Playing with Semantic Memory

Bob Marinier
6/15/11

SoarTech

Modeling human reasoning.
Enhancing human performance.

Motivations for Semantic Memory

- **Expensive rules**
- Data chunking / learning
- Runtime query construction
- Limit the size of WM (possibly to help epmem)
- Take advantage of underlying database
- Pre-load data
- Maintain data across runs

Example (from Soar manual)

```
sp  {smem*sample*query
    (state <s> ^smem.command <sc>
               ^lti <lti>
               ^input-link.foo <bar>)
-->
    (<sc> ^query <q>)
    (<q> ^name <any-name>
         ^foo <bar>
         ^associate <lti>
         ^age 25)
}
```

Semantic Memory Does Not:

- **Retrieve multiple matches at once**
- Support arbitrary partial matching
- Support deep structure matching
- Support variablize attributes
- Support less than/greater than matching
- Support negative queries
- Support special spatial queries
- Prove P=NP
- Solve world hunger

Looping to Retrieve Multiple Matches

- Looping in semantic memory: 3 methods
  - Build up a prohibits set
    - As each item is retrieved, add its LTI to the prohibits
    - For large sets, this doesn't scale well
  - Modify the memories so they don't match anymore
    - Scales much better, but requires data to share some flag
  - Retrieve linked LTIs (walk a list)
    - Scales even better since there is no match cost, but requires data to be structured as a list
- Compared to writing a single expensive rule, writing the many rules to do either of these patterns is a lot of work
  - So I started extending the Dave Ray's bebot library

Generic Bebot Loop Proposal

```
sp {propose*bebot*smem*loop
       (state <s> ^name test-smem-loop
                 -^result)
    -->
       (<s> ^operator <o> +)
       (<o> ^name bebot*smem*loop
            ^query <query>
            ^func my-function-operator
            ^tcnum-attr my-tcnum-attr # optional
            ^next-attr my-next-attr # optional
            ^allow-chunks true # optional; default
 false
            ^init my-init-operator # optional
            ^test my-test-operator # optional
            ^return my-return-operator) # optional
```

Bebot Example: Retrieve first n values

```
sp {propose*bebot*smem*retrieve-n
       (state <s> ^name test-smem-retrieve-n
                 -^result)
    -->
       (<s> ^operator <o> +)
       (<o> ^name bebot*smem*retrieve-n
            ^query <query>
            ^n 3)
       (<query> ^value <v>)
}
```

Example: get two smallest values in a set (Expensive rule)

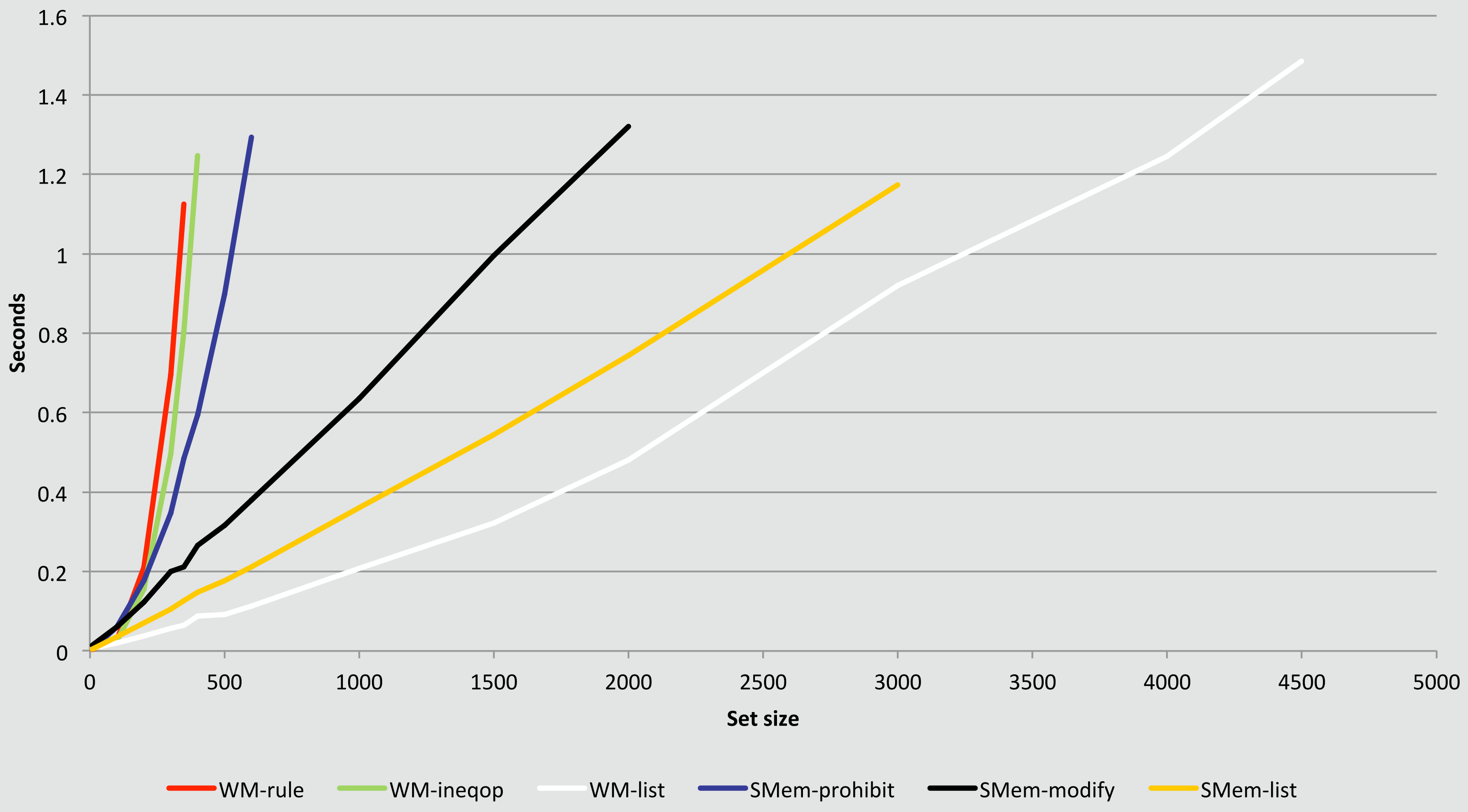
```
sp {search-wm*return-result
    (state <s> ^name search-wm
               ^values <vals>
               ^superstate <ss>)
    (<vals> ^value <v1> {<v2> < <v1>})
   -(<vals> ^value {<v3> <> <v2> < <v1>})
-->
    (<ss> ^min-values <mv>)
    (<mv> ^value <v1> <v2>)
}
```

Example: get two smallest values in a set (Bebot)

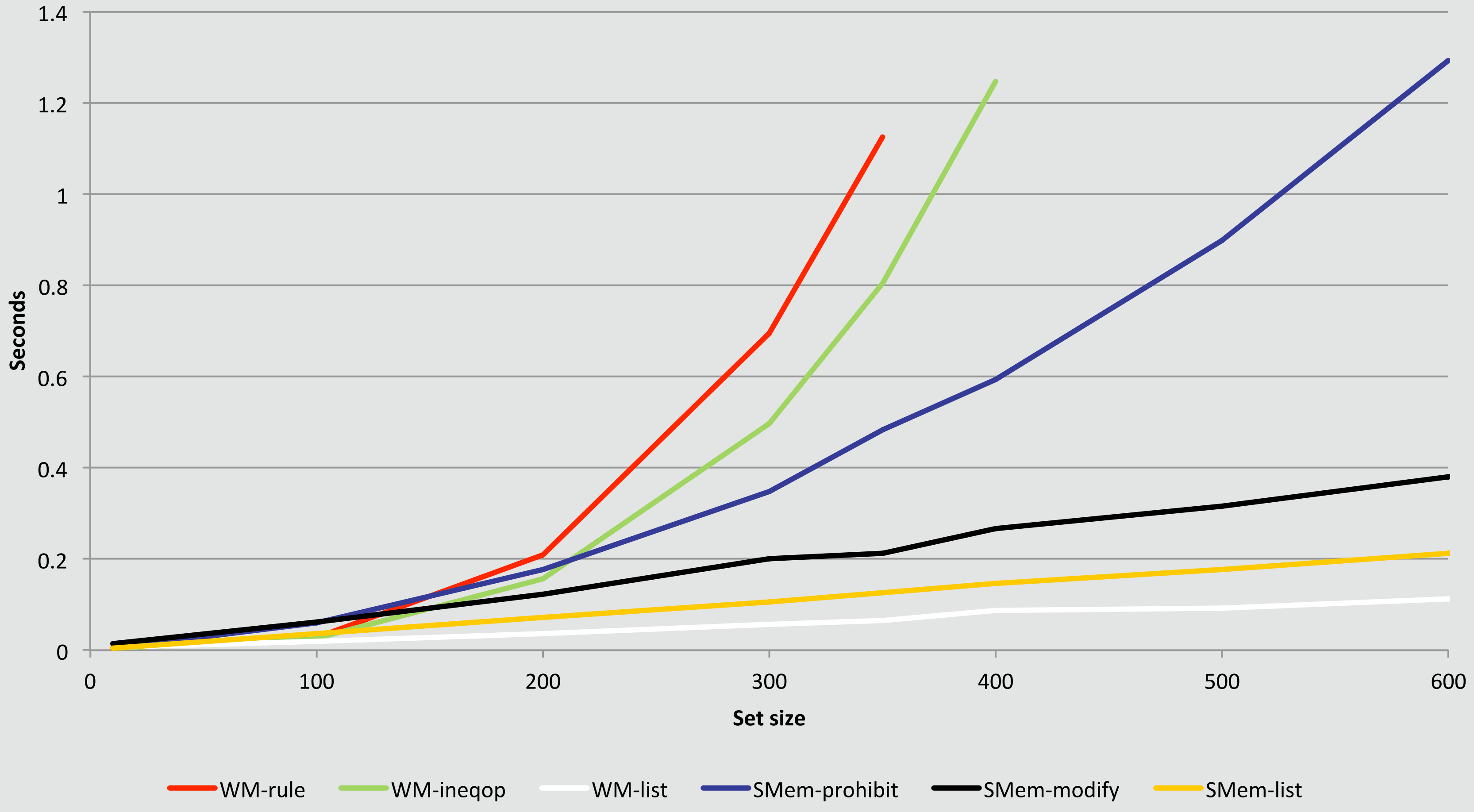
```
sp {propose*bebot*smem*loop
       (state <s> ^name test-smem-loop
                 -^result)
    -->
       (<s> ^operator <o> +)
       (<o> ^name bebot*smem*loop
            ^query <query>
            ^func func-two-min-vals
            ^tcnum-attr tcnum)
       (<query> ^value <v>)}

sp {apply*func-two-min-vals*first
       (state <s> ^operator <o>)
       (<o> ^name func-two-min-vals
            ^object.value <newval>
            ^previous <p>)
      -(<p> ^value)
    -->
       (<s> ^result <p>)
       (<p> ^value <newval>)}

sp {apply*func-two-min-vals*second
   (state <s> ^operator <o>)
   (<o> ^name func-two-min-vals
        ^object.value <newval>
        ^previous <p>)
   (<p> ^value <v1>
       -^value <> <v1>)
-->
   (<s> ^result <p>)
   (<p> ^value <newval>)}
```

```
sp {apply*func-two-min-vals*too-large
       (state <s> ^operator <o>)
       (<o> ^name func-two-min-vals
            ^object.value > <v2>
            ^previous <p>)
       (<p> ^value <v1>
            ^value {<v2> > <v1>})
    -->
       (<s> ^result <p>)}

sp {apply*func-two-min-vals*replace-value
   (state <s> ^operator <o>)
   (<o> ^name func-two-min-vals
        ^object.value {<newval> < <v2>}
        ^previous <p>)
   (<p> ^value <v1>
        ^value {<v2> > <v1>})
-->
   (<s> ^result <p>)
   (<p> ^value <v2> - <newval>)}
```

Performance Results: Time (get 2 smallest values)

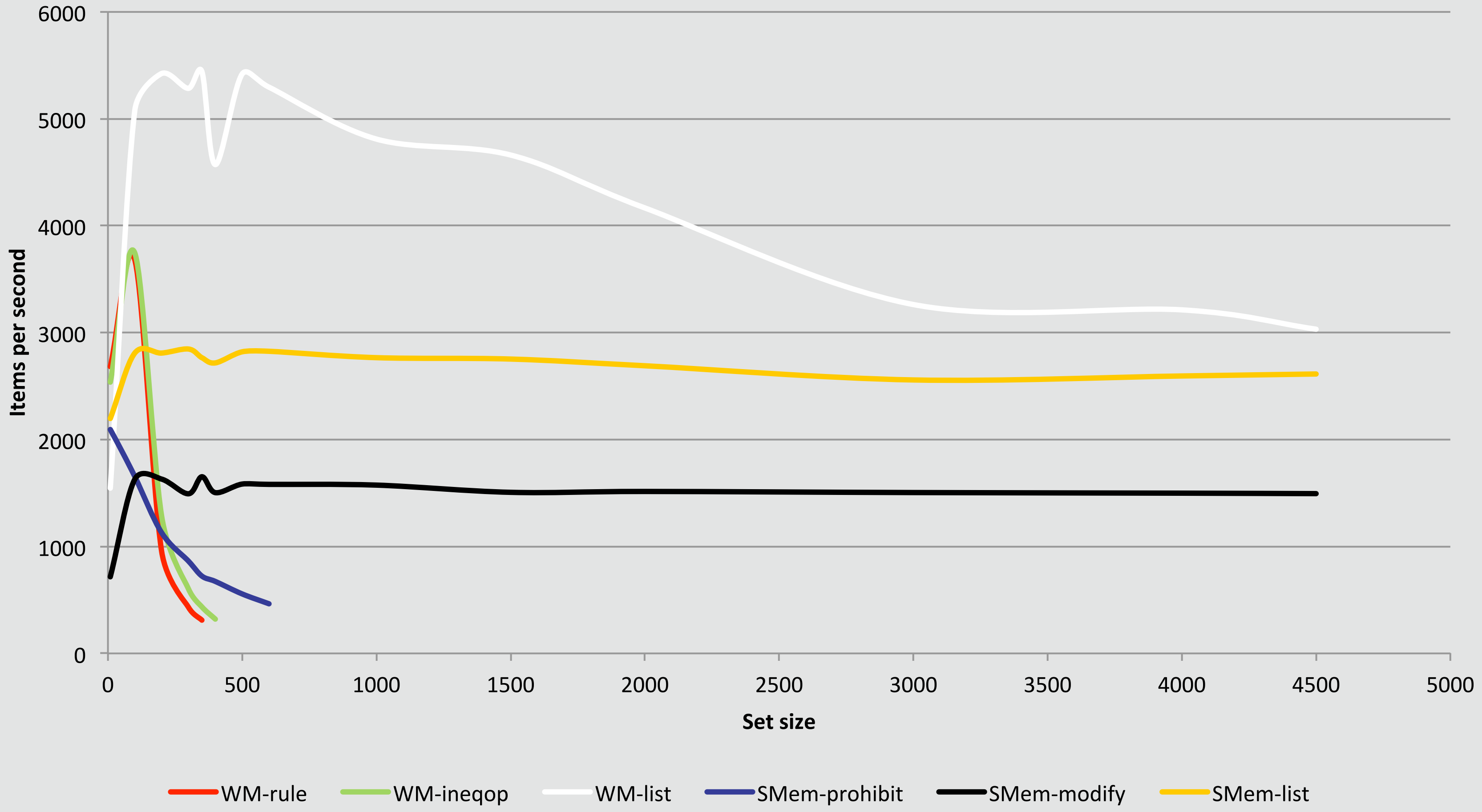


Performance Results: Time (get 2 smallest values)

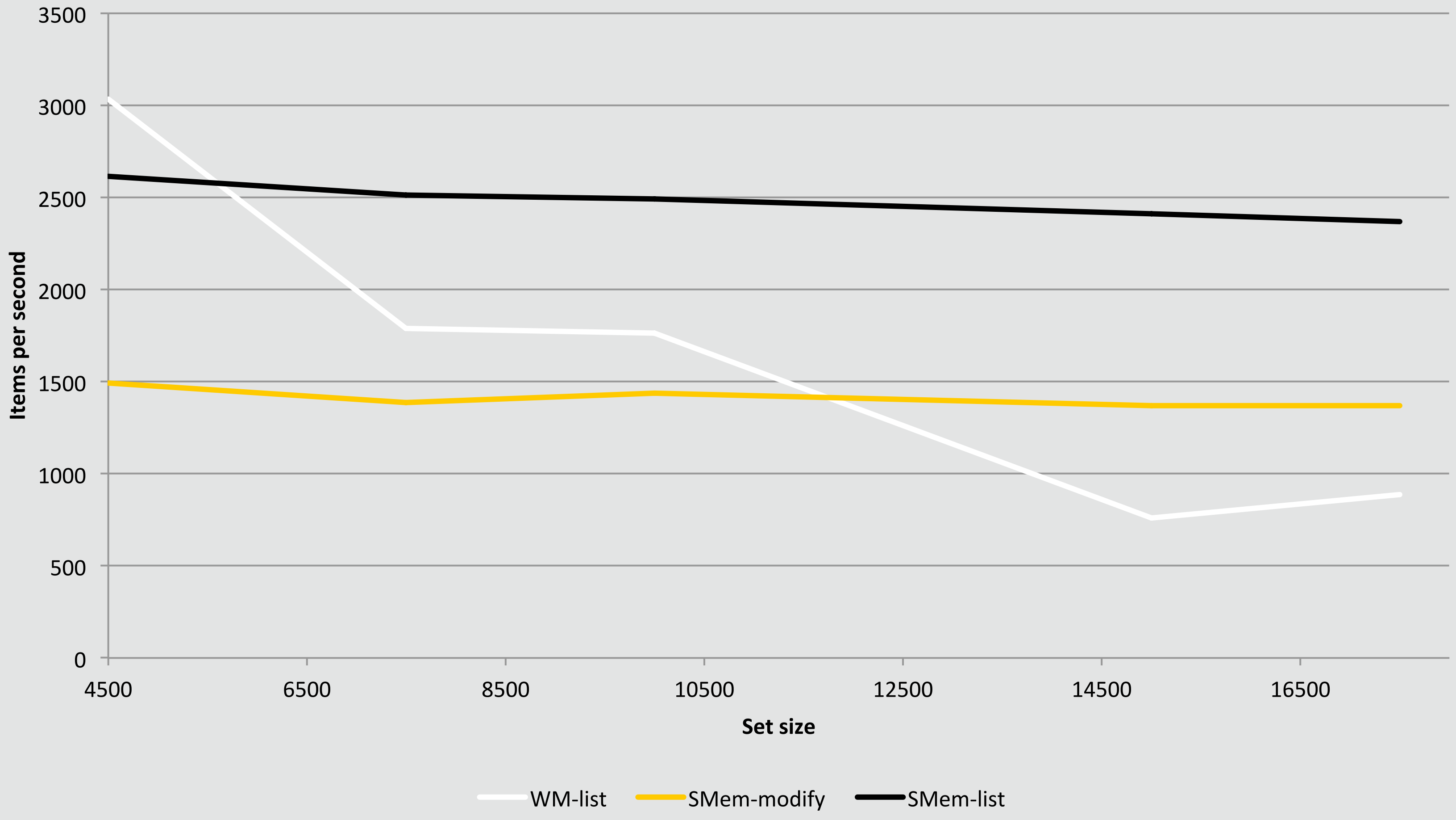


SOARTECH

Performance Results: Throughput (get 2 smallest values)



Performance Results: Throughput (get 2 smallest values)



Should I Use Semantic Memory?

Yes

- Simple rules too expensive
- Data learning
- Runtime query construction
- Can't use WM
  - Need a small WM
  - Can't maintain data in a WM list
- Want database guarantees
- Need to pre-load lots of data
- Need to maintain data across runs

No

- Simple rules work well
- Performance matters and don't need a general solution
- Need to do queries smem doesn't support
- Need a stable system

SOARTECH

More Information

- Soar 9.3.1 manual
- Bebot: https://github.com/daveray/bebot

Nuggets

- Smem really does work
- May be able to capture some common usage patterns in a reusable library
- Even when slow, still maintains reactivity
- Underlying database can be useful
- Starting to understand some use cases where it makes sense

Coal

- Requires more work to use than expensive rules
- Maybe not the best for extensive looping over sets
  - Prohibit approach doesn't really scale
  - Architectural support for iterators/cursors might be nice
- Best uses cases still not well understood