Supervised and Experiential Learning

Practical Work 3 - PCBR
CBR System for a Synthetic Task

Víctor Badenas Crespo
Michael Birkholz
Andrea Masi
Kevin David Rosales Santana

Master in Artificial Intelligence (MAI)

June 2021

Contents

Chapter 1

Introduction

The goal of this project is to develop a prototype case-based reasoning (CBR) system in order to solve a synthetic task. A synthetic task, in contrast with an analytical task, is a task whose solution is made up of an aggregation of various components, rather than simply chosen from a list or estimated via regression. A CBR system operates on the principles that similar problems have similar solutions and that a solution that worked in the past is likely to work again in the future. Moreover, it is capable of adapting one or more solutions in order to meet the requirements of its current problem.

The problem our team has chosen to address was inspired by friends and family members who, upon learning that the word "computer" is in our job title or major, immediately proceed to ask us for one of two things: a) "My computer is broken. Can you fix it?" or b) "I need to buy a new computer. Which one should I get?" The first question lends itself to a more analytical solution, but the second question is a great candidate for a synthetic task since it involves constructing a solution out of a set of components, where each component may have many possible values. As this is just a prototype system, we have placed some limitations on the number of values that may be selected for each component, but our approach is generic enough to work with any number of values. In spite of this, our system is capable of generating a very large number of valid PC configurations, even some that contain components that are not present in any configuration in the case library.

Let us present to you the PCBR system, a system that can recommend a PC configuration based on your responses to a few simple questions. The prototype system that we have developed can, when given a user profile including experience level, preferences, budget, typical applications and additional hard constraints, recommend a suitable PC for purchase and an approximate price estimate. It stores a sparse case library using a flat structure. Retrieval is performed using a weighted k-Nearest Neighbors approach, taking user preferences into account to derive the weights. Reuse of the cases is a multi-phased approach. First, weighted adaptation among the closest neighbors is used to provide a starting point. Then, the case is further customized using the constraints specifically requested by the user and some rules based on domain knowledge. After that, an expert user is required to review the recommendation and revise it if necessary. A case is only retained if it is unique enough from other cases in order to limit the growth of the case base library.

Chapter 2

Requirements Analysis

This section is dedicated to the requirements, their justification and verification.

2.1 Requirements

This introductory section is informational only, i.e., no statement in it constitutes a requirement. It describes some use cases for the system that were used to aid in brainstorming and initial development of the requirements.

The PCBR system is intended to help a user figure out what type of computer system they need based on information they supply to the system through a survey-like questionnaire about their needs and preferences. Using this information, the PCBR system attempts to match their case to similar cases it has in its memory. The solution proposed likely needs to be adapted in order to fit the precise needs of a user. If a customized solution is deemed successful via user feedback, it may be learned by the system and reused when a user with similar needs uses the system.

A possible eventual consumer of this system could be a webpage that relies on this system as a back-end. The initial page that gathers user information must load very quickly as users get frustrated when webpages take longer than 3 seconds to load [1]. However, possibly due to the sunken cost fallacy, after the user has invested some time interacting with the page, they likely have slightly more patience waiting for an answer [2]. Therefore, the second step of actually generating a recommendation may be able to be slightly longer. In the short term (i.e., the duration of this project), the cases are all indexed locally and the solution is simply some customization of one or more of these cases, modified according to a set of rules. However, if this were to evolve past the prototype phase into a fully-functional product, one could envision a system that attempted to validate a potential solution by contacting various websites to locate parts and/or customize a PC through a brand's website. In this latter case, we would expect the user to tolerate a much longer wait, similar to what we experience on travel aggregation sites such as Orbitz.

2.1.1 Functional Requirements

The following are the functional requirements[1] for PCBR:

- **F1:** PCBR shall provide an interface[2] for users to specify their characteristics (casual user, gamer, office worker, etc.). *Motivation:* In order to recommend an appropriate system, PCBR must have information about the user for whom it is creating a recommendation.
- **F2:** PCBR shall provide an interface for users to specify their preferences (priorities, budget, etc.). *Motivation:* The user should be able to indicate which items they care about more than others and how much.
- **F3:** PCBR shall provide basic input fields for non-technical users. *Motivation:* Not all users will be able to provide an informed choice about all answers.
- **F4:** PCBR shall provide advanced input fields for technical users. *Motivation:* Some more technical users may have specific preferences/needs than basic users and should be offered a way to communicate them.
- **F5:** PCBR shall display a recommendation (result) in a human-readable format. *Motivation:* If the user cannot understand the recommendation, it is not useful.
- **F6:** PCBR shall allow an expert (oracle) to customize the solution before proposing it for retention. *Motivation:* We would like to allow the system to learn new rules from a human expert.

2.1.2 Technical Requirements

- **T1:** PCBR shall maintain a database of successful solutions and the profiles of the users who they served. *Motivation:* This system should be able to learn from successful solutions employed in the past and improve itself incrementally, ideally increasing the diversity of solutions offered in the process.
- **T2:** PCBR shall be capable of looking up the past input most similar to the current input and retrieving the recommended solution to that problem. *Motivation:* As part of the Retrieve step, the most similar case or cases to the current problem definition should be retrieved from the case base.
- **T3:** PCBR shall customize the result based on similar solutions, the user's profile and the user's preferences. *Motivation:* In order to more closely align with a user's situation, recommendations are not simply retrieved from the library, but rather customized.
- **T4:** PCBR shall support a custom set of internal rules to constrain solutions (to support things like hardware compatibility, etc.) *Motivation:* There are two types of constraints that need to be enforced: user constraints (like budget/CPU brand preference) and technical constraints (for example, an AMD CPU must have a discrete graphics card instead of integrated graphics).

[1]Note: All requirements are denoted by the word **shall.** Any other modifier (e.g., must, should, etc.) does not constitute a binding requirement and is for informational purposes only.

[2]The definition of "interface" is intentionally left ambiguous in the requirements since the determination of a GUI, CLI, etc., is a design decision.

- **T5:** PCBR shall produce a recommendation within 5 seconds of a request. *Motivation:* This system is intended for use in an application similar to a web back-end, so a relatively short response time is expected.
- **T6:** PCBR shall run on a standard desktop/laptop PC with a reasonable amount of memory (8 or 16 GB). *Motivation:* We should constrain this problem to a hardware configuration that each developer has readily available.
- **T7:** PCBR shall have a mechanism to control the growth rate of its case library. *Motivation:* In order to keep the search times low and the memory usage limited, the case library should be limited in how much it can grow. Additionally, this prevents it from over-specializing its recommendations.

2.2 Verification Plan

Table 2.1 contains the verification plan for our product. Traceability to the requirements is found in the *Requirements* column.

Test ID	Procedure	Method (I/D/T/A)[3]	P/F[4]	Requirements
1	After launching the application, open the *Basic Tab* and ensure that options to specify a user profile are present. Then, open the *Preferences Tab* and ensure that options to define preferences are present. Finally, open the *Advanced Tab* and ensure that there are options that an advanced user may supply but also have options appropriate for a basic user (e.g., "No Preference")	D	P	F1, F2, F3, F4, T6
2	Submit the complete set of profile/preferences, click the "Done" button and ensure that human-readable recommendation is displayed	D	P	F5
3	Once the recommendation is presented, choose the option to customize it and ensure you are able to change a/some component/s before proceeding to the next input	D	P	F6
4	Confirm that a new database file is written upon exit of the application and that it contains information regarding users, preferences and solutions	D	P	T1
5	Inspect the source code and confirm that in the retrieval step, k-Nearest Neighbors looks up the nearest neighbors, using user profile/preference (i.e., problem description) information	I	P	T2
6	First, inspect source code and confirm that the solution is further customized in the Reuse step beyond simply looking up the solution. Expect to see a weighted adaptation approach. Then, inspect source code and ensure that there are custom rules applied to the solution after the weighted adaptation	I	P	T3, T4
7	Run the generator which generates random requests for a large number of iterations and plot the time taken for each request. Additionally, plot the number of cases stored over time. Ensure that the number of cases stored is smaller than the total number of iterations and that it approaches a maximum value. Additionally, when it is close to the maximum value, note the time taken per request and ensure that each request is under 5 s.	D, A	P	T5, T7

Table 2.1: Verification Plan

The graphs of the results of Test 7 are presented in Figures 2.1 and 2.2. The case base stops growing short of 400 cases and the total time for retrieve-reuse-retain for each case is well under 1 s (requirement: 5 s).

[3] **[I/D/T/A]** = Inspection/Demonstration/Test/Analysis
[4] **[P/F]** = Pass/Fail

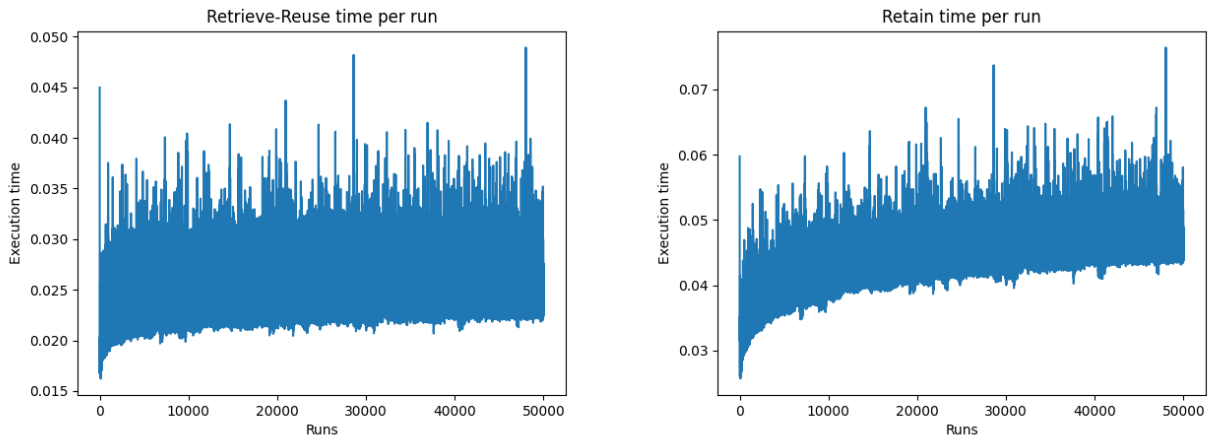

Figure 2.1: Timing results of test 8. The sum of these two times is under 5s each iteration.

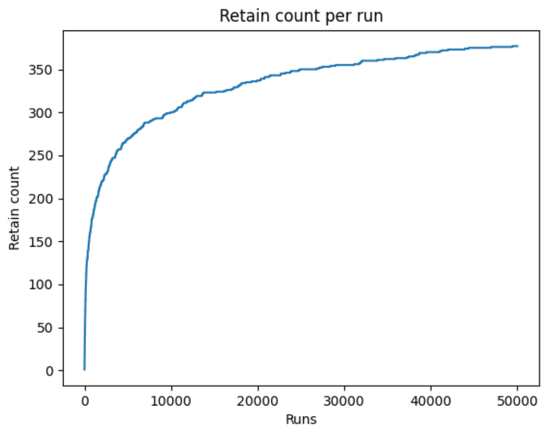

Figure 2.2: Retention count for test 8.

Chapter 3

Architecture

3.1 User Interaction

For the ease of use of the software, a Graphical User Interface or GUI was developed. The GUI was developed using the PyQt5 [3] and human-centered design concepts. The main goal of this GUI is for the user to have an easy and intuitive interface to interact with the software. Entering data in the system is one of the most unintuitive parts of the CLI interface, making use of GUI elements such as radio buttons, labels and text fields, the data entry stage of the data flow has been converted to a more intuitive procedure for the user to do. Additionally, the "yes" and "no" questions for which the user had to write manually "y", "yes", "n", or "no" have now been replaced by buttons, which makes the interaction with the software easier and less prone to typing errors. The advanced controls were programmed to default to neutral values in order to prevent a novice user from even having to visit the Advanced tab.

Using PyQt's capabilities, a table display has also been used to show the user instances and results in a table format as humans are more used to read that format.

There are exactly two points in the process where a user must interact with the system. First, prior to the retrieval phase, they must provide input about who they are and what they want. Soon after, they will receive a response. At that point, the second interaction occurs when the system presents the recommendation to the user. The user (or an expert) must evaluate the case and decide whether it is acceptable, needs modification or is unacceptable in order for the PCBR system to learn from the experience.

3.2 Major Components and Data Flow

As with any CBR system, this system has the typical set of four R's:

- **Retrieve:** Fetches the nearest case(s) from the case base library
- **Reuse:** Adapts the cases to a solution that conforms to the user's preferences and constraints
- **Revise:** An expert user's opinion is solicited in order to affirm or correct the recommended solution
- **Retain:** The solution is stored for future use if it is deemed necessary

In addition to the four R's, we must not forget the case library. The case library stores a set of cases that are split into a set of features that represent the problem (i.e., a user's profile and preferences) as well as a set of features representing the solution (i.e., the PC configuration suggested to the user). A flat case library structure is used since the number of cases is intended to be quite small and ultimately limited in size (enforced by the relevance measure for storing new cases), so the tradeoffs between code complexity and performance benefit do not justify a more complicated approach to storing the cases. Many more details about each of the aforementioned items may be found in Section 4.

In addition to these components, there is also a need for a user interface. The user interacts with the system at three specific points:

- **Request phase:** The system solicits information from the user about who they are and what they care about.
- **Result phase:** The system displays the results to the user.
- **Revision phase:** The system requests feedback from an expert user (oracle) to correct any components in the solution that are not adequate.

There is some data that must flow throughout the entire process, such as the user request, which consists of the user's profile, preferences and constraints. The case library is only consulted during the retrieval and retention phases. Table 3.1 lists the inputs and outputs of each of the major components of the system. Figure 3.1 depicts the architecture of the system graphically, showing the major components, data flow and points where the user must interact with the system.

Retrieve	**Inputs:** User profile, user preferences, case library
	Outputs: Solutions of the three closest cases and distances from input case
Reuse	**Inputs:** User preferences, constraints, solutions of three closest cases and distances from input case
	Outputs: Adapted solution
Revise	**Inputs:** Adapted solution
	Outputs: Expert-modified/approved solution or nothing (do not retain solution)
Retain	**Inputs:** Revised solution, user profile, user preferences, case library
	Outputs: May write case to case library

Table 3.1: Input and output for each CBR component

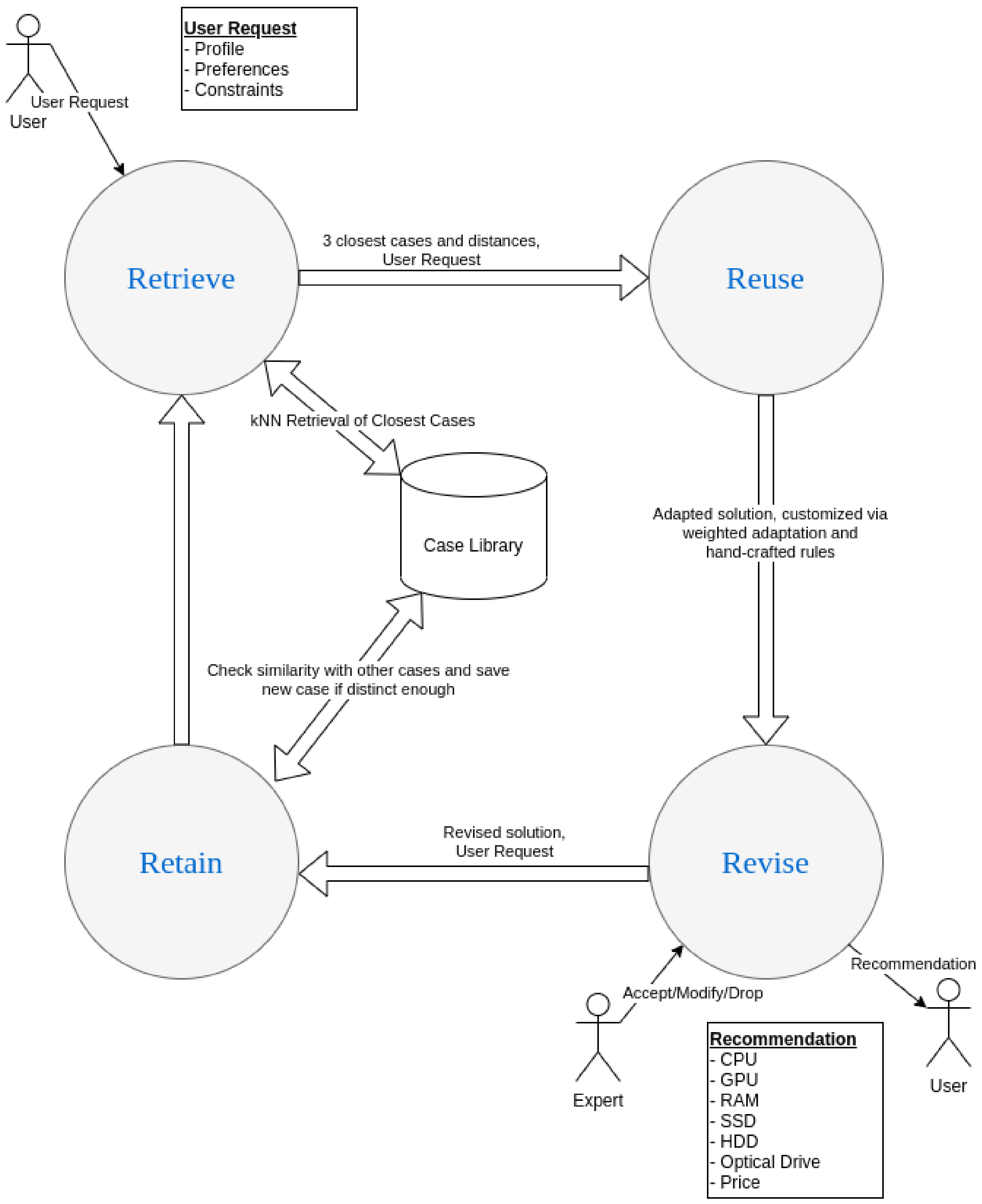

Figure 3.1: Architecture and data flow among components.

Chapter 4

Design and Verification

As in any CBR system, this system implements the four (plus one) R's: (Represent), Retrieve, Reuse, Revise and Retain. The implementation details of each follow.

4.1 Case Representation

Cases are represented by two distinct parts. First, there is a portion that corresponds to a user profile, preferences, etc., which represents the problem. The second part, which corresponds to a particular PC configuration, is the solution that was created for that particular user's case. It is very important to keep track of this distinction at all steps of the process and to have in mind which portion should be used for which step. For instance, when deciding which cases match the current problem, one should compare to the user preferences portion of the cases.

Each feature is converted to numerical attributes of unit range, however, the processing done differs on the real-world representation of that feature. The *CPU* and *GPU* string fields are replaced with their corresponding numerical benchmark value. For `boolean` values, they are converted to `int` where `True` is converted to 1 and `False` to 0. Finally, the *Primary use* field will be changed by numerical integer values representing the strings in 4.1. Once all the values are numerical, fields representing capacities such as RAM, SSD and HDD will be transformed using a log_2 transform so that the distance between different elements is coherent. For example, the distance between 8GB and 16GB should be the same as 16GB and 32GB of RAM. Finally all values are normalized using a *MinMaxScaler*[4] object to be in the range between [0, 1].

4.2 Retrieve

The objective behind the retrieve stage is to search the case library for instances with similar input features as the ones obtained in a request. We will refer to the new instance obtained in the request as x_{new}. When x_{new} is received, the first step on the data flow of the algorithm is the Retrieve step. The Retrieve step is based on an alternate implementation of the traditional k-Nearest Neighbors (KNN) where the voting step is omitted and the results of the different neighbors are not aggregated to return a unique prediction for each instance. Instead of that, k neighbors can be retrieved. In the case of our implementation, we set the number of neighbors to be $k = 3$. The input features of the KNN algorithm are the source features defined in 4.1.

Attribute	Type	Description	Values
ID	int	Integer case identification. Used as key to link a case to its solution.	int
		Target attributes	
CPU	string	CPU Model identifier string. This attribute will be replaced by its corresponding benchmark. This will help compute the distance between CPUs fairly.	CPU models defined in foreign table
RAM	float	Number of GB of System RAM in the computer.	{8, 16, 32, 64, 128}
SSD	float	Number of GB in the Solid State Drive.	{0, 250, 500, 1000, 2000, 4000}
HDD	float	Number of GB in the Hard Disk Drive.	{0, 1000, 2000, 4000}
GPU	string	GPU Model identifier string. This attribute will be replaced by its corresponding benchmark. This will help compute the distance between GPUs fairly.	GPU models defined in foreign table
Optical Drive	bool	Boolean option for including an optical drive or not. It will be represented as int format where True = 1 and False = 0.	{0, 1}
Price	float	Price in € for the components in the instance. Float value ranging from 0 to 10000 inclusive.	[0, 10000]
		Source attributes	
Experience	int	Expertise of the user in computer building and knowledge of computer selection.	{1,2,3,4,5}
WFH	bool	Whether the user is or not working from home. This attribute will be translated to integer range {0, 1} for easy computation of distances.	{0, 1}
Primary use	string	String contaning the main purpose that the user is going to give to the pc. The possibilities are sorted by perceived similarity.	{Home, Work, Production, Programming, ML, Gaming}
Budget	int	An integer representation of the budget of the user. Low, Medium and High encoded as {1, 2, 3} respectively.	{1, 2, 3}
Replace	int	Frequency with which the user replaces computers. {1, 2, 3, 4} represent {1-2, 2-5, 5-7, 7-10} years respectively.	{1, 2, 3, 4}
Office	bool	Whether the user is a user of Microsoft Office or not. Represented as int values.	{0, 1}
Photoshop	bool	Whether the user is a user of Photoshop or not. Represented as int values.	{0, 1}
VideoChat	bool	Whether the user is a user of VideoChat applications or not. Represented as int values.	{0, 1}
ML	bool	Whether the user is a Machine Learning programmer or not. Represented as int values.	{0, 1}
Compilers	bool	Whether the user is a software engineer using compilers such as GCC or not.	{0, 1}
HighPerformanceGames	bool	Whether the user plays games with high computational requirements or not.	{0, 1}
LowPerformanceGames	bool	Whether the user plays games with low computational requirements or not.	{0, 1}

Table 4.1: Feature type and explanations

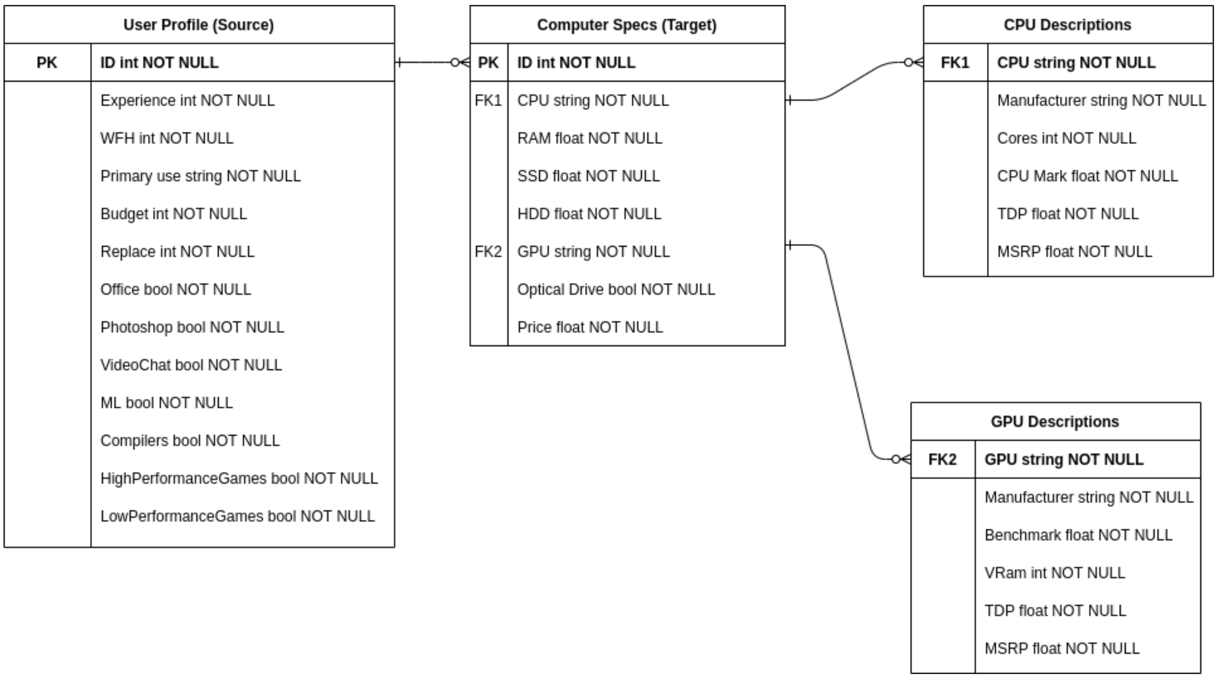

Figure 4.1: Dataset Relations and fields

The KNN algorithm was implemented accepting arrays as target attributes. Using this implementation, once the instances that minimize the Minkowski distance function are found,

$$I_{closest} = \underset{i}{\operatorname{argmin}}\ d(x_{new}, x_i)$$

$$d(x, y) = \frac{\sum_{k=1}^{K} \omega_k * |x_k - y_k|}{\sum_{k=1}^{K} \omega_k}$$

where x_i are the instances in the case library, the top k closest indexes are extracted and the labels corresponding to those instances are returned. Finally, a weight array is defined in order to provide the system of adaptability of the most relevant features to the user depending on their preference input.

In the previous equations, the term ω was introduced. The ω_k value denotes the weight of for the feature k. The weight vector ω for the features used in the KNN is extracted from a vector of preferences p entered to the system by the user. Then a matrix H is defined containing the contribution of each of the preferences to the weight of each of the features. For M features and N preferences, the H matrix is defined as:

$$H = \begin{pmatrix} h_{00} & h_{10} & \cdots & h_{N0} \\ h_{01} & h_{11} & \cdots & h_{N1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{0M} & h_{1M} & \cdots & h_{NM} \end{pmatrix}; h_{ij} \in [-1, 1]; \sum_{j=0}^{M} h_{ij} = 1 \tag{4.1}$$

Or in other words, the contribution of each of the features is required to sum 1 and each of the preferences can contribute positively or negatively to the weight of a feature. Finally, the ω values are obtained through a simple matricial operation $\omega = p \cdot H$. Finally, as we define $p \in [0, 1]$ from this operation, we can see that if $p \in [0, 1]$ and $h_{ij} \in [-1, 1]$ with unit sum for all columns of the matrix H, $\omega \in [-1, 1]$ which will be scaled linearly onto a range $[0, 1]$.

4.3 Reuse

In this synthetic task, new solutions must be proposed. For that reason, the kNN should not just return the most similar case solution to the problem. Therefore, a weighted-adaptation is made so as to create new ones. First, the similarity is obtained using the distances from the user profile (see *User Profile (Source)* in Figure 4.1) to the 3 nearest neighbors (i.e., inputs of the cases in the CBL):

$$sim(x_{new}, y) = \frac{1}{d(x_{new}, y) + 0.1} \tag{4.2}$$

It must be recalled that $d(x, y)$ takes into account the aforementioned explained w_k while computing its distance. After that, the distance is normalized so that $sim(x_{new}, y) \in [0, 1]$ and $\sum_{i=0}^{2} sim(x_{new}, y_i) = 1$. If the input matches exactly one case from the library, then $sim(x_{new}, y_i) \approx 1$.

Finally, the adapted solution (see Figure 4.2) is computed by a weighted average of the different target attributes (see *Computer Specs (Target)* in Figure 4.1) from the nearest neighbors taking the similarity as the importance of each solution to the final adaptation. Consequently, for each target attribute f of the adapted solution:

$$adaptedSolution(f) = \sum_{i=0}^{2} sim(x_{new}, y_i) * y_i(f) \tag{4.3}$$

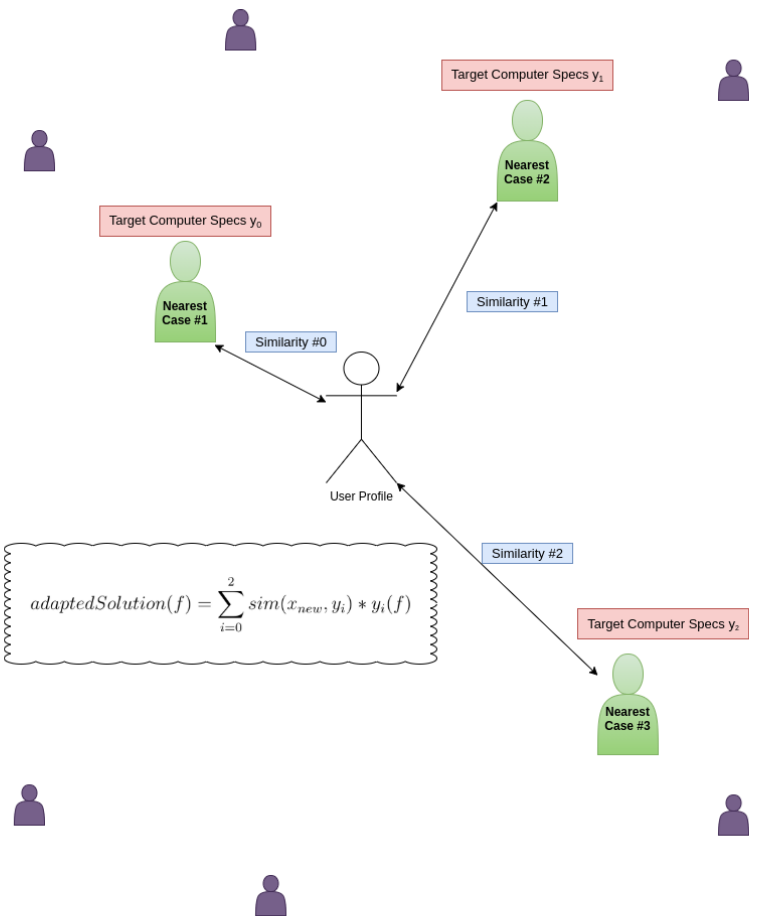
Target Computer Specs y_1
Nearest Case #2
Target Computer Specs y_0
Nearest Case #1
Similarity #1
Similarity #0
User Profile
Similarity #2
$adaptedSolution(f) = \sum_{i=0}^{2} sim(x_{new}, y_i) * y_i(f)$
Target Computer Specs y_2
Nearest Case #3

Figure 4.2: Weighted Adaptation

After the weighted adaptation is complete, each attribute will lie somewhere in the continuous range of [0,1], but most attributes only have a discrete set of values in this range that they may take on. So, at this point, each attribute of the adapted solution is analyzed discretized, or “snapped” to the nearest valid value, yielding valid components.

The next step is to verify that user constraints have been met. This is a very tricky proposition since it is necessarily a multi-step process. We should take care that a later step does not undo a previous adaptation applied to satisfy a constraint. The first step is to create a set of component tables. The highest-priority table contains only user-preferred components. An alternate table includes those preferred components, plus any other acceptable (albeit not first-choice) components. For instance, if the user specifies that they prefer an AMD CPU, the first table contains all AMD CPUs that meet a certain performance benchmark and the second table additionally includes Intel CPUs since the constraint is only a preference, not a hard constraint. During this table construction step, we additionally analyze the user preferences and, through a simple voting scheme, we rank them from most to least important and store them in order in a table.

After these tables are constructed, the hand-crafted rules are applied to the candidate solution. The first rule is very simple and adds an optical drive if one is desired (or removes one if there is one but it was not requested). Next, there are two passes through the lists of priorities to customize the remaining hardware, where the first pass goes from most important to least important and the second pass goes from least important to most important. This is done because there is a set of interchangeable storage (SSD and HDD) where the higher-priority item needs to be populated first. However, in the case of the other components, it is convenient to run the highest-priority rule last so it can override any rule that ran before it.

At this point, the solution should be nearly complete. There is a quick check of a fixed rule to make sure an AMD processor is always accompanied by a discrete graphics card and the rest of the constraints are verified. Moreover, another important constraint is the storage of the suggested PC, since HDD + SDD should result in a value greater than 0. If any fail, it may result in a warning, but since we have not been concerned with budget up to this point, no unmet constraints are expected other than budget. If the price is over the budget, the system checks the relative priorities of performance vs. budget and if budget is more important, it will try to pick cheaper parts for either the CPU, the GPU or both.

The outcome of this process is the solution that is recommended to the user. Note that it is possible that a solution could be non-compliant in some regard, particularly if the initial set of constraints are impossible to meet simultaneously, and it is up to the expert in the next step to make any necessary corrections.

4.4 Revise

In this step, the user is given the opportunity to check the solution and decide if it meets all of their wishes. We take the approach that the user is an oracle and if they accept the solution as-is, we pass along the solution to the next step. If, on the other hand, the user modifies the case, we pass along the modified case. If the user does not want to use this solution, they may also instruct the system to drop the case.

4.5 Retain

If the case from the Revise step is accepted, the system passes both the inputs from the user and the accepted solution to the Retain step. The similarity of the user's answers is checked against existing cases and if it is determined to be sufficiently unique, the case (both user preferences and solution) are stored in the case library.

In more details, a *relevance measure* approach has been used, aiming to retain only the solutions that are different enough from the ones that are stored in the case library. To do so all user inputs and all solutions of the `PCBR` (from both base cases and retained cases) were transformed into numerical values, then concatenated and fed to a newly implemented `Nearest Neighbor` algorithm to get the distance from the first nearest neighbor of every instance.

Some statistics were extracted from the data, in particular: max, min, mean, standard deviation and some percentiles ('25%', '40%', '50%', '75%', '85%', '95%'). At the same time, the concatenation of the current proposed solution and user's input were fed to the `Nearest Neighbor` to predict once more the distance from its first nearest neighbor. With this data, a study of the best possible threshold was made.

The mean together with the standard deviation was discarded since did not provide a good way to extract a meaningful threshold. Then, the different proposed percentiles were studied and the final threshold was selected at the 40th percentile.

This means that a proposed solution will not be stored if the concatenation of the current proposed solution and user's input have a distance value from its first nearest neighbor that is less or equal to the 40th percentile distance value of the dataset, otherwise it will be stored.

In other words, if the "*problem + solution*" point is at least farther than 40% of points present in the case library from their nearest neighbor then it is unique enough to be stored.

This study has been performed with the help of some plots like the one depicted in Figure 4.3. In the first row, the distances from the nearest neighbor of the case library are shown, where every point is an instance and every vertical line corresponds to a different percentile (the first vertical line on the left is the '25th' percentile, then the '40th' and so on...). The second row is meant to give a visual representation of the data, displaying the 2D PCA of the dataset formed by the concatenation of all user inputs and all solutions.

The columns represent iteration 1 and iteration 3 of our `PCBR` system using random data input. It's fairly easy to understand why in the first and second iteration the solutions have been stored while in the third iteration this won't happen.

Finally, if the proposed solution has passed the threshold test, both the user's input and solution are added to the other cases of the library, in memory as numerical values, while a human interpretable copy of them is stored in two different *.csv* files.

Once the user closes the application, those files will be merged to the base cases and a new *.csv* file will be generated saving all the cases stored in the run, in such a way that it can be loaded and used once again.

(a) Iteration 1: Proposed solution stored. (b) Iteration 3: Proposed solution discarded.

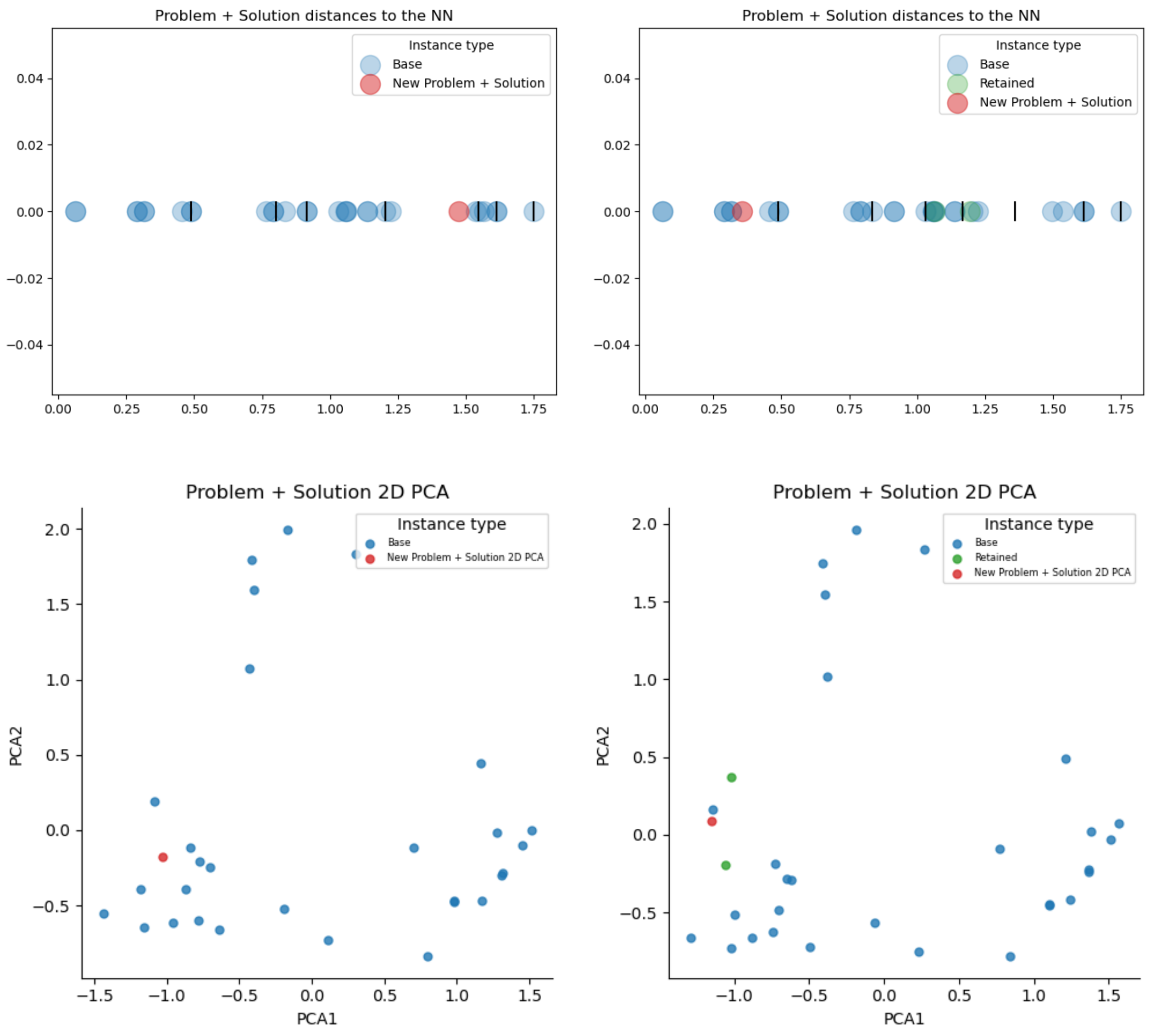

Figure 4.3: Three iterations of the `PCBR` showing "Problem + Solution" distances and PCA.

Chapter 5

Conclusion

5.1 Future Work

During the course of the development of this application, quite a few ideas occurred to us about how to enhance this software in order to add new features or scale up to a viable commercial product.

One of our first ideas was to incorporate BERT [5] in order to allow more natural input from the user. It showed promising results initially, but had problems distinguishing between subtle differences and negation, however. Additionally, it did not provide a mechanism (without full semantic understanding, at least) to enforce constraints. Therefore, we decided upon a survey-like mechanism for user input. However, using BERT seems to be a very simple method to grant the user some extra freedom to answer questions such as, "What tasks do you perform with your computer?" or "What applications do you intend to run?"

There are a number of issues that would need to be addressed in order to scale up to a product suitable as a server back-end. First, this currently runs on a single PC in a single thread and only one user at a time is accessing the database. In order to support a server that can handle possibly tens up to thousands of simultaneous requests, a few steps would need to be taken. First, an access control mechanism to block reads while a write is in progress would be important. Second, the number of writes should be limited or possibly even aggregated and all applied on a schedule, like once an hour, day, etc. If we were to incorporate a feature to search the web for parts and their current market price, instead of MSRP which we used in our implementation, there would likely need to be a caching mechanism for the latest prices in order to both keep results fast and not to get blocked from websites that see our application as attempting a DoS attack.

Hand-crafting adaptation rules is quite tedious and time-consuming. It worked well enough for our simple case, but if we break out every single component, such as motherboards, RAM types, processor package, etc., the hand-crafted rules as they are currently implemented become unsustainable. One idea that may help would be to represent the "solution" portion of the case (i.e., the PC configuration) as a tree-like structure, where only valid combinations of components could be connected. Another point that would help greatly to reduce the complexity from a software perspective would be to take advantage of an object-oriented approach and combine the pre-processed, numeric representations of components with their symbolic representation to avoid converting back and forth constantly.

A final idea that crossed our minds is that we recognize that we have a multivariate optimization problem, which is very difficult to solve optimally. We might be able to kill two birds with one stone by applying an evolutionary algorithm to it, or at least a heuristic search like a beam search. This would serve two purposes. First, it could explore many similar solutions in order to find something that is at least the same or better in all aspects rated as important (budget, performance, etc.). This may be particularly beneficial as new generations of parts are released that can do more at a lower price point than previous generations and the system could learn configurations that incorporated these new components. Second, a less obvious benefit is that the hand-crafted rules may be able to be simplified since they would only need to detect valid/invalid configurations, not attempt to synthesize new valid configurations since that is taken care of by mutation and/or the objective function with a strong penalty on invalid configurations.

5.2 Final Thoughts

Throughout the development of this CBR System, several challenges have been faced as has been explained. Nevertheless, the main objectives have been reached since the aforementioned requirements were satisfied. Therefore, this proposal works as expected, providing an intuitive PC Configuration given a user profile via GUI or CLI. Moreover, as several preprocessing techniques are applied in the retrieval and reuse stage, the user is not required to be an expert.

Furthermore, even if the proposed solution is constructed by using a weighted adaptation and a set of rules to solve constraints, it has demonstrated being sufficiently appropriate, given the size of the problem trying to be solved. On the other hand, the revise and retain stages make sure that the CBL is filled with sensible cases so as to continue providing good-quality solutions in next iterations. The in-depth empirical study that was performed for obtaining the correct retain threshold is one of the most important keys that supports producing these accurate PC Configurations. In other words, retaining the most appropriate cases supports correct learning in human minds and consequently, it is also appropriate for experiential learning. Another consideration in CBR is the number of stored cases in order to resolve requests quickly. Since the upper bound is approximately 400 cases, given the analyzed threshold, the CBR System is able to provide its answer 5 times faster than the initial requirement. This is a very important characteristic of the proposal, since a complete adaptation step is carried out so as to create new complete synthetic solutions that are not present in the CBL. Finally, the main unresolved/partially-resolved problem remaining in this system is the quality-budget multivariate optimization problem. As was mentioned in section 5.1, further techniques such as genetic algorithms should be applied with a previously-studied objective function. Even if it is not actually common in this system, the provided solution may contain a high-quality part combined with low-quality hardware, which usually causes the well-known bottleneck problem in PCs.

In conclusion, this final assignment of *Supervised and Experiential Learning* has brought us insight into how solutions to problems may require extra steps apart from complex classifiers to extract the required knowledge to simulate how an individual learns from experience.

Bibliography

[1] David Kirkpatrick. *Marketing Dive - Google: 53% of mobile users abandon sites that take over 3 seconds to load*. 2016. URL: `https://www.marketingdive.com/news/google-53-of-mobile-users-abandon-sites-that-take-over-3-seconds-to-load/426070/` (visited on 06/02/2021).

[2] Ken Godskind. *SmartBear - 5, 10, 15 seconds? How Long Will You Wait For a Web Page to Load?* 2009. URL: `https://smartbear.com/blog/5-10-15-seconds-how-long-will-you-wait-for-a-web-p/` (visited on 06/02/2021).

[3] Riverbank Computing Limited. *PyQt5 5.15.4*. URL: `https://pypi.org/project/PyQt5/`. (accessed: 15.06.2021).

[4] scikit-learn developers. *sklearn.preprocessing.MinMaxScaler*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`. (accessed: 15.06.2021).

[5] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT*. 2019.

[6] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: `10.5281/zenodo.3509134`. URL: `https://doi.org/10.5281/zenodo.3509134`.

[7] Janet Kolodner. "An introduction to case-based reasoning". In: *Artificial Intelligence Review* 6 (Mar. 1992), pp. 3–34. DOI: `10.1007/BF00155578`.

[8] Agnar Aamodt and Enric Plaza. "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches". In: *AI Commun.* 7.1 (Mar. 1994), pp. 39–59. ISSN: 0921-7126.

Appendix A

Resource Accounting

Who	**Andrea**	**Kevin**	**Mike**	**Víctor**
Meetings	4.5	7	6.5	6
Requirements, Definition	1	8.5	12	9
Representation	0	8	4	8
Retrieval	0	0.5	0	8
Reuse	0	10	16	1
Revise	16	0	1	1
Retain	26	0	2	0
GUI	1	17	1	17
Report	5	7	15	8
Presentation	5	2.5	4	3

Table A.1: Hours per team member per task