Learning Algorithms and Quake

David Ganzhorn, William de Beaumont
University of Rochester
March 19th, 2004

Abstract

We implemented a natural language interface to our Quake agents. During the course of the project we modified an existing chart parser to reconstruct the parse trees for natural language sentences from a grammar and lexicon we designed. The grammar allows for a large amount of flexibility in syntactic structures, and combined with the lexicon it has a great generative power. We take the parse tree, pass it through our semantic module, and end up with quagent level commands, which we pass into our quagent module. The quagent can then follow complex series of commands, including movements, perceiving, intelligent exploratory behavior, displaying it's current map of exploration, picking up and dropping items, and checking it's status. Although the infrastructure is well developed, we were not able to fully implement the semantics module that was capable of dealing with the full expressiveness of our grammar and lexicon.

Motivation

Our primary motivation was to create a natural language interface to our quagent that was sufficiently expressive for us to be able to direct our quagent to do virtually any combination of the fundamental set of

behaviors it is capable of, from taking a step forwards to actively seeking out neeed items. This was our goal both so that we could get experience working with natural language parsing, processing, and generation, but also so that we would be able to reprogram our quagents behaviors via natural language instructions, so that we could reconfigure the quagent more quickly and efficiently.

Methods

Our first step was to modify the chart parser so that we could recover the parse tree; the chart parser we started with would only let us know how many parses it was able to find. By adding links backwards from each nonterminal lexical structure to it's components, we were able to recursively reconstruct the parse tree. We then created a small program to visualize the way our inputs were being parsed:

```
give me some tofu
 give me some tofu
 \__/               GV "give"
     \_/            PR "me"
     \_/            DNP "me"
        \___/       DET "some"
             \____/ N "tofu"
             \____/ NP "tofu"
        \__________/ DNP "some tofu"
 \__________________/ VP "give me some tofu"
 \__________________/ COMMAND "give me some tofu"
 \__________________/ S "give me some tofu"
```

After this, we created a program to take the parse tree output, and condense it into a single string, for further processing:
[S [COMMAND [VP [GV give] [DNP [PR me]] [DNP [DET some] [NP [N tofu]]]]]]
During this process we were also constructing and elaborating on a lexicon and grammar, as well as a preprocessing module. The preprocessor conforms (a small part of) the much broader set of utterances in English into the less expressive grammar and lexicon that we developed. Of course, it cannot conform all English to our grammar and lexicon, but it does help to expand the set of input that our program is capable of processing. See the appendix for our grammar and lexicon.
Lastly we created a quagent with a queue of commands, such that the agent would be able to keep track of multiple instructions and queries. Also, we included the ability to tell the quagent to ignore all previous commands. Then, we added in a small passive learning function that allowed the quagent to determine the optimal exploratory behavior, in terms of simply reaching an unexplored square as soon as possible.
Lastly we implemented the program logic that allows the quagent to listen for new instructions at every cycle, and then to get to work on any commands or queries that have not yet been dealt with.

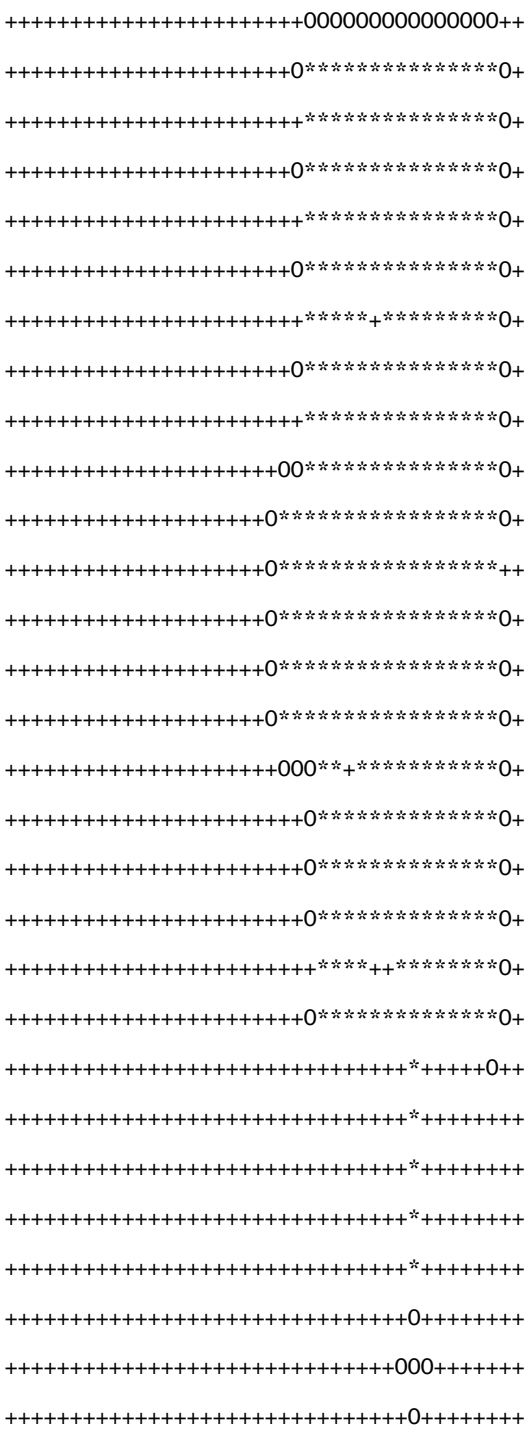
Results

Because we were able to implement only a limited semantics

module, and because we did not have time to implement the more advanced features that we had planned, we were only able to test that our parser and semantics module were capable of turning various simple natural language commands into quagent commands. We were only able to implement the bare bones semantics interpretation, and thus the only commands that (we believe) can make it through the semantic module are, “walk”, “go”, “move”, “explore”, and “die”. However, the infrastructure is mostly developed, and expanding the semantics module to deal with a much larger input set should be trivial.

We had hoped that we would have time to implement a language generation module, that would have allowed the quagent to communicate back to us through our grammar and lexicon, but unfortunately we did not get that implementation finished.

One thing that we did to ensure that our quagent was capable of following commands correctly was to directly command it to explore for several hundred turns, and then to show us the map of what it had found. At each cycle the quagent used value iteration over a subsection of the quagent world, to create a plan that would lead it to new unexplored regions. Here is what the quagent's map looked like::



In this map, the +'s are unexplored regions, the *'s are explored regions, and the O's are walls. At the bottom, the cross shaped section of walls is an example of what happens when the quagent gets stuck on a corner in quake world: it marks all of the surrounding squares as walls, and thus it realizes it is stuck, and so it marks it's current square as a wall. To navigate, the quagent uses value iteration over the world, where unexplored regions are considered to be reward squares. This causes the agent to head for nearby regions of unexplored space, and thus, leads to exploration in general.

Discussions

What we found to be the main bottleneck in this project was the

semantic module. Modifying the chart parser, constructing a grammar, lexicon, and preprocessor, and implementing the quagent back-end, were clear as to what needed to be done. However, implementing a semantic module that was capable of transforming a parse tree into simple, intelligible commands was an issue that we found truly challenging. It formed such a bottleneck that we were not able to implement the more interesting features that we had planned.

One such feature we had hoped to implement was the ability to dynamically define new words in the lexicon of the course of the discourse; using a new word would prompt a question from the quagent, and the description of the world would be used to synthesize new rules to add to the semantic module, to allow for the future interpretation of that word. Also, the ability for the quagent to generate natural language responses to our command and queries, via some process analogous to running the quagent commands into the semantic module in reverse, was another feature we had hoped to be able to implement.

A test that we had planned on running was to have a randomly wandering quagent, a wall following quagent, and an exploratory quagent, all of which we had already implemented in the past, and our new NLU quagent compete against each other to see which quagent could pickup the most items in quake world in a limited span of time. Because we were not able to complete the implementation of the NLU quagent we were unable to run this test, but we thought that it would have been an

excellent imperial test to show what advantages (and disadvantages), were gained by having an NLU interface to the quagent.

Overall, we found that the NLU interface was a difficult and interesting feature to implement, and that deriving semantics from parse trees was extremely difficult.

Appendix

grammar

```
things this grammar doesn't include:
modal verbs, verb-phrase-like clauses, morphology of any
kind
s       -> command
        -> question
        -> statement
        -> s conj s
        -> ifword statement thenword s
command -> vp
question -> wh vp
statement -> dnp vp
dnp     -> det np
        -> num np
        -> pr
        -> dnp conj dnp
np      -> adjp np
        -> np pp
        -> n
adjp    -> adv adjp
        -> adj
these vp rules are 'commented out' by inserting a space in
the arrow
vp      - > iv pps
        - > iv
        - > tv dnp pps
        - > tv dnp
        - > gv dnp dnp pps
        - > gv dnp dnp
        - > adv vp
        - > vp adv
```

```
pps     - > pps pp
        - > pp
vp      -> iv
        -> tv dnp
        -> gv dnp dnp
        -> adv vp
        -> vp adv

this rule is sort of a kludge
adv     -> pp
pp      -> prep dnp
num     -> dig num
        -> dig
```

lexicon

these are the categories

n	**noun**
pr	**pronoun**
det	**determiner (ex the)**
iv	**intransitive verb (ex die) no arguments**
tv	**transitive verb (ex take) one argument**
gv	**"give" verb (ex give) two arguments**
adj	**adjective**
adv	**adverb**
prep	**preposition (ex to)**
conj	**conjunction (ex and)**
wh	**wh- word (ex what)**

agent == n
robot == n
bot == n
tofu == n
gold == n
head == n
kryptonite == n
distance == n
height == n
width == n
depth == n
me == pr
I == pr
you == pr
it == pr
they == pr
one == pr adj
this == pr det
that == pr det

these == pr det
those == pr det
is there a pronoun?
there == pr
someone == pr
something == pr
a == det
an == det
the == det
any == det
some == det
my == det
your == det
find == tv
give == gv
get == tv gv
bring == tv gv
take == tv
drop == tv
is == tv
are == tv
am == tv
die == iv
go == iv
move == iv tv gv
nearest == adj
nearer == adj
closest == adj
closer == adj
farthest == adj
farther == adj
further == adj
slowly == adv
quickly == adv
to == prep
from == prep
for == prep
by == prep
of == prep
and == conj
or == conj
0 == dig
1 == dig
2 == dig
3 == dig
4 == dig

```
5 == dig
6 == dig
7 == dig
8 == dig
9 == dig
times == n
degrees == n
units == n
please == adv
straight == adj adv
forward == adv
ahead == adv
left == adv adj
right == adv adj
hold == tv
wealth == n
tell == gv
see == tv
show == gv tv
turn == iv
walk == iv

technically this is incorrect, but I'm guessing people
won't know that...
fast == adj adv

most == adj
fewest == adj
greatest == adj
least == adj
more == adj
fewer == adj
greater == adj
less == adj

who == wh
what == wh
when == wh
where == wh
why == wh
how == wh

if == ifword
then == thenword

these will be produced by the preprocessor
```

```
pickup == tv
andthen == conj
```