Production Compilation

Niels Taatgen
University of Groningen
Artificial Intelligence

At last year’s PGSS two questions remained concerning production compilation

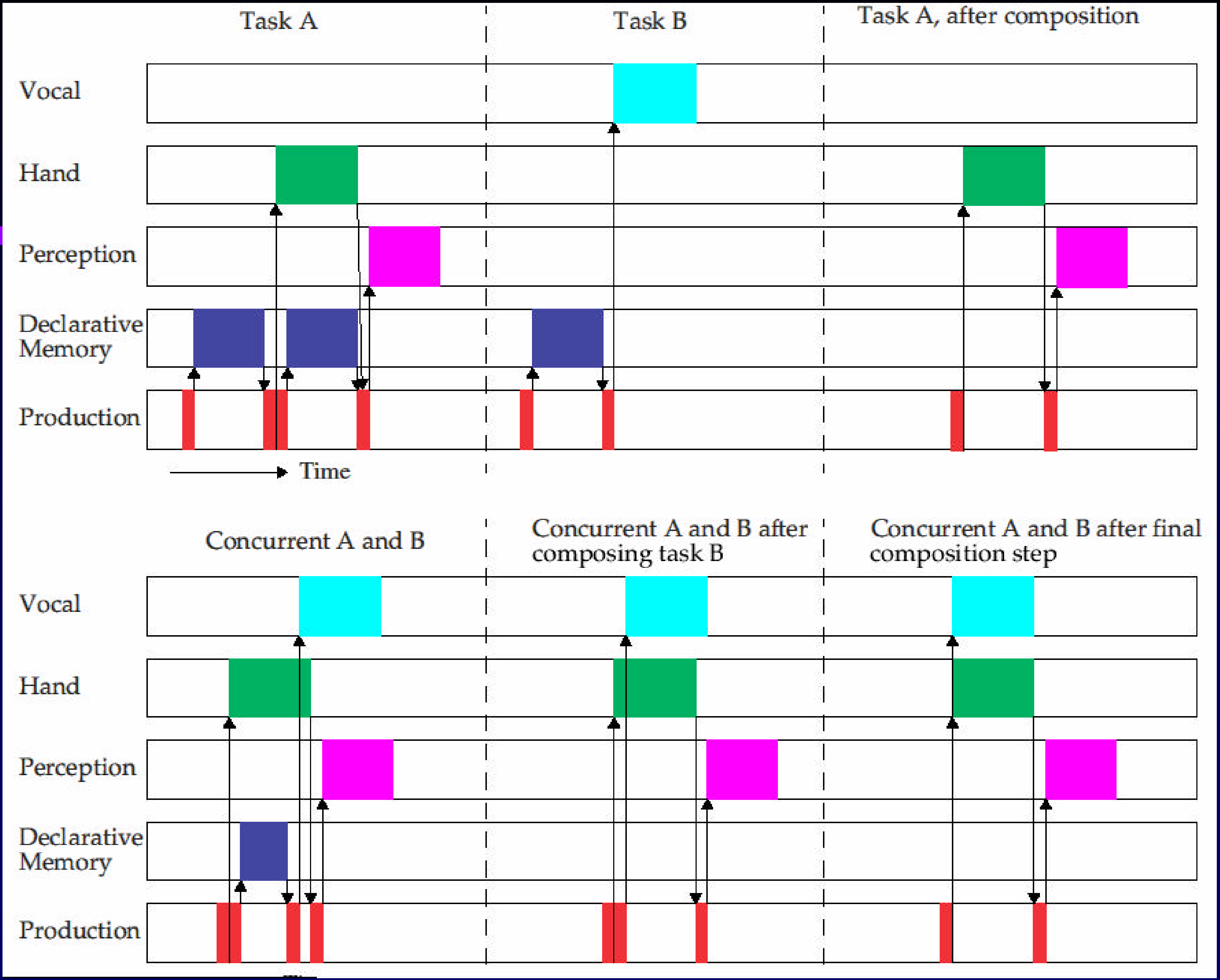
- ✍ How does production compilation handle interaction with ACT-R/PM?
- ✍ How are the parameters learned?

Interaction with ACT-R/PM

- To explore this interaction, Frank Lee and I updated my ACT-R 4 non P/M model of the Kanfer-Ackerman Air Traffic Controller task to an ACT-R 5 model with ACT-R/PM
- This proved to work really well
- Except that ACT-R at some point didn’t get any faster anymore, while participants still improved
- So we decided that a tighter integration of perceptual, cognitive and motor actions was needed

Integration of Cognitive, Perceptual and Motor actions

- To use time as efficiently as possible, you should keep all the modules as busy as possible
- So while a declarative retrieval is going on, you might want to initiate an eye-movement
- ACT-R should do “internal” multi-tasking



Task A
Task B
Task A, after composition
Vocal
Hand
Perception
Declarative Memory
Production
Time
Concurrent A and B
Concurrent A and B after composing task B
Concurrent A and B after final composition step
Vocal
Hand
Perception
Declarative Memory
Production

Problems with this approach

- Hard to inspect whether a certain module is "free" (especially retrieval)
- When you do two tasks at the same time:
  - Do you represent them as two goals, making it necessary to switch between them?
  - Or do you represent them both in a single goal?
  - Or, will this be something for a module behind the goal, the "intention-model"?

Parameter Learning: what we want from it

- Gradual introduction of new rules: after the first opportunity for the rule to be learned, it should take some more practice or experience before it will regularly be used
- Evaluation of new rules
  - If the new rule is better than the parents, it should eventually fire whenever it matches
  - If the new rule is worse than the parents, it should eventually not fire anymore

Current scheme

- A new rule is given prior values for its successes, failures and efforts based on the parent rules
- A penalty is added to the cost of the rule to ensure that it is gradually introduced

Current implementation

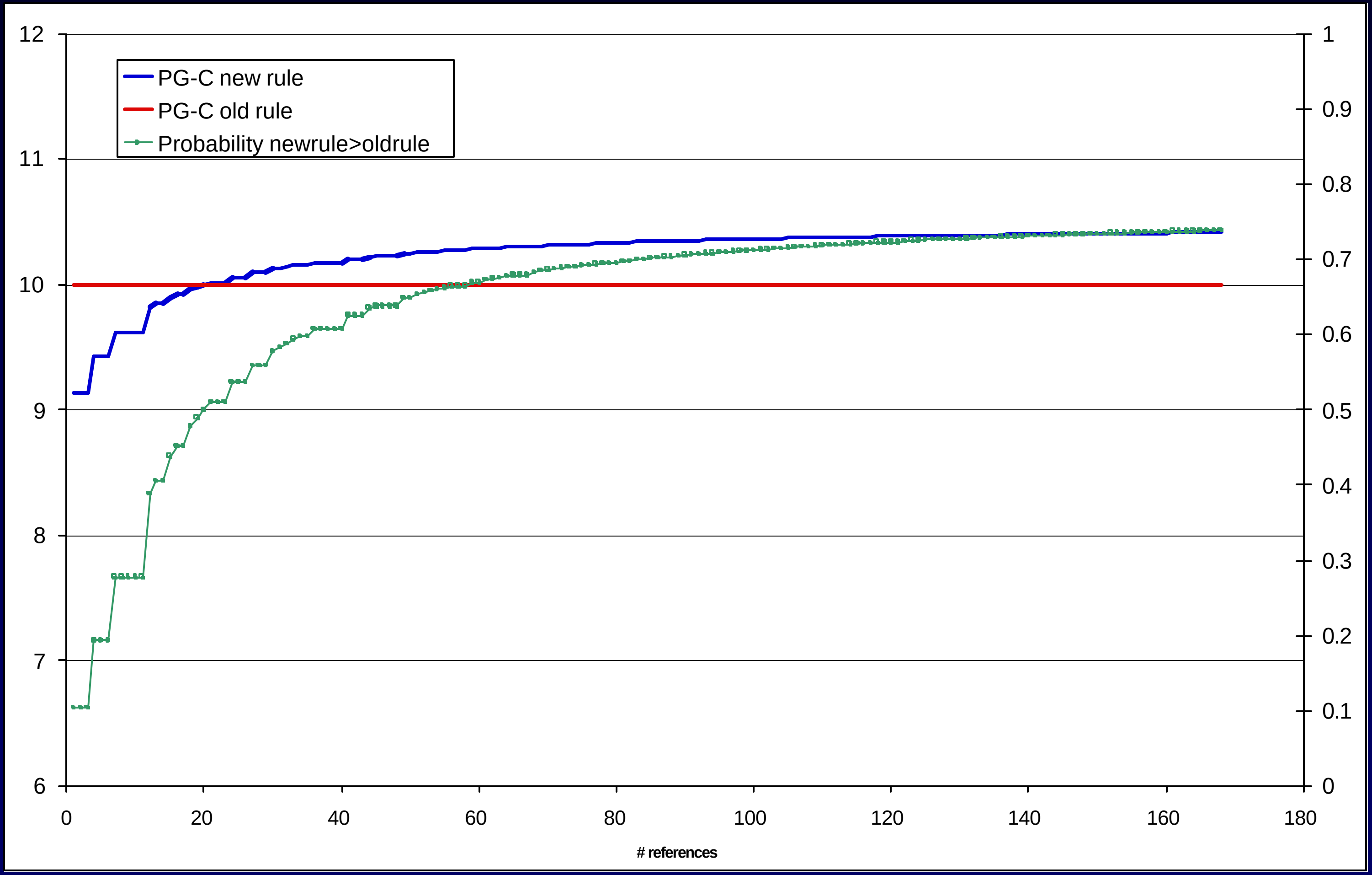
- Basic Utility equation:

$$\text{Utility} = \frac{n \cdot \text{priorUtility} + m \cdot \text{experiencedUtility}}{n + m}$$

- In the current implementation

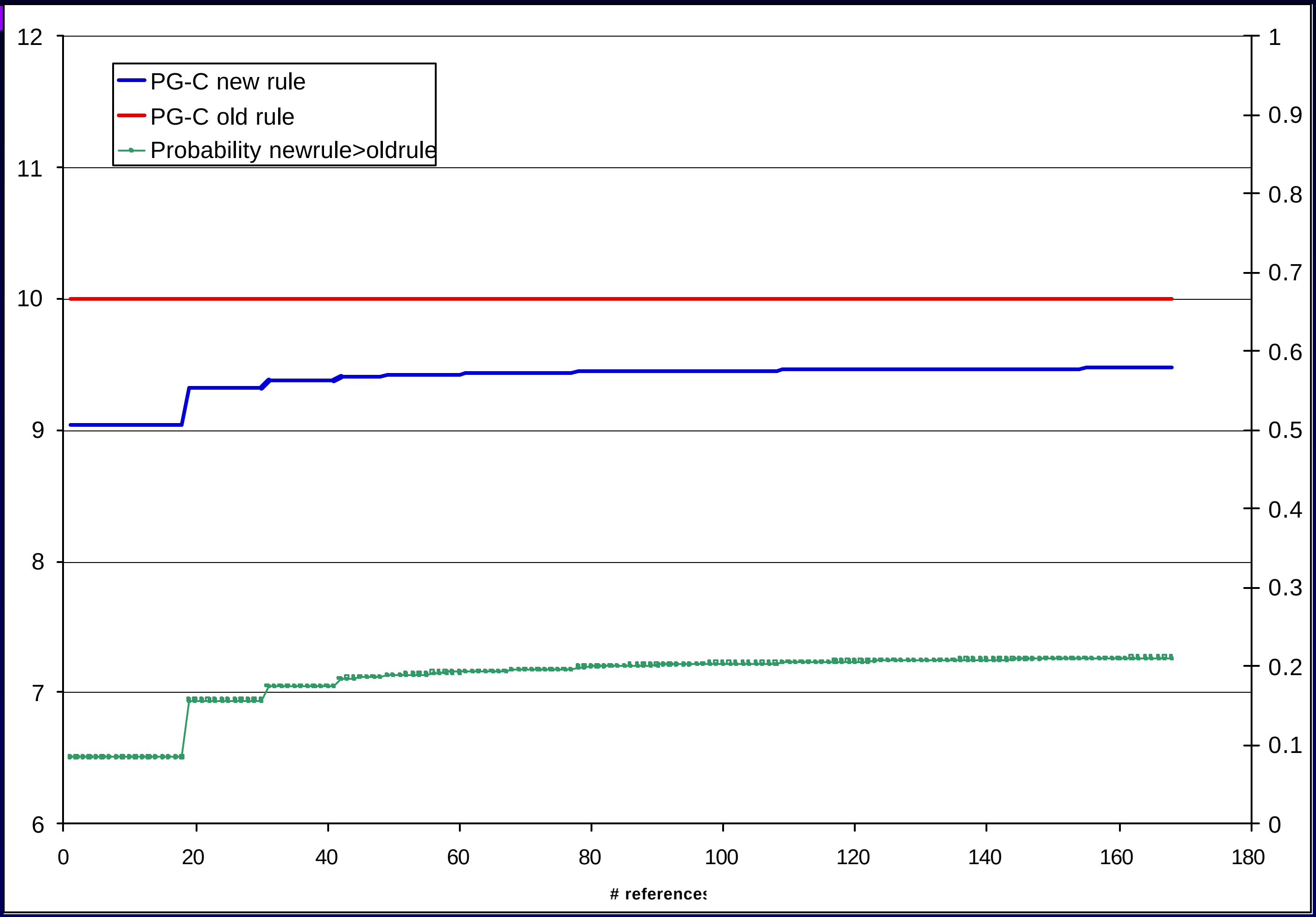
priorUtility = parentUtility - costPenalty

Parameter learning: example



Noise = 0.4 Initial experiences = 10 Penalty = 1.0 Utility new rule = 10.5

Parameter learning: example



Noise = 0.4 Initial experiences = 10 Penalty = 1.0 Utility new rule = 9.5

Problems with current implementation

- Level of noise is critical
  - If it is too low, the new rule will never be tried
  - If it is too high, the rule will be introduced too fast
- Eventually, the new rule will not dominate if it is better, nor will it fade away if it is worse

Current implementation

- Basic Utility equation:

$$\text{Utility} = \frac{n \cdot \text{priorUtility} + m \cdot \text{experiencedUtility}}{n + m}$$

- In the current implementation

priorUtility = parentUtility - costPenalty

Proposal

- Basic Utility equation:

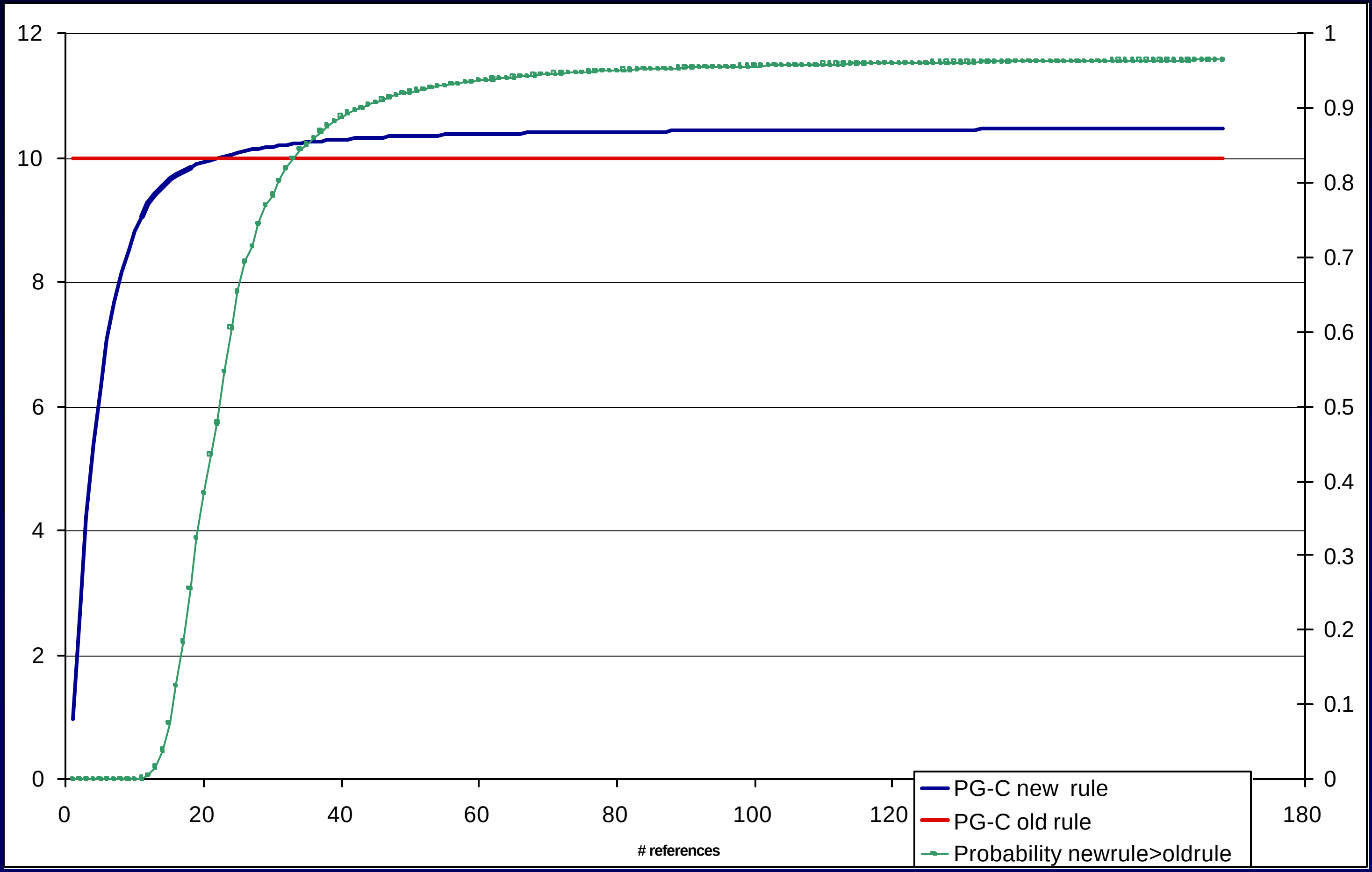
$$\text{Utility} \; ? \; \frac{n \,\text{`}\, \text{priorUtility} \; ? \; m \,\text{`}\, \text{experiencedUtility}}{n \; ? \; m}$$

priorUtility = 0 (initially)

Each time the rule is recreated:

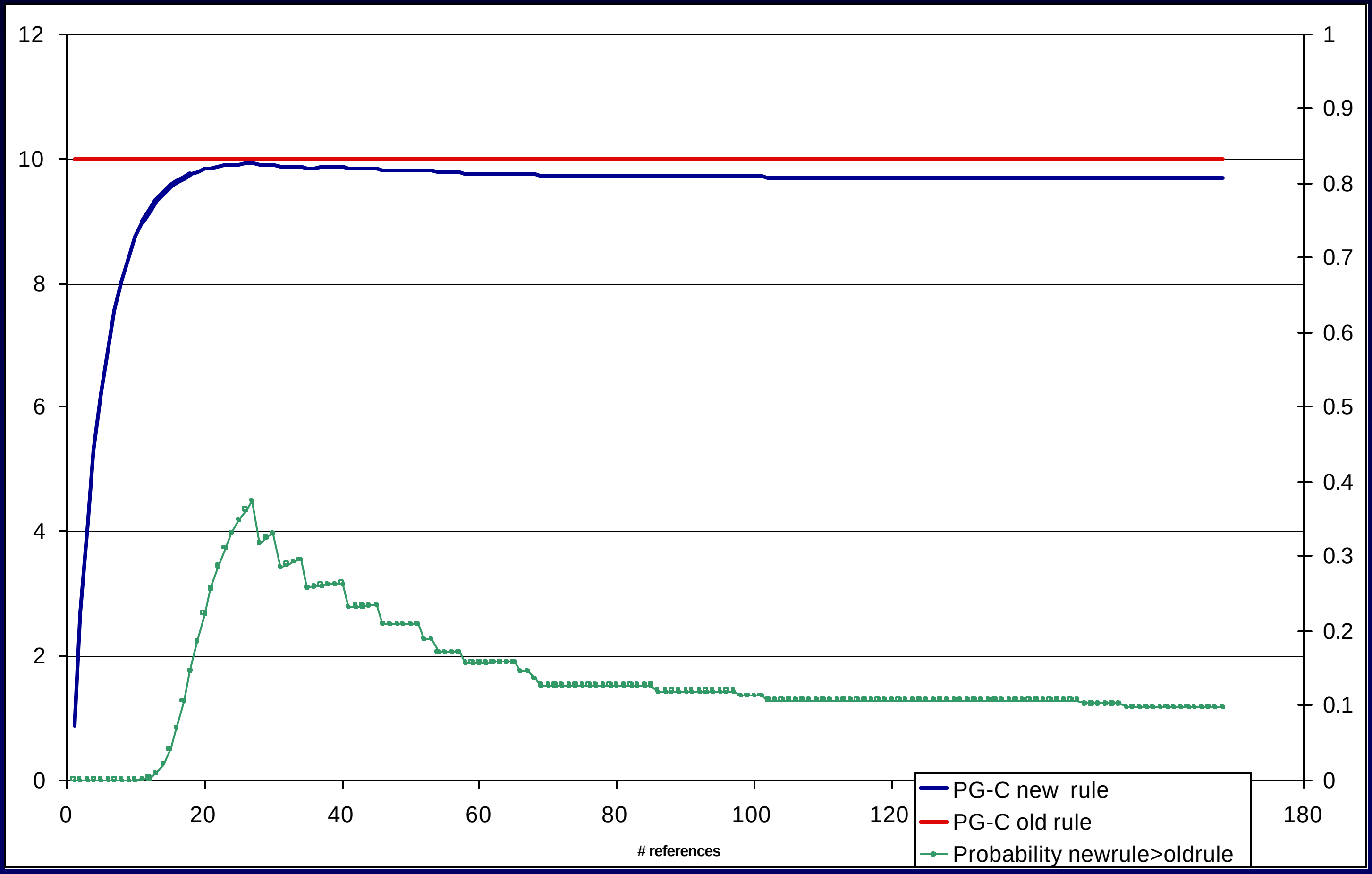
priorUtility = priorUtility + ? (parentUtility - priorUtility)

Parameter learning: example



Noise = 0.1 Initial experiences = 10 ? =0.2 Utility new rule = 10.5

Parameter learning: example



Noise = 0.1 Initial experiences = 10 ? =0.2 Utility new rule = 9.5

Evaluation

- It takes a while for the new rule to be learned
- Rules that are recreated more often are learned faster
- The number of free parameters is the same as the current implementation (2)
- It is more robust, as it is less sensitive to the level of noise