Prometheus AI

Sean Stappas — Supervised by Prof. Vybihal

McGill

Introduction

- Prometheus
  - Model of the human brain.
  - Controls multiple robots in a swarm.

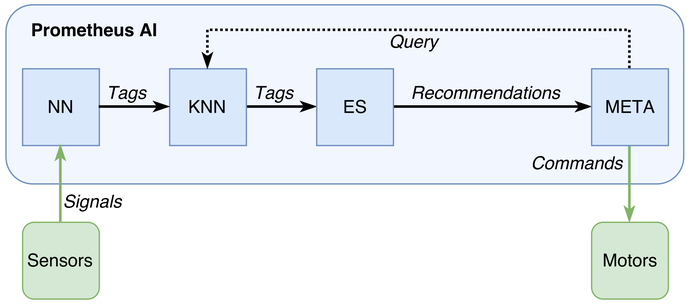
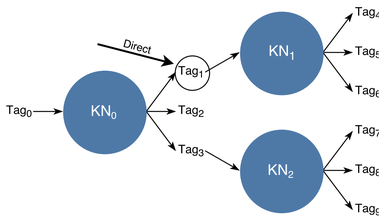
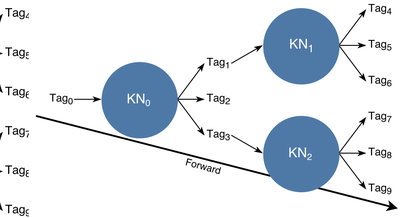


Figure 1: Prometheus AI model with labeled input and output.

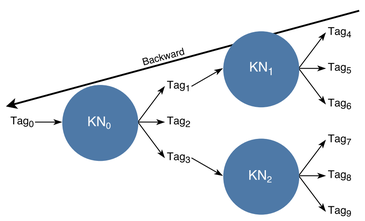
- Neural Netowrk (NN)
  - Low-level signal processor.
- Knowledge Node Network (KNN)
  - Represents memory.



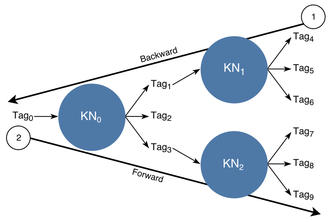
(a) Direct searching.



(b) Forward searching/thinking.



(c) Backward searching/thinking.



(d) Lambda searching/thinking.

Figure 2: Methods of searching and thinking in the KNN.

- Expert System (ES)
  - Logical reasoner.
  - Unaware of context.

$$P(A_1, \ldots, A_n) \tag{1}$$

Table 1: Examples of facts in the ES.

Fact	Meaning
$P(x)$	x is true or active.
$P(x = 1)$	x is equal to 1.
$P(x\ != 1)$	x is not equal to 1.
$P(x > 1)$	x is greater than 1.
$P(x < 1)$	x is less than 1.
$P(\&x)$	x can take any integer value.
$P(*)$	All arguments can take any value.
$P(?)$	One argument can take any value.

$$Fact_1 \cdots Fact_m \rightarrow Tag_1 \cdots Tag_n \tag{2}$$

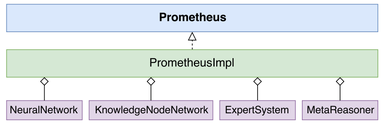
- Meta Reasoner (META)
  - High-level decision-maker.

Problem

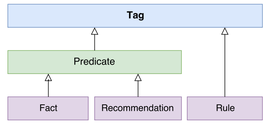
- Design and implement the ES and KNN in Java.
  - Create an initial design.
  - Build a code skeleton.
  - Implement integration and unit tests.
- Supervise undergraduate students working on Prometheus.
  - Provide resources.
  - Review code.

Design & Implementation

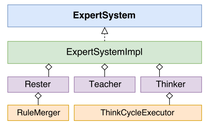
- Dependencies modeled using Google Guice.
  - Framework for modular dependency injection.
  - Allows for easily testable code.
- Tag object is central to the ES and KNN.
  - Represents a unit of information.
  - Can be instantiated as Fact, Rule or Recommendation classes.



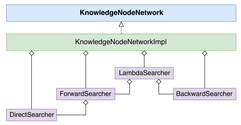
(a) Prometheus UML diagram.



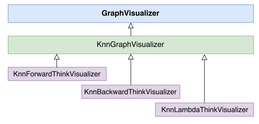
(b) Tags UML diagram.



(c) ES UML diagram.



(d) KNN UML diagram.



(e) Graphing UML diagram.

Figure 3: UML diagrams of the major modules in Prometheus.

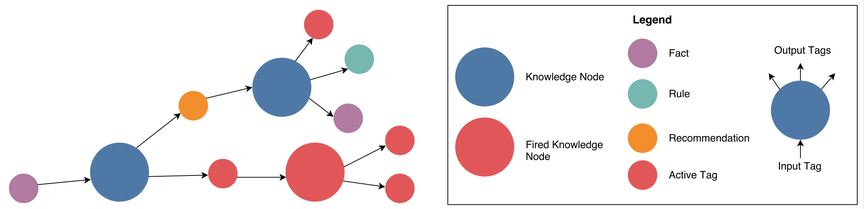


Figure 4: Legend for the visualization of the KNN.

Results & Tests

- Unit tests.
  - Testing individual methods of every class in the KNN and ES.
  - Dependencies mocked using the Mockito library.
  - 90 % coverage.
- Integration tests.
  - Testing end-to-end behavior of the ES and KNN modules.
  - All unit and integration tests written with TestNG and executed with TravisCI.
  - 95 % coverage.

State	Ready Rules	Active Rules	Active Facts	Active Recommendations
Initial	$(A)(B) \rightarrow (D)$ $(D)(B) \rightarrow (E)$ $(D)(E) \rightarrow (F)$ $(G)(A) \rightarrow (H)$ $(E)(F) \rightarrow (\#Z)$		$(A), (B)$	$(\#X), (\#Y)$
⋮	⋮	⋮	⋮	⋮
Final	$(G)(A) \rightarrow (H)$	$(A)(B) \rightarrow (D)$ $(D)(B) \rightarrow (E)$ $(D)(E) \rightarrow (F)$ $(E)(F) \rightarrow (\#Z)$	$(A), (B),$ $(D), (E)$ (F)	$(\#X), (\#Y), (\#Z)$

Figure 5: ES test setup representing simple rules and facts that must be brought to quiescence.

Results & Tests (continued)

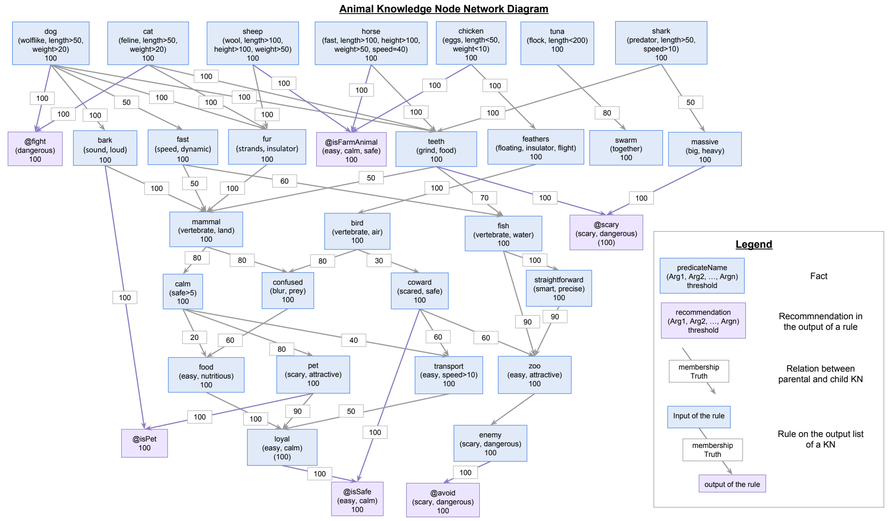
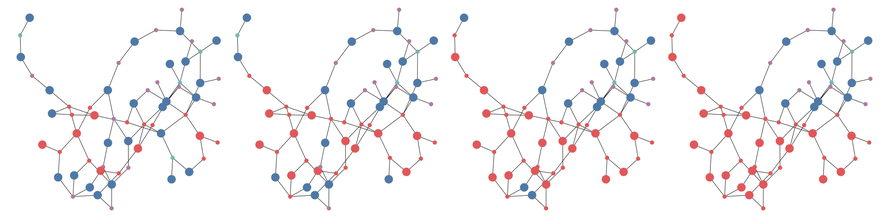


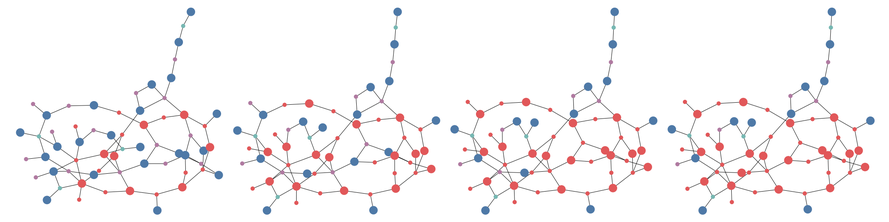
Figure 6: Elaborate test KNN network representing connections between memories of animals and their characteristics. Credit for creating the graph goes to volunteer Laurence Liang.

- Graph visualization tests.
  - Iterations of the KNN searching algorithms are presented visually.
  - Small nodes are Tags, big nodes are Knowledge Nodes (KNs).
  - Red nodes represent active Tags or fired KNs.
  - For non-active Tags, Facts are purple, Rules are blue and Recommendations are orange.



(a) Ply 1. (b) Ply 2. (c) Ply 3. (d) Ply 4.

Figure 7: Forward thinking visualization in the KNN.



(a) Ply 1. (b) Ply 2. (c) Ply 3. (d) Ply 4.

Figure 8: Backward thinking visualization in the KNN.

Conclusion

- Skills learned:
  - Planning and implementing large software project.
  - Time management.
  - People management.
- Possible future work:
  - Implement the missing NN and META layers.
  - Explore further features in the KNN, such as learning and attention.