very different areas, such as artificial intelligence, education, human-computer interaction and software engineering, the process of building an ITS is inherently a challenging task. The construction also highly depends on the views and priorities of the author. They can choose to focus on exercise selection [7, 3], ensuring the system's tutoring decision making is based on sound pedagogical principles, or in error diagnosis [5, 4], using appropriate knowledge structure and algorithms to interpret users' decisions correctly and give proper feedback to the user.

The structure of an ITS varies tremendously between different systems. Since the work in this area is experimental in nature, there is no clear-cut way to construct an ITS's structure [12] although many common elements can be found in the structure of different ITSs. The most common structure is a composition of different modules comprising different parts needed for the tutoring process. The basic components, *Expert Knowledge module*, *Student Model module* and the *Tutoring module*, make up the intelligent part of the system and is where most work in constructing an ITS is had, while a fourth component, the *User Interface module*, has also been identified [9] as essential as it serves as the only communication channel between the user and the system. A simplified model of this structure can be found in the following figure 1.

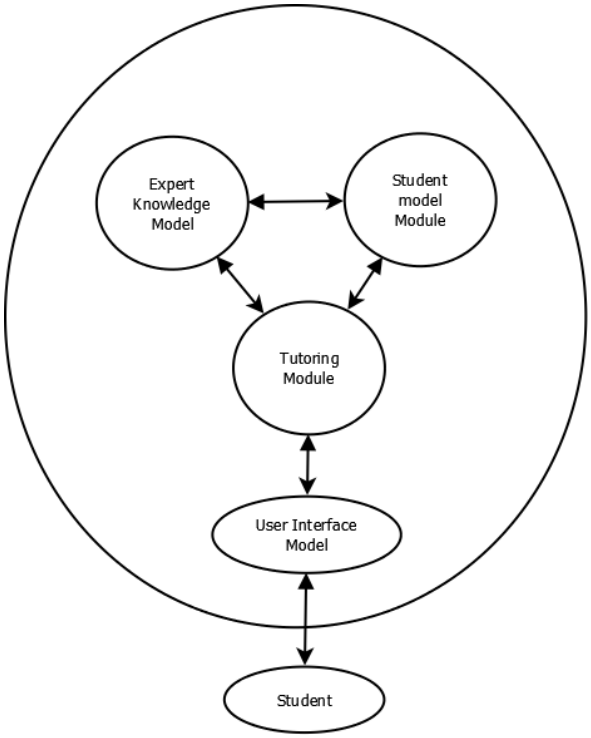

Figure 1. Basic structure of an ITS

KNOWLEDGE ESTIMATION

For the purpose of teaching a student, the system must also ascertain what the student already knows, building the student model, but also of what he/she must know to achieve the taught skill, by having a clearly defined expert knowledge module.

Knowledge Components

In any Intelligent Tutoring System, we can define its main goal the learning of a certain skill, in this case being Propositional Resolution. To facilitate the learning process, this goal can be subdivided into sub-goals, smaller skills that must be learnt first since they are applied in the main skill. These sub-goals are called Knowledge Components, KCs. They can be represented as nodes in an acyclic graph to allow the creation of a hierarchy between them to model their increasing difficulty and required precedence of other KCs.

As defined by Lindsey et al. [8], KCs are provided by experts on the matter which can then be refined by automated techniques. For this problem, we will propose our own KCs required to learn Propositional Resolution, which we will discuss further in this paper.

Zone of Proximal Development

When learning a skill, it is important to take note of what activities to perform. If they are too easy, no new knowledge is attained, if they are too hard, the student cannot complete or understand it, meaning that the exercises recommended can neither be too hard nor too easy. They must be challenging enough to maintain the student's interest and easy enough that they can solve it, which requires exercises to be at a level of required knowledge only slightly higher than the current. These activities belong to what is called the Zone of Proximal Development[15], or ZPD, for short, which comprises of all the activities which give the most learning progress.

MULTI-ARMED BANDITS FOR ITS

When making an ITS specialized in choosing sequences of exercises customized for each student, the most pressing issue is how to define and optimize this sequence. A solution is using Multi-Armed Bandit methods [3, 16]. In short, in a problem with various activities to perform and each with a different unknown reward, the problem lies in choosing whether to explore different activities to see the reward of each one, "Exploration", or exploit the activity that is guaranteed to give a good reward, "Exploitation". Taking a casino analogy into account, multi-armed bandits would be described as trying to find the most profitable slot machine.

Since the bandit must spend money exploring all of them before choosing the best one, the dilemma lies in simultaneously trying new activities to know their payoff while also

selecting the best ones so actual profit can be made. In the context of an ITS, the slot machines are the different exercises that the algorithm recommends to the student, the reward is the learning gain and the choice of "Exploration vs Exploitation" is made by the teaching algorithm, considering the reward information it gets previously.

The usage of these methods allows us to have a weaker dependency on a Student Model, in favor of optimizing the Tutoring module to adapt to each individual student. This, in turn, also allows us to use methods of student model construction that make no assumptions about how students learn and only require information regarding the estimated learning progress of activities, which creates a simple, yet unique model for each user. As such, by focusing on optimization of the MAB algorithm, the system becomes more accurate as student models and the tutoring module become more defined.

There are some particularities in an MAB approach to ITSs. Firstly, the reward for each activity, which is learning progress, does not stay the same, since it depends on the competence level of the student for the exercise, which will stop giving a reward after a certain competence level is achieved. Secondly, rewards are not independent and identically distributed, as we are dealing with humans, which brings various effects into play that can affect the reward, such as distractions, mistakes using the system and mainly different preferences between students.

Thus, every activity a will have a weight w_a which tracks its reward, correlated to the learning progress given by activity a. Each time this activity is performed, this w_a is updated:

$$w_a \longleftarrow Bw_a + ur \tag{1}$$

As shown, the reward given by the activity, r, is added to the current weight of the activity to update the new weight. By altering the parameters B, u, we can change the relevance of the reward or the previous weight. These weights come into use when the system has to choose the next activity to recommend, as each activity is assigned a probability, p_i, for it to be selected, which uses the normalized weight of the activity, $\acute{w}_a$, the exploration rate, y, and a uniform distribution, e_u, to ensure sufficient exploration of activities:

$$p_i = \acute{w}_a(1 - y) + ye_u \tag{2}$$

Since it is needed that all activities have an associated weight to determine their probability, it would be needed to explore all activities to estimate their impact on each knowledge component. This would be very time-consuming and could produce an under-performing learning sequence, so instead, a canonical learning sequence is used to initialize the algorithm, after which this sequence can be optimized. This sequence is determined by an expert, and normally has a higher reward on the introductory low difficulty activities and a lower reward in the more advanced activities.

Two different algorithms using MAB technology can be used, the first one requiring little domain knowledge named Zone of Proximal Development and Empirical Success, ZPDES. The second approach assumes there is a simple relation between the activities and skills of the student, estimates the learning progress obtained at a given point in time and proposes to the student the activities which provide higher learning progress, thus the name of the algorithm being Right activity at the Right time, RiaRit.

ZPDES

This algorithm is inspired by the ZPD and the empirical estimation of learning progress, and, as such, it requires very little domain/user knowledge. To estimate the reward of each skill, that is, the estimated learning that the skill provides, the correctness of the answer given by the student is the only parameter needed.

Instead of comparing the correctness of the answer given at a certain time t with all of the previous answers d, it instead compares the last half of $d/2$ answers with the earlier half of $d/2$ answers given. This allows the measure of the quality of each activity, since we can measure how much progress a certain activity has provided in a short time window and we consider activities with a faster progress to be better than others with a slower progress [2].

Thus, the computation of the learning progress r, where $C_k = 1$ if the exercise at time k is correct, is as follows:

$$r = \sum_{k=t-d/2}^{t} \frac{C_k}{d/2} - \sum_{k=t-d}^{t-d/2} \frac{C_k}{d - d/2} \tag{3}$$

When an activity has already been acquired or when the student is not progressing in any way, which are both extreme cases, the reward given will be zero. To reduce the number of activities that are needed to explore, there is also the added restriction that only activities in the ZPD are selected for the user.

Initially, the ZPD is defined as a graph with every activity ordered by levels of difficulty. Only the most basic skills are included in the ZPD, with more advanced ones having the prerequisite of attaining previous easier skills. For activities already in the ZPD, free exploration is allowed since these are considered to always give some learning progress. After enough progress is attained in a certain activity, it is considered mastered, is removed from the ZPD and the more advanced skills that had the previous one as prerequisite being added to the ZPD.

This algorithm also allows the tutor to limit or expand exploration of activities if needed, depending on whether the set of activities have a clear progression of difficulty between them. If they do, then exploration is limited to force students to follow a specific path between each skill, if not, meaning different students can have very different orders when they are obtaining skills, then wider exploration is allowed in order to accommodate individual differences.

ERROR DIAGNOSIS

For the purpose of constructing and maintaining a stable student model, diagnosing the user's input can contribute significantly to this effort. An ITS can be said to consist of two different loops, an outer loop, which chooses activities for the student by matching their learning progress to adequate activities, and an inner loop, which gives feedback and hints about steps the student must take to solve the activity [18]. An important responsibility of the inner loop is analyzing these

steps the student takes, in order to find exactly where the user made a mistake, and give proper feedback.
Different approaches for error diagnosis have been studied extensively in ITSs [17] and 8 different aspects have been identified in multiple ITSs as relevant for diagnosing student steps:

- **Correctness**: refers to whether or not a student step matches an expected step, with the only possible outcomes being *correct* or *incorrect*;
- **Difference**: similar to correctness but measures the edit distance, how different it is, between the student step and the expected step. This measure is normally a number or percentage, for example, if the only difference between the student step and the correct step is a single character, a '+' for a '-', then the edit distance would be one, since it requires only one edit operation for the student step to be correct;
- **Redundancy**: refers to whether the student step is significant in any way, if the difference between the current student step and the previous one is too small, it can be considered redundant. Possible outcomes are *redundant*, *not redundant* and *unknown*;
- **Type of Error**: refers to classifying errors, for example, classifying $a + (b$ as a syntax error. Possible outcomes depend on the domain of the ITS;
- **Common Errors**: refers to errors students make based on common misconceptions, for example, when forgetting to change the sign when moving an expression to another side of an equation ($a + b = 0 \rightarrow a = b$). Possible outcomes depend on the domain of the ITS;
- **Order**: refers to the order in which the student takes different steps, with the possible outcomes being *correct order*, *incorrect order* and *unknown*;
- **Preference**: refers to existing preferable solutions to problems than the one the student presents, with the possible outcomes being *preferred*, *not preferred* and *unknown*.
- **Time**: refers to the time the student took to submit a step or solve a problem, normally measured in milliseconds.

In regards to diagnostic processes, most ITSs use multiple aspects for diagnosing student responses. The most basic process consists of a single aspect, correctness, only checking whether the answer is correct or not. A more complex process involving more aspects is used in the system *AITS* [6]. By calculating the edit distance between the student and the ideal step, other aspects of the error can be better analyzed by checking the number and the content of the different nodes to determine redundancy and the type of error. Using these aspects, an error is classified according to its completeness and its accuracy, meaning an incorrect student step can be either *complete but inaccurate*, *incomplete but accurate* and *incomplete and inaccurate*. The diagnosis *complete and accurate* never occurs since it means the edit distance is zero, and the student step is equal to the best response.

PROPOSITIONAL LOGIC

Propositional Logic [14] is a branch of logic that deals only with propositions, which are facts represented by symbols. They can be a single affirmation or literal, which is called an atomic proposition (an example of this is the proposition "It is raining"), or a set of these affirmations/literals connected by logical connectives, which is called a compound proposition (an example of this is "It is raining, and it is cloudy" which is a compound proposition of the literals "It is raining" and "It is cloudy" with the logical connective "and").
Propositions are normally represented by uppercase letters (we can represent the propositions "It is raining" and "It is cloudy" as A and B, respectively, and "It is raining, and it is cloudy" as "$A \wedge B$"). Logical connectives are symbols used to connect two literals to create a compound proposition, except for the negation symbol, which is the only connective that operates on a single proposition.
The logical connectives used in Propositional Logic are:

- **Negation**; represented by $\neg$, $\sim$, or "NOT", is the only connective that is used in relation to a single literal and isn't used to connect two literals. Represents denial of a literal, for example: $\sim A$ is only true when A is false.
- **Conjunction**; represented by $\wedge$, &, $\cap$ or "AND". Represents the intersection between two literals, for example: $A \wedge B$ is only true when both A and B are true.
- **Disjunction**, represented by $\vee$, $|$, $\cup$ or "OR". Represents the junction of two literals, for example: $A \vee B$ is true when A or B or both are true.
- **Implication**, represented by $\implies$, ==>, or "IF...THEN". Represents dependence between two literals, for example: $A \implies B$, meaning if A happens then B must happen, is true when B is true or when A is false.

Resolution

Resolution is a rule of inference that leads to a theorem-proving technique in propositional logic and first-order logic. The resolution rule states that, from two different propositions, a new one can be created by uniting both and removing the complementary literals. The new proposition is said to be the *resolvent* of the previous two. This can be seen in the following expression where P_1 and P_2 are propositions and $l \in P_1$ and $\neg l \in P_2$:

$$Res(P_1, P_2) = (P_1 - \{l\}) \vee (P_2 - \{\neg l\}) \tag{4}$$

A set of complementary literals is a set of a literal and its negation, for example A and $\neg A$. The *resolvent* between these two literals is the empty set $\{\}$. If there are other literals in the proposition, the same principle is applied and the complementary literals are removed, for example with the propositions $P = A \vee B \vee \neg C$ and $Q = B \vee C \vee \neg E$, if we apply the resolution rule, we get the expression:

$$Res(P, Q) = ((A \vee B \vee \neg C) - \{\neg C\}) \vee ((B \vee C \vee \neg E) - \{C\})$$

$$Res(P, Q) = (A \vee B) \vee (B \vee \neg E)$$

$$Res(P, Q) = A \vee B \vee \neg E$$

The only restriction for this rule is propositional resolution can only be applied to propositions in the Clausal Nominal Form, CNF, or *clausal form*. A proposition is in the clausal form if it is a conjunction of one or more clauses, where a clause can either be a single literal, A or $\neg B$, or a disjunction of literals, $A \vee B$ or $\neg C \vee D$. The only logical connectives a proposition in the CNF can contain are *AND*, *OR* and *NOT*, and the *NOT* operator can only be used in reference to a single literal. The empty set is also a clause, $\{\}$, and it is equivalent to an empty disjunction.
To convert sentences to the clausal form, the following steps must be taken:

1. Convert any implications to its disjunction form, Ex: $A \implies B$ is equivalent to $\neg A \vee B$;
2. Push the negation symbol inwards using De Morgan's Laws:
   (a) For double negation: $\neg\neg A$ is equivalent to A;
   (b) For negated disjunction: $\neg(A \vee B)$ is equivalent to $\neg A \wedge \neg B$;
   (c) For negated conjunction: $\neg(A \wedge B)$ is equivalent to $\neg A \vee \neg B$;
3. Apply the distributive property of disjunctions, Ex: $(A \wedge B) \vee C$ is $(A \vee C) \wedge (B \vee C)$.

An application of the resolution rule is to prove theorems. With the application of the satisfiability theorem, which dictates that given a set of premises P and a literal C we wish to prove, the premises logically entail the literal C if $P \wedge \neg\{C\}$ is unsatisfiable and vice-versa, we deny our theorem and attempt to reach the unsatisfiable state, the empty set $\{\}$, with the given premises. With the set of premises and a denied conclusion in the clausal form, we must reach the empty set by applying the resolution rule to prove the theorem, for example with $P \wedge \neg C = \{\{A\}, \{\neg A, B\}, \{\neg B, D\}, \{\neg D\}\}$, we can reach the empty clause.

$$P = \{\{A\}, \{\neg A, B\}, \{\neg B, D\}\}$$

$$C = \{\{D\}\}$$

$$P \wedge \neg\{C\} = Q = \{\{A\}, \{\neg A, B\}, \{\neg B, D\}, \{\neg D\}\}$$

$$Res(Q_1, Q_2) = R_1 = \{B\}$$

$$Res(R_1, Q_3) = R_2 = \{D\}$$

$$Res(R_2, Q_4) = \{\}$$

IMPLEMENTATION

The prototype of the system is built on a python Flask application in an internet browser form, found in figure 2, with the user interface being a free *HTML5* template available online and the tutoring module, student model module and expert knowledge module built entirely with Python. For the propositional logic structures, operations and expressions, a python library *logic.py* from the book *AIMA* [14] was used. This library provides a framework for propositional logic in a python context, from which we use structures for logical expressions and operations with logical connectives, such as all the steps of conversion to the clausal form.
It is composed of three different activities the user can choose from or a stream of activities chosen by the system using the ZPDES algorithm, where the possible activities are one of the three. The three activities are:

- Truth or False exercises, where a formula and its supposed CNF formula are shown, and the user must answer whether the CNF formula is correct or not;
- CNF conversion, where a formula is shown and the user must input its clausal form;
- Connecting Clauses: Application of the resolution rule, where a formula in its CNF form is shown and the user must figure out if it is possible to achieve the empty clause by applying the resolution rule and eliminating clauses. This activity consists of selecting clauses to apply the resolution rule, the activity ends when the user can achieve the empty clause with any combination of the existing clauses or if the user selects it is not possible to do so;
- Full Resolution exercise, where a set of premises P and a conclusion C are presented and the user must prove that the conclusion is proven by the premises, by converting $P \wedge \neg C$ to its clausal form, and apply the resolution rule to the remaining clauses to achieve the empty clause $\{\}$. It is the only activity to consist of two different steps, one where the user must input the clausal form of $P \wedge \neg C$ and another one where the user must connect the resulting clauses to achieve the empty clause $\{\}$.

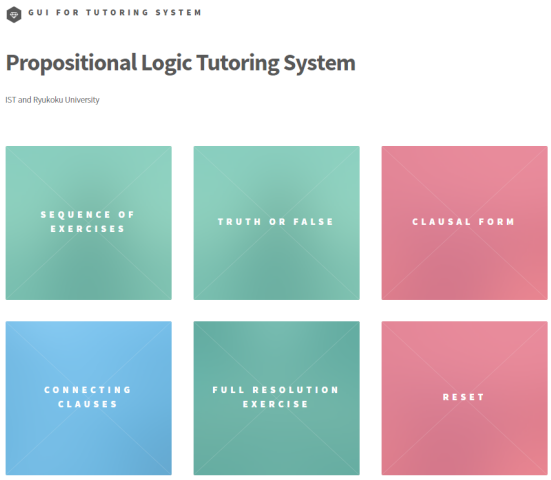

Figure 2. User interface of P-res Tutor

Interaction Loop

When choosing a specific activity function, a random exercise of the chosen type will appear. After this, the user receives feedback for the correctness of the input and can check the solution. When using the stream of exercises option, an exercise is selected based on the student model and presented. After the student's input, the correctness of the exercise is

evaluated by the expert knowledge model, the student model is updated according to the correctness, feedback is shown and a new exercise is selected and presented based on the new student model. The solution for the previous exercise can also be consulted.
This stream of new exercises is only stopped if the student wishes to go to the main menu or if the student model shows that the user has learned everything there is to learn. If a user wishes to start over and reset the student model, there is an option that resets it to its initial state in the main menu.

Knowledge Estimation

To estimate the knowledge of the student regarding Propositional Resolution, we divide it into different Knowledge Components and represent them in the Student model module.

Knowledge Components

The proposed KCs for Propositional Logic and Propositional Resolution and its corresponding KC tree are the following:

1. Atomic propositions, "It is raining" is A;
2. Negation of atomic propositions, "It is not raining" is $\neg A$;
3. Conjunction of propositions, "It is raining and it is cloudy" is $A \wedge B$;
4. Disjunction of propositions, "It is raining or it is cloudy" is $A \vee B$;
5. Implication of propositions and its decomposition, "If it is raining then it is cloudy" is $A \implies B$;
6. Removal of double negation, $\neg\neg A$ is A;
7. Distributive property of the *OR* connective, $(A \wedge B) \vee C$ is $(A \vee C) \wedge (B \vee C)$;
8. Negation of a conjunction, $\neg(A \vee B)$ is $\neg A \wedge \neg B$;
9. Negation of a disjunction, $\neg(A \wedge B)$ is $\neg A \vee \neg B$;
10. Application of the resolution rule, $Res((A \vee C), \neg A) = (P_1 - \{A\}) \vee (P_2 - \{\neg A\}) = C$;
11. Proving a theorem using the resolution rule with proof by contradiction.

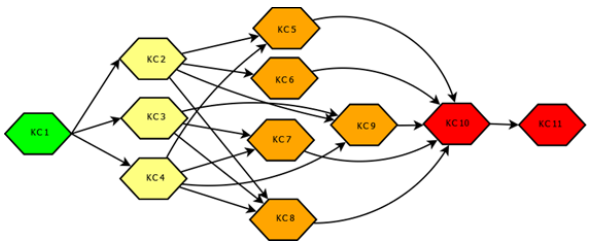

Figure 3. KC tree with skill dependency represented by the directional arrows

Student model Module

For the student model, we define it as a list of 10 values, for every KC except KC1, corresponding to each KCs learning progress. This value is percentage based, ranging from 0%, where the KC has not been attempted yet, to 100%, where the KC has been fully taught. The ZPD is also included in the student model, initialized with KC2, KC3 and KC4, and, as KCs are learned by the user, it is updated with new KCs according to the KC tree.
After every activity, both the KC values and the ZPD are updated. Depending on the correctness c of the exercise, which can be -1 for incorrect answers and 1 for correct ones, and on the learning speed ls, a possible way [13] to update every KC value $u_{k,n}$ would be:

$$u_{k,n+1} = u_{k,n} + ls \times c_{n+1} \tag{5}$$

To give even more importance to recent inputs, we build on this method by including a parameter representing the streak σ of exercises the student has gotten correct in a row, to a maximum of 10 exercises. As with the learning speed, we add an α parameter to adjust the streak's relevance. In practical uses, we use a small value for α, for example 0.1 or 0.2, so at a maximum streak of 10 exercises in a row, it would not add more than 1 or 2 to the KC value. Thus, the method becomes:

$$u_{k,n+1} = u_{k,n} + ls \times c_{n+1} + \alpha\sigma \tag{6}$$

Expert Knowledge Module

This module is built using *logic.py* library, with a change to the CNF conversion function. After all steps of CNF conversion, if we are left with an expression with repeated symbols in conjunction or disjunction with each other, for example $A \vee A$ or $\neg B \wedge \neg B$, these symbols are merged together, to A and $\neg B$ respectively, as having both would be redundant. This module also includes a database of exercises testing every KC. Depending on their content, these are classified according to which KCs they affect prioritizing the higher rated KCs. For example, for a proposition to be classified with KC2 it must only have a NOT connective, $\neg$, and for KC6 it must have two in succession, $\neg\neg$. As such, if an exercise is classified with KC6 we do not classify it with KC2.
Instead of letters, we use sequential numbers for each different letter and, if the exercise is shown to the user, all numbers are replaced with a random letter, for example $1 \implies 2$ can become $A \implies B$ or $Q \implies D$ or any other combination. This is done so the same exercise can be shown multiple times without looking repetitive. The exercises available in the system are mostly randomly generated, created using a script that randomizes the number of clauses and the number and order of logical connectives, keeping into account the validity of the generated expression.

TUTORING MODULE

This module is responsible for exercise selection and uses the ZPDES algorithm for such. With the KC values and the ZPD from the student model, we verify the ZPD for which KCs we can choose, then select a KC with the lowest value. With the KC chosen, we filter the exercises for the only exercises that have an impact on that KC and order them by length since, if an exercise has more clauses or symbols, it is considered

harder. With the learning value for the chosen KC as L_{kc} the number of exercises that impact that KC as N_{kc} and the learning value at which we consider that KC to be taught as $Lmax_{kc}$, we choose an exercise based on the following expression:

$$ExerciseNumber = \frac{N_{kc} \times L_{kc}}{Lmax_{kc}} \quad (7)$$

With the exercise chosen, the activity must also be chosen, which depend on the KC and its learning value:

- Truth and False exercises are always picked for every novice and intermediate KC when they are below 25% taught and have a chance to be chosen between 25% and 75%, after this threshold, these activities stop being chosen;
- CNF conversion exercises have a chance to be chosen for every novice and intermediate KC when they are over 25% taught and are always chosen when they are over 75%;
- Connecting Clauses exercises are only picked for KC10;
- Full Resolution exercises are only picked for KC11.

Error Diagnosis

For error diagnosis, we first check the validity of the answer given, by verifying there are no syntax issues or illegal characters or if there is an empty answer. After this verification there are two approaches we use for error diagnosis, one where we only check for correctness and an approach similar to *AITS* [6] diagnosis process, where we check for the edit distance and the type of error to give better feedback and update the KC values.

When measuring correctness, we compare the expression inputted, or shown in the Truth and False exercises, with the expected expression. If they include the same symbols and connectives in a similar order as the expected expression, for example $A \wedge B$ is equal to $B \wedge A$, the response is deemed optimal and correct. Otherwise, if there is a significant difference between the two formulas, the answer is deemed incorrect.

In the other approach, we verify the edit distance between the input and the ideal response. If there is any difference the answer is checked for its error. Here we make a distinction between basic errors, that students might make in the initial stages, and normal errors, where the student misses some steps of CNF conversion. Basic errors provide detailed feedback according to the type of error while normal errors also provide feedback but also influence the KCs they reference by decreasing their learning value. The types of error are as following:

- Basic Errors:
  - Not enough clauses;
  - Too many clauses;
  - Clause not in original expression;
- Normal errors:
  - KC5 error, if $\implies$ is not simplified;
  - KC6 error, if $\neg\neg$ two negation symbols are not cancelled out;
  - KC7 error, if $\vee$ is not distributed to other conjunctions;
  - KC8 error, if negation of a disjunction is not converted;
  - KC9 error, if negation of a conjunction is not converted.

Firstly we verify the number of clauses to verify basic errors, then we check for any differing combination of symbols not present in CNF. For any error that is found, we display feedback on it and what steps the student missed. The full error diagnosis process can be found in the following figure 4.

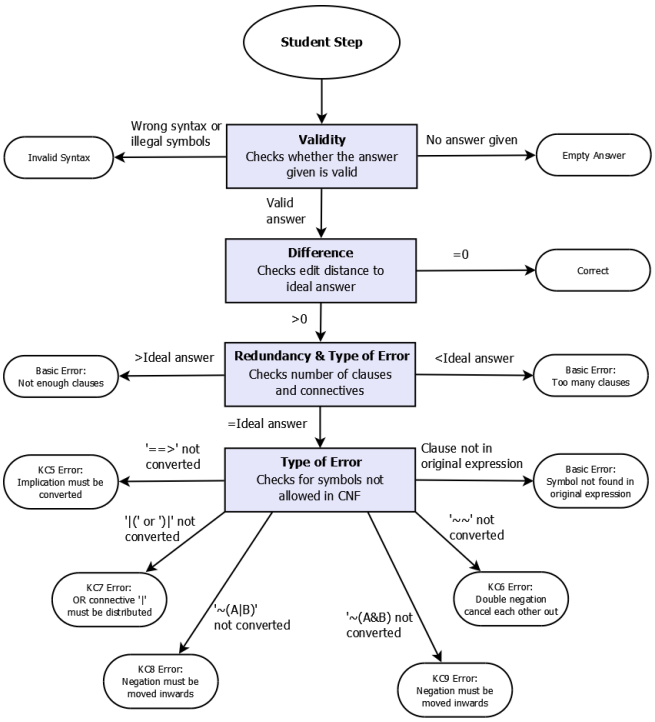

Figure 4. Full process of error diagnosis of P-res tutor

USER STUDY

In order to evaluate P-res tutor for its learning capabilities, more specifically, how much learning progress can be attained with it and how efficient it is at teaching, a user study was conducted. This study consisted of letting users interact with the system for a specified time and evaluating whether they had learned anything after, whether they were engaged during this interaction and whether the system is clear enough for users to understand without outside help.

Two different versions of the system were tested:

- P-res Tutor only with ZPDES algorithm;
- P-res Tutor with error diagnosis integrated with ZPDES algorithm.

The study was conducted using fourteen participants, seven for each version, with most being current or former students of Engineering in Instituto Superior Técnico. Instead of only selecting students of Computer Engineering, we included students from various different courses, such as Mechanical Engineering, Civil Engineering and Electrical Engineering, with most being second year students. In this way, we also manage to assess whether the system can teach a random college-level student a subject normally reserved for computer science students.

Before the study, every participant was given a short theoretical introduction to Propositional Logic, including the meaning of propositions, logical connectives, how to convert a sentence to CNF and how to apply the resolution rule to two propositions. For non-computer science students, this is the first contact they have with logic, more specifically, propositional logic (although some participants commented that they still remembered some logic from learning Philosophy in high school). Thus, we use the system to consolidate the knowledge we introduce in a short, roughly fifteen minutes, lesson.

After this introduction, we also present the interface and the activities of the system to reduce errors unrelated to learning logic. We then give unrestricted access to the system during a minimum of thirty minutes, starting with an initial student model and the stream of exercises. While the student hasn't learned everything, the stream keeps showing new exercises. After the minimum time has passed, the student can choose to stop the study or keep going to finish their learning. Assistance was also provided for the student for any question regarding use of the system while the test was being carried out.

At the end of the study, either from request of the student or from finishing the system, general questions about the interface, the exercise selection and how much propositional logic they have actually learned were asked and the answers recorded. Pen and paper were also supplied to each user.

Results

To evaluate each student's learning progress, our first approach was to register how many KCs had been learned after the interaction but, since most of the participants wished to finish the system and achieve all KCs by continuing to use the system after the thirty minutes, we instead chose to measure the percentage of correctness of each student for both approaches and compare the two populations of students, with the results in figure 5. We also measured the overall correctness of each KC to evaluate whether proper feedback and error diagnosis influence certain KCs learning, with the results in figure 6.

By only looking at the statistical results, we can clearly see the ZPDES plus Error Diagnosis system provided better results and was a more efficient approach since the students using this version achieved the same learning progress with less exercises. These results were expected as this approach provided a better environment for students to not repeat their errors, as proper feedback allowed students to understand where they had made their mistake and correct it in the future. This is also proven by the KC correctness values,

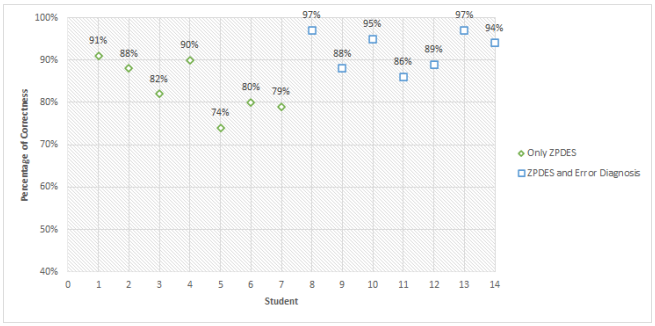

Figure 5. Graph showing the results of the correctness for each student in the user study

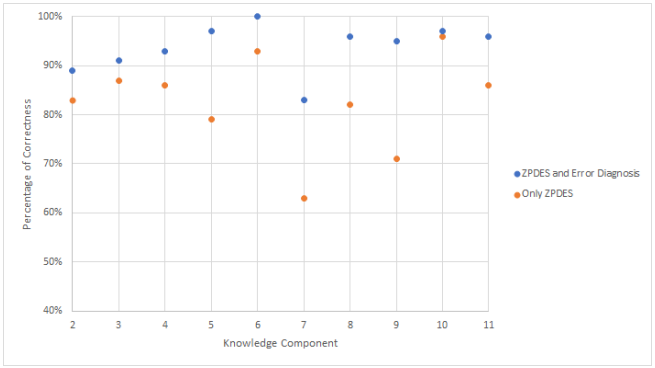

Figure 6. Graph showing the average correctness of each KC for each approach

as KCs that were harder to understand without feedback were now easier to attain and had a better correctness value.

In general, students seemed engaged when using the system, proven by the fact that most wished to end the training even when told the minimum time had passed. As such, these same students commented that they considered they had actually learned the basics of propositional logic and how to perform Propositional Resolution. Students who used the Error Diagnosis version of the system commented that the feedback and explanation of the previous exercise's solution was clear and concise. Users also complimented the clean interface and some commented that they considered the exercises were tailored for their needs.

Regarding criticisms, one of the most prominent opinions was that the system was not very user friendly, as activities and the descriptions of what to do were a bit vague, which left them unsure of what to do. This difficulty was pointed out mostly about the Connecting Clauses activity since it resembled a multiple choice activity. Another criticism was related to a lack of a sense of progress to the stream of exercises. A common comment during user testing was if the system would endlessly present exercises or if it had an end, since the only feedback received was related to the user's input and no feedback is given related to how much learning progress the student has achieved.

Overall, this study was conclusive in evaluating our both approaches with college students, although a more extensive study, with a more diverse population of different back-

grounds and ages would provide more conclusions about the system, its advantages and its shortcomings.

CONCLUSION AND FUTURE WORK

Concluding this dissertation, we sum up the achievements made with this work and discuss future work. We managed to create P-res Tutor, an ITS that is able to provide a customizable and adaptive experience for any student to learn Propositional Resolution. In terms of improvements, the priority would be to improve on the existing activities to make them more user friendly, by changing the activities descriptions or even the activities themselves. A possible improvement would be to change the form of input in CNF conversion exercises, instead of being text-based, all of the possible symbols and connectives would be in selection boxes and the input for the exercise would be created by clicking these boxes. A major extension for P-res Tutor would be to implement First-Order Logic and First-Order Resolution in the system. This would require some extensions to the $logic.py$ library, as it only includes a framework for Propositional Logic. It would also require implementation of the existential quantifier $\exists$ and the universal quantifier $\forall$ and their decomposition to CNF. Furthermore, both different KCs and a different KC tree would have to be created since First-Order Logic includes many concepts, such as predicates, which Propositional Logic does not consider. For this extension, help could be obtained from analysing ITSs which test First-Order Logic [5].

REFERENCES

1. Clancey, W. J. Intelligent Tutoring Systems: A Survey. *Explor. Artif. Intell.* (1986), 1–43.
2. Clement, B., Oudeyer, P. Y., and Lopes, M. A comparison of automatic teaching strategies for heterogeneous student populations. *Proc. 9th Int. Conf. Educ. Data Mining, EDM 2016* (2016), 330–335.
3. Clement, B., Roy, D., Oudeyer, P.-Y., and Lopes, M. Multi-Armed Bandits for Intelligent Tutoring Systems. 20–48.
4. Galafassi, C., Galafassi, F., Reategui, E., and Vicari, R. EvoLogic: Sistema Tutor Inteligente para Ensino de Lógica. 222–233.
5. Grivokostopoulou, F., Perikos, I., and Hatzilygeroudis, I. An intelligent tutoring system for teaching FOL equivalence. *CEUR Workshop Proc. 1009* (2013), 20–29.
6. Grivokostopoulou, F., Perikos, I., and Hatzilygeroudis, I. An Educational System for Learning Search Algorithms and Automatically Assessing Student Performance. *Int. J. Artif. Intell. Educ. 27*, 1 (2017), 207–240.
7. Grivokostopoulou, F., Perikos, I., Hatzilygeroudis, I., Grivokostopoulou, F., Perikos, I., Hatzilygeroudis, I., Tools, A., Grivokostopoulou, F., Perikos, I., and Hatzilygeroudis, I. Assistant Tools for Teaching FOL to CF Conversion To cite this version : HAL Id : hal-01521414 Assistant tools for teaching FOL to CF Conversion.
8. Lindsey, R. V., Shroyer, J. D., Pashler, H., and Mozer, M. C. Improving students' long-term knowledge retention through personalized review. *Psychological science 25*, 3 (2014), 639–647.
9. Mandl, H., and Lesgold, A. *Learning Issues for Intelligent Tutoring Systems*, 1 ed. 1989.
10. Murray, T. Authoring Intelligent Tutoring Systems : An Analysis of the State of the Art.
11. Nkambou, R., Bourdeau, J., and Psyché, V. Building intelligent tutoring systems: An overview. *Stud. Comput. Intell. 308* (2010), 361–375.
12. Nwana, H. S. Intelligent tutoring systems: an overview. *Artif. Intell. Rev. 4*, 4 (1990), 251–277.
13. Reis, M. S. RegexTutor : A Fully Online Intelligent Tutoring System.
14. Russell, S., Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson, 2020.
15. Shabani, K., Khatib, M., and Ebadi, S. Vygotsky's zone of proximal development: Instructional implications and teachers' professional development. *English language teaching 3*, 4 (2010), 237–248.
16. Slivkins, A. Introduction to multi-armed bandits. *Found. Trends Mach. Learn. 12*, 1-2 (2019), 1–286.
17. van der Bent, R., Jeuring, J., and Heeren, B. The diagnosing behaviour of intelligent tutoring systems. In *European Conference on Technology Enhanced Learning*, Springer (2019), 112–126.
18. VanLehn, K. The Behavior of tutoring systems. *Int. J. Artif. Intell. Educ. 16*, 3 (2006), 227–265.