Architectures of Intelligence
Assignment 2

Group 13
Mirko Maragaira & Matteo Wohlrapp

December 4, 2021

Zbrodoff-experiment

Question a

What is the effect of manipulating the parameters below on the pattern of the results? Explain the reason behind it. It is useful to think in advance what the effect might be, instead of just trying it out. Of course, it is still a good idea to also try it out.

Retrieval Threshold, τ (:rt, typical values -1.0 to 2.0)

The retrieval threshold $:rt$ describes how much activation A_i chunk i needs to be retrieved from memory. This means that the higher $:rt$, the higher A_i needs to be and therefore the retrieval of chunk i gets more unlikely. This results in more counting operations of the model and since the extra counting operations decrease the speed of the model, it gets slower. **Figure 1** shows how an increase in $:rt$ affects the runtime of the model for an addend of three. The graph shows, as expected, a higher runtime for higher thresholds. Still, for each block there is a tendency to increase performance because the more often chunks are repeated, the higher their activation value and therefore the higher the probability of the chunk to be retrieved.

Latency Factor, F (:lf, typical values range from .1 to 2)

The latency factor $:lf$ influences how fast chunks can be retrieved from memory. Since counting requires additional requests to the retrieval buffer, it is expected, that especially for the first block the times should increase significantly with increasing $:lf$. **Figure 2** which depicts how the latency factors influence the runtime for the addend three over the different blocks, shows exactly that behavior. Especially in the first block where a lot of counting operations need to be done, the runtime is higher. Because $:lf$ is constant, the blocks two and three, where most of the chunks are already in memory and so only one retrieval request is created have a slightly higher value in comparison to the original data.

Activation noise, s (:ans range 0.1-0.8)

The influence of the activation noise $:ans$ is expected to slow down the model for higher values. That is because the more noise is added, the more likely are situations where A_i without noise would exceed $:rt$, but the added noise decreases A_i to a value below $:rt$. This also means that with higher $:ans$, the count productions need to be used more often which slows down the model. This can also be observed in **Figure 3**, where the model is especially slower for blocks two and three when $:ans$ increases.

Question b

The model you constructed uses two strategies sequentially. First, it fires production rules that try to retrieve the answer from memory. If that retrieval fails, it fires production rules that calculate the answer through counting. This model is inspired by an older model from Gordon Logan. This original model executed both strategies (counting and retrieval) in parallel. That is, it attempted to retrieve the answer and, at the same time, it started to count. The final answer was provided by the strategy that finished first. Would it be possible to implement such a parallel strategy in ACT-R? If yes, what would such a model look like? If no, why is it not possible?

In ACT-R, all modules work in parallel. This would imply, that a parallel execution would be possible. However, both the productions for counting and the production to check if the equation is already known to need access to the retrieval buffer. **Listing 1** and **Listing 2** show that both productions make a request to the retrieval buffer, one trying to retrieve information about the problem and the other to get the next letter. Since the buffer can only handle one request at a time, these two actions can not be executed in parallel. A solution to work around this problem would include the creation of a lot of extra chunks for memory. One could extend the problem chunk to not only hold information about the problem but also about the next letters. This means that one could combine the two requests into one. However, for this solution to work, every saved problem needs to be combined with every letter and next letter combination and also, the retrieval strategy should not be limited to exact hits but also to hits where only part of the request matches.

Question c

If you think about how the brain processes information, do you think it could do counting and retrieval in parallel? What would be requirement for this?

The brain is used to multitasking and working in parallel. Doing several things at once, such as driving a car while typing a number on the cellphone is feasible. What is important though is that the different activities do not require the same brain areas. This means, that if the knowledge for counting and the memory about the problem is stored in the same area, it can be hard for the brain to work on both in parallel. However, the brain, in general, is very adaptive and with training, counting and retrieving in parallel should be possible by connecting the counting process with the retrieval of the problem information.

A. Assignment 2

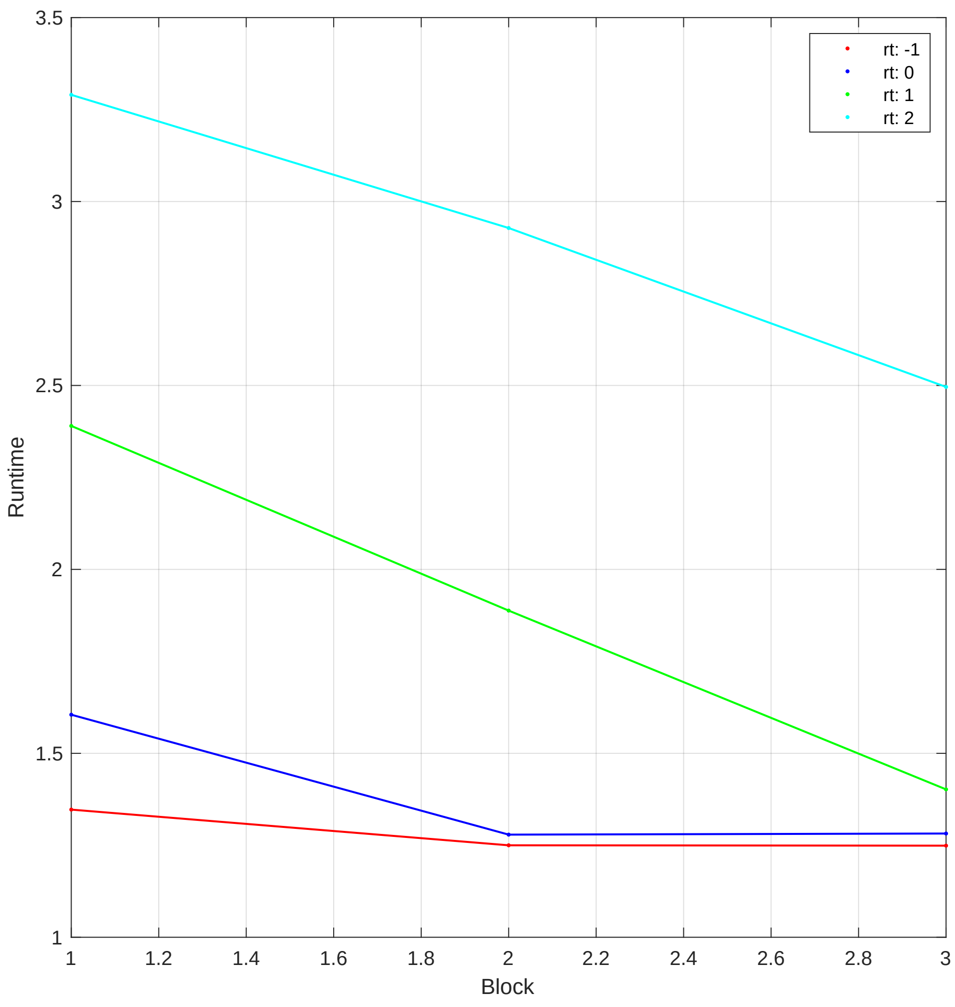

Figure 1: *Graph visualizing the runtime for each block. Different values of :rt are presented in different colors. All other parameters are set to the original values. In this case, three was used to add to the character.*

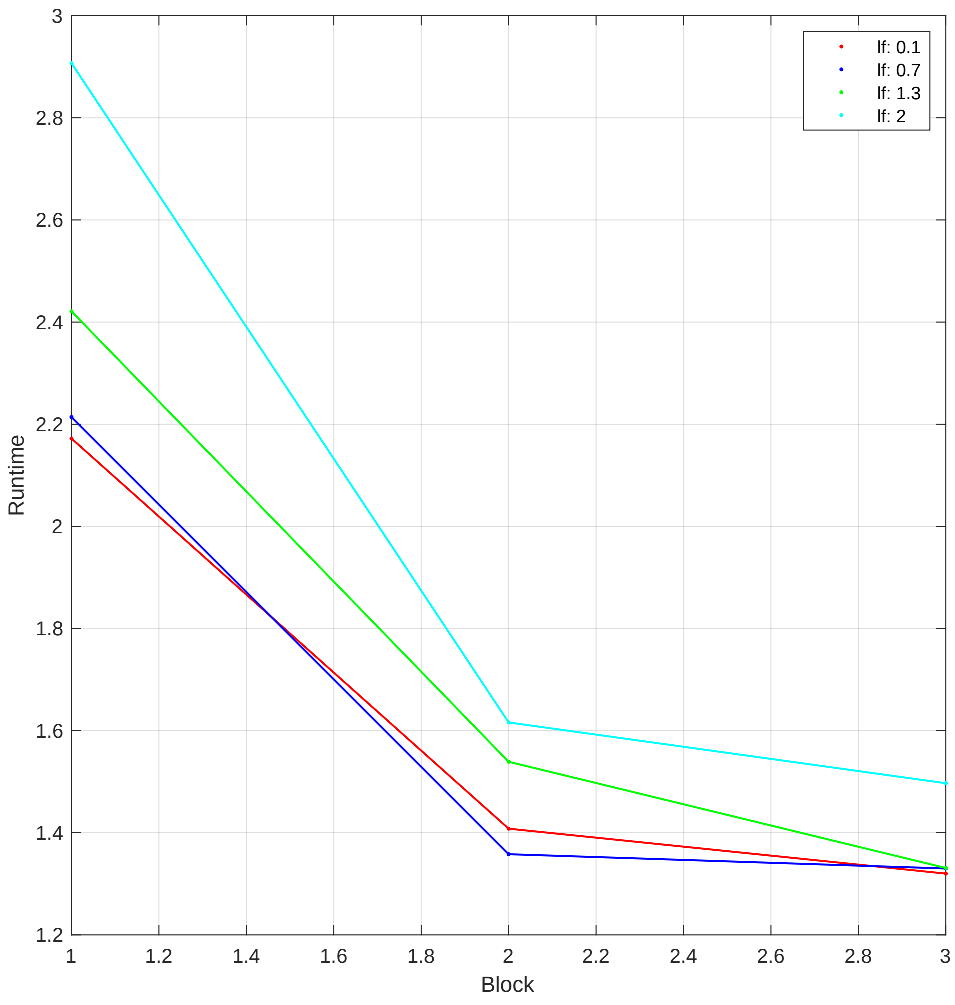

Figure 2: *Graph visualizing the runtime for each block. Different values of :lf are presented in different colors. All other parameters are set to the original values. In this case, three was used to add to the character.*

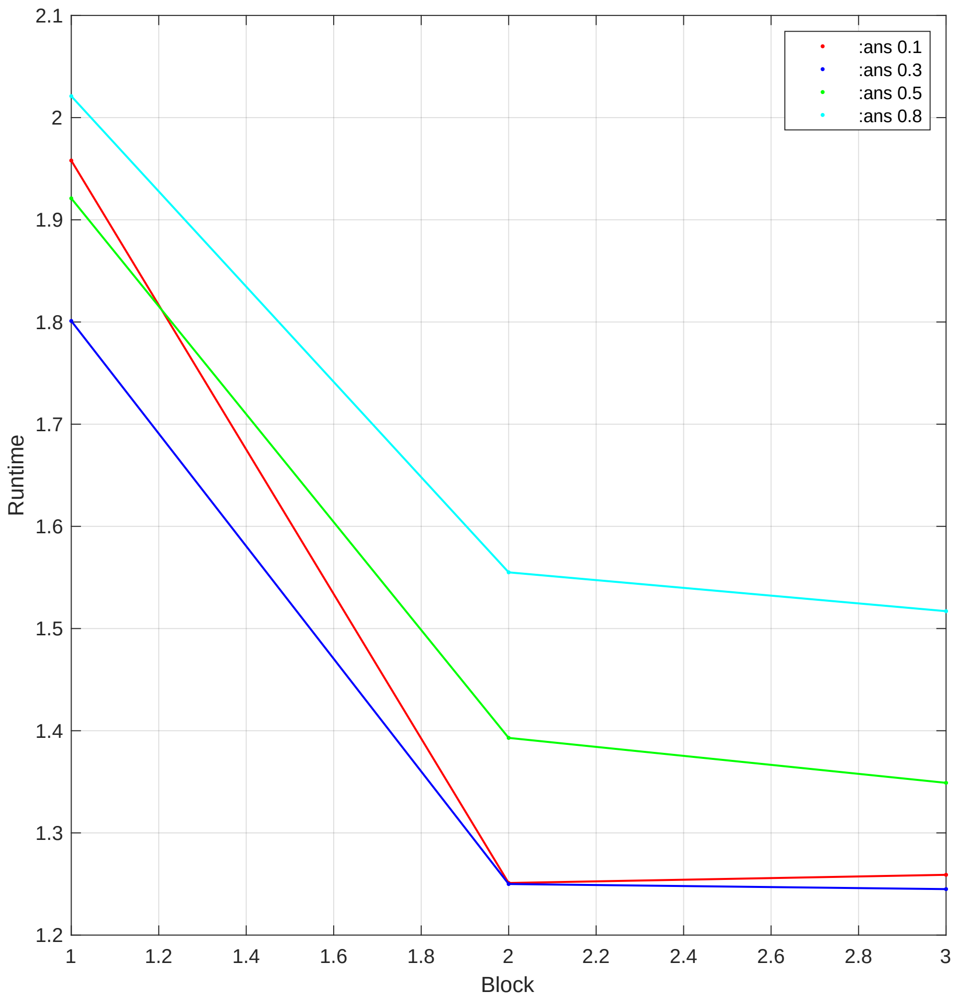

Figure 3: *Graph visualizing the runtime for each block. Different values of :ans are presented in different colors. All other parameters are set to the original values. In this case, three was used to add to the character.*

Listing 1: *encode-third production*

```
; changed
; method to get the letter value for the 3rd visual object.
; The conditions were not modified
; Only the modifications to buffers were changed:
;    - state of result buffer from count to memory -> that way the
   newly defined productions are selected next
;    - retrieval request for the previously attended letters -> to
   check if we have information about this combination saved
(p encode-third
   =goal>
     isa          goal
     state         encode
     target        nil
   =retrieval>
     letter        =let
     vocal-rep     =word
   =imaginal>
     arg1          =a1
     arg2          =a2
   ?vocal>
     preparation  free
   ==>
   +vocal>
     cmd           subvocalize
     string        =word
   =goal>
     target        =let
     state         memory
   +retrieval>
      isa       problem
      arg1      =a1
      arg2      =a2
   =imaginal>
   )
```

Listing 2: *start-counting production*

```
; changed
; production which is selected if the combination of character and
   number was not previously seen, which is indicated by a buffer
   failure
; in that case we have to count as we did in the original version ->
   state is set to counting
; prepare counting by setting the result accordingly
(P start-counting
   =goal>
     ISA          goal
     state        memory
   =imaginal>
     isa          problem
     arg1         =a
     arg2         =val
   ?retrieval>
     buffer    failure
==>
   =imaginal>
     result       =a
   =goal>
     state        counting
   +retrieval>
     ISA          letter
     letter       =a
   )
```