NExt ApplicationS of Quantum Computing

D6.17: QRBS Evaluation Report

Document Properties

Contract Number	951821
Contractual Deadline	31-10-2024
Dissemination Level	Public
Nature	Report
Editors	Gonzalo Ferro, CESGA Andrés Gómez, CESGA
Authors	Gonzalo Ferro, CESGA Samuel Magaz-Romero, UDC Vicente Moret-Bonillo, UDC Andrés Gómez, CESGA
Reviewers	Mohamed Hibti, EDF Sebastiaan O. Brand, ULEI
Date	24-10-2024
Keywords	Software implementation, IDC, application, QRBS
Status	Submitted
Release	1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 951821

History of Changes

Release	Date	Author, Organisation	Description of Changes
0.1	18/07/2024	Gonzalo Ferro (CESGA)	First draft
0.2	17/09/2024	Andrés Gómez, Gonzalo Ferro (CESGA)	Second draft version. Conclusions added.
0.3	23/09/2024	Andrés Gómez (CESGA)	Reviewers added. Minor changes before submission for internal review.
0.4	15/10/2024	Gonzalo Ferro (CESGA)	Added the reviewer comments and corrections. A new appendix for membership functions was included.
1.0	24/10/2024	Andrés Gómez (CESGA)	Final version for submission.

Table of Contents

1.Executive Summary

This report is the last deliverable of use case 6 – Quantum Rule-Based Systems (QRBS) for breast cancer detection - of the NEASQC project. It presents the results of the software evaluation and benchmarking against classical solutions.

The evaluation was done in five steps:

1. Check the programmability of the developed library, i.e., the possibility of developing applications by other programmers.
2. Integration of the library into Eviden Qaptiva environment.
3. Test the proposed algorithms using a basic synthetic case.
4. Benchmark the solution against classical standard for Invasive Ductal Carcinoma (IDC) diagnosis.
5. Evaluate the solution on noisy emulated Quantum Processing Units (QPU).

As a consequence of this work, the QRBS software release was extended with four tutorials using notebooks, some issues that limited the usability were identified and corrected, and additional support for emulators with larger capacity and other capabilities (such as the possibility of emulating noise using Qaptiva software) was introduced.

The main conclusions extracted from the evaluation are:

- The results obtained for all the cases in a noiseless environment are those expected for the defined problems. These values present a high degree of similarity with those of classical models, as shown by the comparison of QRBS with fuzzy logic. In fact, the problem is not inferential; the problem is derived from the classification criteria.
- When comparing classical and quantum methods, it can be observed that the corresponding results are consistent. However, the QRBS approach improves the outputs adding intrinsic uncertainties to the inferences, which are not available in the classical approach. The results obtained show an almost identical behaviour among them, thus reinforcing the hypothesis (in both, qualitatively and quantitatively perspectives) that Quantum Computing is a more general paradigm, with the capacity to encompass the different classical models of symbolic reasoning.
- Noise affects the results catastrophically when is higher than a threshold, producing for the current QPUs useless results. However, for the tested models, a moderate improvement in the quality of the QPUs could produce acceptable results. Of course, the executed models do not justify yet the usage of real QPUs, because they are still small, allowing the classical ideal emulation. Experiments with larger models are needed to assess clearly the effect of the noise.

2.Context

NEASQC WP6 is devoted to research on quantum algorithms for artificial intelligence (AI) and graphs. Within this Workpackage, the Use Case 6 (UC6) is devoted to the development of a Quantum Rule-Based System (QRBS) for Invasive Ductal Carcinoma (IDC) diagnosis. This last deliverable of this UC describes the evaluation of the proposed QRBS software (Magaz-Romero, 2024), making a benchmarking against the standard classical solution for IDC diagnosis.

To have a better understanding of this evaluation, the reader can rely on the contents of previous deliverables of this UC (D6.2 (Moret-Bonillo, Mosqueira-Rey, Magaz-Romero, & Gómez-Tato, 2021), D6.5 (Moret-Bonillo, Mosqueira-Rey, & Magaz-Romero, 2021), D6.9 (Moret-Bonillo, Gomez Tato, et al., 2022), D6.11 (Moret-Bonillo, Gomez Tato, Magaz-Romero, et al., 2023a) and D6.14 (Moret-Bonillo, Magaz-Romero, Mosqueira-Rey, & Alvarez-Estevez, 2023)). They described the requirements and developments of the set of quantum algorithms evaluated in this document.

This evaluation was performed by a CESGA team that was not involved directly in the development of the solution. This guarantees a fair test of the proposed software.

The document is divided into only two parts. In the first section, the process for evaluating the QRBS proposed solution is described. It is composed of 5 steps, devoting a subsection to each of them. The last section discusses the results and extract the main conclusions.

3.QRBS evaluation

The main objectives of this evaluation are:

- Evaluate the usability of the library for developing different QRBS systems.
- Comparing the obtained results with classical software, especially for IDC diagnosis.
- Evaluation of the performance of the software in emulated quantum devices.
- Provide feedback about possible errors or improvements of the library to the UDC.

The evaluation consisted of several steps:

1. Check the programmability of the developed library, i.e., the possibility of developing applications by other programmers.
2. Integration of the library into Eviden Qaptiva environment.
3. Test the proposed algorithms using a basic synthetic case.
4. Benchmark the solution against the classical standard for IDC diagnosis.
5. Evaluate the solution on noisy emulated Quantum Processing Units.

The next subsections describe the performed work and the results of each step.

3.1. Testing programmability

This first step permitted the CESGA team to understand deeply the QRBS software and familiarize themselves with its capabilities. At the same time, this step allowed CESGA to test the usability of the library and its documentation.

For this purpose, two different tutorial notebooks were developed. These notebooks, added now to the QRBS GitHub[1] repository under the **/misc/notebooks** folder, were:

- An introductory tutorial notebook: *01_QRBS_for_dummies.ipynb*. It is a brief tutorial about the Rule-Based System (**RBS**) and how to implement them in **QRBS**.
- A tutorial to QRBS software package: *02_QRBS_Uncertainty.ipynb*. This notebook presents the capability of the software for modelling the evolution of indetermination.

3.1.1. Introductory tutorial notebook: 01_QRBS_for_dummies

This notebook presents the different concepts of a Rule-Based System (**RBS**) and how to implement them in the **QRBS** software. The examples implemented in this notebook are very simple and only deal with categorical facts and rules. Despite this simplicity, the notebook explains how to use the software for creating the corresponding quantum version of a classical **RBS**. This implies the following steps:

- Definition of the facts, the rules and the logical operators the **QRBS** needs.
- Conversion of the inferential circuits obtained by the **QRBS** to their quantum counterpart.
- Provision of the input facts to the **QRBS**.
- Execution of the inference of the **QRBS** and obtain the output.

The development of this first notebook allowed CESGA team to discover several weak points. The original software version automatically generated the inferential circuits and transformed them into quantum circuits, but could only execute them using:

- A fixed *myqlm* [2] Quantum Process Unit (**QPU**): the **PyLinalg** algebra simulator[3].
- 1024 shots for measuring the quantum circuits results.

[1] https://github.com/NEASQC/qrbs
[2] https://myqlm.github.io/
[3] PyLinalg

The **QPU** limitation was very important for different reasons:

- **PyLinalg** is very slow compared with other available **QPUs** in the *myqlm* like **CLinAlg** (see subsection 3.2)
- Both *myqlm* simulators (**PyLinalg** and **CLinAlg**) have a limit on the number of qubits to simulate (not more than 25).
- With a fixed **QPU** the user cannot take advantage of the **Qaptiva** machines, like the installed in CESGA[4], which allow to simulate quantum circuits with higher number of qubits.
- Additionally, the **QRBS** software won't allow, in the future, to send the quantum circuits to real quantum devices.

The number of shots weakness was important too because the precision of the measurements in a quantum system is related to this parameter.

To solve both weak points, CESGA team developed the following Python module and package:

- **selectable_qpu.py:** this module implements the *SelectableQPU* class. This class allows to evaluate and execute a **QRBS** circuit by providing a **QPU** object. Additionally, the number of shots to measure for each generated quantum circuit can be provided as input.
- **qpu/** package: this package allows to instantiate the different available **Qaptiva QPUs** easily. All the instantiated **QPUs** using this package can be used with the *SelectableQPU* class. More information about this package is provided in the subsection 3.2.

3.1.2. Tutorial to QRBS software: 02_QRBS_Uncertainty.ipynb

This notebook presents the main functionality of the **QRBS** software package: modelling the evolution of indetermination propagating through an inferential network using quantum circuits.

As explained in other deliverables, the indetermination in inferential circuits can arise due to:

1. *Imprecision of the facts*: In this case, the facts are not categorical variables. Now the facts can be affected by a degree of belief, probability or even intensity. In the **QRBS**, this is modelled by assigning a number between 0 and 1 to the precision of the facts (the attribute *precision* of a fact object).
2. *Uncertainty of the rules*: in this case, some indetermination appears when the rule is created. Again, this indetermination is modelled as a number between 0 and 1 and in the **QRBS** is associated with the *certainty* of the rule object.

The notebook provides several examples of *Imprecision* and *Uncertainty* and how the **QRBS** package deals with them. Additionally, the notebook explores how the inferential generated circuits are translated into their quantum counterpart depending on the selected model for indetermination propagation. Up to three different models can be used (for more information about them, please refer to section 3.2 of D6.14 (Moret-Bonillo, Magaz-Romero, Mosqueira-Rey, & Alvarez-Estevez, 2023)):

- *Certainty Factor model:* based on the work of (Shortliffe & Buchanan, 1975). This model is the one initially proposed at the beginning of the project, as the first approach to Quantum Rule-Based Systems (Moret-Bonillo, Magaz-Romero, & Mosqueira-Rey, 2022; Moret-Bonillo, Magaz-Romero, Mosqueira-Rey, & Alvarez-Estevez, 2023).
- *Fuzzy logic:* based on the fuzzy logic initially introduced by Lofti A. Zadeh in 1965 (Zadeh, 1965).
- *Bayesian networks:* University of Coruña (UDC) proposal for a classical-inspired quantum approach to QRBS through Bayesian networks (Borujeni et al., 2021).

The logical inferential gates (*OR*, *NOT*, *AND*), the precision and the rules are translated into different types of quantum circuits depending on the selected model.

[4] CESGA QLM

3.2. Integration into Eviden Qaptiva environment: the QPU package

One of the main software contributions of the evaluation part of the **QRBS** is the integration of the **qpu** package to the software. This section explains what the main purpose of this package is and how to use it for configuring **Eviden QPUs**.

The ecosystem of quantum simulation tools of **Eviden Qaptiva** can be split into 2 different well-differentiated (and interconnected) tools:

- **myQLM**: is the quantum software stack developed by Eviden, for writing, simulating, optimizing and executing quantum programs. This software is a freeware Python package which comes with interoperability connectors. With this tool only two basic simulators are available to the user: **CLinAlg** and **PyLinAlg**.
- **Qaptiva 800 series**: is the Quantum Appliance Toolset developed by Eviden. This software product extends the capabilities of myQLM, adding more functionalities, more emulators and the capability of executing on real quantum hardware. In the case of the emulators, it expands the maximum number of simulated qubits beyond 25. For example, CESGA QLM30 guarantees the possibility of simulating circuits up to 30 qubits, but in some cases, it can simulate larger ones. Also, it includes other advanced simulators as a linear-algebra-based quantum emulator (**LinAlg**) or a Matrix Product State (**MPS**) one. Additionally, this software comes with a lot of extra functionalities like circuit rewriting and optimization tools and the possibility for configuring and simulating noisy Quantum Processing Units using the **Noisy Quantum Emulator** (**NoisyQProc** object).

These tools have the same semantics to create quantum circuits, so the user can write a quantum program using **myQLM** and optimize, rewrite and execute the circuit using the **Qaptiva 800 series**. Additionally, a third tool called **Qaptiva Access** allows the user to submit **myQLM** heavy computation to a remote **Qaptiva** machine.

One of the main problems that arises with this approach is dealing with the different functionalities of these tools. To solve this issue, the **qpu/** package integrates several of them to configure different types of **QPU**s easily.

The main module of the **qpu/** package is the **qpu/select_qpu** one. This module implements a wrapper function called *select_qpu*. The input is a Python dictionary, whose scheme is shown in listing 3.1, that configures a **QPU**.

```
qpu_cfg = {
    "qpu_type": str,
    "qpu_name": str,
    "kak_compiler": str,
    "sim_method": {
        "sim_method": str,
        "bond_dimension": int,
        "n_samples": int
    },
    "t_gate_1qb": int,
    "t_gate_2qbs": int,
    "t_readout": int,
    "depol_channel" : {
        "active": bool,
        "error_gate_1qb": float,
        "error_gate_2qbs": float
    },
    "idle": {
        "amplitude_damping": bool,
        "dephasing_channel": bool,
        "t1": int,
        "t2": int
    },
    "meas": {
        "active": bool,
        "readout_error": float
    }
}
```

*Listing 3.1: Python dictionary schema for select_qpu function from **qpu/select_qpu** module.*

The Python dictionary allows the user to configure 3 different types of QPUs:

- Ideal noiseless **QPU**.
- Ideal noiseless **QPU** with circuit rewriting.

- Noisy **QPU** (includes circuit rewriting).

The following subsections explain how to configure the Python dictionary for these different types of **QPU**.

3.2.1. Ideal noiseless QPU

The ideal noiseless **QPU** is basically an algebra simulator. In this case, the quantum circuit will be emulated without any modification or rewriting of the circuit. There are several available ideal **QPU**s on Eviden Qaptiva software:

- **PyLinalg**: lineal algebra simulator based on Python. For selecting it, the string *python* should be provided to **qpu_type** key of the input dictionary.
- **CLinalg**: lineal algebra simulator based on C. For selecting it, the string *c* should be provided to **qpu_type** key of the input dictionary.
- **LinAlg**: lineal algebra simulator only available in **Qaptiva**. For selecting it, the string *linalg* should be provided to **qpu_type** key of the input dictionary. To use it, the user must be in a Qaptiva server.
- **MPS**: **MPS** simulator only available in in **Qaptiva**. For selecting it, the string *mps* should be provided to **qpu_type** key of the input dictionary. To use it, the user must be on a Qaptiva server.

Additionally, the user can submit the circuit via **Qaptiva Access** to a **Qaptiva** hardware. In this scenario, the user can access the **LinAlg** by providing *qlmass_linalg* or the **MPS** by providing *qlmass_mps* to the **qpu_type** key.

For the ideal noiseless **QPU**s the only key that should be provided is the **qpu_type** key. Other keys can be set to *None*.

3.2.2. Ideal noiseless QPU with circuit rewriting

BE AWARE!! To use this type of QPU the user should be connected directly to a Qaptiva hardware

The *select_qpu* function from **qpu/select_qpu** module can build an ideal noiseless **QPU** with some circuit rewriting capabilities. To activate these capabilities, the string *ideal* should be provided to the **qpu_type** key of the dictionary shown in listing 3.1.

In this case, two rewriting **Qaptiva 800** functionalities are enabled:

1. **KAK compression** plugin [5]: merge consecutive one qubit gates into a temporary unitary matrix, and decompose this temporary matrix using a selected pattern. The pattern can be selected by providing the following strings to the **kak_compiler** key:
   - *ZXZ*: the unitary matrix is decomposed using the following pattern: $R_z - R_x - R_z$.
   - *XZX*: the unitary matrix is decomposed using the pattern: $R_x - R_z - R_x$.
   - *ZYZ*: the unitary matrix is decomposed using the pattern: $R_z - R_y - R_z$.
   - *ions* or *ions*: the unitary matrix is decomposed using the pattern: $R_z - R_x\left(\frac{\pi}{2}\right) - R_z - R_x\left(\frac{\pi}{2}\right) - R_z$.
2. **Toffoli Rewritter**: using the **Pattern Manager** plugin from Qaptiva, all the **Toffoli** gates of the quantum circuit will be decomposed in 1 and 2 qubit gates.

For example, figure 1 shows a quantum circuit obtained from a **QRBS**, where the gate M is given by the equation (3.1).

$$M = \begin{pmatrix} \sin\theta & \cos\theta \\ \cos\theta & -\sin\theta \end{pmatrix} \tag{3.1}$$

When **KAK compression** plugin with pattern *ions* is used, it decomposes the operator M into a set of gates with the pattern $R_z - R_x\left(\frac{\pi}{2}\right) - R_z - R_x\left(\frac{\pi}{2}\right) - R_z$, adding the corresponding angles to the three rotations around z. The transformed circuit is shown in Figure 2.

Similarly, the Toffoli rewriter changes all the Toffoli gates into their corresponding decomposition into 2 and 1 qubit gates, producing the final circuit shown in figure 3.

[5] Plugin available in Qaptiva 800.

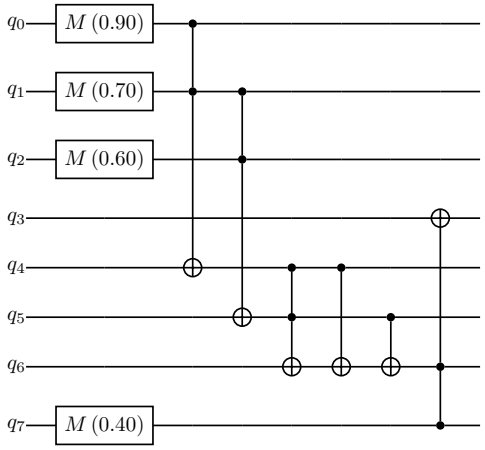


***Figure 1**: Typical quantum circuit from a **QRBS**.*

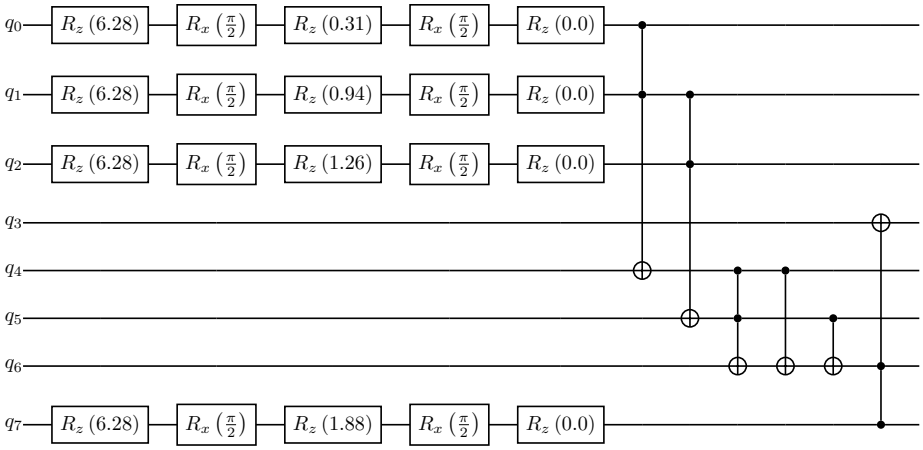


***Figure 2**: One qubit gate decomposition of quantum circuit of figure 1 using **KAK compression** with ions pattern.*

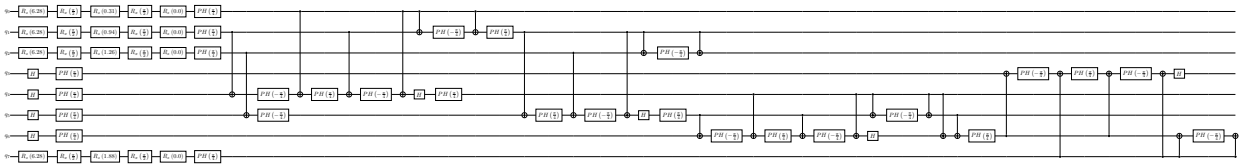


***Figure 3**: Final circuit from Figure 2 after applying Toffoli gate decomposition.*

3.2.3. Noisy QPU

BE AWARE!! To use this type of QPU the user should be connected locally to a Qaptiva hardware

The *select_qpu* function from **qpu/select_qpu** module allows the user to configure noisy simulation, using different **Qaptiva 800 series** functionalities and plugins. To activate these capabilities, the string *noisy* should be provided to the **qpu_type** key of the dictionary shown in listing 3.1.

The noisy simulation model is composed of three different, configurable, parts:

- *Depolarization channel*: in this noisy model, any unitary gate can have a probability of not being applied correctly. To enable it, the **depol_chanel** dictionary should be modified. The **active** key should be set to True. The **error_gate_1qb** and **error_gate_2qbs** keys set the failure probability of one and two-qubit gates respectively.
- *Idle noise*: this part models the behaviour of the qubits during the time that no gate is being applied to them (idle time). The subdictionary key **idle** configures the two different types of idle noises that can be applied:

1. *Amplitude Damping channel*: this is related to the decoherence time T_1 of the qubits. To enable it, the **amplitude_damping** key should be set to True. The T_1 time (nanoseconds) should be provided to the **t1** key.

2. *Phase Damping channel*: this is related to the decoherence time T_2 of the qubits. To enable it, the **dephasing_channel** key should be set to True. The T_2 time (nanoseconds) should be provided to the **t2** key. This case needs the **amplitude_damping** enabled.

- *Readout error*: models the error in the measurement. To configure it, the sub-dictionary **meas** should be used. To enable it, the corresponding **active** key should be set to True. The specific measurement error value should provided to the **readout_error** key.

Once the hardware model noise is modelled, the final step is to configure the type of simulation. This is done by the sub-dictionary **sim_method** shown in Listing 3.1. The most important key is the **sim_method** one that selects the type of simulation:

- **deterministic** simulation: in this case, the complete density matrix evolution for the quantum circuit is simulated. This simulation has a high memory usage but there is no statistical error. To use it the following strings can be provided to the **sim_method** key: *deterministic* or *deterministic-vectorized*. This type of simulation can be used when the number of qubits of the quantum circuits is not too large (for the **QLM 30** in CESGA facilities the thumb rule is no more than 15 qubits).

- **stochastic** simulation: this kind of simulation performs a stochastic sampling over all possible trajectories. To use it the string *stochastic* should be provided to the **sim_method** key. The storage cost is lower than in the **deterministic** but it has some statistical error given by the number of trajectories. The number of trajectories should be provided to the sub-dictionary **sim_method** using the **n_samples** key.

- **Matrix Product Operator (MPO)** simulation: in this case, the simulation uses tensor networks and matrix product state techniques for approximating the final result. To use it, the string *mpo* should be provided to the **sim_method** key. An additional parameter is the bond dimension of the **MPO** that should be provided (as an integer) to the key **bond_dimension** of the **sim_method** sub-dictionary.

3.3. Synthetic case: Basket problem

To perform an initial evaluation of the **QRBS** software, a toy problem was used: *the basketball selection problem*.

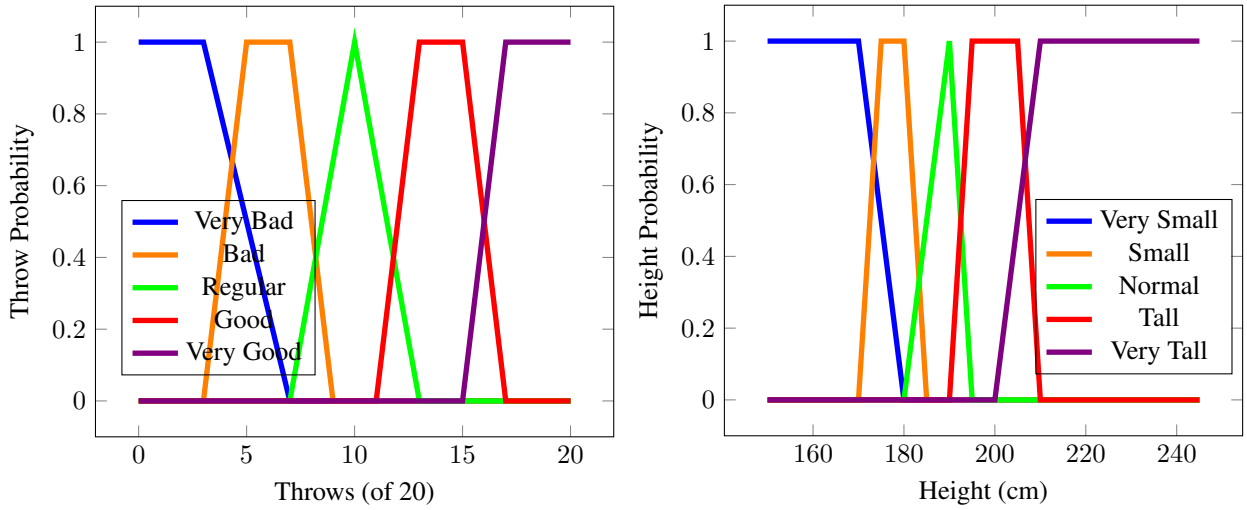
The problem is the following: a basketball coach wants a new player for a team. To select the best player, he decides to evaluate them based on 2 inputs: the height of the player (in cm) and the scored throws (over 20). With these 2 inputs, the coach defines a **RBS** that returns a score between 0 and 100 to evaluate a player's suitability.

First of all, the coach categorizes each one of these 2 inputs into 5 different groups as shown in Table 1 where, for each possible category, the corresponding membership function (see appendix B for definition and notation) is presented: columns *Throws (of 20)* for throws and *Height (cm)* for height.

Throw category	Throws (of 20)	Height Category	Height (cm)
Very bad	1/0-1/3-0/7	Very Small	1/150-1/170-0/180
Bad	0/3-1/5-1/7-0/9	Small	0/170-1/175-1/180-0/185
Regular	0/7-1/10-0/13	Normal	0/180-1/190-0/195
Good	0/11-1/13-1/15-0/17	Tall	0/190-1/195-1/205-0/210
Very Good	0/15-1/17-1/20	Very Tall	0/200-1/210-1/250

***Table 1**: Categorization of the 2 inputs features for the basket problem.*

Figure 4 plots the corresponding membership functions for the categorization of the throws (Figure 4a) and the heights (Figure 4b). To understand how Table 1 and Figure 4 work some examples are provided:



(a) Throws *(b) Heights*

***Figure 4**: Membership functions for Table 1.*

- A player with 12 of 20 throws will be categorized as a *Good player* (so this player will have a one in the *Good* player category and a zero in the others).
- A player with a height of 180 cm will be categorized as a small player (so this player will have a one in the *Small* player category and a zero in the others).
- A player with 16 of 20 in throws will have a 0.5 in *Good* and *Very Good* player categories and zero in the others.
- A player with a height of 192 cm will have a 0.6 in *Normal* a 0.4 in *Tall* categories and zero in the others.

Number	Rule	Output
Rule 1	IF height = normal AND Throws = Good OR height = normal AND Throws = Very Good OR height = tall And Throws = Regular OR height = Very tall And Throws = Regular	Normal
Rule 2	IF height = Tall AND Throws = Good OR height = Very Tall AND Throws = Good	Good
Rule 3	IF height = Tall AND Throws = Very Good OR height = Very Tall AND Throws = Very Good	Very Good
Rule 4	IF shots = Very Bad OR Throws = Bad	Bad
Rule 5	IF height = Very small OR height = Small	Bad
Rule 6	IF height = Normal OR Throws = Regular	Bad

***Table 2**: Rules for the basket **RBS**.*

Secondly, based on his own experience, the basketball coach decides to categorize the players into 4 groups: $I = \{Bad, Normal, Good, Very\ Good\}$ and develop the rules presented in Table 2 for this **RBS**. The problem is that the inputs are not categorical (a player can have a 0.5 in *Good* and *Very Good* throw and a 0.6 in *Normal* and 0.4 in *Tall* categories) so the output facts, $I = \{Bad, Normal, Good, Very\ Good\}$, won't be categorical, they will be affected by some imprecision, $a_I \in [0, 1]$, (a player can be 0.4 *Normal*, 0.6 *Good* and 0 for the other categories).

The final ingredient of the basket **QRBS** is the Table 3. This table presents the membership function that relates a **final score** between 0 and 100 with the output category (*Bad*, *Normal* ...) of a player. Figure 5 plots them for the final score domain. In the proposed **QRBS** the inverse problem is presented: a score for each of the four output categories is obtained and it is needed to convert it to a **final score**.

Final player Category	Final Score
Bad	1/0
Normal	1/0-1/25-0/40
Good	0/25-1/40-1/60-0/75
Very Good	0/60-1/75-1/100

***Table 3**: Final Categorization of the players and corresponding membership function.*

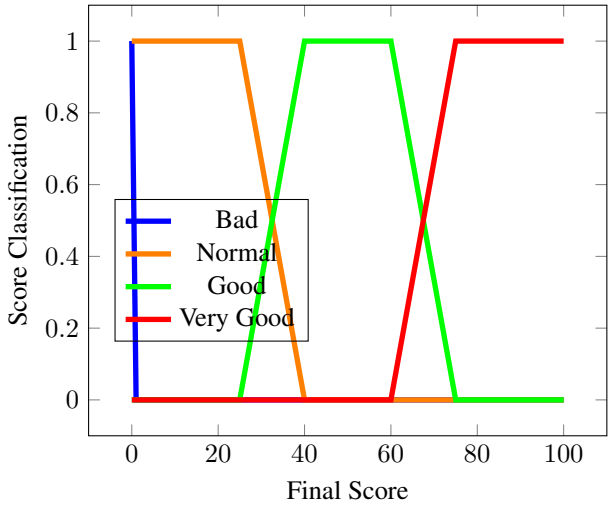


***Figure 5**: Membership function graphic for Scoring categorization showed in Table 3.*

To solve this inverse problem, given the score of the player for each possible output category, a_I, it is mandatory to compute the degree of membership function of the player to each category $p_I(x)$ given by (3.2) where $\mu_I(x)$ is the membership function for the I output fact (see Figure 5).

$$p_I(x) = \min\left(\mu_I(x), a_I\right)) \tag{3.2}$$

Finally, a $Z(x)$ function should be built using a Mamdani implication, defined as

$$Z(x) = \max_I\left(p_I(x)\right)) \tag{3.3}$$

The final score is given by:

$$final\ score = \frac{\int Z(x)\ x\ dx}{\int Z(x)\ dx} \tag{3.4}$$

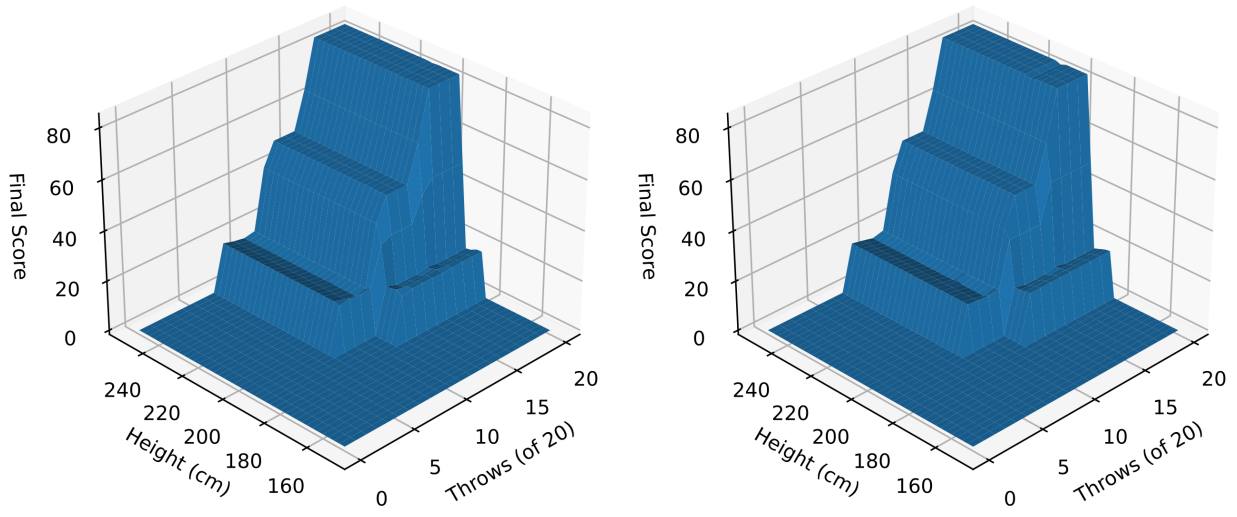
where the integration should be done over the final score domain (between 0 and 1).

For executing this case, a third notebook was developed (*qrbs/misc/03_Basket.ipynb*). It includes a full description of this easy basket example and an implementation using the **QRBS** software. To evaluate it, the basket **RBS** was developed using a fuzzy logic Matlab implementation as well. The corresponding *.fis* Matlab file with the basket **RBS** implementations is presented in Listing A.1 in the annex A.

Name	Throws	Height	Final Score (QRBS)	Final Score (Matlab)
Player 1	16	198	64.6	64.6
Player 2	17	193	59.5	56.1
Player 3	17	188	16.5	16.9
Player 4	15	203	50.0	50.0
Player 5	18	176	0.0	0.0
Player 6	18	186	17.4	17.6

***Table 4**: Evaluation of different players under the basket **QRBS** and the fuzzy logic Matlab implementation.*

Table 2 shows the evaluation for different players, using the **QRBS** and the Matlab fuzzy logic implementations. As can be seen, both final scores are compatible.



(a) QRBS. *(b) Fuzzy logic.*

***Figure 6**: Control surface for the basket problem.*

Figure 6 shows the 2 surface control plots for the basket problem when implemented using the **QRBS** (Figure 6a) and the fuzzy logic Matlab program (Figure 6b). As can be seen, the two surfaces are very similar, as expected, so the **QRBS** works similarly to classical (not quantum) tools.

A more representative view can be found in Figure 7 where the difference between the 2 surface control plots is depicted (so this figure is the difference between Figures 6a and 6b). The surface control plots are not identical but they are very similar and the differences are compatible.

Finally, Figure 8 presents the relationship between the final scores using **Fuzzy logic** and **QRBS**. The dashed line is a slope 1 line for eye-guided purposes.

As a consequence of the previous data, it can be concluded that the basket problem was implemented correctly using the **QRBS** and that this system works in a similar way to classical indetermination propagation models (like **fuzzy logic**).

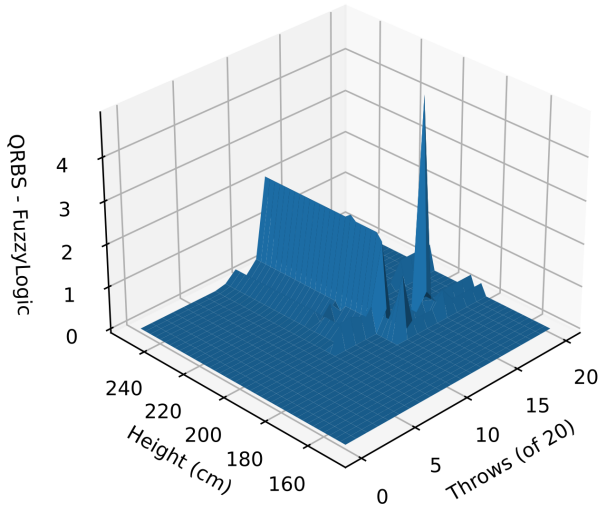


***Figure 7**: Surface control plot difference between **QRBS** and **Fuzzy Logic** ones..*

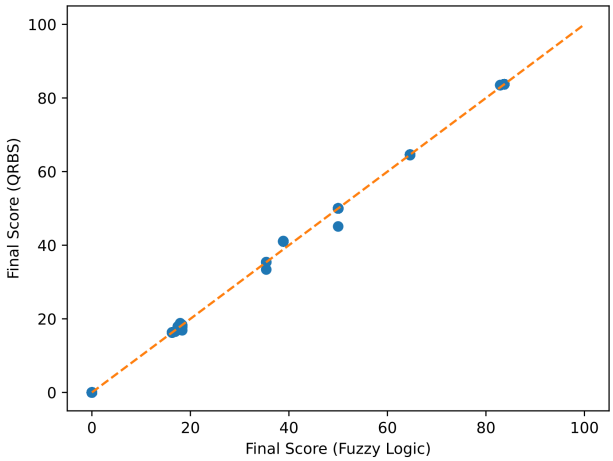


***Figure 8**: Comparison between **QRBS** and **Fuzzy Logic** final scores. The dashed line of slope one is presented for eye guide purposes.*

3.4. QRBS for Invasive Ductal Carcinoma staging

The main objective for UC6 was the development of **QRBS** for breast cancer detection, in concrete to Invasive Ductal Carcinoma staging (**IDC**). Here we briefly glimpse the **IDC** staging system. Please refer to other deliverables for a more detailed explanation.

The **QRBS** developed implementation of the **IDC** staging system, is supported by a knowledge model based on the TNM classification system (Giuliano et al., 2018) where the three sets of variables *T*, *N* and *M* (see Figure 9) are used to classify the current state of the patient (Moret-Bonillo, Gomez Tato, Magaz-Romero, et al., 2023b).

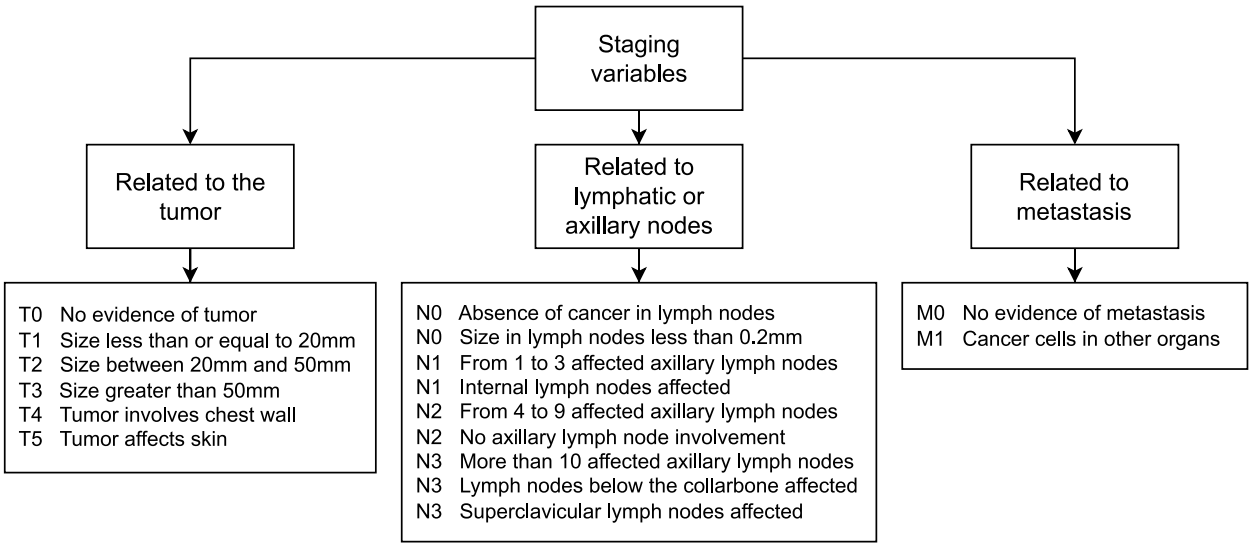


***Figure 9**: Variables or symptoms that are considered in the knowledge model for IDC staging.*

Once the patient has been classified regarding the TNM system, the **IDC** stage is determined through the correspondence of Table 5. However, several TNM possible classifications fit in more than one **IDC** stage (for example, the classification *T0 N1 M0* corresponds to both stages *I-B* and *II-A*).

IDC Stage	*Compatible TNM classification*
I-A	T1 N0 M0
I-B	T0 N1 M0 / T1 N1 M0
II-A	T0 N1 M0 / T1 N1 M0 / T2 N0 M0
II-B	T2 N1 M0 / T3 N0 M0
III-A	T0 N2 M0 / T1 N2 M0 / T2 N0 M0 / T3 N2 M0 / T3 N1 M0
III-B	T4 N0 M0 / T4 N1 M0 / T4 N2 M0
III-C	TX N3 M0
IV	TX NY M1

***Table 5**: Invasive Ductal Carcinoma stages according with TNM classification system.*

An **IDC** application using the **QRBS** software was developed into the framework of the project and well documented in (Moret-Bonillo, Magaz-Romero, Mosqueira-Rey, & Alvarez-Estevez, 2023).

For this evaluation, a fourth notebook was developed (*03_IDC.ipynb*) where the **IDC** was built using the **QRBS** software. The notebook explains the case and how to program it using the **QRBS** software.

The validation test of the **QRBS** implementation of the **IDC** was performed by executing each possible TMN classification and verifying that the precision of the different output stages is compatible with the **TMN** classification of Table 5. In the notebook, the mandatory code for reproducing this validation text is provided, but the user must have access to Qaptiva hardware (connected locally or using the **Qaptiva Access** tool) to execute it. Additionally, the obtained validation results can be found in the notebook. Table 6 summarizes them when the generated quantum circuits were simulated with an ideal noiseless **QPU**. The table presents, for each TMN classification, all the compatible inputs and

the stage or stages that had a 1.0 precision (the other stages had a 0.0 precision). As can be seen, the obtained results reproduce the classical and standard staging presented in Table 5.

The last line of Table 6 presents input symptoms that are not compatible with any TMN classification of Table 5 and there is no valid staging output. In this case, all the outputs for the precision provided by **IDC QRBS** are 0.0.

TNM	Compatible inputs	Precision 1.0
T1 N0 M0	T1N0AM0, T1N0BM0	Stage I-A
T0 N1 M0	T0N1AM0, T0N1BM0	Stage I-B, Stage II-A
T1 N1 M0	T1N1AM0, T1N1BM0	Stage I-B, Stage II-A
T2 N0 M0	T2N0AM0, T2N0BM0	Stage II-A, Stage III-A
T2 N1 M0	T2N1AM0, T2N1BM0	Stage II-B
T3 N0 M0	T3N0AM0, T3N0BM0	Stage II-B
T0 N2 M0	T0N2AM0, T0N2BM0	Stage III-A
T1 N2 M0	T1N2AM0, T1N2BM0	Stage III-A
T3 N1 M0	T3N1AM0, T3N1BM0	Stage III-A
T3 N2 M0	T3N2AM0, T3N2BM0	Stage III-A
T4 N0 M0	T4N0AM0, T4N0BM0	Stage III-B
T4 N1 M0	T4N1AM0,T4N1BM0	Stage III-B
T4 N2 M0	T4N2AM0,T4N2BM0	Stage III-B
TX N3 M0	T0N3AM0, T0N3BM0, T0N3CM0, T1N3AM0, T1N3BM0, T1N3CM0, T2N3AM0, T2N3BM0, T2N3CM0, T3N3AM0, T3N3BM0, T3N3CM0, T4N3AM0, T4N3BM0, T4N3CM0	Stage III-C
TX NY M1	T0N0AM1, T0N0BM1, T0N1AM1, T0N1BM1, T0N2AM1, T0N2BM1, T0N3AM1, T0N3BM1, T0N3CM1, T1N0AM1, T1N0BM1, T1N1AM1, T1N1BM1, T1N2AM1, T1N2BM1, T1N3AM1, T1N3BM1, T1N3CM1, T2N0AM1, T2N0BM1, T2N1AM1, T2N1BM1, T2N2AM1, T2N2BM1, T2N3AM1, T2N3BM1, T2N3CM1, T3N0AM1, T3N0BM1, T3N1AM1, T3N1BM1, T3N2AM1, T3N2BM1, T3N3AM1, T3N3BM1, T3N3CM1, T4N0AM1, T4N0BM1, T4N1AM1, T4N1BM1, T4N2AM1, T4N2BM1, T4N3AM1, T4N3BM1, T4N3CM1	Stage IV
No Stage	T2N0AM0, T0N0BM0, T2N2AM0, T2N2BM0	Neither

***Table 6**: Validation test results for **QRBS** implementation of**IDC**. For each TMN classification, all the compatible inputs and the stages with 1.0 precision are presented.*

3.4.1. Imprecision in QRBS IDC

In the validation test presented in sub-section 3.4, the input symptoms of the **QRBS** implementation of the **IDC** were used as categorical variables (so the precision was set to 0 or to 1). In real-life medicine, the input symptoms can be affected by some indetermination and the practitioners can assign some probability to them based on their own experience. QRBS can propagate, in a natural way, this indetermination (by assigning the desired probability to the input symptoms) and provide a final probability for each output staging.

labels	T0	T1	T2	T3	T4	T5	N0A	N0B	N1A	N1B	N2A	N2B	N3A	N3B	N3C	M0	M1
T0T1/N0/M0	0.3	0.7	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	0.0
T0T1/N1N2/M0	0.3	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.2	0.0	0.0	0.0	0.0	1	0.0
T0T1/N2N3/M0	0.3	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.3	0.0	0.0	1	0.0
T1T2/N0/M0	0.0	0.8	0.2	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	0.0
T1T2/N1N2/M0	0.0	0.8	0.2	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.2	0.0	0.0	0.0	0.0	1	0.0
T1T2/N2N3/M0	0.0	0.8	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.3	0.0	0.0	1	0.0
T2T3/N0/M0	0.0	0.0	0.6	0.4	0.0	0.0	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	0.0
T2T3/N1N2/M0	0.0	0.0	0.6	0.4	0.0	0.0	0.0	0.0	0.8	0.0	0.2	0.0	0.0	0.0	0.0	1	0.0
T2T3/N2N3/M0	0.0	0.0	0.6	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.3	0.0	0.0	1	0.0

***Table 7**: Imprecision input facts for testing indetermination propagation in the IDC **QRBS** implementation.*

To show how the system deals with some degree of imprecision in the input symptoms, several simulations with ideal **QPU** (presented in the last part of the *03_IDC.ipynb* notebook) were performed. Table 7 shows the configuration of

the different input symptoms used. The imprecision was assigned arbitrarily (and without any medical knowledge). The corresponding results are presented in Figure 10 where, for each tested input from Table 7, the obtained Stage precisions are plotted as a heat map.

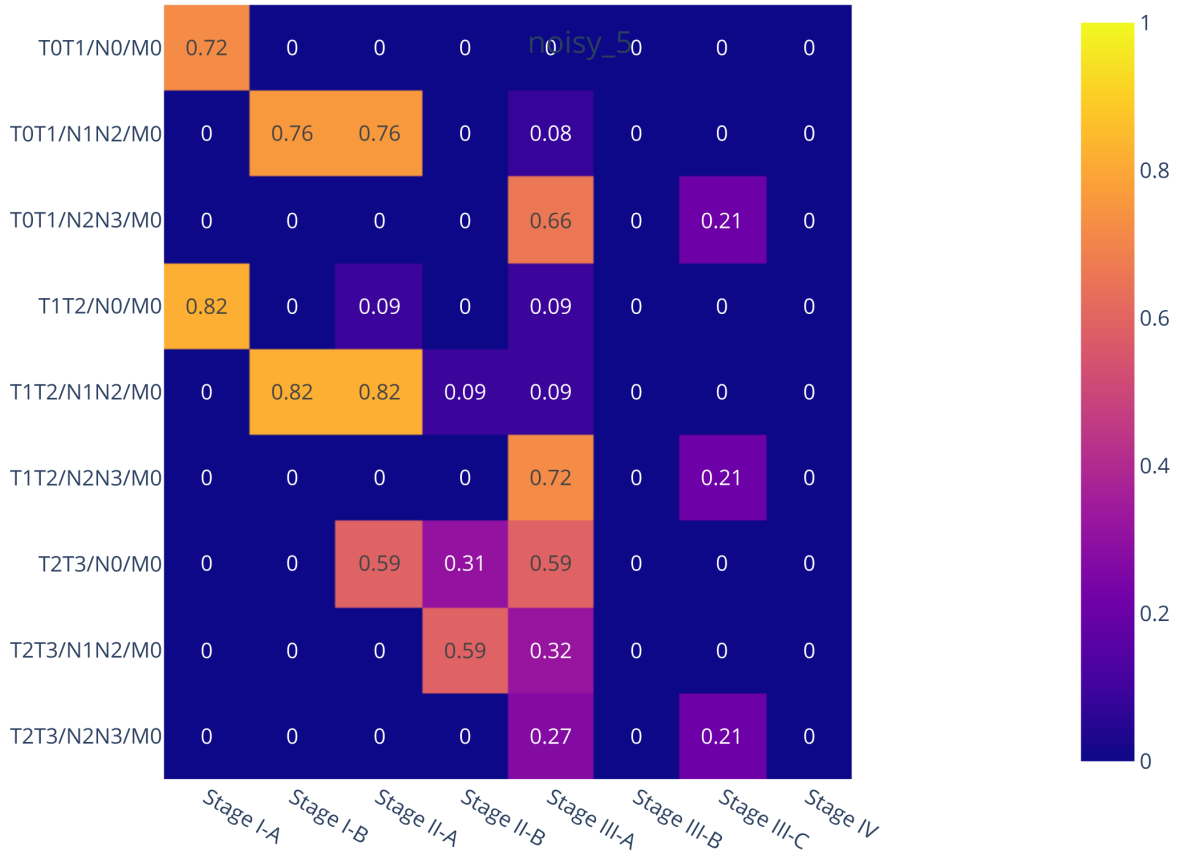


***Figure 10**: Precision of all the stages for the TMN inputs of Table 7.*

As can be seen, the stages that before were categorical (had a 1.0) now have some probability assigned and even some stages that before were impossible, now appear with some probability.

The results obtained do not significantly differ from those that could be expected when considering the subjectivity of the clinician who analyzes the case. The same case evaluated by two different experts would produce slightly different results depending on the practitioner. Therefore, the variability of the outputs is merely a consequence of the clinician's subjectivity when interpreting the results. In other words, we are not talking about mathematical conditional probability, but introducing the concept of subjective conditional probability.

This QRBS functionality is beyond the capabilities of the standard classical software used for this benchmarking and could represent a future advantage of the proposed method.

3.5. Noisy simulations

The last step is the evaluation of the resilience of QRBS to noise. Currently, **QPU**s have an important noise that is decreasing continuously, and with important advances in logical qubits. But maybe in the next years, there will still be noise during the execution of the quantum programs. So, it is important to know how it will affect the programs and, also, to assess the level of noise that it is acceptable for each case. In this section, using emulated noise as described in previous sections, the resilience to noise of QRBS is evaluated using the two previous cases: the synthetic basket problem and the IDC diagnosis.

The main idea was to simulate the problems using **QPU**s with different levels of noise. The hardware model used for building them consists of a *depolarization channel* and *idle* noise, with **amplitude and the phase damping** channels activated. Table 8 shows the settings used in the experiment. The noisy **QPU**s are labelled by the level of noise. The

lowest one is the *noisy_0* with very low error rates for the gates and long decoherence times. This case is the closest to an ideal simulation. The highest level of noise is the *noisy_5* whose parameters were obtained from the median values of the IBM Brisbane QPU. Other parameters for the simulation depend on the case and will be explained in each subsection.

qpu_name	t_gate_1qb (ns)	t_gate_2qbs (ns)	error_gate_1qb	error_gate_2qbs	T1 (ns)	T2 (ns)
noisy_0	35	660	2.27E-09	7.74E-08	2.32E+10	1.33E+10
noisy_1	35	660	2.27E-08	7.74E-07	2.32E+09	1.33E+09
noisy_2	35	660	2.27E-07	7.74E-06	2.32E+08	1.33E+08
noisy_3	35	660	2.27E-06	7.74E-05	2.32E+07	1.33E+07
noisy_4	35	660	2.27E-05	7.74E-04	2.32E+06	1.33E+06
noisy_5	35	660	**2.27E-04**	**7.74E-03**	**2.32E+05**	**1.33E+05**

***Table 8**: Parameter configuration for building the noisy **QPU**s. The bold values correspond to the median values for **IBM Brisbane** QPU.*

3.5.1. Basket problem

The generated circuits by the basket **QRBS** do not demand, in general, a large number of qubits (the widest circuit uses 19 qubits). So, this case was used as a testing bench for noisy simulation.

The main idea was to simulate the basket **QRBS** for the players of Table 4 using **QPU**s with different levels of noise. Table 9 presents other parameter configurations of the noisy simulation experiments for this test.

Parameter	Key	Value
Compression Pattern	kak_compiler	ZXZ
Simulation method	sim_method	stochastic
Number of trajectories	n_samples	10000

***Table 9**: Simulation parameters for the noisy experiments. The table presents the name of the parameter, the corresponding key in the **QPU** configuration dictionary (see Listing 3.1) and the value used.*

To all the noisy **QPUs** from Table 8, a **KAK compression** plugin with the pattern *ZXZ* and a Toffoli rewriter were included. The simulation method was the *stochastic* one because the basket **QRBS** can generate circuits with more than 15 qubits. The number of trajectories was set to 10000 (for the biggest circuits the obtained results do not change when more trajectories were used). The three models for propagation of indetermination were simulated (i.e. *Certainty Factors*, *Fuzzy Logic* and *Bayesian Networks*). Finally, due to a problem with the **Qaptive 800 series** when the *stochastic* simulation is used, the number of shots was set to 0. This implies that instead of sampling from the computed probability distribution, the system returned the desired state probability. The **Qaptive 800 series** randomly raises an error when *stochastic* simulation is used with a number of shots different from 0. It is expected that this issue will be solved with the following versions of the library.

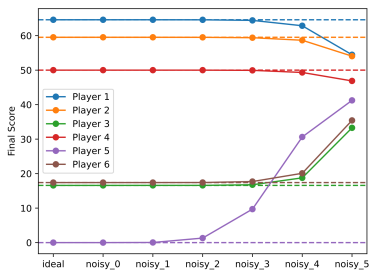


***Figure 11**: Basket **QRBS** noisy simulation results for player of Table 4 under the Certainty factor model. The dashed lines show the ideal scores.*

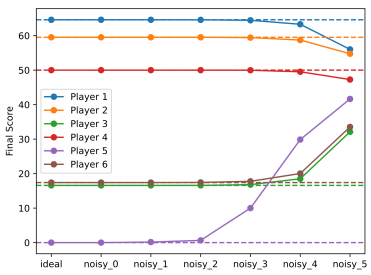


***Figure 12**: Basket **QRBS** noisy simulation results for player of Table 4 under the Fuzzy logic model. The dashed lines show the ideal scores.*

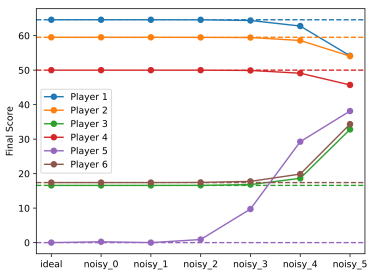


***Figure 13**: Basket **QRBS** noisy simulation results for player of Table 4 under the Bayesian networks model. The dashed lines show the ideal scores.*

The results of the noisy simulation are presented in Figure 11, where the *Certainty factor* model is used, Figure 12, corresponding to *Fuzzy logic*, and in Figure 13 corresponding to the *Bayesian networks*. The dashed lines in these figures correspond to the ideal simulation for each player. As can be seen, until the noisy level *noisy_2* the difference between ideal and noisy simulations is negligible. For the *noisy_3* and *noisy_4* the noisy results are compatible for all players except *player 5* (whose ideal final score is 0). For *noisy_5*, with the parameters of the **IBM Brisbane**, the difference between noise and ideal results is too high and the **QRBS** is not usable.

The noise model used in the simulation is very simple (only idle and depolarizing channels were used) but provides some insight into how the acceptable noise (error rate gates and decoherence times) a quantum device should have to execute the basket **QRBS**. The maximum noise level should be the *noisy_3* because the Final Score order is kept meanwhile in the *noisy_4* model the *Player 5* would have a better score than players *3* and *6*. If only the best final scores are important because some threshold will be added (players with a final score lower than 40 are rejected automatically for example), the *noisy_4* can be usable.

Finally, it is clearly seen that increasing the noise makes that the results for all the players converge. Probably, if larger noise would be simulated, all the scores would converge to a single value, making the results completely useless.

3.5.2. Noisy simulations for QRBS IDC

The same evaluation was performed on the IDC staging system built using **QRBS**. For all the different compatible **TMN** classifications (see Table 5), the **QRBS** IDC was simulated using the noisy **QPU**s presented in Table 8.

tnm	T0	T1	T2	T3	T4	T5	N0A	N0B	N1A	N1B	N2A	N2B	N3A	N3B	N3C	M0	M1
T1 N0 M0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
T0 N1 M0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
T1 N1 M0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
T2 N0 M0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0
T2 N1 M0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0
T3 N0 M0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0
T0 N2 M0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
T1 N2 M0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
T3 N2 M0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0
T3 N1 M0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0
T4 N0 M0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0
T4 N1 M0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0
T4 N2 M0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0
TX N3 M0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
TX NY M1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1

***Table 10**: Tested TNM classification and the corresponding input case used for noisy simulations of IDC **QRBS** implementation.*

Each possible **TMN** classification can be obtained by different combinations of the variables presented in Figure 9. Table 10 shows the combination of variables used for each tested **TMN** classification.

As in the basket problem, to all the noisy **QPU**s from Table 8, two different plugins were added: one for rewriting Toffoli gates and a **KAK compression** one with the pattern *ZXZ*. The simulation method was the *stochastic* one because the IDC **QRBS** can generate circuits with more than 20 qubits. The number of trajectories was set to 500 (for the biggest circuits the obtained results did not change when more trajectories were used). Only the *Certainty Factors* indetermination propagation model was simulated in the noisy simulations. Again the number of shots was set to 0 (so the desired state probability from the computed probability distribution was returned instead of a sampling).

Figure 14 shows the results of the noisy simulations. This plot presents the evolution of the precision of the correct stage for each TMN classification of Table 10 with the different noisy **QPU**s. As can be seen, until the noisy level *noisy_3* the difference between ideal and noisy simulations is negligible. For the *noisy_4* level, the precision of the correct stage decreases but with this level of noise, the outputs can be acceptable. For the *noisy_5* level, with noisy parameters taken from **IBM Brisbane** quantum device, the precision of the correct stage decreases a lot for many of the stages and the system does not work properly.

Figure 15 presents the results of the IDC noisy simulation using heat maps. For each possible TMN classification (y-axis) tested, (see Table 10) the different precision obtained for all the possible stages (x-axis) (see Table 5) is represented as a colour scale between blue (precision 0) and yellow (precision 1).

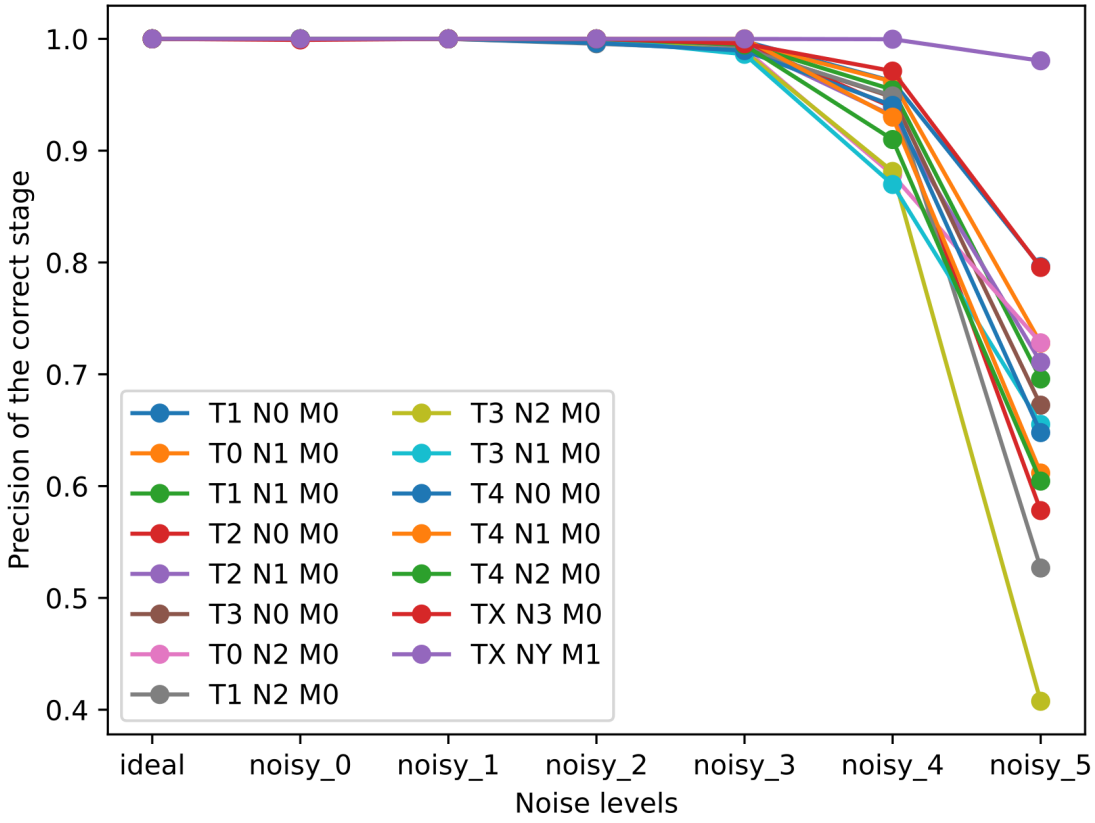


***Figure 14**: Simulation results for the IDC for the different TMN classifications are shown in Table 10.*

As can be seen, the **QRBS** implementation of IDC works properly until *noisy_3* level: the different TMN classifications provide the corresponding correct stage (or stages). For example, until *noisy_3* level, the classification *T0N1M0* delivers a 1 precision for *Stage I-B* and *Stage II-A* and a 0 for the other possible stages (according to the classification presented in Table 5). For the *noisy_4* level, the precision of the correct staging is not 1.0 (in general) and other incorrect stages arise with some precision. For example, the *T0N1M0* produces a precision of 0.95 for *Stage I-B*, 0.94 for *Stage II-A* and a 0.11 for the *Stage III-A* (an incorrect staging for a *T0N1M0* classification). So when the noise level increases, additional bad staging arises.

This is more clearly presented in the *noisy_5* level where the precision of bad stages can be increased to values similar to the precision of the good stages, For example, the *T0N1M0* produces a precision of 0.74 for *Stage I-B*, 0.65 for *Stage II-A*, a 0.62 for the *Stage III-A* a 0.35 for *Stage III-B* and so on.

Although the noise model used in the simulation is very simple (only idle and depolarizing channels were used), some insight about the acceptable noise (error rate gates and decoherence times) a quantum device should have to execute the **QRBS** implementation of the IDC can be obtained. It looks like a quantum device with a *noisy_3* compatible level could execute the software properly. Even if the noise level is compatible with *noisy_4* the software can be executed with some confidence. A device compatible with the *noisy_5* will produce a lot of bad stagings and the software can not be executable in a reliable way.

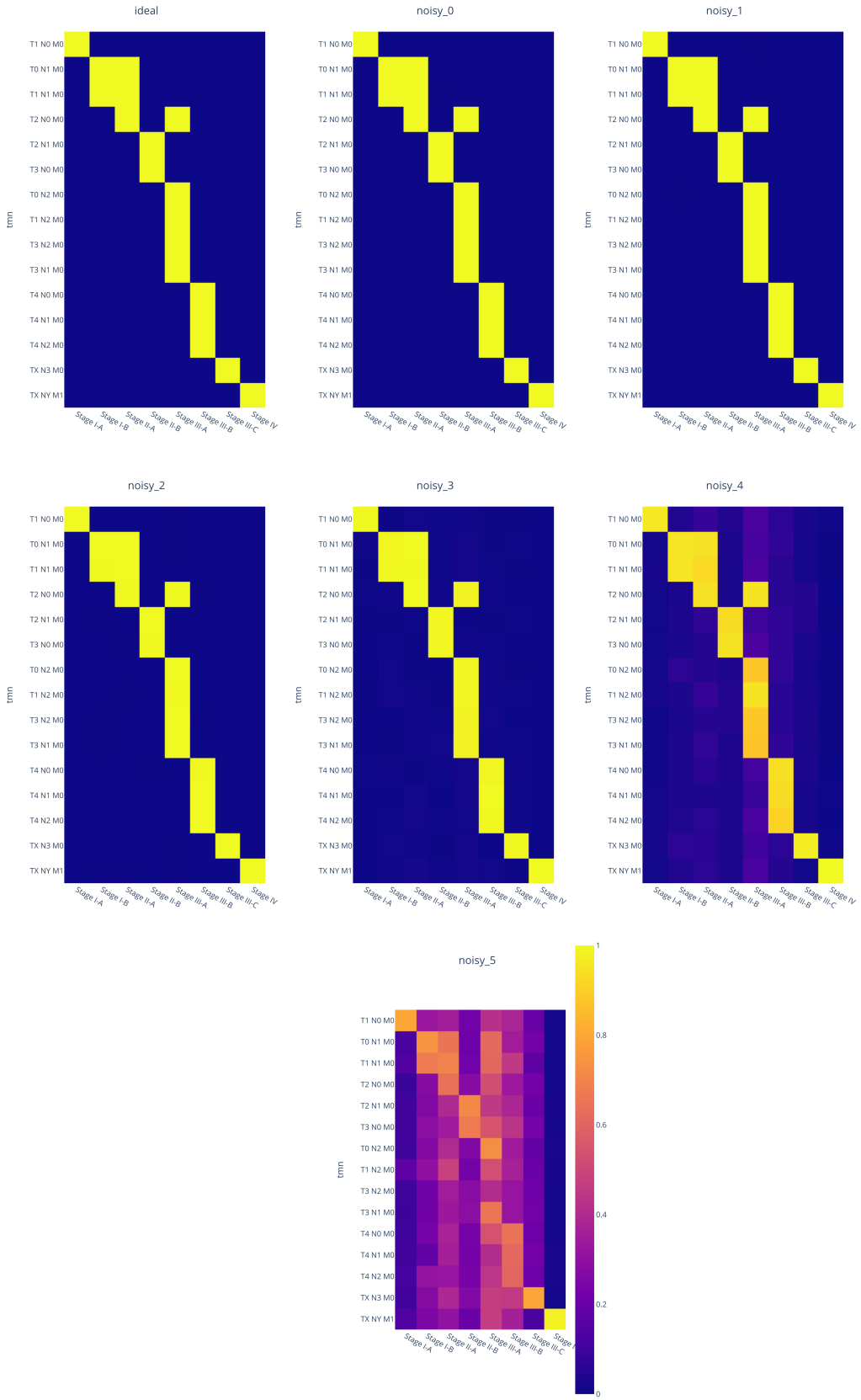


***Figure 15**: Results of the noisy simulations performed on IDC. The heat maps present the precision, range between 0 (blue) and 1 (yellow) obtained for each possible staging (x-axis) for the tested TMN classifications (y-axis).*

4.Conclusions

In this deliverable, we have presented the evaluation of the QRBS software library. The results of the evaluation permit to extract the following conclusions:

- The results obtained for all the cases in a noiseless environment are those expected for the defined problems. These values present a high degree of similarity with those of classical models, as shown by comparing QRBS with fuzzy logic. In fact, the problem is not inferential; the problem is derived from the classification criteria.
- When comparing classical and quantum methods, it can be observed that the corresponding results are consistent. However, the QRBS approach improves the outputs adding intrinsic uncertainties to the inferences, which are not available in the classical approach. The results obtained show an almost identical behaviour among them, thus reinforcing the hypothesis (in both, qualitatively and quantitatively perspectives) that Quantum Computing is a more general paradigm, with the capacity to encompass the different classical models of symbolic reasoning.
- Noise affects the results catastrophically when it is higher than a threshold, producing for the current **QPU**s useless results. However, for the tested models, a moderate improvement in the quality of the **QPU**s could produce acceptable results. Of course, the executed models do not justify yet the usage of real **QPU**s, because they are still small, allowing the classical ideal emulation. Experiments with larger models are needed to assess clearly the effect of the noise.

Also, the evaluation produces several improvements to the original QRBS software. Some tutorials were produced as notebooks, that explain clearly the capabilities of the developed library. These notebooks explain the different **QRBS** concepts, some improvements related to the **QPU** used for solving the quantum circuits generated for the systems, the generation from scratch of an intermediate **RBS** using the library and evaluating it in ideal and noisy simulation (the basketball case). The software was extended with new capabilities that now permit the execution on other emulators using Qaptiva, and in the future, using it to connect to real QPUs.

List of Acronyms

Term	Definition
API	Application Programming Interface
IDC	Invasive Ductal Carcinoma
MPO	Matrix Product Operator
MPS	Matrix Product State
QRBS	Quantum Rule-Based System
RBS	Rule-Based System
UC	Use Case
QPU	Quantum Process Unit
QLM	Quantum Learning Machine

***Table 11**: Acronyms and Abbreviations*

List of Figures

List of Tables

Bibliography

Borujeni, S. E., Nannapaneni, S., Nguyen, N. H., Behrman, E. C., & Steck, J. E. (2021). Quantum circuit representation of bayesian networks. *Expert Systems with Applications*, *176*, 114768. https://doi.org/https://doi.org/10.1016/j.eswa.2021.114768

Giuliano, A. E., Edge, S. B., & Hortobagyi, G. N. (2018). Eighth edition of the ajcc cancer staging manual: Breast cancer. *Annals of Surgical Oncology*, *25*(7), 1783–1785. https://doi.org/10.1245/s10434-018-6486-6

Magaz-Romero, S. (2024). Neasqc: Qrbs. https://github.com/NEASQC/qrbs

Moret-Bonillo, V., Gomez Tato, A., Magaz Romero, S., Mosqueira-Rey, E., & Alvarez-Estevez, D. (2022, October). D6.9: Qrbs software specifications. https://doi.org/10.5281/zenodo.7299193

Moret-Bonillo, V., Gomez Tato, A., Magaz-Romero, S., Mosqueira-Rey, E., & Alvarez-Estevez, D. (2023a, July). D6.11 Preliminary QRBS software and IDC application specification. https://doi.org/10.5281/zenodo.8108580

Moret-Bonillo, V., Gomez Tato, A., Magaz-Romero, S., Mosqueira-Rey, E., & Alvarez-Estevez, D. (2023b, July). D6.11 Preliminary QRBS software and IDC application specification. https://doi.org/10.5281/zenodo.8108580

Moret-Bonillo, V., Magaz-Romero, S., & Mosqueira-Rey, E. (2022). Quantum computing for dealing with inaccurate knowledge related to the certainty factors model. *Mathematics*, *10*(2). https://doi.org/10.3390/math10020189

Moret-Bonillo, V., Magaz-Romero, S., Mosqueira-Rey, E., & Alvarez-Estevez, D. (2023, December). D6.14 Final QRBS software and IDC application. https://doi.org/10.5281/zenodo.10868936

Moret-Bonillo, V., Mosqueira-Rey, E., & Magaz-Romero, S. (2021, December). D6.5 Quantum Rule-Based System (QRBS) Requirement Analysis. https://doi.org/10.5281/zenodo.5949157

Moret-Bonillo, V., Mosqueira-Rey, E., Magaz-Romero, S., & Gómez-Tato, A. (2021). Quantum rule-based systems (qrbs) models, architecture and formal specification (D6. 2). https://www.neasqc.eu/wp-content/uploads/2021/05/NEASQC_D6.2_QRBS-Models-Architecture-and-Formal-Specification-V1.5-Final.pdf

Shortliffe, E. H., & Buchanan, B. G. (1975). A model of inexact reasoning in medicine. *Mathematical Biosciences*, *23*(3), 351–379. https://doi.org/https://doi.org/10.1016/0025-5564(75)90047-4

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353. https://doi.org/doi.org/10.1016/S0019-9958(65)90241-X

A.Matlab code for implementing classical Rube-Based System (RBS)

```
[System]
Name='basket'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=9
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='shots'
Range=[0 20]
NumMFs=5
MF1='throw_very_bad':'linzmf',[3 7]
MF2='throw_bad':'trapmf',[3 5 7 9]
MF3='throw_regular':'trimf',[7 10 13]
MF4='throw_good':'trapmf',[11 13 15 17]
MF5='throw_very_good':'linsmf',[15 17]

[Input2]
Name='skill'
Range=[150 250]
NumMFs=5
MF1='height_very_small':'linzmf',[170 180]
MF2='height_small':'trapmf',[170 175 180 185]
MF3='height_normal':'trimf',[180 190 195]
MF4='height_tall':'trapmf',[190 195 205 210]
MF5='height_very_tall':'linsmf',[200 210]

[Output1]
Name='score'
Range=[0 100]
NumMFs=4
MF1='bad':'linzmf',[25 40]
MF2='regular':'linzmf',[25 40]
MF3='good':'trapmf',[25 40 60 75]
MF3='very_good':'linsmf',[60 75]

[Rules]
1 1, 1 (1) : 1
2 1, 1 (1) : 1
3 1, 1 (1) : 1
1 2, 1 (1) : 1
2 2, 1 (1) : 1
3 2, 1 (1) : 1
1 3, 1 (1) : 1
2 3, 1 (1) : 1
3 3, 1 (1) : 1
```

*Listing A.1: Matlab .fis code for implementing the **RBS** using fuzzy logic.*

B.Membership functions

A membership function is a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1.

In this deliverable, we assume that all membership functions are linear-piecewise. We have used a simplified notation to define the membership functions: each function is a list of pairs where the first element is the degree of membership and the second one is the domain coordinate.

The following membership functions were used in this deliverable:

- With three elements:
  - linear *z-shape*: the first two elements have a degree of membership of 1 and the third a 0. *Example*: 1/0-1/3-0/7 (see Figure 16).

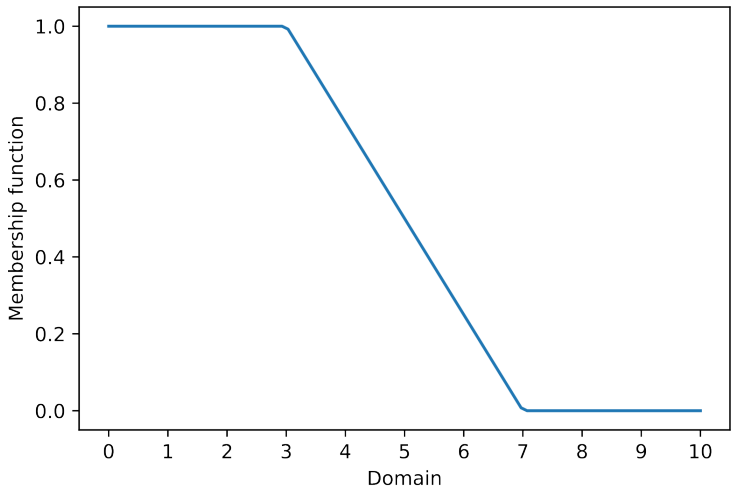


***Figure 16**: Plot with the membership function for a z-shape: 1/0-1/3-0/7*

  - linear *s-shape*: the first element has a degree of membership of 0 and the other two have a 1. *Example: 0/3-1/7-1/10* see Figure 17.

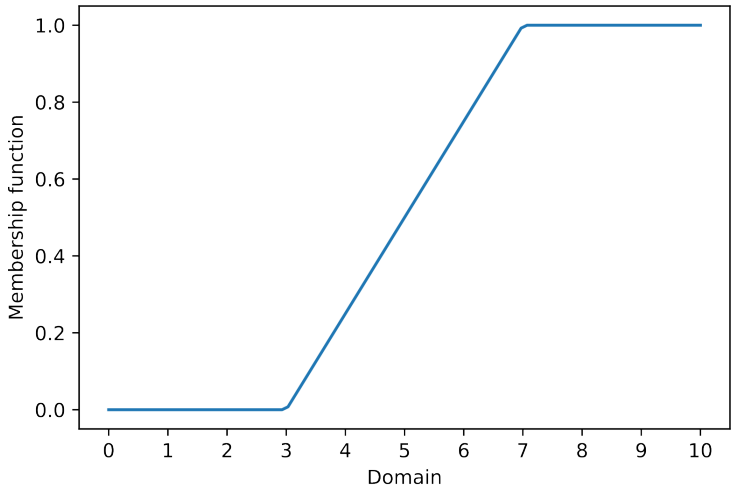


***Figure 17**: Plot with the membership function for a s-shape: 0/3-1/7-1/9*

  - *triangular shape*: the first and the last elements have a degree of membership of 0 meanwhile the second has a 1. *Example*: 0/3-1/5-0/7 (see Figure 18)

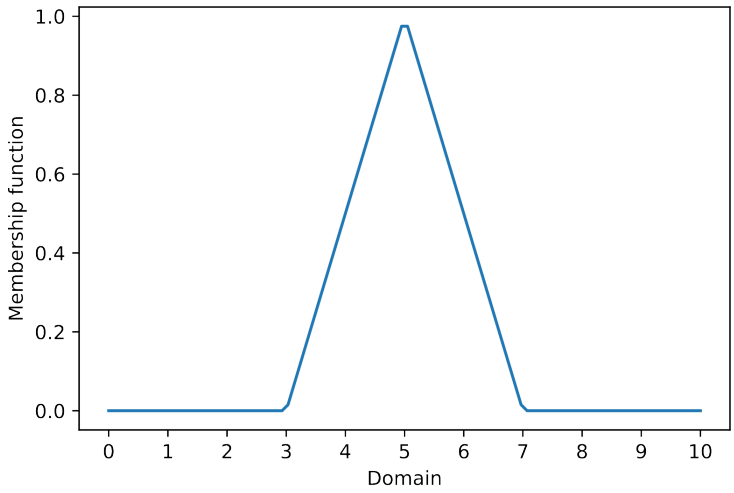


***Figure 18**: Plot with the membership function for a triangular shape: 0/3-1/5-0/7*

- With four elements:
  - *trapezoidal shape*. The two first elements define the first slope and the other two define the second slope. *0/3-1/4-1/6-0/7*

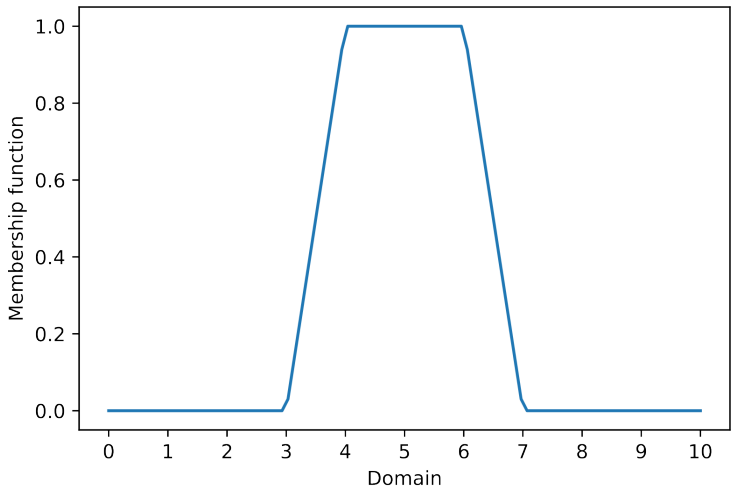


***Figure 19**: Plot with the membership function for a trapezoidal shape 0/3-1/4-1/6-0/7*