NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4173 - Machine Learning

Case-Based Reasoning Methods

Authors:
Olav Røed Meberg
Fridtjof Høyer

Thursday 15th October, 2020

Abstract

This paper will discuss advantages of using Case-Base Reasoning to solve complex problems. It will give an overall view on the fundamentals of Case-Base Reasoning, after a more in-depth picture on the retrieval phase of the 4r cycle. The paper will mainly converge the retrieval with the help of k-d trees. Foundation will present an overall presentation on what and how k-d trees work, how to generate and how to retrive cases from a k-d tree. The later section will explore other methods that can be used instead of k-d trees. Lastly a paper that use k-d trees in the retrieval phase will be presented for a better understanding of the usage of this method. We ask readers to consider the appendix [5] for more figures and graphs when referenced in the main article.

Table of Contents

List of Figures

1 Introduction

The fundamental idea behind Case-Base Reasoning (for readers convince; CBR) is to solve complex problems by reusing a relevant portion of previous solved problems. In other words, CBR is a methodology for solving distinctive problems by evaluating previously successful results to “comparable” or similar problems, thereafter, adapting their solutions to fit new requirements. Hence, CBR solve new problems by revisiting known cases describing a similar problem. The “rational thinking” in this artificial intelligence relies solely on the idea that the more similar two distinct problems are, the more similar their solutions will be.

The most familiar approach of applying CBR to an application is to use the 4r cycle seen in figure 1. This approach was created by Aamodt and Plaza in 1994. The 4r cycle consist of the steps **retrieve**, **reuse**, **revise**, and **retain**. The cycle also contains a case-base. The Case-base is previously successful results stored in long-term memory. The first step of the cycle is the retrieve step, this step finds the most similar case in the case-base given a query. When a case from the case-base is retrieved, adaption of the retrieved case must be implemented if necessary. The adaption of the given case happens in the reuse step. The following stage is the revise phase. This step checks if the solution to the query is correct. If so, the solved query can be stored in the case-base, storing the case in the case-base happens in the retain step.

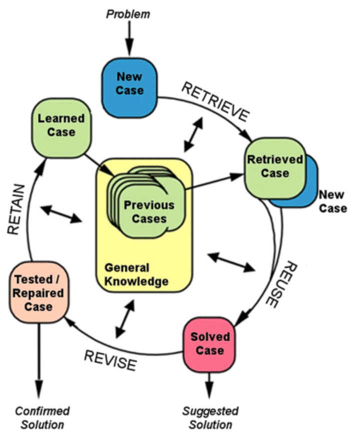

Figure 1: The 4r CBR cycle as proposed by Aamaodt and Plaza

Source: Reuss [2017]

One of the core motivations of implementing the 4r cycle for CBR is to implement new solutions to the case-base. As described if a query is solved it can be added to the case-base if desired. A larger case-base means a greater chance of finding a similar case to a query. Hence, the more cases added throughout the 4r cycle, the bigger the probability is for finding a similar case to a new given query.

Humans tend to think in a rather similar fashion to the CBR methodology. Therefor we would like to invite the reader to join us on a small thought experiment. Imagine yourself as a Grand Master in the game of Chess. You have practiced a lot and played a vast amount of games. In your next game, you play $d4$, your opponent replies with $d5$, ($c4$, $e6$), the Queens Gambit Declined, $nf3$, then $dxc4$. The game has never been played before, but you recognize a very similar position you have seen in another game, and decide to act with $e3$. This move ultimately gave you the victory, and now you always play $e3$ in this position knowing this gives white an advantage.

To get a clear overview over how to approach CBR this paper will have a closer examination of retrieve phase, namely finding a case in the case-base most similar the given query. One of the many methods that can be used in the retrieval procedure is to implement an index structure in the given case-base with the help of k-d trees. The reason for implementing an index-structure is to achieve faster retrieval from the case-base.

2 Foundations

k-d Trees is a way to manage index-based retrieval in CBR, the purpose of this approach is to index the case-base for more accurate, fast, and relevant retrieval. The core fundamentals of index-based retrieval are to first generate an index structure in the given case-base, secondly retrieval from index-base is defined.

k-d Tree (k-dimensional tree) is a space partitioning data structure for organizing points in a k-dimensional space. Say the Case-Base contains a k ordered domain $T_1, T_2, ..., T_k$ for the attributes $A_1, A_2, ..., A_k$. Implementing k-d tree for this case-base can be thought of as implicitly generating a splitting hyperplane for every non leaf node that divides the space into two parts, known as half spaces. In geometry, a hyperplane is a subspace whose dimension is one less than that of its ambient space. By splitting the space into subspace, a k-d tree is a recursively defined binary tree for the case-base. These k-d trees are useful data structures for several applications, such as searches involving multidimensional search keys (e.g. range searches and nearest neighbor searches). The goal of implementing k-d trees in CBR is to find the nearest neighbor to the given query.

If one is familiar with binary space portioning tree there is not much of a difference compared to binary tree. k-d Tree is a binary space partition tree, but with the specific property that all lines of division for a k-d tree must be parallel to the axis. A binary space partition tree, in general, does not need to abide to such restriction.

For the sake of simplicity, generating a k-d tree will be presented in a 2-dimensional space. This implements that we have a case-base containing an ordered domain $T_1, T_2, ..., T_k$ with attributes A_1, A_2. The number of cases that is acceptable for linear retrieval at the end of the procedure is called buckets, defined by b. The definition of a leaf node is therefore given by:

if: $|CB| \leq b$ then tree(CB) is a leaf node, called bucket

Suppose that we have a $CB = \{A(9,1), B(3,4), C(4,6), D(6,5), E(2,3), F(8,7), G(7,9), H(9,6), I(1,2), J(4,5)\}$, these parameters can be seen plotted in figure 2. we set the parameter b, i.e.: The size of the buckets to 3.

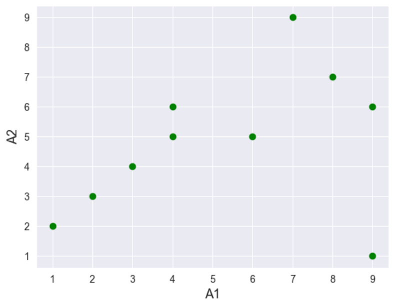

Figure 2: Geometric view of domain

Source: Produced by authors

The first procedure of creating a k-d tree is to choose a value A_i that partitions the domain. At each node one has two degrees of freedom, one can partition the domain either by choosing an attribute or by the choice a value. In our example the split criteria chosen is a value, namely the median.

The k-d tree first uses the median of the first axis and then, in the second layer, the median of the second axis. This can also be implemented for higher dimensions. Let the A_1-axis be the start axis by choice. The ascending sorted A_1-values are: $\{1, 2, 3, 4, 4, 6, 7, 8, 9, 9\}$, trivial the median is 6. Now we divide the datapoints by the A_1-value equal to 5. This divides the datapoints into two separate groups as seen in the appendix, figure 9. Thereafter this procedure continues in the two separated groups, but now we divide the groups with regards to the A_2-axis. The resulting geometric view can be seen in figure 3.

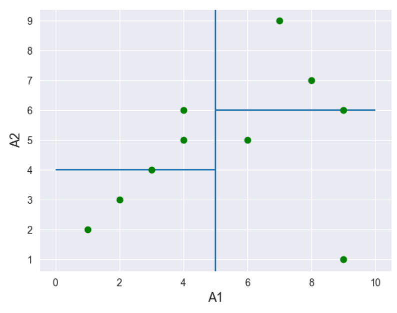

Figure 3: Final spatial seperation of geometric view

Source: Produced by authors

The goal of choosing a value A_i is to minimize the length of the path to the buckets, as well as separating distinct cases high up in the tree. Thus, one chooses the inner nodes in such way to obtain a tree that is most likely balanced. The resulting tree can be seen in figure 4.

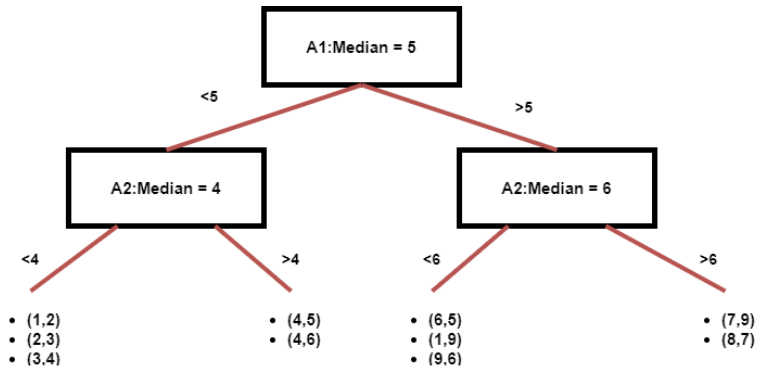

Figure 4: Generated k-d tree

Source: Produced by authorss

Each node has two degrees of freedom, a choice of attribute A, and a choice of a value that split the domain. This can be seen in the pseudocode 1.

There are mainly two ways of defining a splitting point, median splitting and maximum splitting. Median splitting was described in the example above. Maximum splitting selects the value where the biggest gap between attributes occur.

Algorithm 1 Generating k-d tree

```
procedure CREATE-TREE($CB$)
   if $|CB| \leq b$ then
     return leaf node with base $CB$
 else
     $A_i := pick$attribute(CB);
     $v_i := pick$value(CB,A$_i$);
     Return:
     Tree with root labelled with $A_i$ and $v_i$ that has two subtrees
     CreateTree($\{x_1, ..., x_k \in$CB|x$_i \leq$v$_i$ $\}$) and
     CreateTree($\{x_1, ..., x_k \in$CB|x$_i > v_i$ $\}$)
```

A typical method for attribute selection where the domain is real-valued uses the one that has the largest interquartile distance. Interquartile distance divides the domain into a lower half of the distribution containing 25 % quartile and an upper bound which contains 75 % of the quartile. The greater the distance of these quartiles the more sizeable the dispersion of the attribute values is selected. Applying to tree construction, the attribute most scattered to the local similarity measure is used. It denotes that we select an attribute for discriminating purposes where the respective quartiles have the lowest local similarities, this correlates to the greatest distance. Richer and Weber [2013]

Until now we have created rectangle representations of the domain, as seen in figure 3. The attribute most suitable to the query may not be on the tree, but rather somewhere in the real-value geometric view, that is no longer considered. The constructions of trees described are considering hyper-rectangles, while the similarity considers hyper-balls. The relationship between these two situations can be seen in figure 11. The optimal situation is that the hyper-balls are within the hyper-rectangles for finding the most suitable query in the retrieval. To check if the hyper-balls is within the hyper-rectangle as seen in figure 11 **(a)** or if the ball overlap the bounds as seen in figure 11 **(b)**, two tests are presented:

1. The BWB test: is the ball within the bounds?
2. The BOB test: Does the ball overlap the bounds?

In figure 12 the BWB test is visualized. As seen a bounding box of the hyper-circle is created, this is a box is used for checking if the hyper-circle is within bounds. Using a bounding box instead of looking directly at the hyper-circle makes this procedure more facile. The reason for implementing a BWB test is to be sure that there is no attribute A in the neighboring subtree that is more like the query than the found A most similar case. For the BOB test the same procedure of investigation is performed. If a more similar case is found within a neighbor bucket the ball size of the query is decreased. In figure 10 one can see an example of how a query can have a closer neighbour in the neighbouring subtree given the same geoemtric view earlier described.

After making the k-d tree the actual retrieval phase in the 4r cycle can be implemented. Let us say we have a query point q witch we want to find the nearest neighbor to. By using the tree, one can traverse to the correct leaf node. While in an inner node the BOB test can be implemented to check if it is necessary to search other subtrees. When arriving to a leaf node, also called bucket, the BWB test check if the search terminates or backtracking must take place. In the appendix the psudocode for retrival is presented, this can be seen in psudocode 3.

In section 3 the properties of creating a k-d trees, such as runningtime, advantages and disavdvantages is discussed and summarized.

3 Alternative Metodes

This part of the article will be discussing sequential retrivel and two-level Retrieval as alternative methods to approach earlier described in foundations 2. Please note that section will emphasise the retrieval step of in the 4r cycle, solely. The paper will present sequential retrieval and two-level retrieval and compare the advantages and challenges they present with respect to the k-d tree.

3.1 Sequantial Retrival

The first method to be examined is the basic algorithm, sequential retrieval. In short, sequential retrieval calculates the similarity of the query and all other cases in the case-base to retrieve the best match based on similar cases.

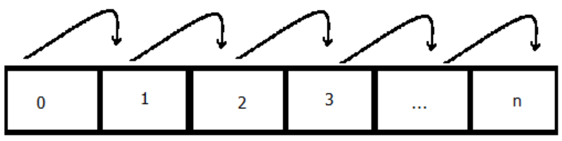

Figure 5: Sequential retrieval in an array

Source: Produced by authors

This method applies a rather naïve approach and uses brute force to find the k-nearest neighbour with the qualities needed for the query. The paper includes a suitable pseudocode 2 seen bellow to help readers understand the fundamental principles of this action in the 4r cycle. The algorithm will return a list of size k containing the most similar cases to our query given a similarity function. It is important to note that the similarity function is dependent on the query and case-base. Every case in the case-base of size n is iterated over. Hence, the runtime for sequential retrieval is $\mathcal{O}(n)$ (BC = WC) .

Algorithm 2 Sequential Retrivel

```
procedure SequentialRetrive(Object $q$, Integer $k$, List $CB$, SM $sim$)
    result = [ ]                                   ▷ Empty list containing results
    for case $\in CB$ do
     sim$c_i$ = sim$(q, c_i)$                      ▷ Check similarity with given similarity function
     result = result $\bigcup x(simc_i,\ c_i)$
     result = $sort$(result)
     result = $prune$(result, $k$)
     return result
```

As promised the advantages and disadvantages will be described more in detail now. The big upside of this school of thought is that sequential retrieval is easy to impenitent compared to more advanced algorithms, where there is a need for an index structure. This method also works for all similarities. However, upsides usually come with complementary disadvantages as a trade of. Sequential retrieval has as earlier described a run time of $\mathcal{O}(n)$ since the algorithm must visit every single case in the case-base. Thus, the method has some problems when the case-base get large. Richer and Weber [2013]

3.2 Two-Level Retrieval

The next method to examine can be used in the retrieval phase of the two-level retrieval method. The idea of this method is to use a two ways procedure called: Many are Called, and Few are Chosen (MAC FAC, respectively). This procedure gives the opportunity to not have a complex similarity computation for every case in the case-base. The first step of this retrieval is the MAC procedure. This step will select a subspace of the domain, of the case-base by a certain predicate SIM. Initial step for candidate picks $C_q \subseteq CB$ by:

$$C_q := \{p \in CB \mid SIM(q, p)\}$$

It is important to mention that SIM is not a measure, but return a binary predicate (rational retrieval). In this procedure it is important to select enough, but not too many cases given this SIM predicate. The most important thing is to not exclude relevant cases in this procedure (not many objects p close to q query) Richer and Weber [2013], which contributes to a smaller error. One way to do this can be to only look at one important attribute and select cases regarding the attribute that satisfies a given demand.

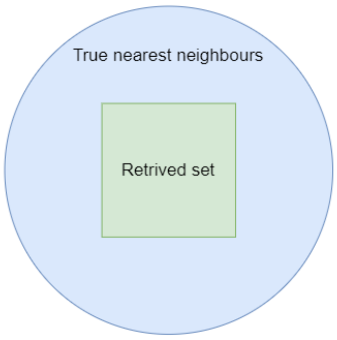

Figure 6: Shows a representation of how error that can occur during the MAC procedure. The rectangle describes the retrieved set and the circle describes the true nearest neighbours. the α-error witch is showed in the figure gives a visual representation of how a case p witch is similar to the query q is not selected

Source: Produced by authors

In the MAC stage of the prosses is a rather simple, computationally cheap match prosses. Michael M. Richter and Rosina O. Weber describe this as a "wide- net" in Case-Based Reasoning Richer and Weber [2013]. The processes take place between the query case in the working memory (RAM) and long term stored memory. From this a small number accurate and inaccurate cases continues. The FAC stage take use of simple matches in parallel on each of the results obtained in the MAC phase to select the most suitable match(es) (several cases if sufficient to match criteria) Richer and Weber [2013].

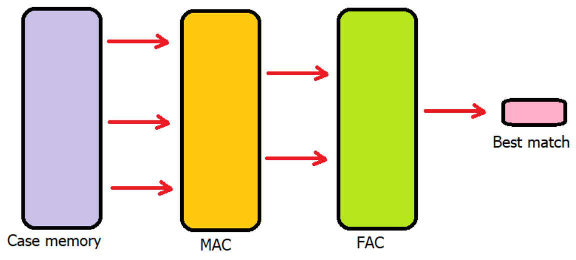

Figure 7: Simplified phases of the process

Source: Produced by authors

Let us showcase this with an example inspired by Case-Based Reasoning by Michael M. Richter and Rosina O. Weber Richer and Weber [2013] on how to predicate SIM.
Partial equality:

$$SIM(q, p) \Leftrightarrow q \wedge p, \text{ agree on at least one attribute.}$$

Local similarity:

$$SIM(q, p) \Leftrightarrow q \wedge p, \text{ sufficiently similar for all attributes. Requires a defined threshold } \theta \text{ and providing flexibility.}$$

Partial local similarity:

$$SIM(q, p) \Leftrightarrow q \wedge p, \text{ sufficiently similar for one attribute. Ok for threshold } \theta \text{ and attribute.}$$

The advanteges of two-level retrieval are hence a greater perfomance, the proposed approach significantly reduces execution time for larger case bases Richer and Weber [2013]. This is becuses only a few cases are selcted redusing the computaion needed. However by doing so, a clear disatvanteges is that this may harm the accuracy somewhatRicher and Weber [2013].

3.3 Methods Compared to k-d Trees

Advantages and disadvantages of using k-d trees compared to the two methods that has been discussed, namely sequential retrieval and the two-level retrieval method are due for compersion. In the foundation 2 the paper states a general aspect of how k-d trees functions, however an overall comparison is needed.

One of the main ideas with different methods for the retrieval step in CBR is that different case-bases need purpose-built methods. How the case-base is structured will deem what algorithm is preferred. Examples of case-base representations may be image-, object-oriented-, textual- and attribute representation. The algorithms in question, including k-d trees uses attribute representation. For doing retrieval in other case-base representations new methods need to be considered.

k-d Trees has a boundary that the case-base needs to contain an ordered domain, to be able to implement an index-base representation. This is not the case for sequential retrieval and two-Level Retrieval. Another disadvantage of building a k-d tree is that it may be labor intensive as is understandable given the description of creating a k-d tree in section 2. The key advantage of implementing k-d tree is to increase the running time of finding a suitable case. The best case-retrieval from a case-base is $\mathcal{O}(log_2(n))$ with n being the number of cases in the case-base. Compared to sequential retring with the best and worst runtime of $\mathcal{O}(n)$. Implementing a k-d tree makes the retrieval considerably more efficient in a case-base containing many cases. Nevertheless, if a k-d tree requires backtracking the worst-case retrieval effort is $\mathcal{O}(n)$, investigating the whole tree. Unfortunately, situation with few backtrackings seldom occur in practice, this restricts its applicability in practice. Two-level retrieval may be of use with an unfortunate unordered domain. The method is trickier to implement but decreases the runtime overall for larger case-bases. We note that the MAC face is the most problematic to implement and must be handled with the upmost care. There are also possible to combine the two alternative methods and apply the sequential algorithm in the FAC phase due to the smaller amount of data need to be processed.

Method	Dataset size	Representation	Error	Best Case	Worst Case	Iimplementation
Sequential	Small	Attribute	Negligible	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Easy
Two-Level	Large	Attribute	α	Depends	$WC \gg BC$	Hard
k-d Tree	Large	Attribute	Negligible	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	Hard

What type of method you want to use in the retrieval phase of the 4r cycle depends on the given problem and the representation of the case-base. Every method has advantages with complementary disadvantages, and it may be a difficult task deciding what type method is more efficient.

4 Towards Case-Based Reasoning with k-d Trees for a Computer Game of Soccer

This section is dedicated to introducing the paper "Towards Case-Based Reasoning with k-d Trees for a Computer Game of Soccer" by Georgii Mola Bogdan and Maxim Mozgovoy from the University of Aizu, Aizuwakamatsu, Japan.

In their application, they are trying to solve the task of making an AI for the game of soccer to compete in 2-dimensional version of soccer, RoboCup. The game tries to accomplish much of the same as FIFA by EA Sports for readers familiar with this game. Every player as well as the ball are represented by a float (coordinate) on a finite field with defined goals and limits as shown in figure 8. The game uses k-d trees on retrieval for the purpose of computing speed. When playing in real-time, the computing time needs to be reduced to a minimum to accomplish somewhat human-like traits.

The dataset contains of five recorded matches, where the opponents are six different Japanese teams. Statical information about the matches, such as team formations etc. is also known and included in the dataset. The games where recorded at a rate of 25 fps. and conveniently represented as a sequence of frames. Any given frame consists of a two-dimensional player – and ball coordinates with vectors of direction. The goal is to investigate how "easy" and computationally possible it is to find a case in the case-base for any giving situation in real time on the playing field with the help of k-d trees. The argument for using k-d trees is the capability to execute searches within a defined range on multiterminal data. While playing, finding coordinates matching exact previous cases is both unreliable and unrealistic, much because of the infinite different positions possible. The k-d tree method allows for sufficient computing speed to search for closest matches for 23 element vectors for the players and the ball $(11 + 11 + 1)$ giving a total of 46 values. This rate of frames where later reduced to approximately 1800 frames per match giving a total case base of roughly 7500 frames and cases. The authors note that that learning from a smaller, limited dataset are an important additional goal.

Appending new elements to a balanced k- d tree takes a run time of $O(log_2(n))$, there n is the numbers of leaves. The complexity of the querying an axis-parallel is $O(n^(1 - 1/k) + m)$ where m is the reported number points and k is the dimension ($k = 46$ in this case). The paper in question states that k-d tree is unable to provide any long-term planning for future actions, however, mentions that this drawback is noncrucial for the game of football, for the purpose virtual Robocup.

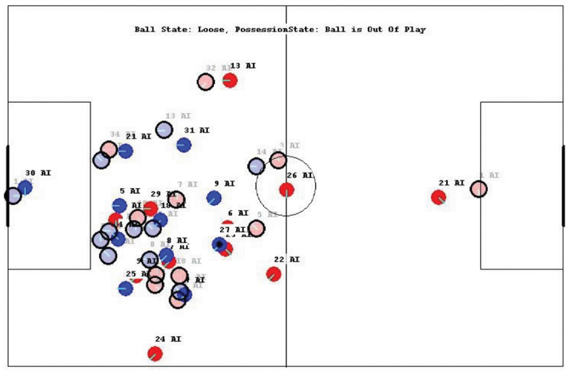

Figure 8: Situations matched as a result of a global query. An example of how the paper uses k-d trees. It shows a overlapped images of query situations and the corresponding retrieved matching results for a global search procedure with a range of 8 meters.

Source: Georgii Mola Bogdan and Maxim Mozgovoy

The paper concludes that a k-d tree approach for match retrieval is feasible in terms of computational performance and ability to find sufficient similarities in the case-base, though the dataset for this specific application was too small to be used reliable for decision making purposes.

Mola Bogdan and Mozgovoy [2019]

5 Conclusions and Further Work

This paper gives and overall introduction on the use of CBR, and how CBR can be represented with the 4r cycle. The target of this paper is the retrieve phase of the 4r cycle. A sufficient method in this phase depends on the given problem as well as the given case-base. The paper describes; generating, retrieval, advantages and disadvantages of using k-d trees for the retrieval of cases given a query. The key advantage of implementing a k-d tree to a given case-base is to lower the time complexity of retrieval. This implementation comes with a cost of many disadvantages such as high effort of building, order domain only, and few backtrackings in the tree seldom occur. For this reason, the paper discusses two new method for retrieval, namely sequential retrieval and two-level retrieval. The paper does not conclude in what method is more efficient for retrieval, but rather that most efficient retrieval procedure depends on the given case-base.

As described, one of the most important key results in this paper is to find out what method is most suitable to use given a case-base. This paper describes three different methods for retrieving a case from a given case-base. While these methods can be efficient there are many other retrieval methods that can be used. Because of this it is important to investigate retrieval methods before starting to implement CBR. New retrieval methods, algorithms and procedures are also created so it is important to not only consider highly developed and mature methods.

An example of a new retrieval method that is created is written about in the article "An Efficient Case Retrieval Algorithm for Agricultural Case-Based Reasoning Systems, with Consideration of Case Base Maintenance" Zhai [2020]. This article discusses a whole new way of retrieving cases from a case base, and it is published July 14, 2020. This shows that it is small limitation on what and how one can retrieve cases from a case-base.

* * *

Bibliography

GeeksforGeeks. K dimensional tree — set 1 (search and insert). URL https://www.geeksforgeeks.org/k-dimensional-tree/.

RHucker Marius. Tree algorithms explained: Ball tree algorithm vs. kd tree vs. brute force. URL https://towardsdatascience.com/tree-algorithms-explained-ball-tree-algorithm-vs-kd-tree-vs-brute-force-9746debcd940.

G. Mola Bogdan and M. Mozgovoy. Towards case-based reasoning with k-d trees for a computer game of soccer. In *2019 IEEE International Conferences on Ubiquitous Computing Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS)*, pages 570–572, 2019.

Dick M.-Termath W. et al Reuss, P. Case-based reasoning: potential benefits and limitations for documenting of stories in organizations. 2017. doi: https://doi.org/10.1007/s41449-017-0086-3.

Michael M. Richer and Rosina O. Weber. *Case-Based Reasoning, A Textbook.* Springer, 2013.

GURCHETAN SINGH. Introductory guide to information retrieval using knn and kdtree. URL https://www.analyticsvidhya.com/blog/2017/11/information-retrieval-using-kdtree/.

Wikipedia. Hyperplane. URL https://en.wikipedia.org/wiki/Hyperplane.

J.-F.M.; Martínez N.L.; Xu H Zhai, Z.; Ortega. An efficient case retrieval algorithm for agricultural case-based reasoning systems, with consideration of case base maintenance. 2020. doi: https://www.mdpi.com/2077-0472/10/9/387#cite.

Appendix

This appendix showcase different picture and a psudocode for better understanding of the overall paper.

A References

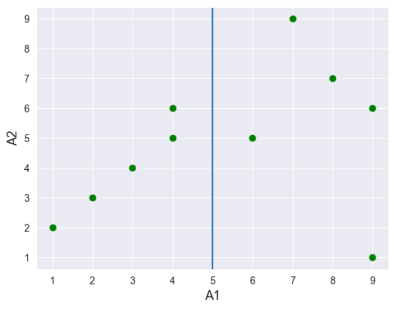

Figure 9: Drawing median of A_1 equal to 5 in the geometric view

Source: Produced by authors

Algorithm 3 Generating k-d tree

```
procedure RETRIVE($k$-d tree, Case Base $CB$, Integer $b$)
    if $K$ is a leaf node(bucket) then                                   ▷ $K$ is a node
        for each object $C$ og $K$ do
            if sim(Q,C) $> scq[m]$ then ▷ $scq$ denotes the order of nearest neighbors to the query.
                insert C in $scq$
            else
                if $A_i$ is the attribute and $v_i$ the value that labels $K$ then
                    if Q($A_i$) $\leq v_i$ then         ▷ the atribute inside the query is denoted as Q($A_i$)
                        retrive($K_{\leq}$)
                        if BOB test satisfy then
                            retrive($K_{\leq}$)
                        else Retrive($K_{>}$)
                            if BOB test satisfied then
                                retrive($K_{\leq}$)
                                if BWB test satisfied then
                                    terminate retrival with scq
                                else return
                                    return leaf node with base
```

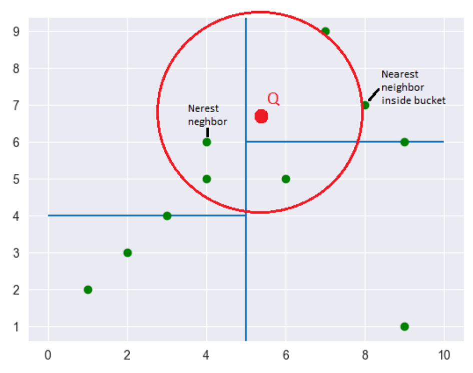

Figure 10: Finding nearest neighbour in geometric view given a query

Source: Produced by authors

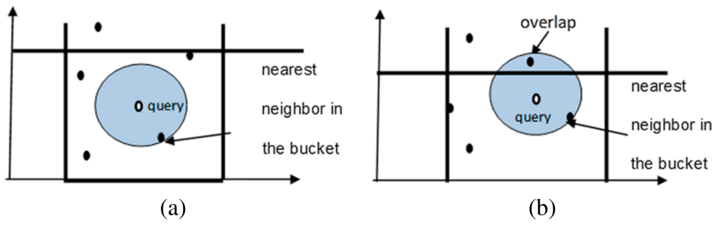

Figure 11: (a) BWB: ball within bounds, (b) BOB: ball overlap bounds

Source: Richer and Weber [2013]

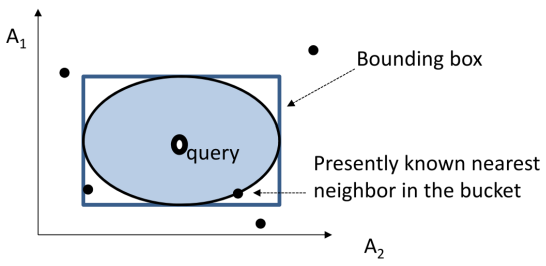

Figure 12: Passed test

Source: Richer and Weber [2013]