SCHOOL OF SCIENCE AND ENGINEERING

INTELLIGENT DISCRETE MATH TUTORING SYSTEM

Capstone Project Final Report

Soukaina CHRIFI ALAOUI

Supervised by:

Dr.Violetta Cavalli Sforza

Fall 2015

INTELLIGENT DISCRETE MATH TUTORING SYSTEM

Capstone Final Report

Approved by the Supervisor

_____

Dr. Violetta Cavalli Sforza

ACKNOWLEDGEMENTS

Firstly, I wish to express my deepest gratitude to my supervisor, Dr. Violetta Cavalli Sforza, for her thought out guidance and constant support.

I place on record as well, my sincerest thanks to Dr. Naeem Sheikh who gave a considerable amount of his time helping in this project.

My greatest thanks go to my friends, Mohamed Nidal Ouazzani for supplying me with testing data while he is working on his own project, Saad Loubaris, Anasse Darfaoui and Idriss Said Alaoui for their valuable advice and constant helpfulness.

I take this opportunity to express gratitude to all of Al Akhawayn faculty, members and students that helped me throughout the different stages of my curriculum, and to my mother, aunts and little brother for their unconditional support and attention.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The main objective of this capstone project is to develop a tutoring assistant that can help discrete math students in their practice by analysing the problem statement and providing guidance accordingly when the student gets stuck.

This tutoring system is part of a collaborative project, the other part being the discrete problem generator on which Mohamed Nidal Ouazzani is working. The aim of the two parts is to design a system that can randomly generate discrete math problems, and provide guidance throughout the user's practice. Thus, Mohamed Nidal Ouazzani's system is providing the input to my tutoring system, this input being a set of randomly generated exercises and their structures in a text format, and my system's output is those problems' statements, comments on the answers provided by a user, and hints to guide him/her in case they respond incorrectly or get stuck.

The tutoring assistant parses the input given by the discrete problem generator, to evaluate the answers fed into it given the different types of problems and to provide guiding hints. It also uses symbolic mathematics for answer evaluation as linguistic parsing alone is not enough.

1. INTRODUCTION

The goal of this capstone project is to develop a tutoring assistant to help discrete math students practice exercises on permutations and combinations. For the tutoring assistant to offer the necessary guidance to a struggling student, it is necessary to consider both the mathematical problem to be solved and the solution proposed by the student.

This tutoring assistant makes use of a problem generator to test students' knowledge. This generator is developed by Mohamed Nidal Ouazzani as his capstone project. Ouazzani's discrete problem generator provides my program with a set of exercises organized in text files, accompanied by metadata that indicates their structure considering the agreed upon mathematical model commonly used by both of us. My system uses natural language processing techniques to parse the generated files and extract the problems structure.

Evaluating an answer goes through two phases. The first step is generating the appropriate formula for solving the exercise at hand; using the information given by Ouazzani's program and the internal knowledge the tutor has about such information. The second step is evaluating to which degree the student's answer matches the response found by the tutor, first in terms of the formula alone (regular expressions are used for this aim) and then in terms of computational value (the SymPy library takes care of that, because the numbers are too big to be handled without it).

All testing input was provided from Ouazzani's Discrete Problem Generator, however this tutoring system can, fundamentally, be used with exercises from any other sources (even manually provided) as long as they respect the format of input that will be explained later and use the same set of annotations to indicate the exercise's structure.

2. STEEPLE ANALYSIS

This intelligent discrete math tutoring assistant is a prototype that can be developed, later on, into a more widely accessible system that will help students better understand and practice the different mathematical concepts they study in a manner similar to working with a human tutor. The software can be even extended to other fields, such as Calculus, Probabilities, Physics, Chemistry, etc.

The main societal impact of the tutoring assistant is to support one of the 14 Grand Challenges of Engineering, as defined by the National Academy of Engineering, namely personalized learning. Personalized learning is indeed a growing field of interest in recent years, as research keeps advising that learning has to be tailored to individuals to give optimum results. This capstone project falls into the category of "intelligent" web-based educational systems, in which many advances have been made. Carnegie Learning's Cognitive Tutor Algebra is one of the most successful of these educational systems [1]. Other systems exist in the fields of computer programming, Physics and geometry. In the field of languages, the recommender system for learning English as a second language is one of the systems conceptualized to tailor reading lessons for individuals depending on the mistakes they make [2].

The tutor tries to serve the purpose of individualized learning by offering a computer system that will assist users in their practice and learning of discrete counting theorems at their personal pace, whether they just want to get extra preparation for their exams, expand their knowledge independently, or cannot afford formal courses.

3. METHODOLOGY

The first step in building the tutoring system was to construct a mathematical model that embodies the possible structures of the domain considered, in this case combinations and permutations problems. This mathematical model was developed in collaboration with Mohamed Nidal Ouzzani, and his supervisor, Dr. Nizar Naeem Sheikh, since they are working on a Discrete Math Problem Generator that was used in the testing phase of my own system.

Parsing the problem statements enables the system to extract the terminology used in the exercise to represent the elements of the model. The link between the terminology of the exercise and the elements of the model is used to provide hints to the student. The parsing is done using regular expressions, and Python is the programming language used for the implementation. SymPy, Python's library for symbolic mathematics, is also used in the system to provide mathematical expressions evaluation.

Finally, the assistant has to consider if the student attempt at solving an exercise is on the right track. If the answer initially provided by the student is not correct or the student expresses a need for help, the system uses a process of question and answer to guide him/her to the/a correct answer. To relieve the system from spending processing time and power on heavy parsing, and potentially difficult and inaccurate understanding of student responses in natural language (English), the student has to answer in a definite format given the instructions of the program (i.e. the answer should be given as a formula, and not as a numerical value and parentheses have to be correctly matched).

Django is the main web framework used for enabling access to the application via the web. AngularJS is the framework with which the front-end UI was developed.

4. THE MATHEMATICAL MODEL

Our mathematical model considered five types of problems that are typically taught in an undergraduate discrete math class, namely permutations, combinations, r-permutations, r-combinations and permutations with constrained repetition.

The mathematical model was meant to be in a graph form, branching into the different five types of problems, yet providing links that relate one type to another. The reason for that was that students can get confused between the different types and the program will be able to guide them from a wrong answer to a correct one, without them having to completely change their answer. Knowing those links is also beneficial to the student's learning. However, we later decided on using a more basic model that the form of a tree that embeds some of the intelligence necessary for guiding the student into it (such as hints and guiding questions). Another part of the intelligence is within the program itself (such as differentiating between different answers that have similar formula patterns). The reason behind this change of mind (using a tree instead of a graph) was pedagogical. During meetings held with Dr. Sheikh, who is one of the faculty members teaching the Discrete Math class at AUI, he explained that the model would rather be built considering how a student, typically, thinks about solving a certain problem. That way, the program can follow the same line of logic, and provide the necessary guidance accordingly, if errors were spotted.

Thus, for our considered domain, permutations and combinations, the model represented in Figure 1 below was considered:

- The problems are divided according to order (i.e. order matters and order doesn't matter)
- Then within these two branches, a sub-division is made: repetitions allowed and repetitions not allowed.
- Each of these branches can have sub-divisions, depending on the constraints of the exercise (e.g. unconstrained number of repetitions).

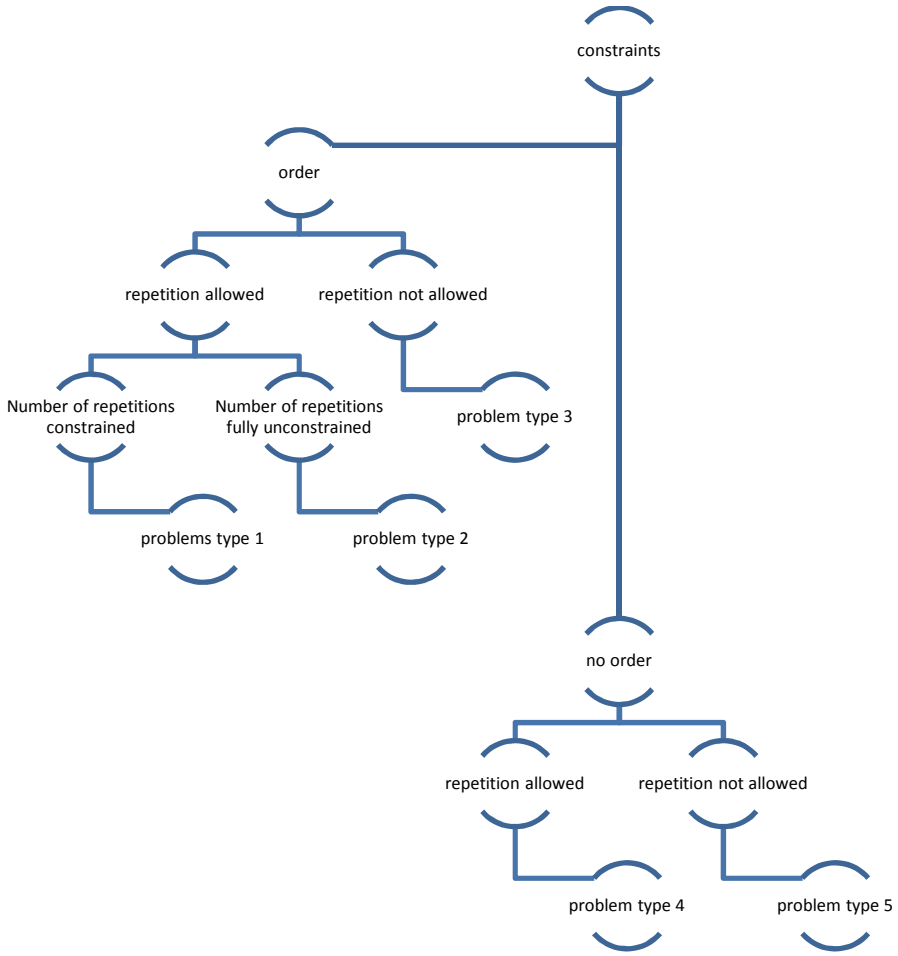

Figure 1: Mathematical model

The types of problems mentioned above are explained in the following table Figure 2:

Problem type	Explanation and examples
Type 1	Number of rearrangements of a given multi-set of symbols (symbol i occurs n_i times and has to be used exactly n_i times – sum of n_i 's is n)

	Formula: $\frac{n!}{n_1!n_2!\ldots n_k!}$! Explanation: n! permutations if treat symbols distinctly divided by repetition factor for each type of symbol that don't make string any different Example: how many rearrangements of Ouarzazate are there?
Type 2	Placing n objects into k places where each object can be used an unlimited number of times Formula: n^k Explanation: n choices at each of the k blanks Example: typing a string of length k using an alphabet with n letters
Type 3	Distinct objects in the set to draw from and the objects you select : permuting k over n objects Formula: $n!\ /\ (n-k)!$ Explanation: permutation of n objects over the permutation of the remaining objects which order doesn't matter. Example: creating a line of K objects out of N distinct objects, with k<n
Type 4	Choosing k objects out of n distinct types of objects without an order Formula: $(k\ +\ n-\ 1)!\ /\ k!\ (n-1)!$ Explanation: total of k objects divided into n regions by using n-1 bars (dividers) thus (n+k-1) symbols, we choose k places for the objects and n-1 for the bars. Example: buying 10 fruits given 6 different types of fruit available
Type 5	Choosing k objects out of n distinct objects without considering order Formula: $\frac{n!}{k!\ (n-k)!}$ Explanation: the same explanation as when order matters, but, since the order of the k objects is not considered here, the formula is divided by k! Example: choosing 11 out of 20 players for a team

Figure 2: Table of problem types

5. PROJECT ARCHITECTURE

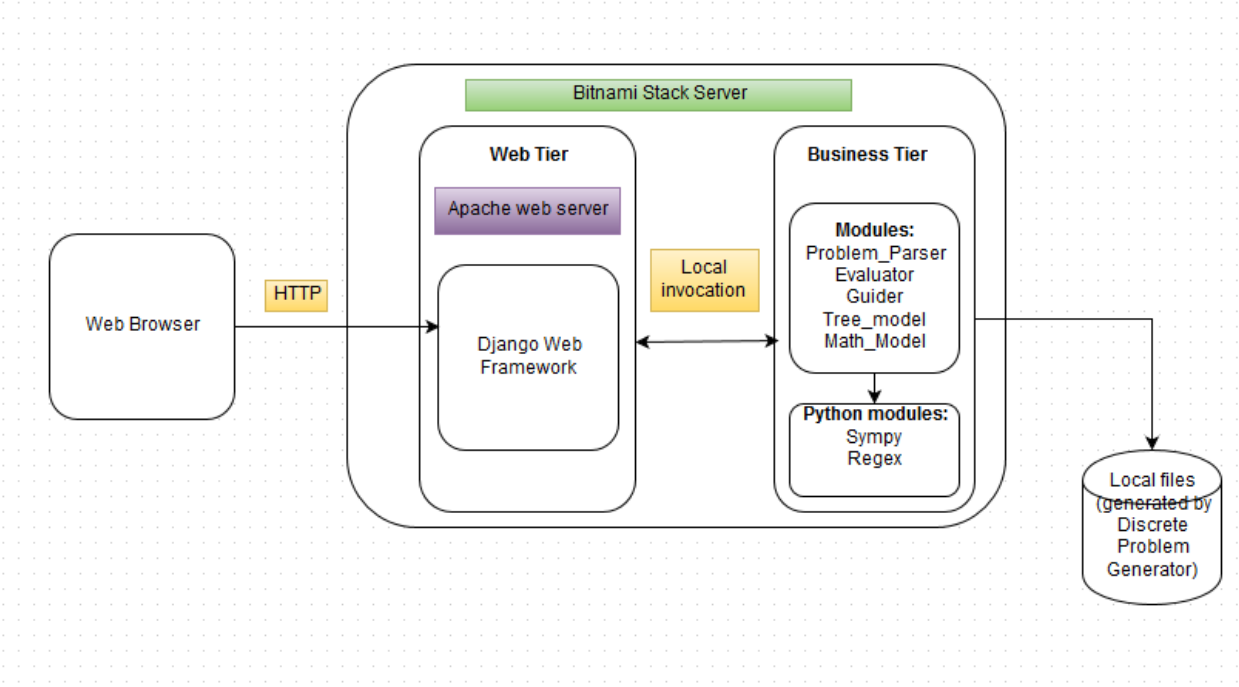

Figure 3: System architecture

As illustrated in Figure 3 above, the project is organized following the MVC paradigm, and as multi-tier application. The business tier contains the model part of the project along with external Python modules it depends on. The web tier surrounds the controller, embedded into Django web framework. The views are also under Apache web server, specifically under the "templates" folder of Django.

Figure 4 below explains the components of the system further as it shows the different packages of the program.

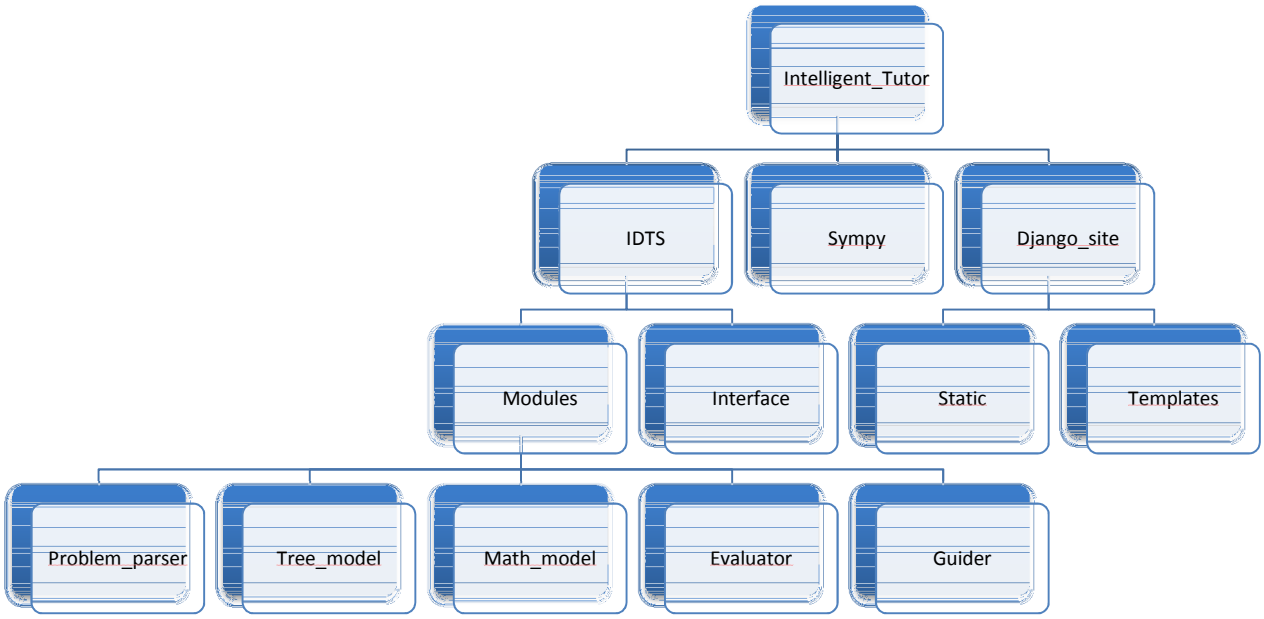

Figure 4: Package diagram

The package “modules” contains the plain classes of the model. This part was initially the focus of my capstone. The different classes under this package are detailed in the implementation section. The “interface” gathers the prototypes of all the functions needed to be exposed to the controller, and potentially other parties that want to use the tutor as a web service. This is the class that is be used in all interactions with the web tier, as it offers flexibility when updating the other classes and preserves the intelligence of the system from being exposed freely.

The views are divided into static and dynamic components, under Django’s package “templates” and developed using AngularJS.

The invocation of the functions is local for the time being, since all components of the project exist in the same directory.

6. IMPLEMENTATION

The majority of this section focuses on the different modules that constitute the model of the project, since it is the fundamental concern of my capstone. However, light is also be shed on other components.

6.1.Problem Parser

This module takes care of two things:

- Finding the appropriate file to take from the generated output of Mohamed Nidal Ouazzani's program given what the user has chosen to work on (i.e. permutations, combinations with their underlying categories or random exercises) and parsing input from the text files. Sets of exercises are taken from a directory of files generated by Ouazzani's program.
- Extracting the needed information from the files and storing it inside instances of a defined data structure that organizes it in a meaningful way for the system. The class defining the data structure, titled 'Problem' exposes attributes:
    - *Problem statement*: This attribute stores the problem description as a string.
    - *Problem type*: The value can range from 1 to 5 that are mapped to the mathematical model described above.
    - *Number of variables*: Most problem types have 2 variables, except for problem type 1 that can have from 1 to multiple variables depending on the problem constraints.

- *Problem variables*: This attribute is modeled by a tuple that maps the variables to their values.
- *Problem formula*: This attribute is set inside the program by the Evaluator module given the problem type and its variables.

Problems extracted from a file are stored under this structure, as the system processes them. The problem statement defined above is actually not the one displayed to the user. It contains a number of annotations and needs to undergo some processing that strips all those markups embedded within it before being sent to the controller for display.

As an example, we can take the following problem statement in Figure5:

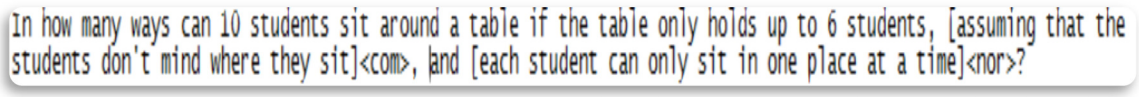

Figure 5: Example of problem statement as generated by the Discrete Problem Generator

F

This is the form it is generated with. The meaning of the embedded annotations will be explained later on. However, as it goes through processing, the annotations are no more visible, and the statement is displayed as shown in Figure 6.

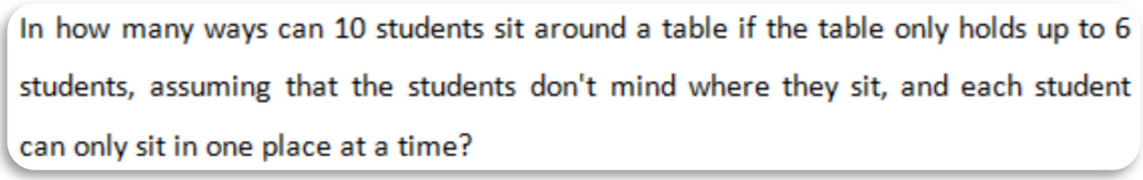

Figure 6: Example of a problem statement after processing

This problem, once stored inside an instance of the 'Problem' data structure, would have the format shown in Figure 7.

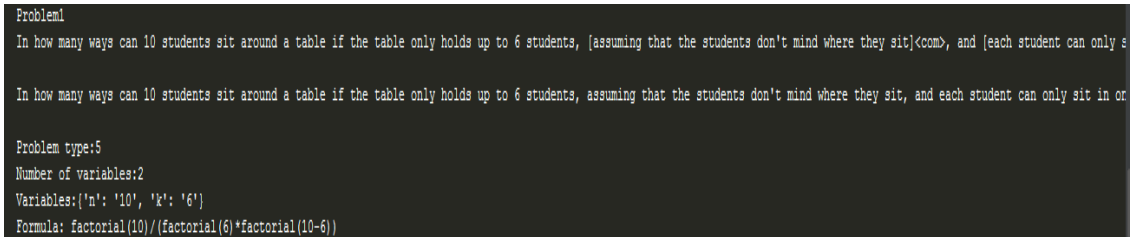

Figure 7: Problem data structure example

6.2.The Evaluator

This module takes care of generating the appropriate formula for solving an exercise given the type it maps to according to the mathematical model considered. It also takes care of computations, both for getting the numerical solution of the problem, and for computing the value of user's answer. This module uses SymPy, the symbolic mathematics library in Python, for numerical evaluation because of its built-in functions and its ability to handle large numbers without falling into an overflow situation.

Additionally, this module evaluates the correctness of the answer given by the user. First, the module matches the user's answer to the regular expressions representing the formulas that can be used for solving such a problem. The module considers various possible ways the solution can be written in, so that the user is not limited in the way s/he can think of the problem. For example, an answer to a combination-with-repetition problem can be given in its raw form as $\frac{factorial(n+k-1)}{factorial(k)*factorial(n-1)}$ or the user may choose to compute the values inside the brackets or to do some reductions. Because of the multiple formats the answer can be given in, this module uses helper functions while performing the matching. Going back to the mathematical model explained in the previous section, problems type 4 and 5 have very similar formulas; hence, a dedicated function takes care of differentiating if the answer given is for type 4 or for type 5. We can run into a similar problem with exercises type 1 and 3 in certain cases, thus, there is a dedicated function to evaluate the difference between them if needed.

Once the answer is validated at this level (i.e. the formula evaluated as right), the module moves to numerical validation using SymPy. SymPy computes the expression the student provided and compares it to the solution the system generated. This is used in case the student provided the right formula but made a mistake in the numerical values. If the answer is correct on both levels, then it is an accepted answer. Otherwise, guidance is provided depending on which level the mistake occurred in, but this is taken care of in another module, namely the Guider.

6.3. Tree Model

This module defines the structure and functions of the tree that is used to implement the mathematical model described above and used in the project. This module's implementation started by using the already existing search tree implementation in Python [10] and Tree library implementation in Python [11], and then merging and tweaking them depending on the needs of my system. The node structure is altered for this purpose and some functions are added or overridden as needed. A node structure looks as shown in Figure 8:

```
def __init__(self, key, data):
    self.left = None
    self.right = None
    self.data = data
    self.key = key
    self.parent = None
    self.hint = None
```

Figure 8: Structure of the 'Tree_Model' nodes

The nodes are defined as BST nodes with a left and right child. The choice of using a BST instead of a regular tree was made to take advantage of the simplicity of building such a data structure, the availability of a support for its functions via TreeLib in Python and its efficiency when it comes to search algorithms. The key is an integer value that allows inserting in such a way to get the same structure of the model, since no other piece of information can guarantee an ordering of the nodes.

The attribute "data" contains the constraint the node represents, such as order or repetition. The "hint" is the guiding question that is used by the Guider module. A detailed explanation on how this attribute is used is provided in the sub-section about the Guider module below.

Those attributes are embedded in the nodes as to support the system's extendibility. The system only supports tutoring for discrete math as of now but if its functionalities were to be extended to Calculus or Probabilities, all what will be needed is building the model to follow accordingly and change the Evaluator module to follow it.

6.4. Math Model

This module extends the Tree Model above to construct the mathematical model that the tutor uses. It initializes the nodes to fit the model's description and builds a BST out of them.

6.5. Guider

The Guider is invoked by the Evaluator if the latter finds that answer provided by the user is wrong. This module then uses the Math Model class to determine in which stage the user has taken a wrong turn (i.e. where the mistake was committed), using a post-fix traversal algorithm (since leaf nodes are the ones to be checked first). The node representing the wrong turn gives the hint needed at this stage. The hint is a question that is returned to the controller to be displayed. The user is, thus, asked to modify his/her answer accordingly. The new answer is evaluated again via the Evaluator module. This process repeats until the answer fed to the system is right, or the user decides to skip the given exercise.

In case the user is stuck, and does not know how to modify his/her answer. S/he can request to be given a hint. In this case, the Guider searches the problem statement for the keyword or phrase that can provide help given the mistake committed. This is possible thanks to the annotations embedded inside the problem statement and that can be summarized in five variations:

- <perm> : indicates a permutations problem (order is important)
- <com>: indicates a combinations problem (order is not important)
- <lre>: indicates a constrained repetition (only relevant to problem type 1)
- <rep>: indicates that repetition is allowed in this problem
- <nor>: indicates that no repetition is allowed.

Those annotations follow certain phrases or keywords marked between brackets. Just like in Figure 2 above. Those phrases/keywords constitute the hint to be shown to the user, either by highlighting it inside the already displayed problem statement or underlying it.

If we consider the same problem presented as an example before, Figure 9 is an example of a sample interaction with the tutor.

```
Problem1
In how many ways can 10 students sit around a table if the table only holds up to 6 students, assuming that the students don't mind where
they sit, and each student can only sit in one place at a time?

Formula:factorial(10)/(factorial(6)*factorial(10-6))
Please type your answer as a formula with numbers plugged in it, not as a number
Answer: factorial(10)
Should order be considered in this problem or not?
Answer: factorial(10+6-1)/(factorial(6)*factorial(10-6))
Not quite! You are correctly deducing that order doesn't matter in this problem, but is repetition allowed or not?
Answer: factorial(10)/(factorial(6)*factorial(16))
Your logic is correct, just take a look again at your numbers
Answer: factorial(10)/(factorial(6)*factorial(4))
Correct answer! You rock! :D
```

Figure 9: Sample interaction with the tutor

In the sample interaction above, the formula generated is shown right below the problem statement just for reference. Normally it is not displayed to the user.

The first answer given by the user would map to a type 1 problem answer, given the mathematical model considered, while the problem is a type 5. The algorithm used by the Guider module uses the mathematical model tree structure to point-out after which constraint node the mistake was made. In this case, this node would be the root. Therefore, the mistake is related to the "order or no order" constraint. The same logic applies for the second answer given in that sample interaction, since it is an answer to type 3 problems.

The third answer given by the user is a problem type 5 answer. The “wrong turn” happened after the “order not important” node, which means that the mistake is made considering repetition. The before last answer has the correct formula, but a wrong denominator value. The feedback is given in that regard.

6.6.The views

The web front-end is developed using AngularJS framework alongside HTTP, CSS and JavaScript. It is a single-page UI that includes 3 sections, displayed or hidden according to the input and action of the user.

- The first section is displayed by default in the home page. It allows the user to choose the problem types s/he wants to work on and the number of exercises s/he wants in each set (Figure 10).
- This same page has a section that displays the problem statement and offers the user the means to answer it: a field for typing his/her answer and a button to submit his/her answer once s/he finishes typing it (Figure 11).
- The guidance section (marked feedback below) displays the questions that are given throughout the guidance and includes a “give me a hint” button that offers the user extra help by highlighting key words/phrases that could help him/her discover the right answer (Figure 12).

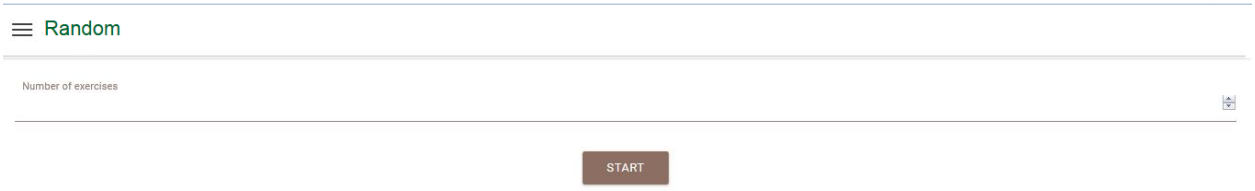

Figure 11: Index page first section: choice of category and number of exercises

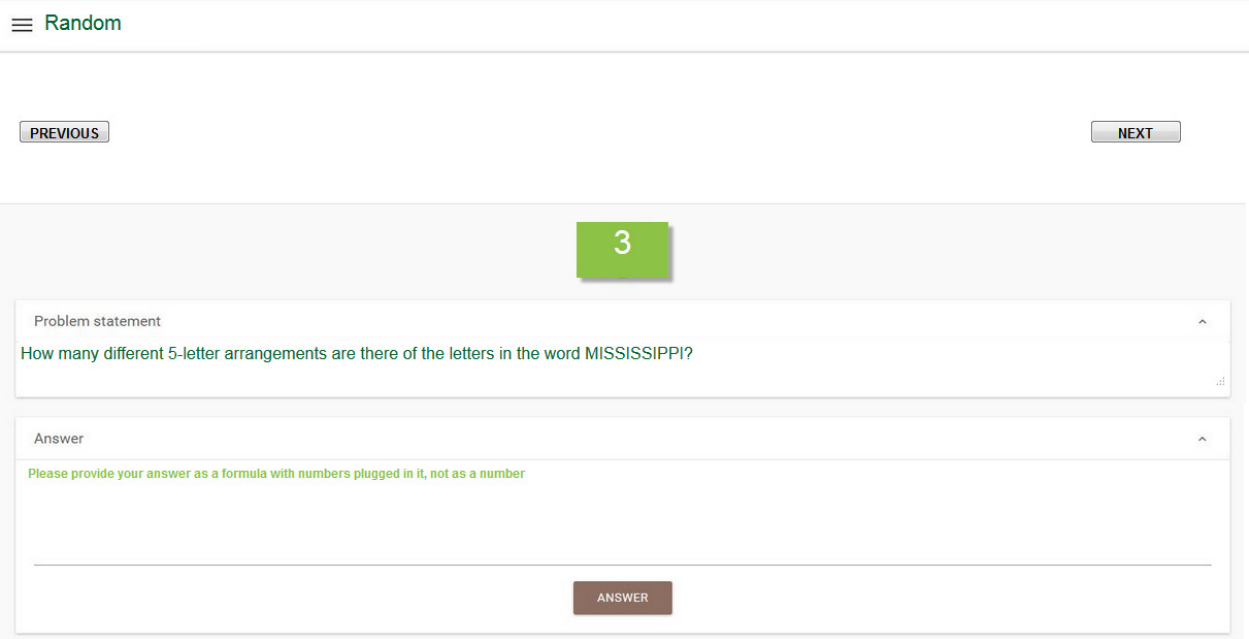

Figure 10: Index page second section: problem and answer field

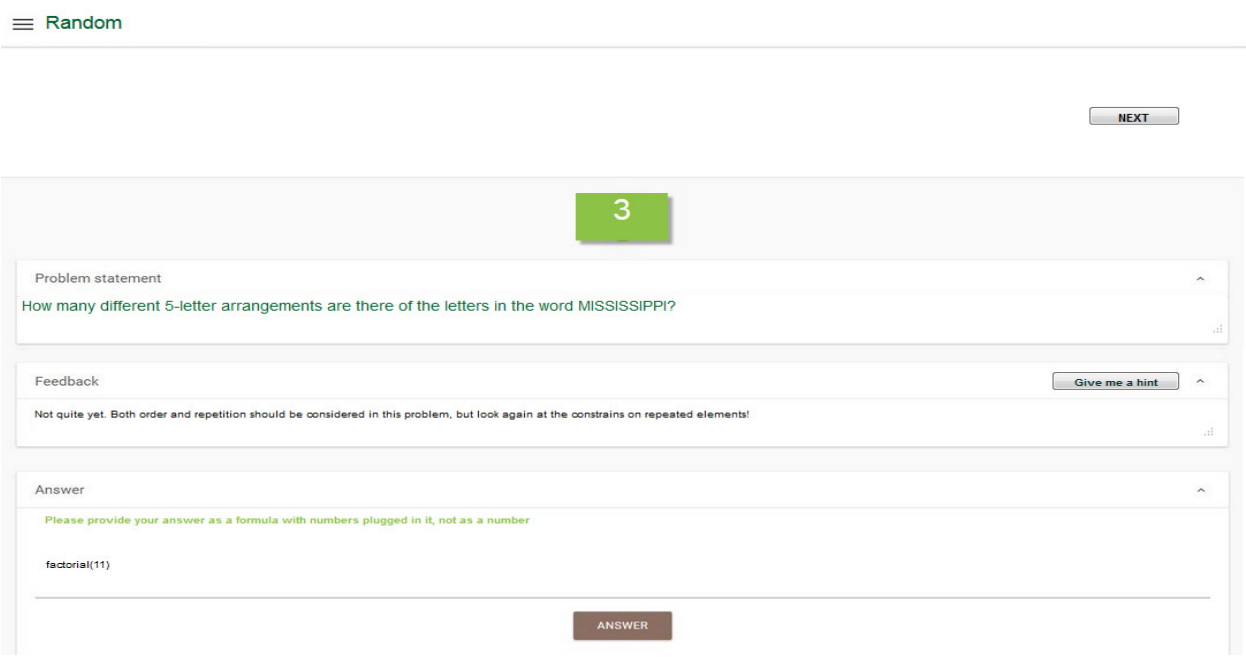

Figure 12: Index page third section: guidance

6.7. The Web Framework

Initially, Flask was the web framework considered for this project, because of its simplicity and nice support. However, using such micro-framework was hard given all what the application needed. One of the major problems was that I could not make SymPy work inside Flask development environment. Thus, instead, I used Django, which is a high level Python web framework based on Jinja 2 template. The support Django gives might be excessive for the purposes of this application, but it is a very well documented framework and has a very active community forum, two things that favored it over other Python based frameworks. Besides, its advanced functionalities can be very useful if the application is to be further enhanced.

6.8. The Deployment Environment

The application is going to be tested inside a local server. The web server, namely Apache 2.0, is set up on top of Bitnami LAMP (Linux, Apache, MySQL and PHP) stack on a Linux OS. The Apache web server hosts a virtual environment that supports both the web framework.

This environment installs a full Python environment locally and, hence, gives the opportunity to run the application as if it is deployed on a remote server. The environment contains all dependencies needed, such as Python 3.4., Django, Jinja2, WSGI, Werkzeug, PIP, and for this project, SymPy as well.

7. FUTURE WORK

Since a lot of work was done on the back-end part of the project, there are many improvements that can be made in the front end. First, the web interface can be improved to be more friendly and intuitive. It will be beneficial to include a revision section that reminds the user of the different formulas in case they want to look at them again.

This project can also be improved regarding how it would communicate with the Discrete Problem Generator developed by Mohamed Nidal Ouazzani. Instead of having a local directory of pre-generated files, a request could be made from the tutor to the generator and the tutor's input will be dynamically created at that time.

It will also be useful to make the system accessible through a mobile platform.

8. CONCLUSION

Working on this system was a rewarding experience for me. It allowed me to exercise what I learned in my computer science curriculum on a practical level and to learn new skills and new technologies as well.

Designing and implementing this project was challenging in many ways. I was working with technologies I had not used before, and such tutoring system required a lot of though, especially that no similar software had developed before for discrete mathematics (there are tutoring systems that use the same principle of guiding questions and hints, but for other fields such as Physics and Calculus).

References

[1] Koedinger, Kenneth R. "Intelligent Cognitive Tutors as Mode Ling Tool and Instructional Model." Carnegie Learning. June 1, 1998. http://pact.cs.cmu.edu/koedinger/pubs/Intelligent Tutors.pdf.

[2] "14 Grand Challenges for Engineering." NAE Grand Challenges for Engineering. 2015. http://www.engineeringchallenges.org/cms/challenges.aspx.

[3] Assef, Vinicius. "Web Frameworks for Python." The Python Wiki. October 1, 2015. Accessed September 1, 2015. https://wiki.python.org/moin/WebFramework

[4] Bird, Steven, and Ewan Klein. *Natural Language Processing with Python*. First ed. Beijing: O'Reilly, 2009.

[5] Das, M. K. *Discrete Mathematical Structures for Computer Scientists and Engineers*. Oxford, U.K.: Alpha Science International, 2007.

[6] "Django." The Web Framework for Perfectionists with Deadlines. 2015. https://www.djangoproject.com/.

[7] Jackson, Peter, and Isabelle Moulinier. *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization*. 2nd Rev. ed. Amsterdam: John Benjamins Pub., 2007.

[8] Kuchling, A.M. "Regular Expression HOWTO." Regular Expression HOWTO — Python 2.7.10 Documentation. 2015.

[9] "The Most Intelligent Python IDE." Python IDE & Django IDE for Web Developers : JetBrains PyCharm.

[10] "Welcome to SymPy's Documentation." SymPy 0.7.6.1 Documentation. 2014. Accessed October 1, 2015.

[11] "Django Documentation", Django, 2015. http://Django.pocoo.org/