An approach to automatic learning assessment based on the computational theory of perceptions

M. Gloria Sánchez-Torrubia [a,*], Carmen Torres-Blanc [a], Gracian Trivino [b]

[a] Applied Mathematics Department, Universidad Politécnica de Madrid Boadilla del Monte, Madrid, Spain
[b] European Centre of Soft Computing, Mieres, Asturias, Spain

A B S T R A C T

E-learning systems output a huge quantity of data on a learning process. However, it takes a lot of specialist human resources to manually process these data and generate an assessment report. Additionally, for formative assessment, the report should state the attainment level of the learning goals defined by the instructor.

This paper describes the use of the granular linguistic model of a phenomenon (GLMP) to model the assessment of the learning process and implement the automated generation of an assessment report. GLMP is based on fuzzy logic and the computational theory of perceptions. This technique is useful for implementing complex assessment criteria using inference systems based on linguistic rules. Apart from the grade, the model also generates a detailed natural language progress report on the achieved proficiency level, based exclusively on the objective data gathered from correct and incorrect responses. This is illustrated by applying the model to the assessment of Dijkstra's algorithm learning using a visual simulation-based graph algorithm learning environment, called GRAPHs.

1. Introduction

Assessment is a key part of any learning process. It is, however, resource intensive, and is not easy to automate if the criteria to be implemented are complex. Additionally, one of our goals as instructors is to examine what the key features of an effective e-learning system for teaching mathematical concepts and algorithms are. We defined eMathTeacher (Sanchez-Torrubia, Torres-Blanc, & Krishnankutty, 2008), a set of specifications that we used to build the GRAPHs[1] environment (Sanchez-Torrubia, Torres-Blanc, & Escribano-Blanco, 2010b), geared to graph algorithms and based on visual simulation (Malmi et al., 2004). Tools like these generate a huge amount of data, and it is impracticable to process data manually. Additionally, a formative learning process assessment should specify the level of attainment of the learning goals defined by the instructor. This may require the definition of complex criteria. This, plus the fact that the assessment should be as fast as possible, is another obstacle to manual data processing.

To solve this problem, we devised an interdisciplinary solution. This solution was to combine an e-learning system based on visual simulation with an automatic assessment expert system based on fuzzy inference. This hybrid system would output a formative assessment based on natural language (NL) reports. For this purpose, we designed a general linguistic granular model of learning assessment. Based on this design, we modelled the assessment of Dijkstra's algorithm learning based on visual simulation in GRAPHs. The implementation of this model generates a personal natural language report on the level of learner attainment across one or more algorithm simulations.

1.1. Visualization and eMathTeacher compliance

The 1981 *Sorting out Sorting* video (Baecker, Sherman, & Group, 1981) and the BALSA system (Brown & Sedgewick, 1984) introduced the use of visualization as algorithm learning support. It would be out of the question to detail the many applications developed since then, most of which are available on the web. For an extensive review of the state of the art on this topic, see (Shaffer et al., 2010). Most of these applications are applets that render visualizations of several algorithms. An important group is composed of systems that integrate a script language to build visualizations. This applies to systems like XTANGO/POLKA (Stasko, 1998), Swan (Shaffer, Heath, & Yang, 1996), ANIMAL (Rößling & Freisleben, 2002), JAWAA (Akingbade, Finley, Jackson, Patel, & Rodger, 2003), AlViE (Crescenzi & Nocentini, 2007) or JHAVÉ (Naps, 2005), which also include a collection of ready-made visualizations. The systems closest to our approach are JHAVÉ and, especially, TRAKLA2 (Malmi et al., 2004). JHAVÉ visualizations include

* Corresponding author. Tel.: +34 913367428; fax: +34 913367426.
E-mail address: gsanchez@fi.upm.es (M.G. Sánchez-Torrubia).

[1] http://www.dma.fi.upm.es/java/matematicadiscreta/GRAPHs/GRAPHs.jnlp.

pop-up questions addressing algorithm execution details and are useful for assessing user knowledge. TRAKLA2 includes user authentication and is designed to assess learner knowledge through the visual simulation of an algorithm on system-generated data. After the algorithm has been simulated, the process is compared with the model and the system outputs the number of correctly executed steps. The drawback of this method is that if there is an error at the start of the simulation, all the subsequent steps will be incorrect. This would result in an extremely low grade. The current trend in this field is towards the construction of hypertextbooks, which, apart from theoretical explanations, contain algorithm visualizations and practical exercises with self-assessment (Rößling, Mihaylov, & Saltmarsh, 2011). If such hypertextbooks are then integrated into a learning management system (LMS) (like Moodle, for example), the self-assessment functions could be integrated into the system to build an automated learning assessment system (Rößling et al., 2010).

The construction of visualization systems has gone hand in hand with theoretical research into their impact on learning and how best to improve their effectiveness (see, i.e., (Hundhausen, Douglas, & Stasko, 2002; Naps et al., 2002; Naps et al., 2003). When designed and used under the right conditions, visualization technologies for algorithm teaching have proved to be a very positive learning aid. Learners who were actively involved in visualization consistently outperformed other learners who viewed the algorithms passively (Hundhausen et al., 2002). Thus, when using an e-learning tool, the program should request continuous user-side interaction to rule out laziness and force learners to predict the next step. This was the key idea behind the eMathTeacher set of requirements.

Since the late 1990s, we have developed several web applications for graph algorithm learning based on visualization and aimed at promoting active learning (Sanchez-Torrubia, Torres-Blanc, & Lopez-Martinez, 2009; Sanchez-Torrubia et al., 2010b). These applications were designed on the basis of the philosophy underlying the eMathTeacher specifications. An e-learning tool is eMathTeacher compliant (Sanchez-Torrubia et al., 2008; Sanchez-Torrubia et al., 2009) if it works as a virtual math trainer. In other words, it has to be an online learning tool with built-in self-assessment that helps students (users) to actively learn math concepts or algorithms independently, correcting their mistakes and providing them with clues to find the right answer. Basically we design tools that act as *nagging* teachers working next to the student. The effectiveness of these specifications for building mathematical concept and algorithm learning support tools was examined in Sanchez-Torrubia et al. (2008), Sanchez-Torrubia et al. (2010b). Throughout the algorithm simulation, learners using eMathTeacher-compliant tools can be sure that all the steps they have taken are correct. Learner proficiency then can be assessed based on the mistakes they make in each step before they enter the correct value. This will prevent a grade from misrepresenting the learner's real proficiency level as a result of a mistake in the early stages of the simulation.

1.2. Learning process and formative assessment

A key part of the learning process is assessment. There are two types of assessment: summative and formative assessment. Summative assessment occurs at the end of a learning process and serves the purpose of determining the learner's level of knowledge. On the contrary, formative assessment is part of the learning process and takes place during this process. This type of assessment provides learners with feedback about what they know, what their weaknesses are and what they need to work on. Its goal is to let learners know how they are doing rather than to award grades. Therefore this assessment must be ongoing throughout the entire learning process.

Learning processes that generate a lot of data, such as courses implemented by e-learning tools, are hard to assess, and this takes up a great deal of instructors' time. Additionally, formative assessments should give learners detailed feedback about their strengths and weaknesses. Besides, rapid feedback increases the effectiveness of formative assessment. This, plus the huge amount of data to be processed, makes automation a number one priority. Automation will require an expert system capable of implementing similar criteria to the standards applied by an expert instructor. Natural language is what human beings (and therefore learners) understand best. On this ground, formative assessment should take the format not only of numerical data but also of reports written in natural language.

To automate formative assessment, we set ourselves the challenge of building a model that reproduces an instructor's reasoning for learning assessment. The expert system based on this model will provide learners with a natural language report on their attainment at each step of and throughout the learning process. Besides, the numerical grades output by the system will not be a direct result of counting correct and incorrect responses; the responses will be aggregated taking into account the importance of different error types, thereby emulating the instructor's criteria.

1.3. State of the art of fuzzy assessment

Fuzzy inference techniques are based on fuzzy set theory, fuzzy logic and approximate reasoning. They enable experts to express their reasoning in natural language and simulate this reasoning using linguistic rules and variables.

Fuzzy logic techniques have been used since 1995 (Biswas, 1995) to develop different learner knowledge assessment methods. Most of these methods aim to assess learners compared with a group, i.e., assign grades to fit a predefined reference curve. They are known as grading on a curve methods (Bai & Chen, 2008a; Bai & Chen, 2008b; Law, 1996; Li & Chen, 2009; Saleh & Kim, 2009). Other authors report methods for assessing learners directly. The first was presented by Biswas. His method is based on fuzzy set theory and similarity functions, where the grade of each response is represented by a fuzzy set. Chen and Lee (1999) report an improvement on Biswas' method and Chen and Wang (Chen & Wang, 2009) modify this method by using interval-valued fuzzy sets to represent the grade of each response. Ma and Zhou (2000) present a method aimed at assessing project-based learning. In this method, both the assessment criteria and the grades assigned to each project are decided by means of fuzzy group decision making techniques. Recently, Tay and Lim (2011) presented an assessment system, based on fuzzy IF–THEN rules with rule refinement, to obtain a finer-tuned assessment. One weakness of all these methods is that instructors have to participate in each and every assessment, as it is they that provide the fuzzy system input data. The expert is then irreplaceable. For example, Biswas' method multiplies the instructors' workload, as they have to give not one but six satisfaction levels for each response.

In Sanchez-Torrubia et al. (2010a) and Sanchez-Torrubia and Torres-Blanc (2010) we presented preliminary solutions to the problem of instructorless automatic assessment, where assessment criteria were not based merely on counting correct and incorrect responses. The proposed solutions are fuzzy inference systems based on Mamdani's direct method. The drawback of these systems is that they suffer either major defuzzification-induced information losses during the inference process or, if intermediate defuzzifications are omitted, an exponential increase of the computational complexity.

As already mentioned, our goal is to build an automated system that emulates the entire assessment process enacted by an instructor. This process ranges from counting objective data on correct

and incorrect responses, through the generation of a formative assessment report, to the assignment of a final grade. Of all the fuzzy techniques, the methodology that best suits our purpose is the granular linguistic model of a phenomenon (GLMP) (Martínez-Cruz, Van der Heide, Sánchez, & Trivino, 2012). We have used this methodology to build the GLMP of learning assessment that represents instructor reasoning. In this article, we present the model and its implementation, which provides formative learning assessment through reports written in natural language.

1.4. Problem-solving specifications

When a learner simulates an algorithm in the GRAPHs environment (see Section 2.1), the system generates an XML interaction log. Based on this XML file, our computer system must generate a natural language report stating how correct the simulation is. The report must contain different detail levels, ranging from the correctness of the execution of each step, through the correctness of each of the key aspects of the algorithm, to an overall appraisal of the simulation including a numerical grade. Additionally, if the learner completes several simulations one after the other, the report must also include an assessment of the learning process.

The article is organized as follows. In Section 2 we detail the background required to be able to understand the article as a whole. Section 3 focuses on the description of the GLMP applied to learning assessment. Section 4 tailors the GLMP to the assessment of Dijkstra's algorithm learning through simulation in G RAPHs. Section 5 describes the methodology used to build the assessment reports. Finally, Section 6 addresses the response provided by the expert system based on real data, and Section 7 outlines the conclusions.

2. Background

2.1. GRAPHs. Generation of assessment expert system input data

GRAPHs (Sanchez-Torrubia et al., 2010b) is an environment devised to visually simulate an algorithm running on a graph. This environment was designed to integrate eMathTeacher-compliant tools (Sanchez-Torrubia et al., 2008; Sanchez-Torrubia et al., 2009), that is, the learner simulates algorithm execution using the respective inputs. It is also important to highlight that, in an eMathTeacher-compliant tool, algorithm simulation does not proceed until the user enters the correct answer.

The environment runs on Java Web Start, features a language selector (currently English and Spanish) and is an extendible system, including a kernel and series of algorithms that will exploit its functionalities. The graphical interface was designed to be user-friendly, intuitive and visually attractive (see Fig. 1).

Users simulating an algorithm execution in GRAPHs have to manipulate several data structures depending on which algorithm step they are executing. At the end of the simulation, the system generates an XML interaction log (Sanchez-Torrubia & Torres-Blanc, 2010) that records both user errors and correct actions labelled by type. In the following we describe the structure of the interaction log file, showing the different types of user input and their action codes. The errors to be taken into account in the assessment will be defined depending on these codes and then normalized for use as system input data (see Section 4.1). These inputs are described by the request message identifier tag if they are correct or by the generated error message identifier tag if they are incorrect.

In particular, users simulating Dijkstra's algorithm execution have to manipulate fixed and unfixed nodes, select the active node, and update distances and predecessors among the set of (unfixed) nodes adjacent to the active node. Each of the correct and incorrect actions in the interaction is characterized by a tag. This tag corresponds to its identifier tag in the *properties* file, enabling the internationalization of a Java application (see Table 1).

2.2. Granular linguistic model of a phenomenon (GLMP)

The concept of granular linguistic model of a phenomenon (GLMP) is part of our contribution to Zadeh's computational theory of perceptions (CTP) (Zadeh, 1999). We have developed this concept as a result of a research line aimed at developing computational

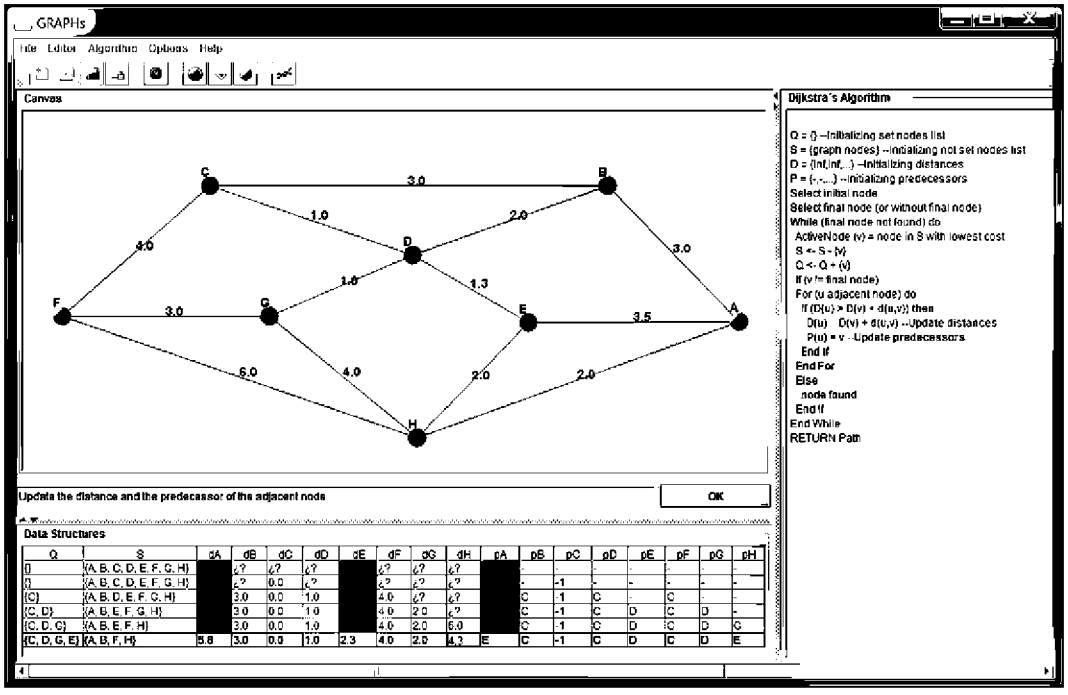

Fig. 1. GRAPHs simulating Dijkstra's algorithm.

Table 1
Pseudocode actions and GRAPHs outputs (∗ is MSG or ERR, depending on whether it corresponds to an error or a correct action).

Action code	Pseudocode action	XML tags
C_{1-1}	Selection of active node	<MSG_SEL_ ACTIVE_NODE_∗>
C_{1-2}	End of algorithm check (while condition)	<ALERT_FINISH_∗>
C_{1-3}	Final node check (if condition)	<ALERT_ACTIVE_IS_FINAL_∗>
C_{2-3}	Distance and predecessor update	<MSG_SEL_ UPDATE _MED_∗>
C_{2-1}	Set of adjacents check (for loop)	<ALERT_ADJACENT_∗>
C_{2-2}	Selection of adjacent Time taken	<MSG_SEL_ADJACENT_NODE_∗>

systems able to generate linguistic descriptions of data. Elsewhere, we have applied this concept to describe the behaviour of traffic on a roundabout (Trivino et al., 2010), generate financial reports from Internet data (Mendez-Nunez & Trivino, 2010), describe the surface of Mars (Alvarez-Alvarez, Sanchez-Valdes, & Trivino, 2011) and generate linguistic reports assessing the results of simulation sessions in a learner driver trainer (Eciolaza, Trivino, Delgado, Rojas, & Sevillano, 2011).

Let us introduce several earlier definitions to explain the concept of GLMP:

- **Designer**. We use the term designer to refer to a person or team of people that interprets the available data and produces the linguistic utterance. We say that the designer is the person who perceives and speaks when the computer perceives and speaks. In other words, the machine emulates the designer's way of thinking. Later on, we will see how the designer uses his/her own perceptions to create the GLMP.

Here, the *designer* is an expert teacher, and our aim is to automate learning process assessment. The *designer* uses her own experience to interpret the available data and then create a computational model that conforms to her assessment criteria.

- **Computational Perception (*CP*)** is the computational model of a unit of information acquired by the designer about the phenomenon to be modelled. In general, *CP*s correspond to specific parts of the phenomenon at a particular granularity level. A *CP* is a couple (A,W), where:
  - $A=(a_1,a_2,\ldots,a_n)$ is a vector of linguistic expressions (NL words or sentences) that represents the whole linguistic domain of *CP*. Each a_i describes the value of *CP* in a particular situation with a specific granularity level. These sentences can be either simple, e.g., a_i = "*The student's exercise is good*" or more complex, e.g., a_i = "*The student has progressed over the last month*".
  - $W=(w_1,w_2,\ldots,w_n)$ is a vector of validity degrees, also called weights, $w_i\in[0,1]$ assigned to each a_i in the specific context. The concept of validity depends on the application, e.g., it is a function of the truthfulness and relevancy of each sentence in its context of use.
- **Perception Mapping (*PM*)**. We use *PM* to create and aggregate *CP*s. There are many types of *PM*, and we will use several in this paper.

A *PM* is a tuple (U,y,g,T), where:
- $\boldsymbol{U}$ is a set of input *CP*s $U=\{u_1,u_2,\ldots,u_n\}$, where $u_i=(A_{u_i},W_{u_i})=\{(a_1^{u_i},w_1^{u_i}),(a_2^{u_i},w_2^{u_i}),\ldots,(a_{n_{ui}}^{u_i},w_{n_{ui}}^{u_i})\}$. In the special case of a first-order perception mapping (1-*PM*), U is a variable defined in the input data domain, e.g., the value $z\in\mathbb{R}$ provided by a thermometer.
- $\boldsymbol{y}$ is the output *CP* $y=(A_y,W_y)=\{(a_1^y,w_1^y),(a_2^y,w_2^y),\ldots,(a_{n_y}^y,w_{n_y}^y)\}$.
- $\boldsymbol{g}$ is the aggregation function with output $W_y=g(W_{u_1},W_{u_2},\ldots,W_{u_n})$, where W_y is the vector $(w_1^y,w_2^y,\ldots,w_{n_y}^y)$ of validity degrees assigned to each linguistic expression in y and W_{u_i} are the validity degrees of the input perceptions. Fuzzy logic researchers have developed many different types of aggregation functions. For example, $\boldsymbol{g}$ could be implemented using a set of fuzzy rules. In the special case of 1-*PM*s, g fuzzifies the data, i.e. g is built using a set of membership functions: $W_y=\left(\mu_{a_1^y}(z),\mu_{a_2^y}(z),\ldots,\mu_{a_{n_y}^y}(z)\right)=(w_1^y,w_2^y,\ldots,w_{n_y}^y)$, where $\mu_{a_j^y}(z)$ is the validity degree assigned to a_j^y and z is the input data.
- $\boldsymbol{T}$ is a text generation algorithm that allows generating the sentences in A_y. In simple cases, T is a linguistic template, e.g., "*The temperature in the room is {high | medium | low}*".

The **granular linguistic model of a phenomenon** consists of a network of *PM*s. Each *PM* receives a set of input *CP*s and propagates a *CP* upwards. We say that each output *CP* is explained by the *PM* using a set of input *CP*s. In the network, each *CP* covers specific aspects of the phenomenon with a set granularity level.

Using different aggregation functions and different linguistic expressions, the GLMP paradigm enables the *designer* to model his/her perceptions computationally.

Note that the output of PM_i is designed as CP_i in our notation, i.e., edges in the graph inherit the id-number from the generating node.

Fig. 2 shows an example where the *designer* uses several *PM*s and *CP*s to explain the top-order computational perception (2-PM_6) about energy consumption efficiency. The *designer* can use this top-order perception to answer a general question about the monitored phenomenon. Fig. 2 includes several selected sentences that describe the current situation accounting for the data provided by the sensors.

3. Granular linguistic model for the assessment phenomenon and its applications

In this section we will look at how the *designer*, i.e., the teacher, uses a GLMP to create a generic model of her own perception of the assessment process. Using different aggregation functions and different linguistic expressions, the GLMP paradigm provides enough

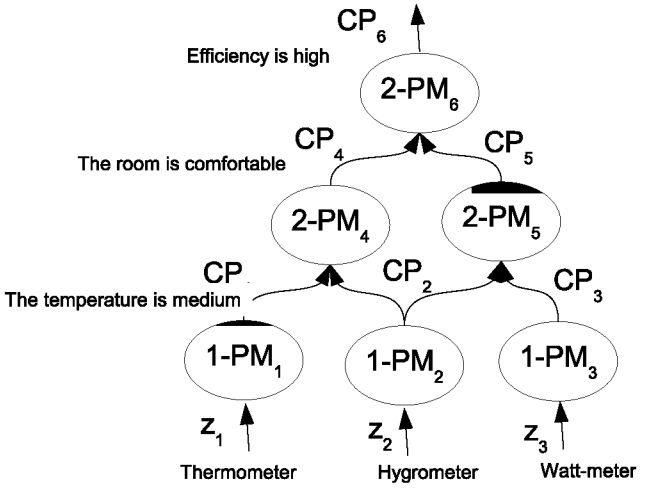

Fig. 2. Basic example of GLMP together with several generated sentences.

resources to design an automatic assessment tool for use in e-learning systems. The case study (in Section 4) details how to apply these concepts to assess the simulation of Dijkstra's algorithm.

When they execute the assess action, instructors group the learner errors according to the concept to which they are related and assesses them according to criteria of importance, repetition, etc. Instructors then aggregate these errors using an experience-based inference method to output the final grade. Instructors often perform much of this error clustering, assessment and aggregation almost automatically based on their knowledge and experience. This is the groundwork for building a GLMP to generically represent the process enacted to perform this task. This general model will then have to be tailored to each case by defining its constituent *CP*s and *PM*s. This way, the assessment task can be automated using objective and easily calculated input data.

3.1. Top-order computational perception

Designing a GLMP is an iterative process that starts with the definition of the top-order computational perception. Here, it is a matter of answering a general question about a learning process:

"What level of proficiency do students demonstrate/acquire after completing a set of exercises?"

A standard response to this question could be

"The correctness level is satisfactory in most of the important exercises, and the grade obtained is 8.1".

This would be generated using the template:

"The correctness level is {*unsatisfactory* | *satisfactory* | *very satisfactory*} in {*Few* | *Some* | *Most*} of the important exercises, and the grade obtained is {*fgrade*}".

Therefore, the *top-order-CP* will correspond to the final grade. This includes a numerical grade and an overall natural language assessment of all the corrected exercises.

3.2. First-order computational perceptions

Each 1-*CP* will represent one of the elementary errors that can occur during the completion of an exercise. The input data, normalized in [0,1], can be a quotient of either correct and incorrect actions or errors and the total number of occurrences of that type of action. 1-*PM* fuzzifies this datum using linguistic labels to output the vector of weights of 1-*CP*.

Each of these 1-*CP*s will represent the answer to the question

"What is the size of error E_i?"

And a standard response will be

"Error E_i is low",

which will be generated by the template

Error E_i is {*low*|*medium*|*high*}.

The 1-*CP*s can also indicate whether or not there is a critical error (see, i.e., Section 4.1.2).

3.3. Second-order computational perceptions

At the next level, elementary errors represented by 1-*CP*s are clustered depending on the types or concepts to which they are related. Note that this is not necessarily a classification, as one and the same error can express a weakness concerning more than one concept or have an impact on the performance of more than one task.

In this case, the 2-*CP* will represent the concept comprehension or task performance quality including errors that explain this perception. These 2-*CP*s will answer a question like

"How many E_{type} errors are there?"

And a standard response would be

"There are few E_{type} errors",

which will be generated by the template

There are {*very few*|*few*|*several*|*many*} E_{type} *errors.*

The 2-*CP* is the output of the 2-*PM* that is built from the explanatory *CP*s used as input. Its output is the result of fuzzy aggregation and will lead to a vector of validity degrees corresponding to the linguistic expressions that belong to the output 2-*CP*.

The aggregation function associated with each *PM* will implement assessment criteria like *"a few errors like this are permissible as they are probably the result of a minor slip-up"*, *"this is a major error and must be penalized"*, or *"these two errors together are a sign that this concept has been misunderstood"*. In the example GLMP below, we will show how to implement these criteria in practice using linguistic labels and if–then rules.

Similarly, the *CP*s at one level (and possibly from a previous level) can be regrouped to represent the comprehension of a concept or the skill at executing a more complex task.

As mentioned in Section 1.3, the designs proposed in (Sanchez-Torrubia et al., 2010a and Sanchez-Torrubia & Torres-Blanc, 2010) suffered from sizeable information losses or high computational complexity. With GLMP, however, the system processes errors without any information loss, as each *CP* is explained based on the weight vectors of subordinate *CP*s. Additionally, computational complexity is very low as the operations are performed on weight vectors instead of the whole fuzzy sets. On the other hand, the report on each error type is generated based on the linguistic expressions and validity degrees, or vector of weights. This is the output of the respective *PM*.

3.4. Summarizer and top-order computational perceptions

As mentioned earlier, the *top-order-CP* is able to answer a general question about the assessment phenomenon. This way, it can represent the assessment of a single exercise or a set of exercises, as described in Section 3.1, whichever is preferred. The set of exercises can represent the assessment of several topics or the learning process of one and the same concept over time. In the second case, the exercises completed last should carry a greater weight than exercises completed first in order to give consideration to learning progress. In the first case, each exercise will be rated depending on the relative importance of the topic under assessment. In both cases, the different assessment ratings of the various exercises will be represented by a vector of importance that will attach the respective weight to each exercise within the summary.

As mentioned in Section 3.1, the *top-order-CP* will answer a question like

"What level of proficiency do students demonstrate/acquire after completing a set of exercises?",

using a template like:

"The correctness level is {*unsatisfactory* | *satisfactory* | *very satisfactory*} in {*Few* | *Some* | *Most*} of the important exercises, and the grade obtained is {*fgrade*}.

The first part of this template has the linguistic structure of a *fuzzy summary* that can be expressed as follows (Yager, 1982, 1991; Zadeh, 1983):

The correctness level is S in Q B exercises

where

- Q is a fuzzy quantifier (Yager, 1993; Zadeh, 1983).
- B is a vector of importance (Kacprzyk & Yager, 2001; Kacprzyk & Zadrozny, 2005; Yager, 1996)
- S is a summarizer (Yager, 1982; Yager, 1991).

The validity degree of these linguistic summaries and the selection of the most informative summary will be calculated in each case using different techniques (see, i.e., Section 5.2).

If the model is applied to assess a single exercise, the *top-order-CP* will be the highest-ranking of the second-order *CP*s described in Section 3.3 and will answer a question like

What is the correctness level of the exercise?

The template will be:

The correctness level of the exercise is {poor | fair | good | very good | excellent} and the grade obtained is {grade}.

3.5. Applications

The above learning assessment model can be applied to any assessment system providing data on correct and incorrect responses. To tailor this model to a particular case, the designer will first have to select the elementary errors that the assessment is to take into account. These errors and the basic criteria on which they rely (linguistic labels) will define the 1-*PM*s that constitute the GLMP groundwork. The elementary errors will then be grouped by type, and the assessment criteria will be defined for each error type by means of the aggregation functions of the respective 2-*PM*s. This process can be repeated as often as necessary until the *top-order-CP* representing the overall assessment is reached.

The case study in Section 4 details how this model is built in the particular case of the assessment of Dijkstra's algorithm learning through a set of simulations performed in the GRAPHs environment. As described in Section 2.1, this learning environment creates an XML interaction log every time a learner simulates an algorithm running on a graph. The elementary errors, identified by the tags of a particular algorithm log, will serve to define the 1-*CP*s of an algorithm assessment GLMP. Depending on the algorithm in question, we will define the assessment criteria that will then be built into the model structure and functions. Once the model has been implemented, the data on correct and incorrect responses obtained from the log generated by the algorithm simulation will be used as input data for the algorithm simulation automatic assessment system.

Another interesting application of this model is the *intelligent* assessment of multiple-choice examinations. Here *intelligent* means capable of implementing differential criteria rather than just totting up correct and incorrect responses. Evidently, it would be too labour-intensive to tailor this model to assess just one examination, unless fixed errors and criteria (which can be kept unchanged for years) are used to build the model. Once the model has been built, it remains merely to detail the test items that provide the data on correct and incorrect responses for each 1-*CP*. Specifically, this model could exploit the built-in automatic correction facilities of learning management systems (LMS) by implementing instructor criteria for assessing multiple-choice examinations. As these systems capture data automatically, the assessment process is fully automated.

Similarly, this model could be applied to implement formative self-assessment systems using data generated by completing the hypertext-book exercises as mentioned in Section 1.1.

4. Granular linguistic model of Dijkstra's algorithm simulation assessment

Fig. 3 shows the diagram of the GLMP used to assess the proficiency level acquired after simulating Dijkstra's algorithm in GRAPHs environment.

In the following sections, we will detail each of the elements that constitute the *PM*s and *CP*s illustrated in Fig. 3. We will describe the rule sets used to aggregate the input variables that explain each 2-*CP* and the method used to build the sentences for inclusion in the natural language final report.

4.1. 1-CPs

The 1-*CP*s interpret student actions performed during learning sessions. As explained in Section 2.1, this information is available in an XML interaction log file created by the e-learning computational system (Sanchez-Torrubia & Torres-Blanc, 2010). Table 2 shows the relationship between the action codes (corresponding

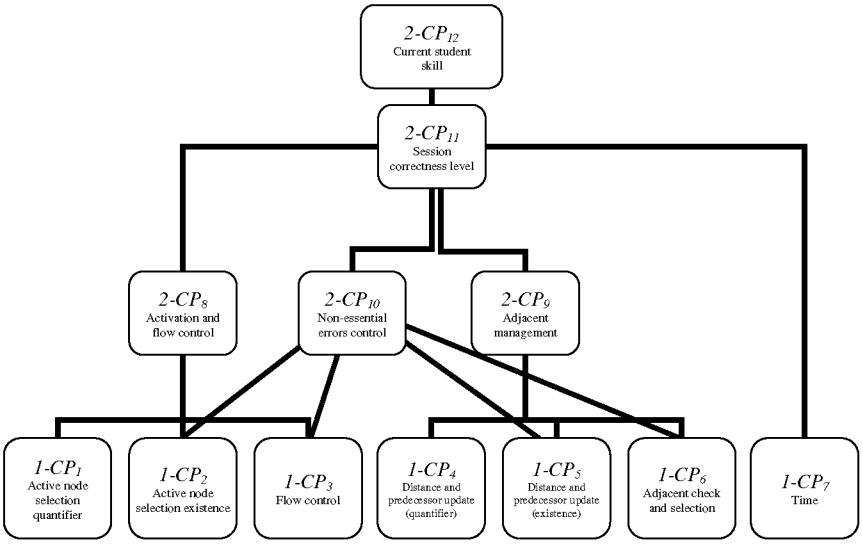

Fig. 3. Granular linguistic model of the assessment of a set of Dijkstra's algorithm simulations.

Table 2
GRAPHs outputs and assessment system inputs.

Action code	C_{1-1}	C_{1-1}	C_{1-2}	C_{1-3}	C_{2-3}	C_{2-3}	C_{2-1}	C_{2-2}	
Input datum	E_{1-1}	E_{1-1}	E_{1-23}		E_{2-3}	E_{2-3}	E_{2-12}		Time
First-order computational perceptions	1-CP_1	1-CP_2	1-CP_3		1-CP_4	1-CP_5	1-CP_6		1-CP_7

to XML tags as shown in Table 1) and the 1-*CP*s in the GLMP (note that some action codes have been grouped by similarity).

The data contained in the codes generated by the interaction log described in Section 2.1 are normalized and used to define the 1-*CP*s. E_i and T are normalized as follows:

$$E_i = \min\left\{\frac{Number\ of\ errors\ C_i}{Number\ of\ correct\ acctions\ C_i}, 1\right\},$$
$$Time = \frac{time\ taken}{maximum\ time}. \quad (1)$$

In an eMathTeacher-compliant tool, algorithm simulation does not continue unless the user enters the correct answer. For this reason, when the quotient in E_i (Eq. (1)) is greater than or equal to 1, the error rate indicates that the learner has no understanding of the algorithm step, and the data item is truncated at 1. This way, all the variables are defined in [0,1].

The interaction log elements are then grouped by 7 types (see Table 2) for the purpose of analysing the data in order to obtain the algorithm simulation correctness level. Each type outputs a 1-*CP* that describes the correctness level achieved for each action type performed during algorithm simulation.

4.1.1. 1-CP_1 active node selection (quantifier)

This 1-*CP* represents the number of errors during the selection of the next active node (which corresponds to the node nearest to the start node that has not yet been fixed). This is the most important step in each iteration, as it is the algorithm's basic principle. This 1-*CP* will have the highest weight in the final assessment. 1-CP_1 is the result of the fuzzification of error E_{1-1} and is obtained using 1-PM_1.

The input datum for 1-PM_1 is defined by Eq. (2):

$$E_{1\ 1} = \min\left\{\frac{Number\ of\ tags\ C_{1\ 1}\ error(<\text{MSG.SEL.ACTIVE.NODE.ERR}>)}{Number\ of\ tags\ C_{1\ 1}\ correct(<\text{MSG.SEL.ACTIVE.NODE.MSG}>)}, 1\right\}. \quad (2)$$

According to the definition of 1-*PM*, the 1-PM_1 of *level of the active node selection error (quantifier)* is a tuple (U,y,g,T), where:

U is a variable defined in the input data domain, i.e., $z = E_{1-1}$ (Eq. (2)).
$y = (A_{e_{1\ 1}}, W^1_{e_{1\ 1}}) = \{(a^1_1, w^1_1), (a^1_2, w^1_2), (a^1_3, w^1_3), (a^1_4, w^1_4)\}$, where $A_{e_{1\ 1}}$ are linguistic expressions that represent the *level of active node selection error* (e_{1-1}) and are built with four linguistic labels: {very small, small, large, very large}.

The linguistic labels membership functions {very small, small, large, very large} are illustrated in Fig. 4(a).

4.1.2. 1-CP_2 active node selection (existence)

In view of the importance of this step, we decided that error-free action performance should be given special consideration. The input datum for perception 1-CP_2 is the same quotient as described in Section 4.1.1. In this case, however, it describes exclusively whether or not $E_{1-1} = 0$. Its main application will be to award 10 or the *perfect* grade in the final assessment.

The 1-PM_2 of the *existence of an active node selection error* is a tuple (U,y,g,T), where:

U is a variable defined in the input data domain, i.e. $z = E_{1-1}$ (Eq. (2)).
$y = \left(A_{e_{1\ 1E}}, W^2_{e_{1\ 1E}}\right) = \{(a^2_1, w^2_1), (a^2_2, w^2_2)\}$, where $A_{e_{1\ 1E}}$ are linguistic expressions that represent *the existence of active node selection error* (e_{1-1E}) and are built with two linguistic labels: {null, not null}.
g is a two-component function whose image, on the numerical value $z = E_{1-1}$ (Eq. (2)), is the vector of validity degrees $W^2_{e_{1\ 1E}} = (\mu_{null}(z), \mu_{nnull}(z)) = (w^2_1, w^2_2)$ where $W^2_{e_{1\ 1E}} = \begin{cases} (1,0) & \text{if } E_{1-1} = 0 \\ (0,1) & \text{if } E_{1-1} \neq 0 \end{cases}$.
T *the active node selection error is* {*null* | *not null*}.

4.1.3. 1-CP_3 flow control

This 1-*CP* studies the proficiency level with respect to the algorithm flow controls (*while* and *if* statements) that discern whether or not the final node has been reached and whether or not the algorithm has finished. Because of the number of times these statements are repeated and as the response is *yes* or *no* in both cases, a few errors may be due to the occasional slipup. If the number of errors is above a certain limit, however, this is a sign of disregard for or unawareness of the foundations of any algorithm. On this ground, the errors are not penalized until a set limit is reached (see Fig. 4(b)). Otherwise, the overall understanding of the algorithm will be considered to be insufficient. The input datum 1-PM_3 is given by Eq. (3).

The flow control 1-PM_3 is a tuple (U,y,g,T), where:

U is the input data $z = E_{1-23}$ (Eq. (3)).
$y = (A_{e_{1\ 23}}, W^3_{e_{1\ 23}}) = \{(a^3_1, w^3_1), (a^3_2, w^3_2)\}$, where $A_{e_{1\ 23}}$ are linguistic expressions that represent the *flow control errors* (e_{1-23}) and are built from two linguistic labels: {acceptable, unacceptable}.
g is a two-component function whose image, on the numerical value $z = E_{1-23}$ (Eq. (3)), is the vector of weights $W^3_{e_{1\ 23}} = (\mu_{acc}(z), \mu_{unacc}(z)) = (w^3_1, w^3_2)$.
T *the flow control error is {acceptable | unacceptable}.*

$$E_{1-23} = \min\left\{\frac{No\ of\ tags\ C_{1-2}\&C_{1-3}\ errors(<\text{ALERT\_FINISH\_ERR}>\ \&\ <\text{ALERT\_ACTIVE\_IS\_FINAL\_ERR}>)}{No\ of\ tags\ C_{1-2}\&C_{1-3}\ correct(<\text{ALERT\_FINISH\_MSG}>\ \&\ <\text{ALERT\_ACTIVE\_IS\_FINAL\_MSG}>)}, 1\right\}. \quad (3)$$

g is a four-component function whose image, on the numerical value $z = E_{1-1}$ (Eq. (2)), is the vector of weights or validity degrees $W^1_{e_{1\ 1}} = \left(\mu_{very-small}(z), \mu_{small}(z), \mu_{large}(z), \mu_{very-large}(z)\right) = (w^1_1, w^1_2, w^1_3, w^1_4)$ obtained from the linguistic label membership functions.
T *the active node selection error is* {*very small* | *small* | *large* | *very large*}.

The membership functions for the linguistic labels {acceptable, unacceptable} are illustrated in Fig. 4(b).

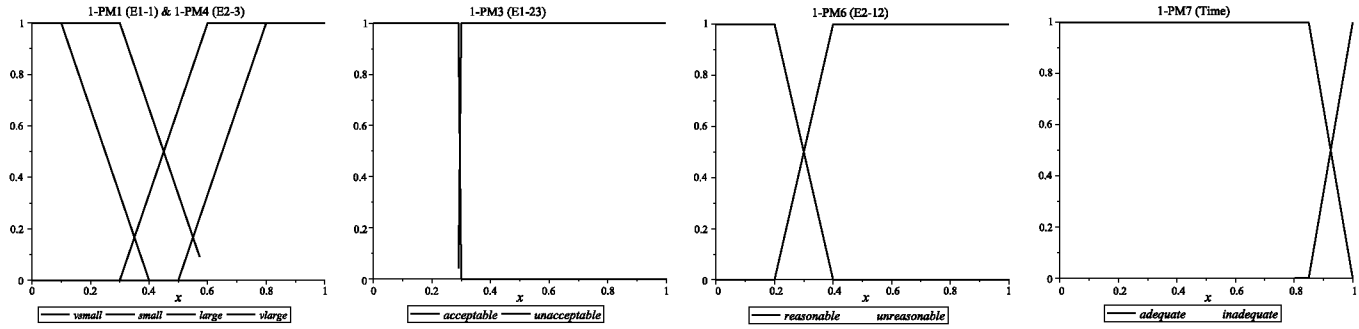

Fig. 4. Membership functions for the linguistic labels of 1-*PMs* for variables E_{1-1} and E_{2-3} (a), E_{1-23} (b), E_{2-12} (c) and *Time* (d).

4.1.4. 1-CP_4 distance and predecessor update (quantifier)

This quantity reflects the quotient between the number of correct and incorrect responses when updating the distances and predecessors of adjacent nodes to the active node. This is another very important step in the iteration, as it will calculate the distance and store the best path (retrievable from the predecessors). Therefore, this perception will also carry an important weight in the final assessment. The 1-PM_4 input datum is given by Eq. (4):

$$E_{2\ 3} = \min\left\{\frac{No\ of\ tags\ C_{2\ 3}\ error(< \text{MSG\_SEL\_UPDATE\_MED\_ERR} >)}{No\ of\ tags\ C_{2\ 3}\ correct(< \text{MSG\_SEL\_UPDATE\_MED\_MSG} >)}, 1\right\}. \tag{4}$$

The 1-PM_4 of *level of distances and predecessors update error (quantifier)* is a tuple (U,y,g,T), where:

U is the input data $z = E_{2-3}$ (Eq. (4)).
$y = \left(A_{e_{2\ 3}}, W^4_{e_{2\ 3}}\right) = \{(a^4_1,w^4_1),(a^4_2,w^4_2),(a^4_3,w^4_3),(a^4_4,w^4_4)\}$, where $A_{e_{2\ 3}}$ are linguistic expressions that represent the *level of distance and predecessor update error* (e_{2-3}) and are built with four linguistic labels: {very small, small, large, very large}.
g is a four-component function whose image, on the numerical value $z = E_{2-3}$ (Eq. (4)), is the vector of weights $W^4_{e_{2\ 3}} = (\mu_{very-small}(z),\mu_{small}(z),\mu_{large}(z),\mu_{very-large}(z)) = (w^4_1,w^4_2,w^4_3,w^4_4)$.
T the distance and predecessor update error is {very small | small | large | very large}.

The membership functions for {very small, small, large, very large} are illustrated in Fig. 4(a).

4.1.5. 1-CP_5 distance and predecessor update (existence)

As with the selection of the active vertex, we give special consideration to the fact that this action is error free. The input datum of 1-PM_5 is E_{2-3} but exclusively describes whether or not $E_{2-3} = 0$. Its main application will be to award 10 or the *perfect* grade in the final assessment.

The 1-PM_5 of the *existence of distance and predecessor update error* is a tuple (U,y,g,T), where:

U is the input data $z = E_{2-3}$ (Eq. (4)).
$y = \left(A_{e_{2\ 3E}}, W^5_{e_{2\ 3E}}\right) = \{(a^5_1,w^5_1),(a^5_2,w^5_2)\}$, where $A_{e_{2\ 3E}}$ are linguistic expressions that represent *the existence of distance and predecessor update error* (e_{2-3E}) and are built with two linguistic labels: {null, not null}.
g is a two-component function whose image, on the numerical value $z = E_{2-3}$ (Eq. (4)), is the vector of validity degrees $W^5_{e_{2\ 3E}} = (\mu_{null}(z),\mu_{nnull}(z)) = (w^5_1,w^5_2)$, where $W^5_{e_{2\ 3E}} = \begin{cases}(1,0) & \text{if } E_{2-3}=0\\ (0,1) & \text{if } E_{2-3}\neq 0\end{cases}$.
T the distance and predecessor update error is {null | not null}.

4.1.6. 1-CP_6 adjacent check and selection

This 1-*CP* aims to find out if the student has understood that the only vertices modified in this algorithm step are the adjacents to the active vertex that have not yet been fixed. As with 1-CP_3, this step is very often repeated, meaning that a few errors may be due to occasional mistakes. The 1-PM_6 input datum is given by Eq. (5).

The 1-PM_6 of *adjacents check and selection* is a tuple (U,y,g,T), where:

$$E_{2-12} = \min\left\{\frac{No\ of\ tags\ C_{2-1}\ \&\ C_{2-2}\ errors(< \text{ALERT\_ADJACENT\_ERR} > \&\ < \text{MSG\_SEL\_ADJACENT\_NODE\_ERR} >)}{No\ of\ tags\ C_{2-1}\ \&\ C_{2-2}\ correct(< \text{ALERT\_ADJACENT\_MSG} > \&\ < \text{MSG\_SEL\_ADJACENT\_NODE\_MSG} >)}, 1\right\}. \tag{5}$$

U is the input datum $z = E_{2-12}$ (Eq. (5))
$y = \left(A_{e_{2\ 12}}, W^6_{e_{2\ 12}}\right) = \{(a^6_1,w^6_1),(a^6_2,w^6_2)\}$, where $A_{e_{2\ 12}}$ are linguistic expressions that represent the *adjacent check and selection errors* (e_{2-12}) and are built from two linguistic labels: {reasonable, unreasonable}.
g is a two-component function whose image, on the numerical value $z = E_{1-12}$ (Eq. (5)), is the vector of weights $W^6_{e_{2\ 12}} = (\mu_{re}(z),\mu_{unre}(z)) = (w^6_1,w^6_2)$.
T the adjacent check and selection error is {reasonable | unreasonable}.

The membership functions for {reasonable, unreasonable} are illustrated in Fig. 4(c).

4.1.7. 1-CP_7 time

This perception rates how long it takes the student to simulate the algorithm. As the main goal is error-free simulation, this item will only influence the final result if it is close to the maximum acceptable time. The 1-PM_7 input datum is given by Eq. (6):

$$Time = \frac{time\ taken}{maximum\ time}. \tag{6}$$

The 1-PM_7 of *time* is the tuple (U,y,g,T), where:

U is the input datum z = *Time* (Eq. (6)).
$y = \left(A_{time}, W^7_{time}\right) = \{(a^7_1, w^7_1), (a^7_2, w^7_2)\}$, where A_{time} are linguistic expressions that represent the *time* it takes to simulate the algorithm and are built from {adequate, inadequate}.
g is a two-component function whose image, on the numerical value of z = *Time* (Eq. (6)), is the vector of validity degrees of $W^7_{time} = (\mu_{ad}(z), \mu_{inad}(z)) = (w^7_1, w^7_2)$.
T the time taken to perform the algorithm simulation is {adequate | inadequate}.

The membership functions for {adequate, inadequate} are illustrated in Fig. 4(d).

4.2. 2-CP

At a second level, we define three 2-*CP*s explained by 1-*CP*s (see Fig. 3): 2-CP_8 *activation and flow error*, 2-CP_9 *adjacent management error* and 2-CP_{10} *existence of essential errors*. According to the general model described in Section 3, each of these 2-*CP*s represents a concept or work area within Dijkstra's algorithm simulation or, as in the case of 2-CP_{10}, a special circumstance related to the quality of the simulation execution, which the professor intends to take into account. These three 2-*CP*s explain 2-CP_{11} that represents the *simulation correctness level*, which is used to output the assessment of a full simulation of Dijkstra's algorithm.

4.2.1. 2-CP_8 activation and flow error

2-CP_8 sets out to describe node activation and flow control errors and is explained by 1-CP_1 and 1-CP_2 (perceptions describing active node selection error $E_{1\text{-}1}$) and 1-CP_3 (perception describing *while* and *if* statement simulation error $E_{1\text{-}23}$). Fig. 5 illustrates the membership functions used for the linguistic labels of *activation* and *flow* error.

The 2-PM_8 of *activation and flow error* is a tuple (U,y,g,T), where:

U is the set of input *CPs* 1-CP_1, 1-CP_2 and 1-CP_3, $U = \left\{\left(A_{e_{1\ 1}}, W^1_{e_{1\ 1}}\right), \left(A_{e_{1\ 1E}}, W^2_{e_{1\ 1E}}\right), \left(A_{e_{1\ 23}}, W^3_{e_{1\ 23}}\right)\right\}$.
$y = \left(A_{e_{af}}, W^8_{e_{af}}\right) = \{(a^8_1, w^8_1), (a^8_2, w^8_2), (a^8_3, w^8_3), (a^8_4, w^8_4), (a^8_5, w^8_5)\}$, where $A_{e_{af}}$ are linguistic expressions that represent the *level of activation and flow error* (e_{af}) and are built with five linguistic labels: {very low, low, medium, high, very high}.
g is a five-component aggregation function, whose image is the vector of validity degrees $W^8_{e_{af}} = g\left(W^1_{e_{1\ 1}}, W^2_{e_{1\ 1E}}, W^3_{e_{1\ 23}}\right) = g\left((w^1_1, w^1_2, w^1_3, w^1_4), (w^2_1, w^2_2), (w^3_1, w^3_2)\right) = \left(w^8_1, w^8_2, w^8_3, w^8_4, w^8_5\right)$.
T the activation and flow error is {very low | low | medium | high | very high}.

Fig. 5 shows the membership functions for labels {very low, low, medium, high, very high}.

As mentioned in Section 4.1.1, the selection of the active node is a key feature of Dijkstra's algorithm. On the other hand, only a very small number of flow control errors is acceptable (see Section 4.1.3). Additionally, the errors are somehow accruable, and a build-up of either of the two types should be penalized. These features are reflected in the following Mamdani-type rule set:

(1) IF $e_{1\text{-}23}$ is unacceptable AND $e_{1\text{-}1}$ is very large AND $e_{1\text{-}1E}$ is no error THEN e_{af} is very high.
(2) IF $e_{1\text{-}23}$ is unacceptable AND $e_{1\text{-}1}$ is very large AND $e_{1\text{-}1E}$ is error THEN e_{af} is very high.

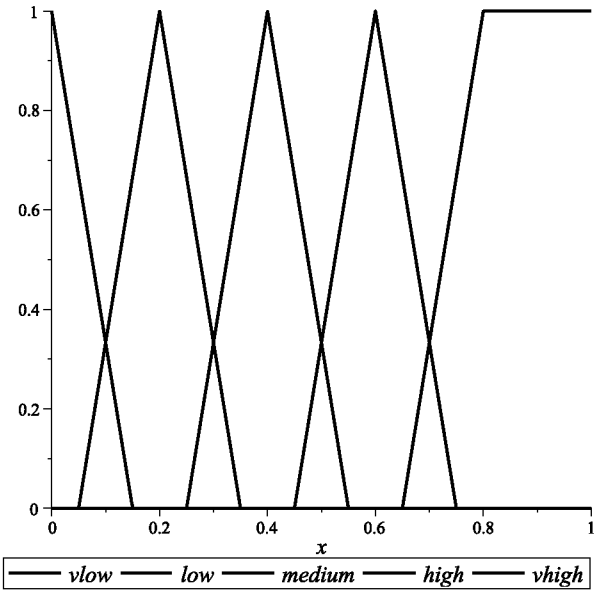

Fig. 5. Membership functions for the linguistic labels of 2-PM_8 and 2-PM_9.

(3) IF $e_{1\text{-}23}$ is unacceptable AND $e_{1\text{-}1}$ is large AND $e_{1\text{-}1E}$ is no error THEN e_{af} is very high.
(4) IF $e_{1\text{-}23}$ is unacceptable AND $e_{1\text{-}1}$ is large AND $e_{1\text{-}1E}$ is error THEN e_{af} is very high.
(5) IF $e_{1\text{-}23}$ is unacceptable AND $e_{1\text{-}1}$ is small AND $e_{1\text{-}1E}$ is no error THEN e_{af} is high.
(6) IF $e_{1\text{-}23}$ is unacceptable AND $e_{1\text{-}1}$ is small AND $e_{1\text{-}1E}$ is error THEN e_{af} is very high.
(7) IF $e_{1\text{-}23}$ is unacceptable AND $e_{1\text{-}1}$ is very small AND $e_{1\text{-}1E}$ is no error THEN e_{af} is medium.
(8) IF $e_{1\text{-}23}$ is unacceptable AND $e_{1\text{-}1}$ is very small AND $e_{1\text{-}1E}$ is error THEN e_{af} is high.
(9) IF $e_{1\text{-}23}$ is acceptable AND $e_{1\text{-}1}$ is very large AND $e_{1\text{-}1E}$ is no error THEN e_{af} is very high.
(10) IF $e_{1\text{-}23}$ is acceptable AND $e_{1\text{-}1}$ is very large AND $e_{1\text{-}1E}$ is error THEN e_{af} is very high.
(11) IF $e_{1\text{-}23}$ is acceptable AND $e_{1\text{-}1}$ is large AND $e_{1\text{-}1E}$ is no error THEN e_{af} is high.
(12) IF $e_{1\text{-}23}$ is acceptable AND $e_{1\text{-}1}$ is large AND $e_{1\text{-}1E}$ is error THEN e_{af} is high.
(13) IF $e_{1\text{-}23}$ is acceptable AND $e_{1\text{-}1}$ is small AND $e_{1\text{-}1E}$ is no error THEN e_{af} is very low.
(14) IF $e_{1\text{-}23}$ is acceptable AND $e_{1\text{-}1}$ is small AND $e_{1\text{-}1E}$ is error THEN e_{af} is low.
(15) IF $e_{1\text{-}23}$ is acceptable AND $e_{1\text{-}1}$ is very small AND $e_{1\text{-}1E}$ is no error THEN e_{af} is very low.
(16) IF $e_{1\text{-}23}$ is acceptable AND $e_{1\text{-}1}$ is very small AND $e_{1\text{-}1E}$ is error THEN e_{af} is very low.

We find that rules 1, 3, 9 and 11 above do not affect the design, as they can never fire. The rule set will be used to calculate weights $W^8_{e_{af}}$. For example, w^8_1, which is the validity degree of the expression *The activation and flow error is very low*, is calculated according to Eq. (7). Similar calculations are used to output the other validity degrees of this perception and 2-CP_9 and 2-CP_{11}.

$$w^8_1 = max\left(min\left(w^1_2, w^2_1, w^3_1\right), min\left(w^1_1, w^2_1, w^3_1\right), min\left(w^1_1, w^2_2, w^3_1\right)\right). \quad (7)$$

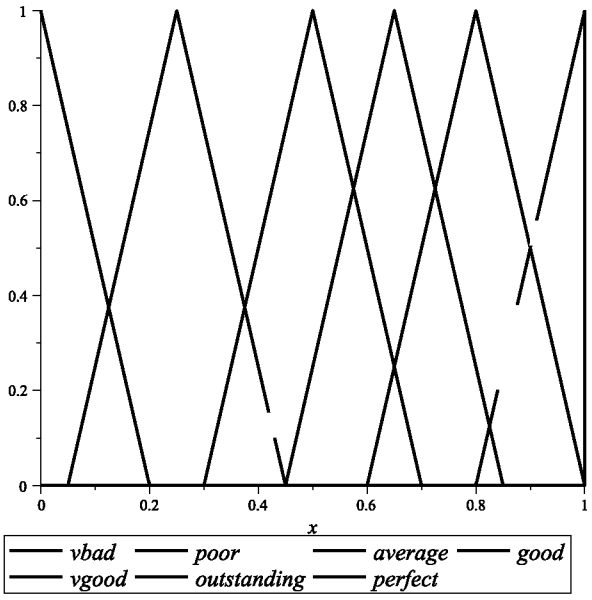

Fig. 6. Membership functions for 2-PM_{11}.

Of the five linguistic expressions that 2-CP_8 produces, one will be chosen for entry in the report to be delivered to the student. This expression will be chosen (see Section 5.1) using Takagi-Sugeno fuzzy reasoning (Takagi & Sugeno, 1985). This same method of selection will be used for 2-CP_9 and 2-CP_{11}.

4.2.2. 2-CP_9 adjacent management error

2-CP_9 describes adjacent management errors and is explained by 1-CP_4 and 1-CP_5 (perceptions describing distance and predecessor update errors E_{2-3}) and by 1-CP_6 (perception describing adjacent check and selection error E_{2-12}).

The 2-PM_9 of *adjacent management error* is a tuple (U,y,g,T), where:

U is the set of input *CP*s, 1-CP_4, 1-CP_5 and 1-CP_6, $U = \left\{\left(A_{e_{2\ 3}}, W^4_{e_{2\ 3}}\right), \left(A_{e_{2\ 3E}}, W^5_{e_{2\ 3E}}\right), \left(A_{e_{2\ 12}}, W^6_{e_{2\ 12}}\right)\right\}$.

$y = \left(A_{e_{am}}, W^9_{e_{am}}\right) = \{(a^9_1, w^9_1), (a^9_2, w^9_2), (a^9_3, w^9_3), (a^9_4, w^9_4), (a^9_5, w^9_5)\}$, where $A_{e_{am}}$ are linguistic expressions representing the *level of adjacent management error* (e_{am}) and are built with five linguistic labels: {very low, low, medium, high, very high}.

g is a five-component aggregation function whose image is the vector of weights $W^9_{e_{am}} = g\left(W^4_{e_{2\ 3}}, W^5_{e_{2\ 3E}}, W^6_{e_{2\ 12}}\right) = g((w^4_1, w^4_2, w^4_3, w^4_4), (w^5_1, w^5_2), (w^6_1, w^6_2)) = (w^9_1, w^9_2, w^9_3, w^9_4, w^9_5)$.

T the adjacent management error is {very low | low | medium | high | very high}.

Fig. 5 shows the membership functions for labels {very low, low, medium, high, very high}. The rules are designed as described in Section 4.3.1, although some corrections are necessary due to the characteristics of 1-CP_6. The following shows the rules that affect the result only.

1. IF e_{2-12} is unreasonable AND e_{2-3} is very large AND e_{2-3E} is not null THEN e_{am} is very high.
2. IF e_{2-12} is unreasonable AND e_{2-3} is large AND e_{2-3E} is not null THEN e_{am} is very high.
3. IF e_{2-12} is unreasonable AND e_{2-3} is small AND e_{2-3E} is null THEN e_{am} is medium.
4. IF e_{2-12} is unreasonable AND e_{2-3} is small AND e_{2-3E} is not null THEN e_{am} is high.
5. IF e_{2-12} is unreasonable AND e_{2-3} is very small AND e_{2-3E} is null THEN e_{am} is medium.
6. IF e_{2-12} is unreasonable AND e_{2-3} is very small AND e_{2-3E} is not null THEN e_{am} is high.
7. IF e_{2-12} is reasonable AND e_{2-3} is very large AND e_{2-3E} is not null THEN e_{am} is very high.
8. IF e_{2-12} is reasonable AND e_{2-3} is large AND e_{2-3E} is not null THEN e_{am} is high.
9. IF e_{2-12} is reasonable AND e_{2-3} is small AND e_{2-3E} is null THEN e_{am} is very low.
10. IF e_{2-12} is reasonable AND e_{2-3} is small AND e_{2-3E} is not null THEN e_{am} is low.
11. IF e_{2-12} is reasonable AND e_{2-3} is very small AND e_{2-3E} is null THEN e_{am} is very low.
12. IF e_{2-12} is reasonable AND e_{2-3} is very small AND e_{2-3E} is not null THEN e_{am} is very low.

4.2.3. 2-CP_{10} non-essential errors

2-CP_{10} identifies the simulations with no essential and very few or no other errors and is explained by 1-CP_2, 1-CP_3, 1-CP_5 and 1-CP_6. Its goal is to award 10 or the perfect grade in the final evaluation.

The 2-PM_{10} of *no errors in essential steps* is a tuple (U,y,g,T), where:

U are 1-CP_2, 1-CP_3, 1-CP_5 and 1-CP_6, $U = \left\{\left(A_{e_{1\ 1E}}, W^2_{e_{1\ 1E}}\right), \left(A_{e_{1\ 23}}, W^3_{e_{1\ 23}}\right), \left(A_{e_{2\ 3E}}, W^5_{e_{2\ 3E}}\right), \left(A_{e_{2\ 12}}, W^6_{e_{2\ 12}}\right)\right\}$.

$y = \left(A_{e_{ne}}, W^{10}_{e_{ne}}\right) = \{(a^{10}_1, w^{10}_1), (a^{10}_2, w^{10}_2)\}$, where $A_{e_{ne}}$ are the linguistic expressions that represent *the existence of errors in essential steps* (e_{ne}) and are built with two linguistic labels: {null, not null}.

g is a five-component aggregation function whose image is the vector of weights $W^{10}_{e_{ne}} = g\left(W^2_{e_{1\ 1E}}, W^3_{e_{1\ 23}}, W^5_{e_{2\ 3E}}, W^6_{e_{2\ 12}}\right) = (w^{10}_1, w^{10}_2)$ where $W^{10}_{e_{ne}} = \begin{cases} (1,0) & \text{if } w^2_1 = w^3_1 = w^5_1 = w^6_1 = 1 \\ (0,1) & \text{otherwise} \end{cases}$.

T the error in essential steps is {null | not null}.

4.2.4. 2-CP_{11} simulation correctness level

Then we aggregate 2-CP_8 (activation and flow control), 2-CP_9 (adjacent management), 2-CP_{10} (non-essential errors) and 1-CP_7 (time) to build a perception that describes the level of Dijkstra's algorithm simulation correctness.

The construction of the 2-*PM* is a two-phase process. First, we obtain the validity degrees for the linguistic expressions that will produce part of the natural language report. Second, we get a numerical rating (*Grade*) of the simulation under assessment. This numerical variable will be used as an input datum for the 2-PM_{12} of *learning outcomes*.

The 2-PM_{11} of level of simulation correctness is a tuple $(U,(y,y_G),(g,g_G),T)$, where:

U are 2-CP_8, 2-CP_9, 2-CP_{10} and 1-CP_7, $U = \left\{\left(A_{e_{af}}, W^8_{e_{af}}\right), \left(A_{e_{am}}, W^9_{e_{am}}\right), \left(A_{e_{ne}}, W^{10}_{e_{ne}}\right), \left(A_{time}, W^7_{time}\right)\right\}$.

$y = \left(A_{as}, W^{11}_{as}\right) = \{(a^{11}_1, w^{11}_1), (a^{11}_2, w^{11}_2), (a^{11}_3, w^{11}_3), (a^{11}_4, w^{11}_4), (a^{11}_5, w^{11}_5), (a^{11}_6, w^{11}_6), (a^{11}_7, w^{11}_7)\}$, where A_{as} are the linguistic expressions that represent *the algorithm simulation correctness level (assessment) achieved by the student* and are built from {very bad, poor, average, good, very good, outstanding, perfect}.

y_G = *Grade* is a numerical variable that expresses the numerical grade of the algorithm simulation.

g is a seven-component aggregation function whose image is the vector of weights $W_{as}^{11} = g\left(W_{e_{af}}^{8}, W_{e_{am}}^{9}, W_{e_{ne}}^{10}, W_{time}^{7}\right) = g\left((w_1^8, w_2^8, w_3^8, w_4^8, w_5^8), (w_1^9, w_2^9, w_3^9, w_4^9, w_5^9), (w_1^{10}, w_2^{10}), (w_1^7, w_2^7)\right) = \left(w_1^{11}, w_2^{11}, w_3^{11}, w_4^{11}, w_5^{11}, w_6^{11}, w_7^{11}\right)$.

g_G is the numerical grade of the simulation. It is obtained using the weighted mean of the weights:

$$Grade = g_G(W_{as}^{11}) = \frac{\sum_{j=1}^{7} w_j^{11} * Ac_j}{\sum_{j=1}^{7} w_j^{11}}, \tag{8}$$

where Ac_j are the centroids of the membership functions of the linguistic labels {very bad, poor, average, good, very good, outstanding, perfect} (see Fig. 6).

T the correctness level achieved by the student is {very bad | poor | average | good | very good | outstanding | perfect}, and the grade obtained in this simulation is {Grade∗10}.

To aggregate the information from the subordinate perceptions, we take several things into account:

- Time is not an important factor, unless it is very close to the permitted limit (see Fig. 4(d)), and has no influence whatsoever in cases where there are very few errors.
- The perfect label will only be associated with exercises in which there is no essential and very few non-essential errors, i.e., $W_{e_{ne}}^{10} = (1, 0)$. In this case, we do not take into account any of the other subordinate perceptions.
- More adjacent management than activation and flow errors are permitted, as the selection of the next active vertex is the concept that is considered critical in Dijkstra's algorithm.
- Errors are to some extent accruable, also taking into account the criteria expressed in the previous point.

The above conditions are summarized in a set of 100 rules, of which only some examples are given:

1. IF e_{ne} is not null AND *time* is adequate AND e_{af} is very low AND e_{am} is very low THEN *assessment* is outstanding.
2. IF e_{ne} is not null AND *time* is inadequate AND e_{af} is very low AND e_{am} is very low THEN *assessment* is outstanding.
4. IF e_{ne} is not null AND *time* is inadequate AND e_{af} is very low AND e_{am} is low THEN *assessment* is very good.
5. IF e_{ne} is not null AND *time* is adequate AND e_{af} is very low AND e_{am} is medium THEN *assessment* is good.
6. IF e_{ne} is not null AND *time* is inadequate AND e_{af} is very low AND e_{am} is medium THEN *assessment* is average.
8. IF e_{ne} is not null AND *time* is inadequate AND e_{af} is very low AND e_{am} is high THEN *assessment* is poor.
21. IF e_{ne} is not null AND *time* is adequate AND e_{af} is medium AND e_{am} is very low THEN *assessment* is average.
51. – 100. IF e_{ne} is null THEN *assessment* is perfect.

Additionally, 2-CP_{11} can be considered as a *top-order-CP* when the goal is to assess a single algorithm simulation. In this case, the perception will respond to the question:

What correctness level does the student achieve in this algorithm simulation?

4.3. Top-order-CP learning outcomes

The *top-order-CP*, 2-CP_{12}, will describe the results achieved in a set of Dijkstra's algorithm simulations. It is explained by a set of 2-CP_{11}s obtained from several simulations performed by a student.

We will use a vector of importance $(b_1, \ldots, b_n)$ to build the aggregation functions of this 2-PM_{12}. To select this vector, we assume that students gradually make fewer mistakes as they learn more about the algorithm until they are capable of almost always solving the problem without making any mistakes. For this reason, more importance will be attached to later simulations.

As for 2-CP_{11}, the construction of this perception is a two-phase process. First, we obtain weights for the linguistic expressions that will output the natural language report on the set of simulations under assessment. Second, we get a numerical *fGrade* that corresponds to the grade for that set of simulations.

The 2-PM_{12} of *learning outcomes* is a tuple $(U, (y, y_G), (g, g_G), T)$, where:

U are the 2-CP_{11}s provided by the simulations under assessment.

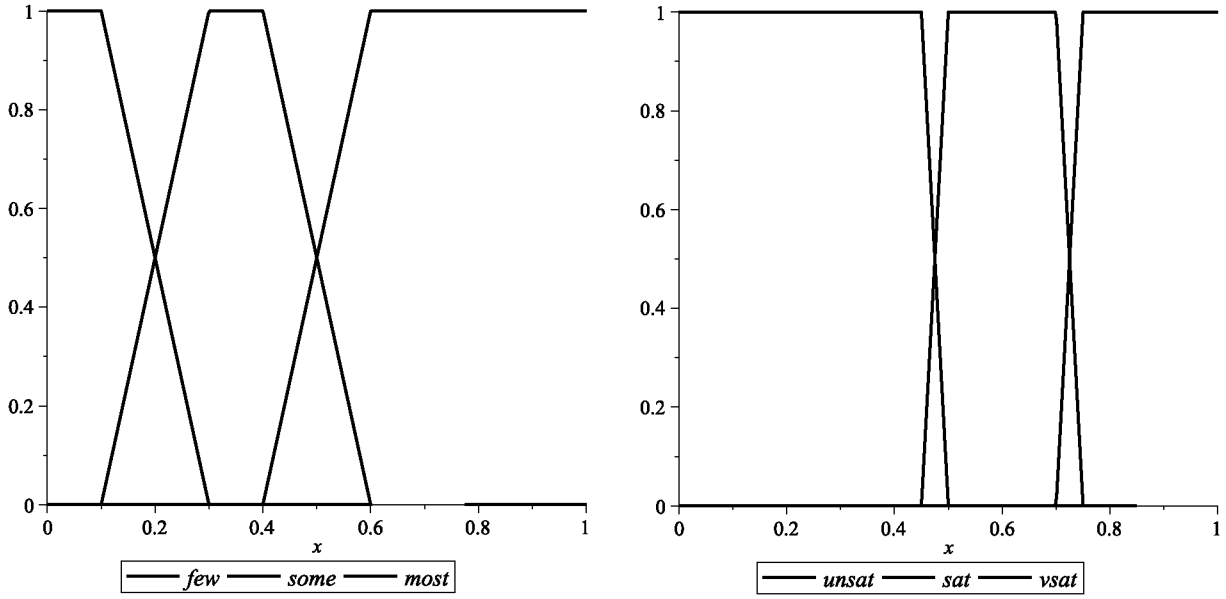

Fig. 7. Membership functions for quantifiers and attributes.

$y = (A_{QS}, T_{QS})$, where A_{QS} are the nine linguistic expressions that represent the *learning outcomes* obtained in a set of n simulations and are built using the quantifiers Q, {few, some, most} and the attributes S {unsatisfactory, satisfactory, very satisfactory}.

$y_G = fGrade$ is a numerical variable that expresses the numerical grade of a set of n simulations.

g is an aggregation function $T_{QS} = g(Grade_1, \ldots, Grade_n)$. The validity degrees T_{QS} are calculated according to Eq. (11), as explained in Section 5.2.

g_G is a weighted average of the inputs with weights given by the vector of importance $(b_1, \ldots, b_n)$, whose image is the numerical grade of the set of simulations

$$fGrade = g_G(Grade_1, \ldots, Grade_n) = \frac{\sum_{j=1}^{n} b_j * Grade_j}{\sum_{j=1}^{n} b_j}. \tag{9}$$

*T the correctness level achieved by the student is {unsatisfactory | satisfactory | very satisfactory} in {few | some | most} of the important simulations, and the grade obtained is {fGrade*10}.*

Fig. 7 illustrates the membership functions for the quantifiers {*few, some, most*} and the membership functions for the attributes {*unsatisfactory, satisfactory, very satisfactory*}.

This *top-order-CP* represents the model answer to the following general question on a set of simulations completed by the student:

"What correctness level does the student achieve in the set of simulations?"

Note that, using the vector of importance $(1,1,\ldots,1)$, this same model can be applied to study the level of correctness achieved by a group of students.

5. Preparing assessment reports

When processing each interaction log, the system generates a series of weight vectors W_x^i with $i = 1,\ldots,11$ corresponding to each of the *CP*s. These weight vectors are used to generate the different levels of the assessment report. Simulation assessment also generates a table that contains the validity degrees of the nine possible quantifier and attribute combinations. In this section, we describe the criteria used to select the linguistic expression that will be entered in the report.

5.1. Assessment report for one simulation

The perception 2-CP_{11} contains the numerical grade that the student receives for an algorithm simulation. As mentioned earlier, this grade is obtained by applying Eq. (8), and the numerical grade output will be *Grade*∗10. Of the seven linguistic expressions that 2-CP_{11} produces (Section 4.2.4), we will choose the linguistic label that has the maximum membership degree of *Grade* (in the event of a draw the most favourable label is chosen). This expression will be part of the natural language report that will be delivered to the student. For each of the other 2-*CP*s, the linguistic expression is selected similarly, that is, we select the one whose label has a maximum membership degree of:

$$\frac{\sum_{j=1}^{n_k} w_j^k * Ac_j^k}{\sum_{j=1}^{n_k} w_j^k}, \tag{10}$$

where w_j^k and Ac_j^k are, respectively, the degrees of validity and centroids for 2-CP_k. For 1-*CP*s we select the linguistic expression whose validity degree is maximum, choosing the best in the case of a draw.

This way, we get a detailed report that indicates the parts of the simulation that the student finds harder, as well as where his or her strengths lie.

5.2. Report on a set of simulations

The *top-order-CP* (2-CP_{12}) template can output nine different expressions depending on the selections of the three quantifier labels and the three attribute labels. The objective is to assess the validity and relevance of each of these reports and choose the best. The validity degree of each of these summaries is calculated using the following equation (Yager, 1982; Yager, 1991; Yager, 1995; Zadeh, 1983):

$$T_{QS} = \mu_Q\left(\frac{\sum_{i=1}^{n} b_i * \mu_S(Grade_i)}{\sum_{i=1}^{n} b_i}\right), \tag{11}$$

which expresses the validity degree of summary *"Q B y's are S"*, where the validity degree of "y_i *is S*" is $\mu_S(y_i)$. In Eq. (11), Q is one of the quantifiers {*few | some | most*}, B is the importance vector and S is one of the linguistic labels {*unsatisfactory | satisfactory | very satisfactory*}.

One option for selecting the best summary could be to choose the one that has the best validity degree T_{QS}. But, as Yager suggests in (Yager (1991), Yager (1995)), this is not always the one that contributes the most relevant information. The selection of the most relevant linguistic expression and, especially, the selection of the combination of the most relevant expressions is an open question. In this paper we will use the Yager's measure of informativeness (Yager, 1982). For the linguistic summary *"Q y's are S"*, depending on the type of quantifier Q, the measure of informativeness is formulated as follows:

$$I_{QS} = \max(T_{QS} * Sp(Q) * Sp(\bar{S}), (1 - T_{QS}) * Sp(\bar{Q}) * Sp(S)), \tag{12a}$$

when Q is a monotonically non-increasing quantifier;

$$I_{QS} = \max(a * T_{QS} * Sp(Q) * Sp(S), (1 - a) * T_{QS} * Sp(Q) * Sp(\bar{S})), \tag{12b}$$

when Q is an unimodal quantifier and a is the center of maximum of Q;

$$I_{QS} = \max(T_{QS} * Sp(Q) * Sp(S), (1 - T_{QS}) * Sp(\bar{Q}) * Sp(\bar{S})), \tag{12c}$$

when Q is a monotonically increasing quantifier.

In these equations Sp indicates the specificity of the associated fuzzy set (Yager, 1998).

In this paper we propose a method for calculating the informativeness of the summary *"Q B y's are S"*, as an extension of the measure proposed by Yager for application to our particular problem. To add the importance B within the summary we considered that each of the weights of B represents a number of repetitions b_i of the ith simulation. Accordingly, the set of simulations on which the label S is applied is substituted by a new set where the ith element is repeated b_i times. This changes the specificity of the above label on the new set. Note that this does not constitute a change in the validity degree T_{QS} of the summary *"Q B y's are S"* calculated in Eq. (9), as the validity degree of this summary is equal to the validity degree of "$Q\ \bar{y}$*'s are S*", where $\bar{y}$ are the elements of the new set. The method that we propose for calculating the informativeness of summary *"Q B y's are S"* changes, not the equations proposed by Yager, but the universe of discourse to which these equations are applied. The labels $Q\ S$ that have the greatest informativeness I_{QS} will provide the summary that will be chosen for the report.

6. Examples and results analysis

6.1. Report on a simulation

The student simulates Dijkstra's algorithm on a graph, where eleven vertices have to be fixed to find the shortest path between the two specified vertices. Data extracted from the XML interaction log (see Section 4.1) are $E_{1-1} = 1/11$, $E_{1-23} = 1/23$, $E_{2-3} = 3/10$, $E_{2-12} = 0/10$ and $Time = 0.8$. GLMP is used to prepare an assessment report based on these data:

The vectors of validity degrees of the 1-*CP*s and linguistic expressions selected for the natural language report follow:

$W^1_{e_{1\ 1}} = (1,1,0,0)$ and the selected expression is a^1_1: *the active node selection error is very small.*
$W^2_{e_{1\ 1E}} = (0,1)$, *the active node selection error is not null.*
$W^3_{e_{1\ 23}} = (1,0)$, *the flow control error is acceptable.*
$W^4_{e_{2\ 3}} = (0.33,1,0,0)$, *the distance and predecessor update error is small.*
$W^5_{e_{2\ 3E}} = (0,1)$, *the distance and predecessor update error is not null.*
$W^6_{e_{2\ 12}} = (1,0)$, *the adjacent check and selection error is reasonable.*
$W^7_{time} = (1,0)$, *the time taken to perform the algorithm simulation is adequate.*

The vectors of validity degrees of the 2-*CP*s, the weighted means, the linguistic label membership degrees over the weighted mean and the linguistic expressions selected for the natural language report follow:

$W^8_{e_{af}} = (1,1,0,0,0)$, the weighted mean is $\frac{\sum_{j=1}^{5} w_j^8 * Ac_j^8}{\sum_{j=1}^{5} w_j^8} = 0.1$. The linguistic label membership degrees over 0.1 are (0.33,0.33,0,0,0), and the linguistic expression selected for the report is a^8_1: *The activation and flow error is very low.*

$W^9_{e_{am}} = (0.33,1,0,0,0)$, the weighted mean is $\frac{\sum_{j=1}^{5} w_j^9 * Ac_j^9}{\sum_{j=1}^{5} w_j^9} = 0.15$. The linguistic label membership degrees over 0.15 are (0,0.67,0,0,0), and the linguistic expression selected for the report is *The adjacent management error is low.*
$W^{10}_{e_{ne}} = (0,1)$, *The error in essential steps is not null.*

And, finally, the *top-order-CP* produces the following vector of validity degrees, linguistic expression and numerical grade:

$W^{11}_{as} = (0,0,1,0,0.33,1,0)$, $Grade = g_G(W^{11}_{as}) \frac{\sum_{j=1}^{7} w_j^{11} * Ac_j}{\sum_{j=1}^{7} w_j^{11}} = 0.731$.
The linguistic label membership degrees over 0.731 are (0,0,0,0.5929,0.6571,0,0) and the linguistic expression selected for the report is *The correctness level achieved by the student is very good, and the grade obtained in this simulation is* 7.31

Table 3
Data of the learning process interaction log with its respective numerical assessments.

E_{1-1}	E_{1-23}	E_{2-3}	E_{2-12}	Time	*Grade*∗10
3/7	0/15	3/11	3/28	0.85	4.09
1/5	1/10	2/7	1/18	0.73	6.99
3/9	0/19	6/16	0/40	0.31	4.94
1/11	1/23	3/10	0/10	0.8	7.31
0/9	1/19	1/6	0/8	0.49	9.40

Although all the perceptions can generate their respective part of the report, we will select the linguistic summaries for 2-CP_8, 2-CP_9 and 2-CP_{11}, unless a great deal of detail is required. This way, the output of the GLMP for the simulation whose data are $E_{1-1} = 1/11$, $E_{1-23} = 1/23$, $E_{2-3} = 3/10$, $E_{2-12} = 0/10$ and $Time = 0.8$ is:

The activation and flow error is very low and the adjacent management error is low, the correctness level achieved by the student is very good, and the grade obtained in this simulation is 7.31.

6.2. Report on the learning process

Let us now look at how the Dijkstra's algorithm learning process assessment report is obtained when the student completes more than one simulation. In this case, the importance of a simulation grows the more previous simulations the student has completed. To do this, we define the vector of importances (see Section 5.2) as $B = (1^2, 2^2, 3^2, \ldots)$. In this example, the student has completed five simulations, and Table 3 shows the interaction log data.

Table 4 shows the validity degrees and the informativeness of the linguistic summaries (see Section 5.2).

We select the two summaries with the greatest informativeness to generate the following report:

In this process of 5 simulations (where importance = i^2), the correctness level achieved by the student is very satisfactory in most of the important simulations (truthfulness = 1), satisfactory in some of the important simulations (truthfulness = 1), and the grade obtained is 7.79.

6.3. Report on a set of simulations

When the aim is to assess the level of learning demonstrated by a group of students in a Dijkstra's algorithm simulation examination, we can apply a variant of *top-order-CP*, 2-CP_{12}. We define the vector of importance (see Section 5.2) as $B = (1,1,1,\ldots)$ in order to assess the exercises of all the students equally. Table 5 shows the data obtained by a group of eight students.

Table 6 shows the validity degrees and informativeness of the linguistic summaries (see Section 5.2).

By slightly amending the template for adaptation to the current case and choosing the two summaries with the greatest informativeness, we generate the following report:

In this group, the correctness level achieved by the students is very satisfactory in most cases (truthfulness = 0.823), satisfactory in some cases (truthfulness = 1), and the average for this group is 7.23.

6.4. Analysis of assessor results

In the example shown in Section 6.2 (see Table 3), we find that the arithmetic mean of the grades obtained is 6.55; however, as indicated in the report, the grade provided by the GLMP is 7.79. This difference is due to the importance attached to the later simulations. Note also that the result of the third simulation is lower, which has a negative influence on the final grade. If we were to switch simulations 2 and 3, the final grade would be 8, that is, the model rewards the positive progress of learning.

Table 4
Validity degrees and informativeness of the linguistic summaries for the learning process described in Table 3.

T_{QS}	Unsat	Sat	Very sat	I_{QS}	Unsat	Sat	Very sat
Few	1	0	0	Few	0.0303	0.1374	0.0739
Some	0	1	0	Some	0	**0.1683**	0
Most	0	0	1	Most	0.0189	0.1657	**0.1847**

Table 5
Interaction log data for a group of students with their respective numerical assessments.

E_{1-1}	E_{1-23}	E_{2-3}	E_{2-12}	Time	Grade
3/11	1/23	3/10	1/10	0. 51	6.63
1/11	1/23	3/10	0/10	0.75	7.31
1/11	1/23	4/10	2/10	0.65	5.88
0/11	1/23	2/10	0/10	0.8	9.40
1/11	1/23	0/10	0/10	0.69	8.70
0/11	1/23	0/10	0/10	0. 95	10
2/11	1/23	2/10	4/10	0.77	2.45
1/11	2/23	1/10	0/10	0.86	7.44

Table 6
Validity degrees and informativeness of the linguistic summaries for the set of simulations described in Table 5.

T_{Q5}	Sat	Very sat	I_{Q5}	Unsat	Sat	Very sat	
Few	0.8750	0	0	Few	0.1	0.1576	0.0995
Some	0.1250	1	0.1769	Some	0.0306	**0.1931**	0.0519
Most	0	0	0.8231	Most	0.0714	0.1774	**0.2048**

The resulting grade in simulation 7 shown in Table 5 is 2.45, because it exceeded the maximum permitted number of errors in *adjacent check and selection* (1-CP_6), (see Fig. 4(c)), which means that the *adjacent management* (2-CP_9) error is *high*. As there are also *active node selection* (1-CP_1) errors, which is the most penalized error, the output of the GLMP returns a very low numerical grade.

Grade 10 is reserved for exercises where there are only a very few non-critical errors. In this case, time is not taken into account (see simulation 6 in Table 5). In other cases, time only has an influence when it is greater than 85% of the maximum permitted limit (see Fig. 4(d)). For example, if the time taken in simulation 3 in Table 5 had been 99%, the result would have dropped from 5.88 to 5.

As mentioned beforehand, the most penalized error is E_{1-1}, which corresponds to the selection of the nearest unfixed vertex as new active vertex. This error is important because of Dijkstra's algorithm strategy, where each subpath of the minimum path is also a minimum path. In other words, if V is a vertex situated in the minimum path from O to G, then the subpath from O to V is also a minimum path. Therefore, the vertex that is fixed in each iteration is the one for which the minimum path has just been found. The grading of simulations 4 and 5 (see Table 5), for example, illustrates the implementation of this criterion. Simulation 4 contains two errors compared with 10 correct actions in E_{2-3} and 0 errors in E_{1-1}, whereas simulation 5 has one error compared with 11 correct responses in E_{1-1} and 0 errors in E_{2-3}. However, simulation 4 is graded higher than simulation 5.

The examples described in this section clearly illustrate that automatic assessment using the proposed method is able to implement complex instructor-specific criteria (in this case specified by the designer) using exclusively objective and easily calculated data.

7. Conclusions

In this paper we have designed a granular linguistic model of the computer-assisted learning assessment process. This model is able to automate complex assessment criteria based on objective data, relieving human assessors of a laborious and repetitive task. The model uses fuzzy inference systems mainly based on linguistic rules. This is the feature that makes it most suitable for implementing instructor knowledge- and experience-based criteria, where learning goals are usually expressed in natural language. Additionally, the model is geared up to output a detailed natural language report on the whole learning process for the purpose of formative assessment.

We designed an expert system that uses this model to assess the Dijkstra's algorithm learning process through visual simulation in the GRAPHs e-learning environment. This application was implemented and tested on real data as shown in Section 6. This application shows the capabilities of the model described in Section 3. As demonstrated, the model implements complex criteria that aggregate errors, which it discriminates by type and number.

Additionally, we propose a method for calculating how informative a summary containing importance-discriminated data is, and we use this informativeness to choose the summary to generate the learning progress report.

This paper outlines the first contribution of its kind in the field of automatic e-learning assessment. The ideas explained can be developed using other aggregation functions and other types of report, meaning that its scope of application is easily extendible.

References

Akingbade, A., Finley, T., Jackson, D., Patel, P., & Rodger, S. H. (2003). JAWAA: Easy web-based animation from CS 0 to advanced CS courses. *SIGCSE Bull, 35*(1), 162–166.

Alvarez-Alvarez, A., Sanchez-Valdes, D., & Trivino, G. (2011). Automatic linguistic description about relevant features of the Mars' surface. In *Proc. 11th Int. Conf. Intell. Syst. Des. Appl. (ISDA)* (pp. 154–159). Córdoba, Spain.

Baecker, R., Sherman, D., & Group, U. o. (1981). Sorting out sorting. *Dynamic Graphics Project*.

Bai, S. M., & Chen, S. M. (2008a). Evaluating students' learning achievement using fuzzy membership functions and fuzzy rules. *Expert Systems with Applications, 34*(1), 399–410.

Bai, S. M., & Chen, S. M. (2008b). Automatically constructing grade membership functions of fuzzy rules for students' evaluation. *Expert Systems with Applications, 35*(3), 1408–1414.

Biswas, R. (1995). An application of fuzzy sets in students' evaluation. *Fuzzy Set Systems, 74*(2), 187–194.

Brown, M. H., & Sedgewick, R. (1984). A system for algorithm animation. *SIGGRAPH Computer Graphics, 18*(3), 177–186.

Chen, S.-M., & Lee, C.-H. (1999). New methods for students' evaluation using fuzzy sets. *Fuzzy Set Systems, 104*(2), 209–218.

Chen, S.-M., & Wang, H.-Y. (2009). Evaluating students' answerscripts based on interval-valued fuzzy grade sheets. *Expert Systems with Applications, 36*(6), 9839–9846.

Crescenzi, P., & Nocentini, C. (2007). Fully integrating algorithm visualization into a cs2 course: A two-year experience. *SIGCSE Bull, 39*(3), 296–300.

Eciolaza, L., Trivino, G., Delgado, B., Rojas, J., & Sevillano, M. (2011). Fuzzy linguistic reporting in driving simulators. In *Proc. 2011 IEEE Symp. Comput. Intell. Veh. Transp. Syst. (CIVTS)* (pp. 30–37). Paris, France, April 11–15.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing, 13*(3), 259–290.

Kacprzyk, J., & Yager, R. R. (2001). Linguistic summaries of data using fuzzy logic. *International Journal of General Systems, 30*(2), 133–154.

Kacprzyk, J., & Zadrozny, S. (2005). Linguistic database summaries and their protoforms: Towards natural language based knowledge discovery tools. *Information Sciences, 173*(4), 281–304.

Law, C. K. (1996). Using fuzzy numbers in education grading system. *Fuzzy Set Systems, 83*(3), 311–323.

Li, T.-K., & Chen, C.-M. (2009). A new method for students' learning achievement evaluation by automatically generating the weights of attributes with fuzzy reasoning capability. In *Proc. 8th Int. Conf. Mach. Learn. Cybern.* (pp. 2834–2839). Baoding, 12–15 July 2009.

Ma, J., & Zhou, D. (2000). Fuzzy set approach to the assessment of student-centered learning. *IEEE Transactions on Education, 43*(2), 237–241.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., & Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Information and Education, 3*(2), 267–288.

Martínez-Cruz, C., Van der Heide, A., Sánchez, D., & Trivino, G. (2012). An Approximation to the Computational Theory of Perceptions using Ontologies. *Expert Systems with Applications, 39*(10), 9494–9503.

Mendez-Nunez, S., & Trivino, G. (2010). Combining semantic web technologies and computational theory of perceptions for text generation in financial analysis. In *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ)* (pp. 1–8). Barcelona, 18–23 July.

Naps, T. L. (2005). JHAVE: Supporting algorithm visualization. *Computer Graphics and Applications, 25*(5), 49–55.

Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., et al. (2003). Evaluating the educational impact of visualization. *SIGCSE Bull, 35*(4), 124–136.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., et al. (2002). Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull, 35*(2), 131–152.

Rößling, G., McNally, M., Crescenzi, P., Radenski, A., Ihantola, P., & Sánchez-Torrubia, M. G. (2010). Adapting moodle to better support CS education. In A. Clear, & L. R. Dag (Eds.), *Proc. 2010 ITiCSE, Work. group rep. (ITiCSE-WGR '10)* (pp. 15–27). Ankara (Turkey) 26–30 April: ACM, New York, NY, USA.

Rößling, G., Mihaylov, M., & Saltmarsh, J. (2011). AnimalSense: combining automated exercise evaluations with algorithm animations. In *Proc. 16th Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE '11)* (pp. 298–302). Darmstadt, June 27–29: ACM, New York, NY, USA.

Rößling, G., & Freisleben, B. (2002). ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages and Computing, 13*(3), 341–354.

Saleh, I., & Kim, S.-i. (2009). A fuzzy system for evaluating students' learning achievement. *Expert Systems with Applications, 36*(3), 6236–6243.

Sanchez-Torrubia, M. G., Torres-Blanc, C., & Cubillo, S. (2010). Design of a Fuzzy Inference System for Automatic DFS & BFS Algorithm Learning Assessment. Comput. Intell. Found. Appl. In *Proc. 9th Int. FLINS Conf.*, (pp. 308–313). Emei, Chengdu, China, August, 2–4.

Sanchez-Torrubia, M. G., Torres-Blanc, C., & Escribano-Blanco, S. (2010b). GRAPHs: A learning environment for graph algorithm simulation primed for automatic fuzzy assessment. In *Proc. 10th Koli Calling Int. Conf. Comput. Educ. Res. (Koli Calling '10)* (pp. 62–67). Koli National Park, Finland, October 28–31.

Sanchez-Torrubia, M. G., & Torres-Blanc, C. (2010). A Mamdani-type fuzzy inference system to automatically assess Dijkstra's algorithm simulation. *International Journal of Information for Theory and Applications, 17*(1), 35–48.

Sanchez-Torrubia, M. G., Torres-Blanc, C., & Krishnankutty, S. (2008). Mamdani's fuzzy inference eMathTeacher: A tutorial for active learning. *WSEAS Transactions on Computer, 7*(5), 363–374.

Sanchez-Torrubia, M. G., Torres-Blanc, C., & Lopez-Martinez, M. A. (2009). PathFinder: A visualization eMathTeacher for actively learning Dijkstra's algorithm. *Electronic Notes in Theoretical Computer Science, 224*, 151–158.

Shaffer, C. A., Cooper, M. L., Alon, A. J., Akbar, M., Stewart, M., Ponce, S., et al. (2010). Algorithm visualization: The state of the field. *ACM Transactions on Computer Education, 10*(3), 9:1–9:22.

Shaffer, C. A., Heath, L. S., & Yang, J. (1996). Using the Swan data structure visualization system for computer science education (ACM, Ed.). *SIGCSE Bull., 28*(1), 140–144.

Stasko, J. (1998). Smooth continuous animation for portraying algorithms and processes. In J. D. J. Stasko (Ed.), *Software visualization* (pp. 103–118). Cambridge: MIT Press.

Takagi, T., & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man and Cybernetics, 15*(1), 116–132.

Tay, K. M., & Lim, C. P. (2011). A fuzzy inference system-based criterion-referenced assessment model. *Expert Systems with Applications, 38*(9), 11129–11136.

Trivino, G., Sanchez, A., Montemayor, A. S., Pantrigo, J. J., Cabido, R., & Pardo, E. G. (2010). Linguistic description of traffic in a roundabout. In *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ)* (pp. 1–8). Barcelona, Spain, July, 18–23.

Yager, R. R. (1982). A new approach to the summarization of data. *Information Sciences, 28*(1), 69–86.

Yager, R. R. (1991). On linguistic summaries of data. In G. Piatetsky-Shapiro (Ed.), *Knowledge discovery in databases*. Cambridge, MA: MIT Press (pp. 347–363).

Yager, R. R. (1993). Families of OWA operators. *Fuzzy Set Systems, 59*(2), 125–148.

Yager, R. R. (1995). Fuzzy summaries in database mining. In *Proc. 11th Conf. Artif. Intell. Appl.* (pp. 265–269). Los Angeles, California, February, 20–23: IEEE Computer Society Press.

Yager, R. R. (1996). Quantifier guided aggregation using OWA operators. *International Journal of Intelligent Systems, 11*(1), 49–73.

Yager, R. R. (1998). Measures of specificity. In O. Kaynak, L. A. Zadeh, B. Turksen, & I. J. Rudas (Eds.), *Computational Intelligence. Soft Computing and Fuzzy-Neuro Integration with Applications* (pp. 94–113). Berlin: Springer Verlag.

Zadeh, L. A. (1983). A computational approach to fuzzy quantifiers in natural languages. *Computers and Mathematics with Applications, 9*(1), 149–184.

Zadeh, L. A. (1999). From computing with numbers to computing with words. From manipulation of measurements to manipulation of perceptions. *IEEE Transactions on Circuits and Systems, 45*(1), 105–119.