Knowledge Node Network in Prometheus Project

by

Si Yi Li

Laurence Liang

1. Introduction

1.1 Prometheus Project

Professor Joseph Vybihal's Prometheus project has a goal of developping a fully autonomous intelligent system. In other words, this system can perceive its environment and take the actions to maxize the chance of success for a given objective without involving any human actions. The point of developping such system is, for example, to contribute a self-determining rescue robot, a personal assistant, or an independent space exploration robot. Prometheus project has three main components (Figure 1):

- AI – which attempts to merge cognitive science and computer science research
- Simulators – algorithmic world to test the capabilities of the AI
- Robots – real swarm-based and individual-based machines running the AI

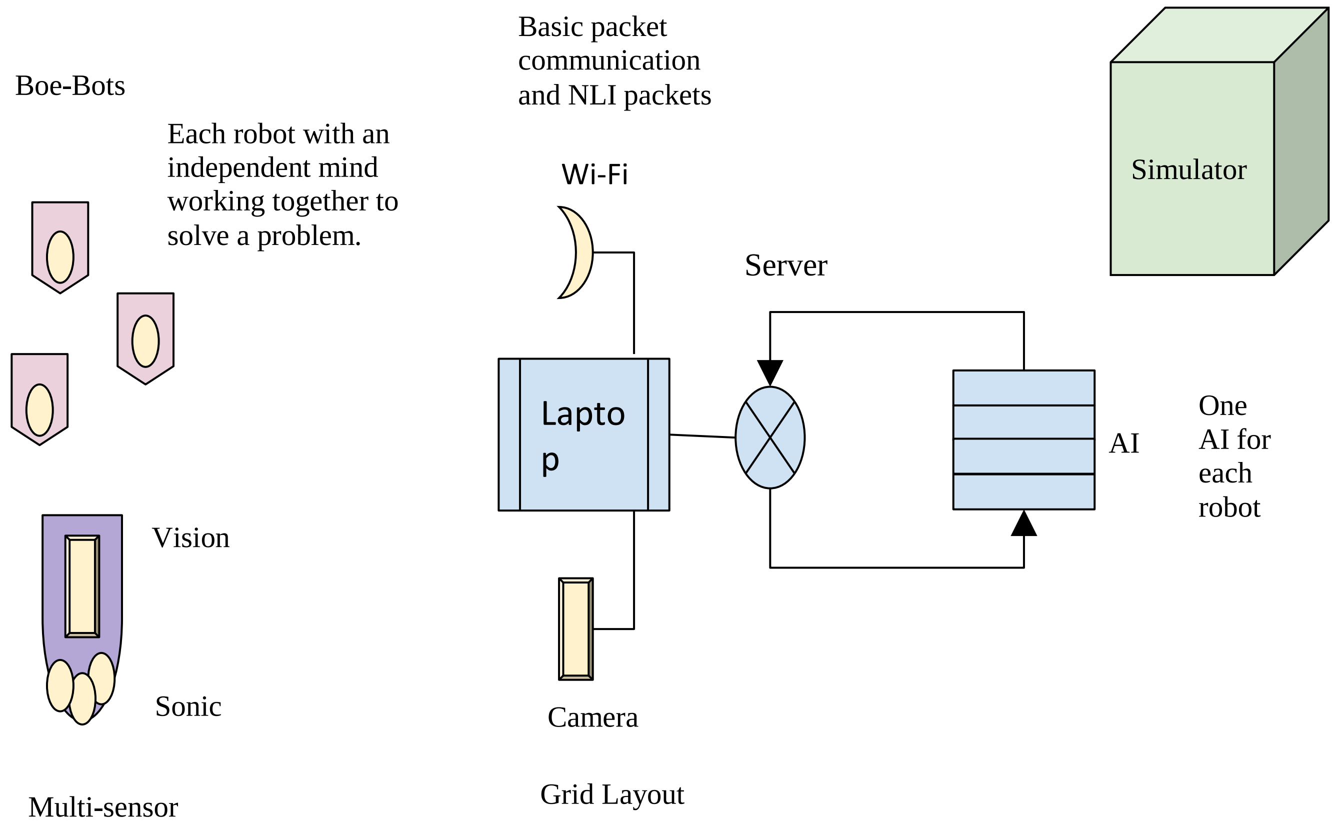

Boe-Bots
Each robot with an independent mind working together to solve a problem.
Vision
Sonic
Multi-sensor
Basic packet communication and NLI packets
Wi-Fi
Lapto p
Camera
Grid Layout
Server
Simulator
AI
One AI for each robot

Figure 1 The entire Prometheus project in a picture

1.2 AI

The AI component of the Prometheus is subdivided into four different layers (Figure 2):

- Neural Network (NN)
- Knowledge Node Network (KNN)
- Expert System (ES)
- Meta Resonner (MR)

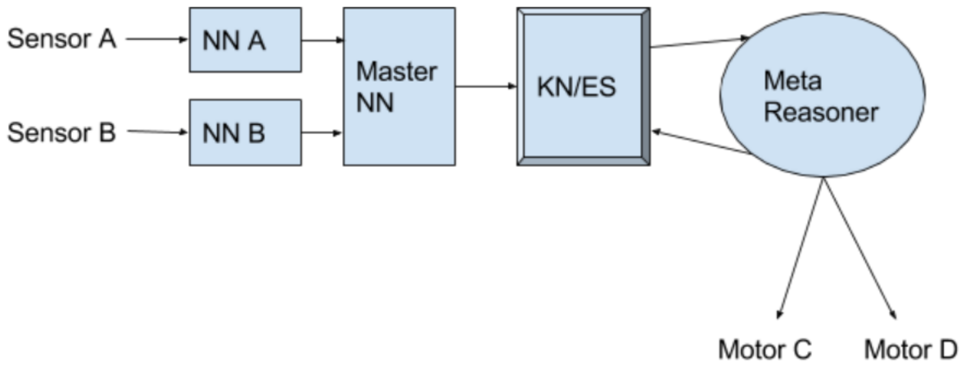


Figure 2 The four different layer of the AI

The role of NN is to receive raw signals from the sensors and convert them into string tuple in form of [String, mesurement], whereas mesurement is the probability for the receiving signals to be the object discribing by the String. Then, the string tuples are sent to the next layer, the KNN, which is the memory of the AI. The KNN searches and outputs any information that is related to the input string tuples. The information is then sent to the ES, the logic reasonner. The ES evaluates and connects every piece of information in order to find out any possible recommendation of action for the MR. After the MR receives the recommendation from the ES, the MR decides, based on the current reality data structure, if the robot should perform any of the actions. If none of the actions has performed, the MR will activate the KNN for another round of searching with a different expected output.

1.3 Knowledge Node Network

One of students in Vybihal's lab, Sean Steppas, has previously made a prototype for the Knowledge Node Network, and here is a brief description of KNN concept with his protytpe (For more detail, please go check Sean Steppas's report).

1.3.1 Knowledge Node

As mentioned earlier, the KNN is the memory layer of the AI, which attempts to imitate the structure of the human brain. Information in the KNN is stored in a data structure called the Knowledge Node (KN) (Figure 3).

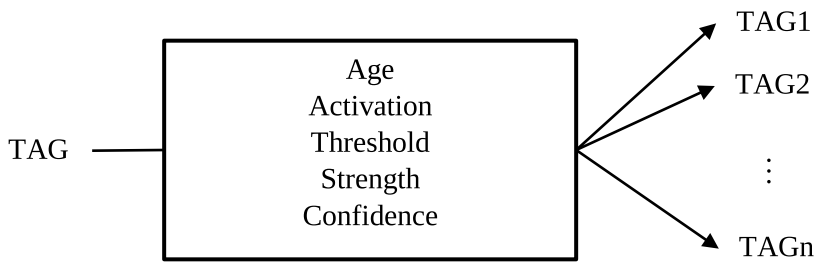


Figure 3 A diagram of the Knowledge Node

As shown in the above image, the Knowledge Node is a Class composed of a TAG, Age, Activation, Threshold, Strength, Confidence and a list of output TAGs (TAG1, TAG2, … TAGn).

TAG is a Class where the information is stored in form of a String. Information itself can be classified into three types which are: Fact, Recommendation and Rule. An example of a Fact can be "dog", a Recommendation can be "@give a bone" and a Rule can be "dog -> @give a bone". With the use of Object Oriented Design, three subclass are designed under the TAG Class which are also named Fact, Recommendation and Rule to classify what type of information the KN is stored.

Age is a Double that indicate how long does the information has been stored in the memory

Activation is a Double that mesure the importance of that piece of information

Threshold is a Double that use to compared with activation. If the value of activation is above the value of threshold, then the correspond KN will have the action of fire.

Strength is an Integer that use to increase the actual activation.

Confidence is a Double that use to evaluate the authenticityof the information.

List of output TAGs (TAG1, TAG2, … TAGn) is any other piece of information that has a relation with the current information. For example a Fact with information "dog" may have a Fact with "4 legs" in its output list.

1.3.2 Searching

Just like a human can think, the KNN also process way of searching to output the best possible information related to the input from the Neural Network. There are in total five way of searching: forward searching, forward searching with ply input, backward searching, baward searching with ply input and lambda searching.

During **forward searching**, if a TAG in KNN possesses the same information as the input of KNN, then the corresponding KN is excited, which means its activation increase with a certain value. If this activation is above the threshold, then the knowledge node will get fired, in other words, the TAG of the KN is put on the KNN active TAG list for becoming the inputs of the ES later, and every TAG on the output list of the KN has an increase in its activation by a certain value in its KN (Figure 4). Forward

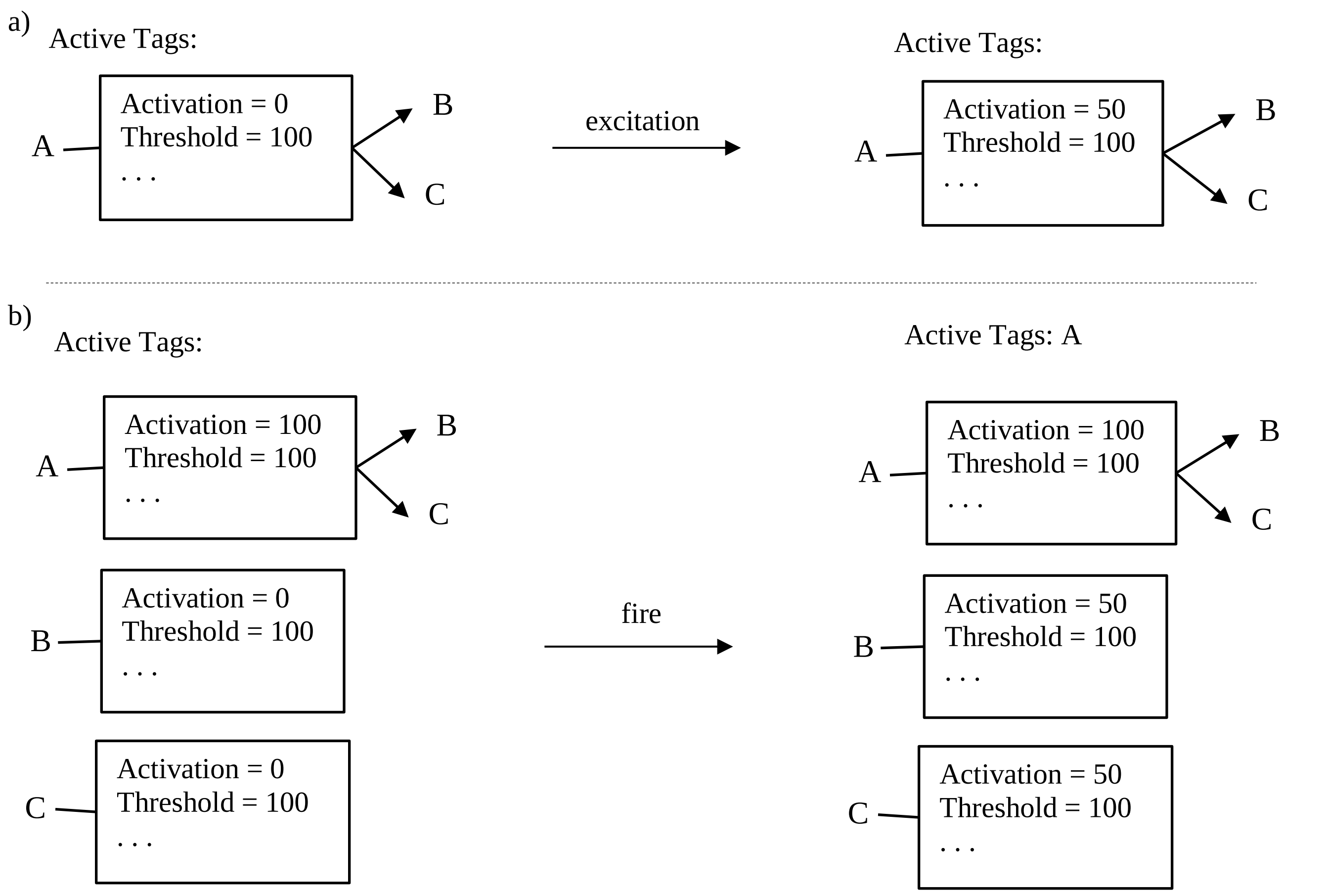


Figure 4 Excitation and Fire action during forward searching.
a) is an illustration for excite a KN and b) for firing a KN

searching without the ply parameter fire all the KNs once they have their activation above their threshold. This mimics how human think when they have unlimited time. On the other hand, **ply** is a parameter that tells forward searching should stop firing KNs at which layer, even though their activation is above their threshold. For example, in figure 4, if ply = 0, only A will be on the active TAG list even if B and C have activation above 100. This mimics the situation when an animal would have few time to think.

During **backward searching**, the matching information is compared with the TAGs on the output list of the KN instead of attaching TAG. In other words, if a certain number of the TAGs in the output list of a KN match with the input information, then the corresponding TAG will be put on the active Tag list of the KNN (Figure 5). The score for minimum of matching is calculated by AI, and will change depend on how accurate the information it needs. The ply parameter for backward searching work the same as for forward. But in this case, it tells until which layer the searching mechanism should stop comparing the output list of a KN.

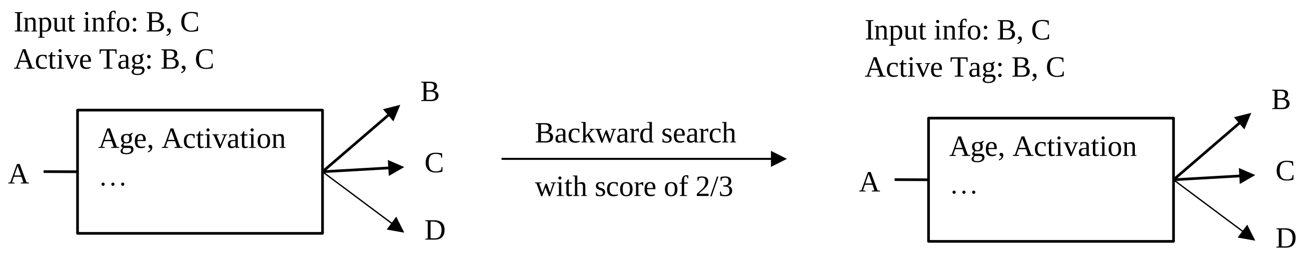


Figure 5 The mechanism of backward searching

Lambda searching is the combination of forward and backward searching. It is used to check if two or more information exist in the KNN, and if they do, what their relationship is. The mechanism is a bit like trying to find a path between two nodes in a graph. For a given piece of information, lambda search use the backward mechanism to find out if there is any TAG (or information) is above. Then, for each of the parent TAGs above the given information, forward search is used to find out if the wanted information also has the same parent TAG (Figure 6).

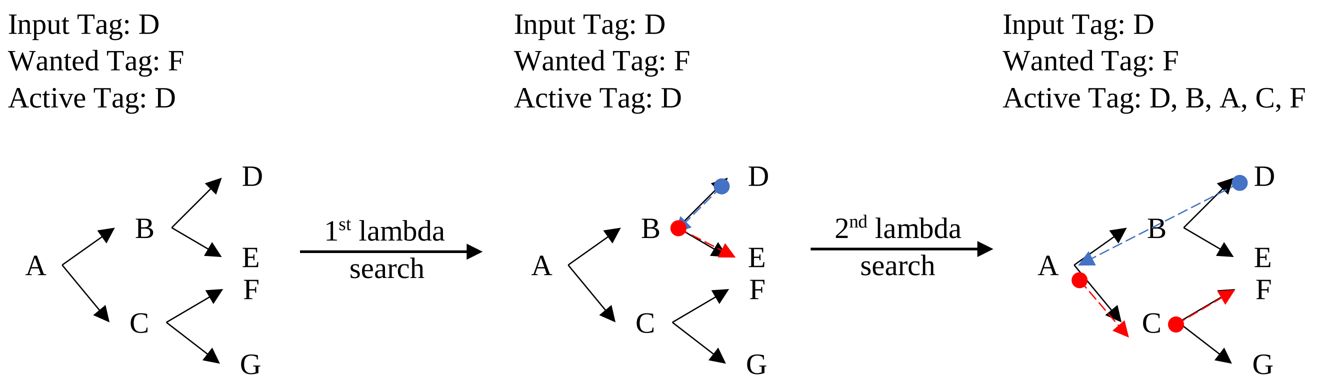


Figure 6 The mechanism of lambda saearch. Blue arrow indicate backward search and red arrow indicate forward search during lambda search. Note: only TAG of the KN is shown.

1.4 Previous process and remain problem

Seans Steppas has previously implanted the class of TAG and KNN. He also implanted forwad and backward search with or without ply parameter. However, lambda search is still not programmed yet. Moreover, how one knows which information in the KNN output is more important than the others is still unclear.

2 Completion of KNN

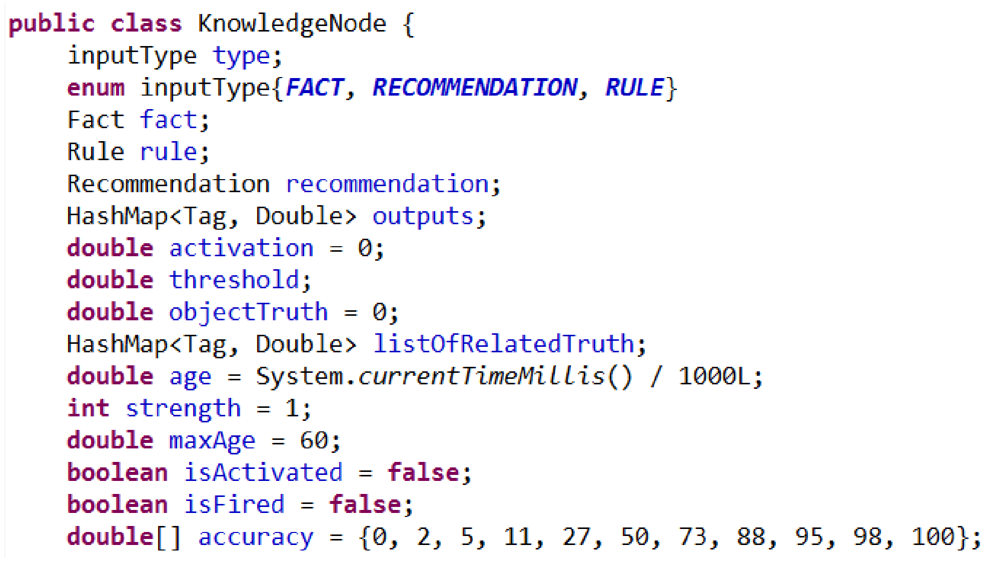
The complete code for TAG, KN and KNN are not shown in this report, please go check the java files.

2.1 Change in the TAG class

My collegue, Isaac Sultan, who is responsible for the development of the ES, has made a major change in the TAG file. Since TAG is also involved in the KNN, a quick summary of these change is needed.

As mentioned in the introduction, Fact and Remmendation are just a single word, and this is not detailled enough for the Expert Systems to choose the best actions for the robot to perform. For example, if the Fact is a “dog”, what kind of dog is that, what is its name, height, or preference? To solve such problem, Isaac create a parameter called argument within the information string, and the format of the information become “predicateName(Arg$_1$, Arg$_2$, …, Arg$_n$)”. For example, instead of just “dog”, it become “dog(type=husky, age=3, weight=28kg)” where “dog” is the predicateName and “type=husky”, “age=3”, “weight=28kg” are Arg_1 Arg_2 and Arg_3 respectively. The same change applies to Recommendation. The advantage of having argument is that there can be two or more KNs that have the same info but different arguments in KNN, and each one of the KN has different list of output TAGs. For example, if the inputs from the Neural Network are “dog” and “north pole”, then KNN, with a greater chance, will output information with “dog: that contain the an argument “husky” something like “dog(type=husky)”. *For more information please check Isaac’s report.*

2.2 Modified KN class from Sean’s version



```
public class KnowledgeNode {
    inputType type;
    enum inputType{FACT, RECOMMENDATION, RULE}
    Fact fact;
    Rule rule;
    Recommendation recommendation;
    HashMap<Tag, Double> outputs;
    double activation = 0;
    double threshold;
    double objectTruth = 0;
    HashMap<Tag, Double> listOfRelatedTruth;
    double age = System.currentTimeMillis() / 1000L;
    int strength = 1;
    double maxAge = 60;
    boolean isActivated = false;
    boolean isFired = false;
    double[] accuracy = {0, 2, 5, 11, 27, 50, 73, 88, 95, 98, 100};
```

Figure 7 A list of important parameter in the Knowledge Node class

Activation, **Threshold, Age** and **Strength** have the same definition as mention in the introduction.

fact, **recommendation**, **rule** represent the three different TAG possible attached on the KN like the one on Figure 3. If a KN process of the them, the other become NULL.

Type indicate what kind of TAG or information is stored in the KN. It can only be Fact, Recommendation or Rule.

Outputs is the output list of the KN. It is a hashmap beacause it marks down the output TAG with the corresponding membership truth to the KN. Membership truth is the probability for a KN to have the indicated TAG on its outputlist. For example a KN with Fact, “dog(type=husky)”, has a Fact, “fur(color=white)” with 80 membership truth (or chance) on its outputs, however another KN with Fact, “dog(type=chiwawa)” can have the same a Fact, “fur(color=white)”, but with 40 membership truth on its outputs.

ObjectTruth is the value for indicating the confidence of a KN from some given inputs. It is calculated differently according to the searching method that is used. The concept of objectTruth is like how probability is evaluated in mathematics. For example in math, the chance of getting a heart of aces first and a king of diamonds second in 52 poker cards is $\frac{1}{52}$ x $\frac{1}{51}$ = $\frac{1}{2652}$. In our case, let’s say the accuracry of identify the object to be a “dog” by the Neural Network is 80%. In KNN, a KN with Fact “dog(Type=husky)” has a Fact “fur(color=white)” on its output list with 50 membership truth. This means, if the object is 100% true a husky dog, then this dog has 50% chance of having white fur. Since the accuracy is only 80% by the NN, the objectTruth for having a husky dog is 80% and the objectTruth for getting something with white color fur is 40% (80%*50% = 40%).

ListOfRelatedTruth is a HashTable for marking down all the active KN with its TAG that can be used to calculcate the object truth of the given KN. These active KNs can eighter be an active parental KN (the given KN’s TAG is found on the output list of the parental KN) or an active child KN (the KN’s TAG is found on the given KN’s output list) depending on which searching method is used. Being active means the KN has activation above its threshold and its TAG is put on the activeTag list of the KNN (a list of result information to send to Expert System).

maxAge is the max time for KN to be stored in the KN. If a KN has age greater or equal to the maxAge then it will be erase.

isActivated become true when the KN’s TAG is on the activeTag list (Figure 4b, TAG A). It is checked during forward search.

isFired become true when forward search fire a KN (Figure 4b, TAG B and C). It is checked during forward search.

Accuracy is a 10 values calculated from the Sigmoid Function (the reason is explained in Vibyhal’s KNN document) to evaluate the probability for having the same information as the one stored in the KN when it is compared with the output of the Neural Network. For example if an output of the NN is (“dog”, 9), then a KN that have a Fact, “dog(type=husky)”, will have object truth=98 and the activation increases by 98.

2.3 Tuple

Tuple has the format (String, int). It is used to mimic the output of the Neural Network as mention before. The int, whose range was from 1 to 10, indicated the probablity for having the object described by String.

2.4 Knowledge Node Network Class

mapKN is a hash map for storing the inforamation tag and the corresponding knowledge node.

inputTags is a hash map for storing the informtation received from the neural network, and their confidence.

activeTags is a hash map for storing any resulting tags after a search method and their confidence. Tags in this hash map become the input of the Expert System later on.

2.5 Forward search with and without ply

Forward search without ply parameter

In the first for loop, information stored in each KN’s tag is compared with the input of the KNN. The KN possesses a FACT, then the predicate name of the Fact is compared with the String of the tuple, and if it processes a Recommendation or a Rule, the whole recommendation or rule is compared with the

input from NN: (A, 10)
inputTag:
activeTag:

input from NN: (A, 10)
inputTag: (A, 100)
activeTag: (A, 100), (B, 80), (C, 60), (D, 40)

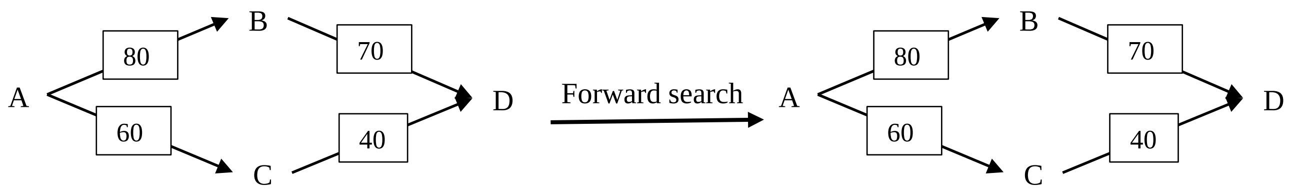


Figure 8 The KNN before and after forward search. Note that, only TAG is shown. The number in the bracket is accurarcy for input from NN and objectTruth for inputTag and active Tag.

String of the Tuple. If there is no KN process the info as the String, a new TAG, according to the content of the string, will be created and added to the activeTag list, and its confidence will be 0. When the information matches to the input, then the corresponding KN will excite according to the int value in the Tuple. In the excitation, a high int value can lead to a higher increase in the activation of the KN, and the objectTruth of that KN is the correpsonding the accuracy value of the int value. In the following while loop, the search method check if there is any KN that has activation above its threshold but not been fired yet. If so, then this KN will be fired, and its objectTruth is the product of its parental KN's objectTruth and its membershipTruth found on the outputlist of the parental KN's output list. In case if the KN has more than 1 parental KN, then the average objectTruth is used (Figure 8). At the end, all the related TAG are put on the activeTag list.

Forward search with ply parameter

The first for loop works the same as the one in forward search without ply. The difference lies in the second for loop. Instead of while loop, this second for loop only checks if there is any KN that has activation above its threshold but not been fired yet within limited ply cycle. In order words, even though there are still KNs that have activation above its threshold but not been fired yet in the KNN after ply cycle of checking, these KNs will stay the same. The objectTruth of the KN is calculated the same way as the one without ply. So, if forward search with ply=0 is used in the figure 8 instead, only (A, 100), (B, 80), (C, 60) will be on the activeTag.

2.6 Backward search with or without ply parameter

Backward search without ply parameter

The first for loop in backward search works in a similar way as the one in forward search, but the difference is that there is no excitation and fire for a KN. This is because backward search does not

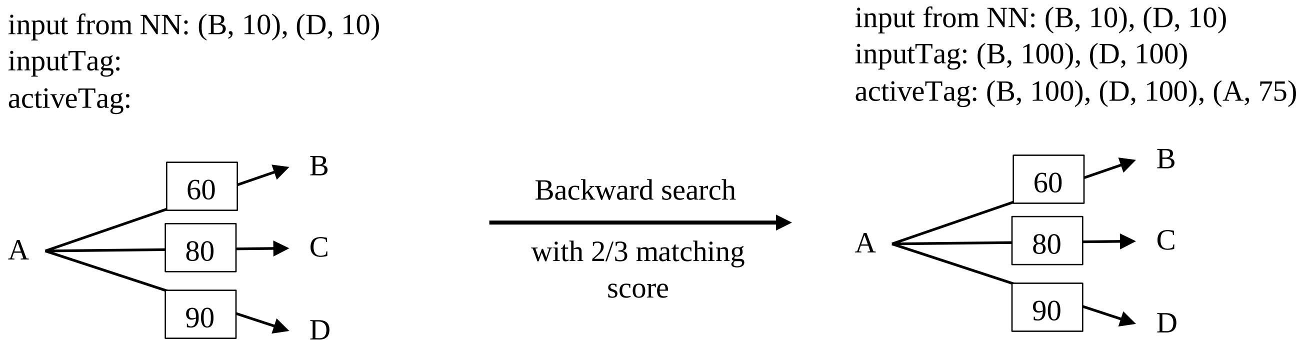

input from NN: (B, 10), (D, 10)
inputTag:
activeTag:
A
60
80
90
B
C
D
Backward search
with 2/3 matching score
input from NN: (B, 10), (D, 10)
inputTag: (B, 100), (D, 100)
activeTag: (B, 100), (D, 100), (A, 75)
A
60
80
90
B
C
D

Figure 9 The KNN before and after backward search. Note that, only TAG is shown. The number in the bracket is accurarcy for input from NN and objectTruth for inputTag and active Tag. Note that, only TAG is shown.

evaluate the activation and the threshold parameter, it only cares how many TAG on the output list of a given KN are also found on the active tag list. The while loop checks if this number of matching TAG on the output list of a given KN is greater or equal to the score parameter. If yes, the tag of that given KN will be put on the active list, and its objectTruth is calculated as the product between objectTruth of the matching TAG's KN on its outputlist and the corresponding membershipTruth of that TAG. If more than one matching is needed, then the objectTruth become the average of these products (Figure 9). The while loop stop when there are no more such KN.

Backward search with ply parameter

Again the first for loop in backward search with ply parameter is the same as the one mentioned above. The difference is that the while loop become a second for loop. This for loop only check, within certain number of cycles, if a given KN has a number of matching TAG between the output list and the active tag list greater or equal to the score. After the the given number cycle of checking, even though there is still such a KN in the KNN, the TAG of that KN will not be put on the active tag list. The objectTruth of the KN that meet the matching score is also calculated the same way as in figure 9.

2.7 Lambda search

As mentioned in the introduction, the lambda search combines both forward and backward search mechanisms. Therefore, the first for loop works exactly the same as the one in backward search. For every input tag found in the KNN, the second for loop first finds out all the parental or grand-parental KNs of the KN that have the input tag (KNs that hava a descendant path to the given KN). This mimics the mechanism of backward search. The second task the second for loop does is that for every found parental or grand-parental KNs, it perform a breath first search, to find out if there is any KN process information that are match to the wanted one. If yes, the objectTruth of the KN the processes the wanted information is calculated as shown in figure 10. This mimic the forward search mechanism. All the

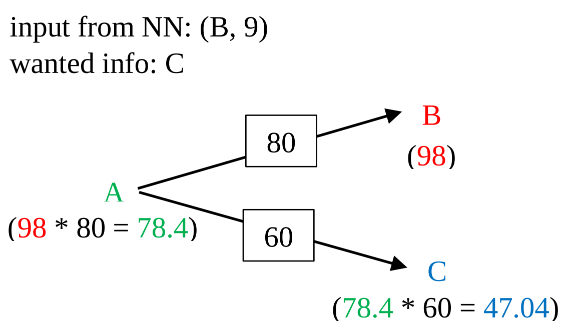


Figure 10 An example of possible path found during lambda search and how each KN's objectTruth in the path is calculated. The black number represent membershipTruth, and red, green and blue number represent the objectTruth of the correspond KN

TAGs that contribute to the backward and forward path will be stored temporarily. The path that has the best objectTruth of the wanted KN will add all its TAGs to the active list (Figure 11).

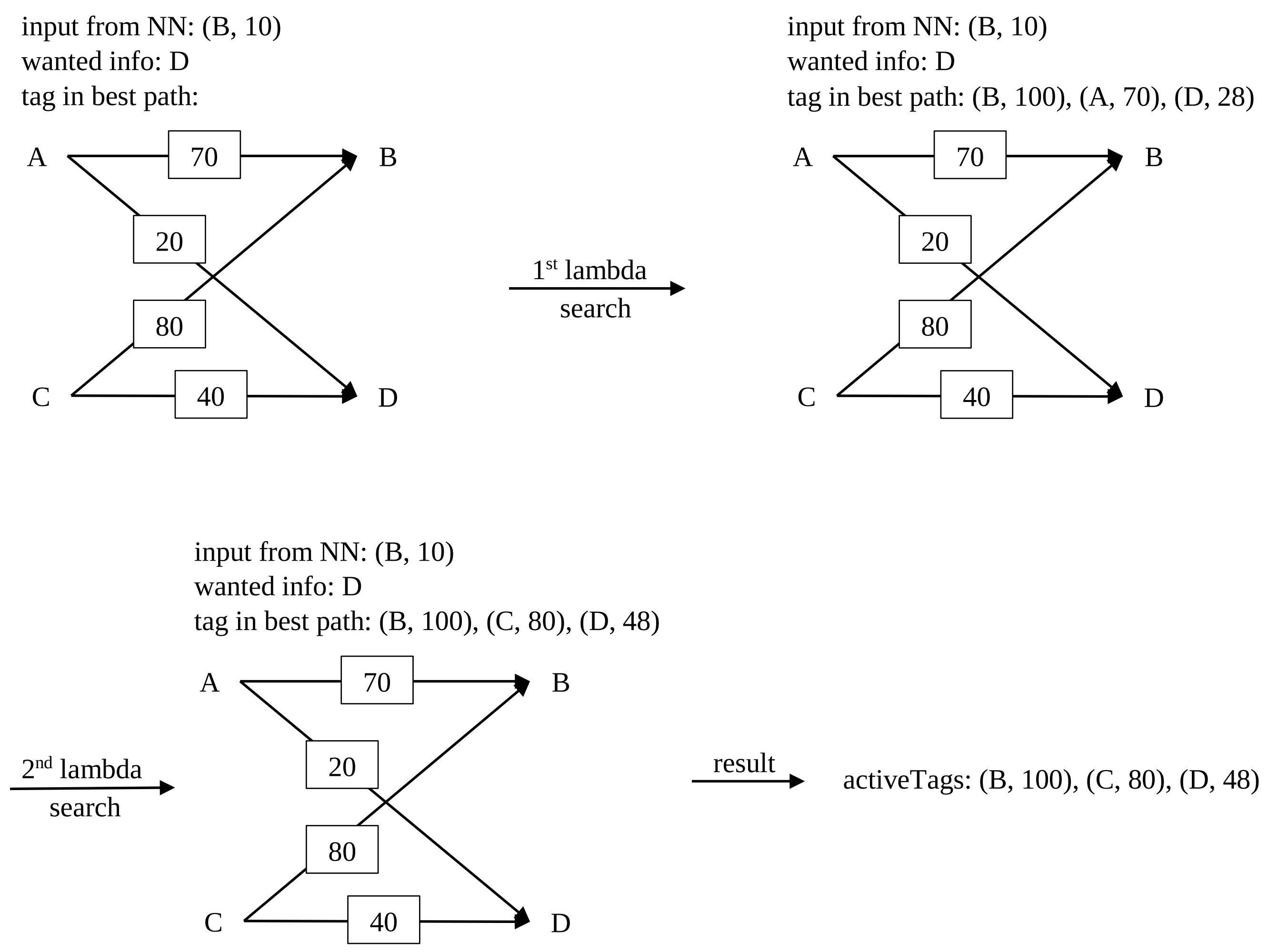


Figure 11 The mechanism of lambda search and the result after lambda search. Note that only TAG of the KN is shown

3 Experiment

To test the new KNN class, my colleague, Laurence Liang, made an animal database. A diagram of the database is illustrated in figure 12. We first use this database to create an animal Knowledge Node Network. Next, we selected some input information to test each searching method. The actual results of each searching method is then compared with the expected results by using TextNG.

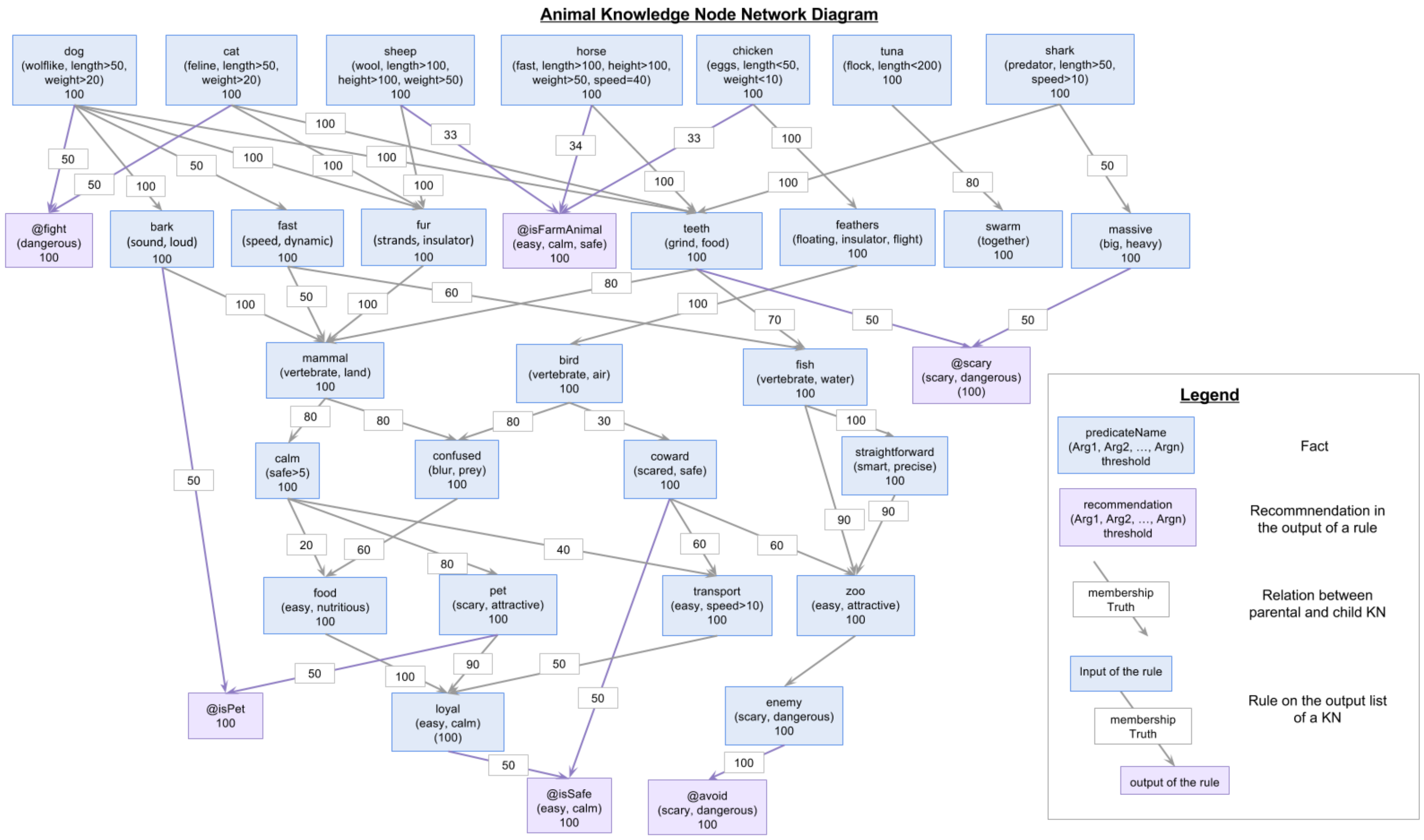

Animal Knowledge Node Network Diagram
dog (wolflike, length>50, weight>20) 100
cat (feline, length>50, weight>20) 100
sheep (wool, length>100, height>100, weight>50) 100
horse (fast, length>100, height>100, weight>50, speed=40) 100
chicken (eggs, length<50, weight<10) 100
tuna (flock, length<200) 100
shark (predator, length>50, speed>10) 100
@fight (dangerous) 100
bark (sound, loud) 100
fast (speed, dynamic) 100
fur (strands, insulator) 100
@isFarmAnimal (easy, calm, safe) 100
teeth (grind, food) 100
feathers (floating, insulator, flight) 100
swarm (together) 100
massive (big, heavy) 100
mammal (vertebrate, land) 100
bird (vertebrate, air) 100
fish (vertebrate, water) 100
@scary (scary, dangerous) (100)
calm (safe>5) 100
confused (blur, prey) 100
coward (scared, safe) 100
straightforward (smart, precise) 100
food (easy, nutritious) 100
pet (scary, attractive) 100
transport (easy, speed>10) 100
zoo (easy, attractive) 100
@isPet 100
loyal (easy, calm) (100)
enemy (scary, dangerous) 100
@isSafe (easy, calm) 100
@avoid (scary, dangerous) 100
Legend
predicateName (Arg1, Arg2, …, Argn) threshold
Fact
recommendation (Arg1, Arg2, …, Argn) threshold
Recommnendation in the output of a rule
membership Truth
Relation between parental and child KN
Input of the rule
membership Truth
output of the rule
Rule on the output list of a KN

Figure 12 The diagram of the animal knowledge node network.

Forward search test result

Input information: (dog, 10), (cat, 10)

Result active tags: {calm(safe>5)=55.0}, {dog(wolflike,length>50,weight>20)=100.0}, {zoo(easy,attractive)=45.0}, {bark(sound,loud)=100.0}, {straightforward(smart,precise)=50.0}, {enemy(scary,dangerous)=40.5}, {loyal(easy,calm)=24.2}, {teeth(grind,food)=100.0}, {[bark(sound,loud), pet(scary,attractive)]=>[@isPet(easy,calm,bark)]=36.0}, {fast(speed,dynamic)=50.0}, {[cat(feline,length>50,weight>20), dog(wolflike,length>50,weight>20)]=>[@fight(dangerous)]=50.0}, {transport(easy,speed>10)=22.0}, {fur(strands,insulator)=100.0}, {mammal(vertebrate,land)=68.75}, {food(easy,nutritious)=22.0}, {cat(feline,length>50,weight>20)=100.0}, {fish(vertebrate,water)=50.0}, {pet(scary,attractive)=44.0}, {confused(blur,prey)=55.0}, {[massive(big,heavy), teeth(grind,food)]=>[@scary(scary,dangerous)]=50.0}, {[enemy(scary,dangerous)]=>[@avoid(scary,dangerous)]=40.5}, {[coward(scared,safe), loyal(easy,calm)]=>[@isSafe(easy,calm)]=12.1}

Forward search with ply test result

Input information: (“dog”, 10), (“cat”, 10)

Ply: 1

Result active tags: {fast(speed,dynamic)=50.0}, {[cat(feline,length>50,weight>20), dog(wolflike,length>50,weight>20)]=>[@fight(dangerous)]=50.0}, {fur(strands,insulator)=100.0}, {mammal(vertebrate,land)=68.75}, {dog(wolflike,length>50,weight>20)=100.0}, {cat(feline,length>50,weight>20)=100.0}, {bark(sound,loud)=100.0}, {fish(vertebrate,water)=50.0}, {teeth(grind,food)=100.0}, {[massive(big,heavy), teeth(grind,food)]=>[@scary(scary,dangerous)]=50.0}, {[bark(sound,loud), pet(scary,attractive)]=>[@isPet(easy,calm,bark)]=50.0}

Backward search test result

Input information: (“calm”, 10), (“coward”, 10)

Score: 0.5

Result active tags: {calm(safe>5)=100.0}, {fast(speed,dynamic)=40.0}, {fur(strands,insulator)=80.0}, {mammal(vertebrate,land)=80.0}, {coward(scared,safe)=100.0}, {bird(vertebrate,air)=30.0}, {feathers(floating,insulator,flight)=30.0}, {dog(wolflike,length>50,weight>20)=60.0}, {bark(sound,loud)=80.0}, {chicken(eggs,length<50,weight<10)=30.0}, {sheep(wool,length>100,height>100,weight>50)=80.0}

Backward search with ply test result

Input information: (“calm”, 10), (“coward”, 10)

Score: 0.5

Ply = 2

Result active tags: {calm(safe>5)=100.0}, {fast(speed,dynamic)=40.0}, {fur(strands,insulator)=80.0}, {mammal(vertebrate,land)=80.0}, {coward(scared,safe)=100.0}, {bird(vertebrate,air)=30.0}, {feathers(floating,insulator,flight)=30.0}, {bark(sound,loud)=80.0}

Lambda search test

Input information: (“mammal”, 10)

Wanted information: (“fish(vertebrate,water)”)

Result active tags: {fur(strands,insulator)=100.0}, {mammal(vertebrate,land)=100.0}, {cat(feline,length>50,weight>20)=100.0}, {fish(vertebrate,water)=70.0}, {teeth(grind,food)=100.0}

4 Future direction

The progress of Knowledge Node Network has now included three different search methods: forward, backward and lambda search. It also has defined what the confidence of a knowledge node is after each search method. However, there are still many improvements needed in order for the Knowledge Node Network to be completed. For example, we need to find out a way to deteremine which knowledge node has more strength than the others. As mentioned in the introduction, strength can make a knowledge node fire earlier even though it may has a lower confidence than others. This parameter is the key for the AI to be able to learn from the past. We also need to find out a way to store new information that were not in the Knowledge Node Network before. Not every single piece of information from the Neural Network can be found in the KNN, and this information should not just be ignored. Having a way to store unknown information can help the AI upgrade its memory and to be able to handle more different situations, thus making it more versatile and general purpose.