



# An Analysis of the Performance of the NGS (New Goal System)

Jacob Crossman

[jcrossman@soartech.com](mailto:jcrossman@soartech.com)

Soar Workshop 2007

Work by: Thom Bartold, Mike Quist

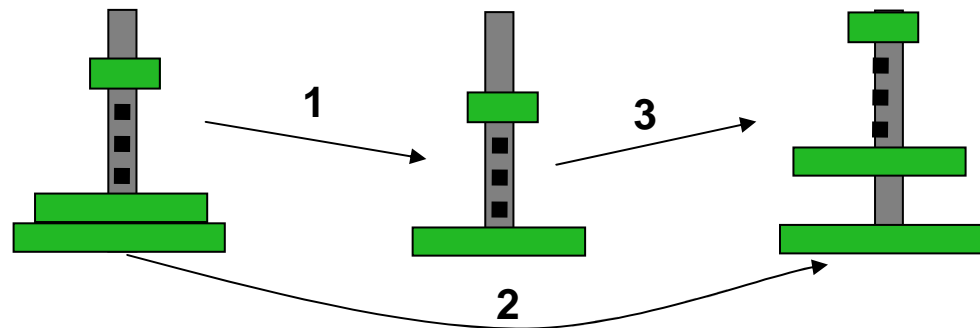
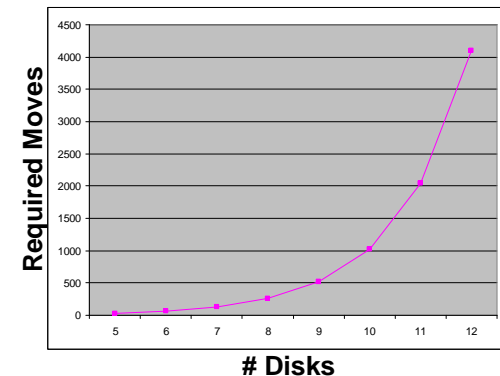
Support: Glenn Taylor, Dave Ray

# Introduction







- Historically, there have been two major categories of goal management processes in Soar programs:
  - The built-in sub-states (impasse driven subgoaling) (e.g. the “University of Michigan Approach”)
  - Declarative goal representations on the top state (e.g. “Radical Randy”)
- Soar Technology has developed a library of code for the declarative goal approach we call the “New Goal System (NGS)”
- Motivation for this talk: we wanted to know how the NGS will scale as problems become more complex and require more goals
  - This is mainly a performance evaluation
  - This is NOT an evaluation of usability or multi-tasking, etc

# The Problem: The Towers of Hanoi Algorithm

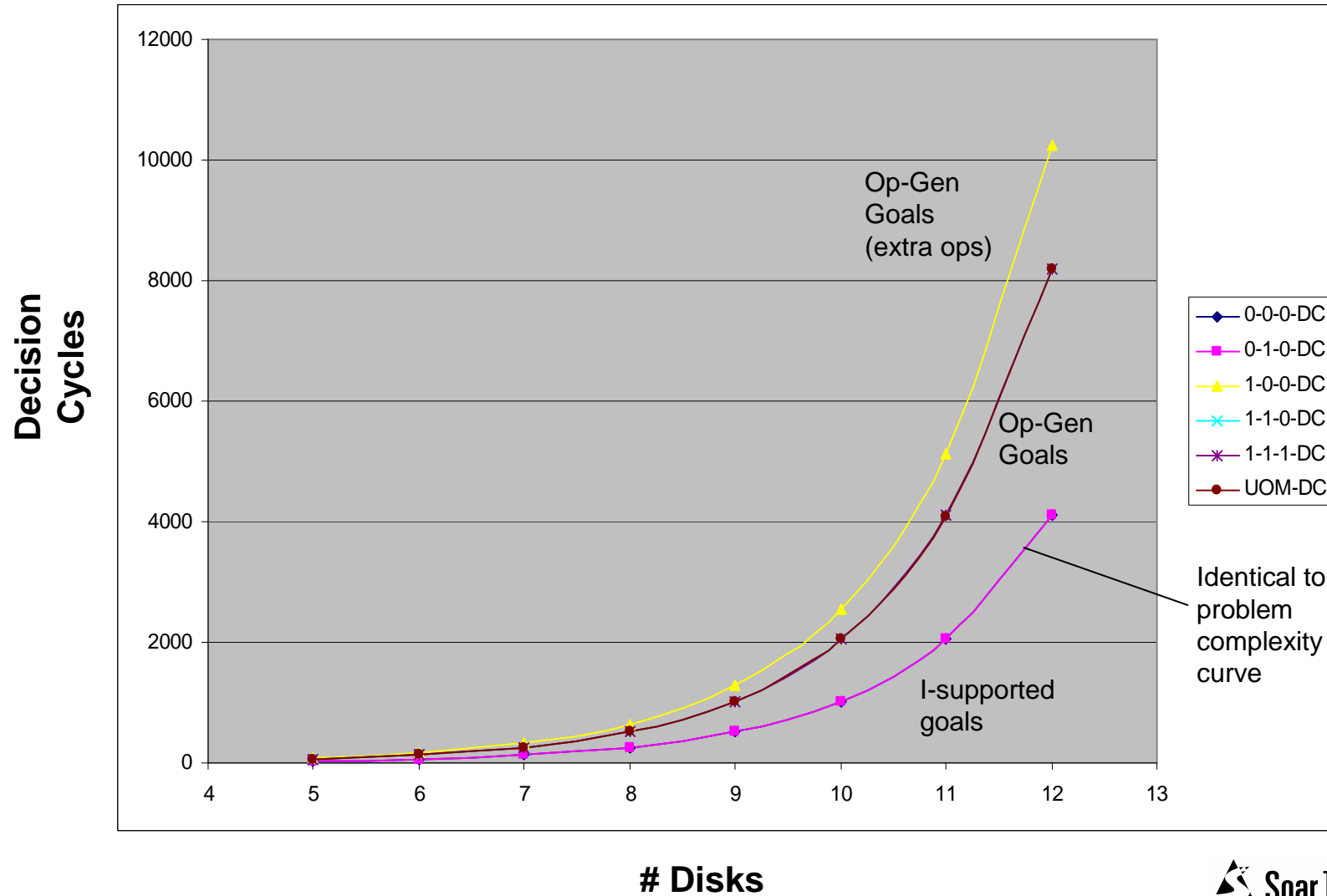
- A recursive algorithm which optimally solves Towers of Hanoi for any number of disks
  - Use means end analysis
  - Move-stack is decomposed into three actions, move smaller stack to free peg, move disk, move smaller stack from free peg
- Start state is all disks on one tower
- Final state is all disks on goal tower
- Requires  $2^n - 1$  disk moves



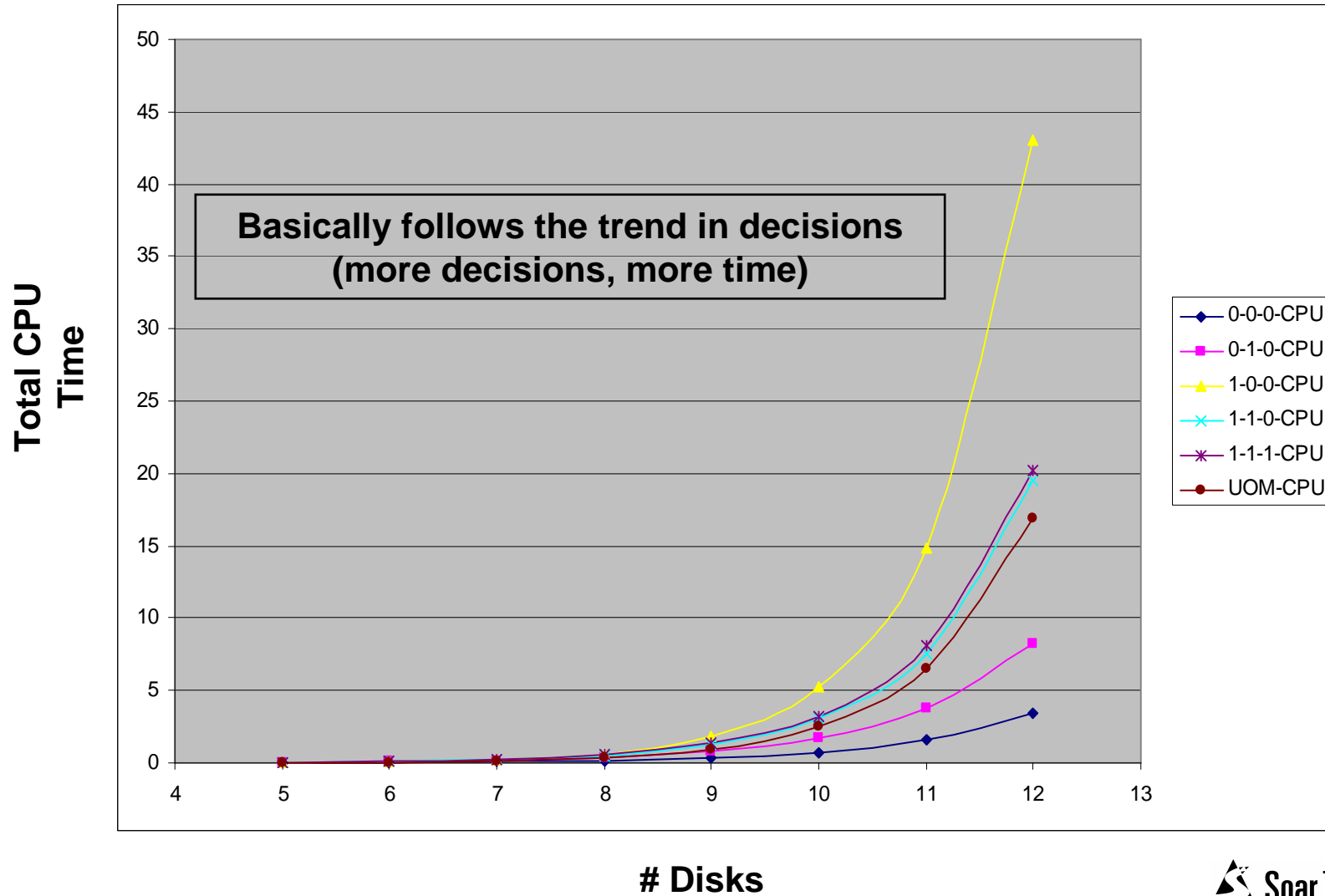
# The Experiment configurations

	Operators Create Goals	Uses Named Conditions	Uses Transforms	Shorthand
NGS-Light (i-support) (Radical Randy)	No	No	No	0-0-0 
NGS-Conditions (i-support)	No	Yes	No	0-1-0 
NGS-Light (w. operators)	Yes	No	No	1-0-0 
NGS-Conditions (w. operators)	Yes	Yes	No	1-1-0 
NGS-Full (uses transforms w. operators)	Yes	Yes	Yes	1-1-1 
UM – UOM	Yes	No	No	UOM 

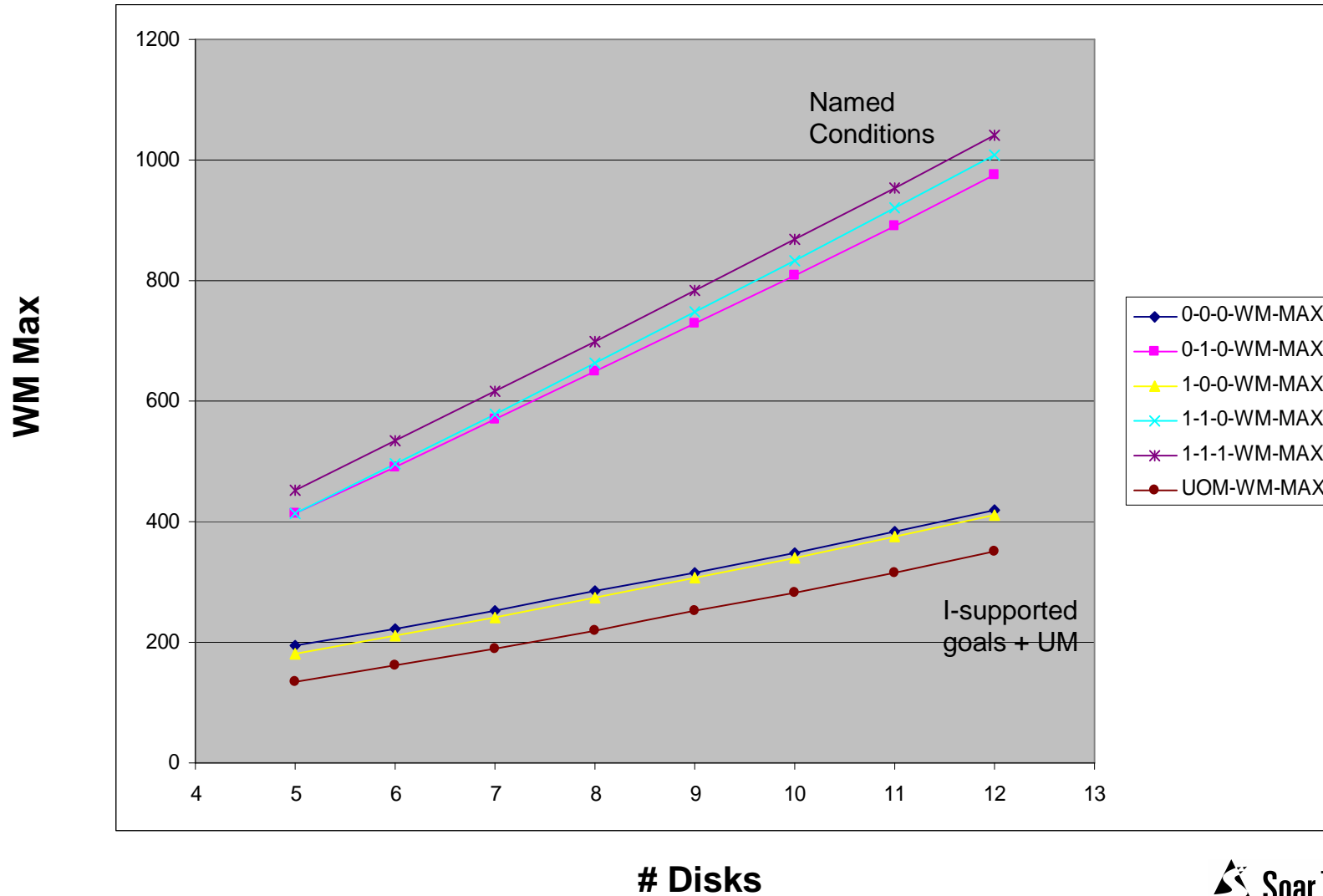
# Decision Cycles



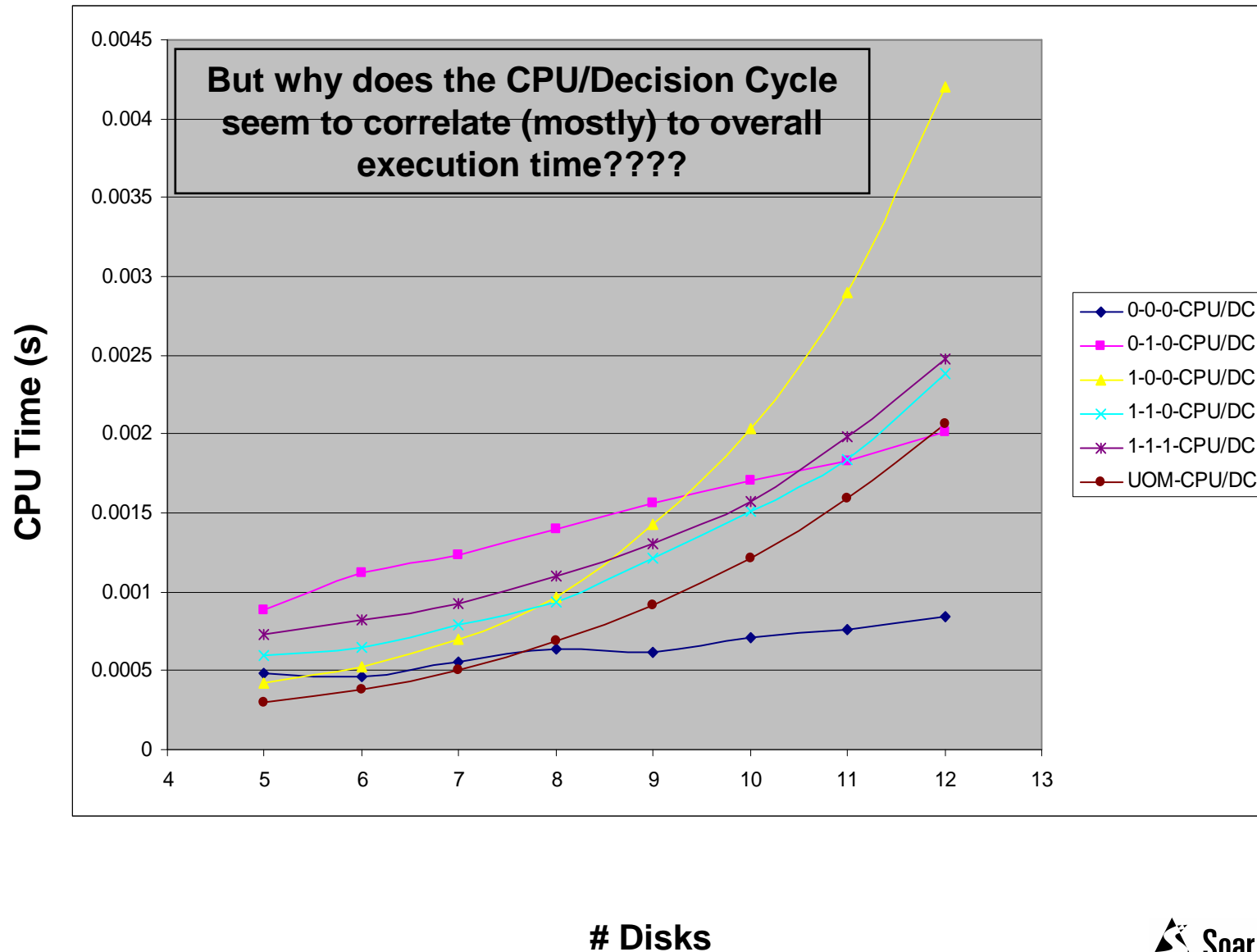
# CPU time, Timeout=0



# WM max



# CPU/Decision Cycle (DC)

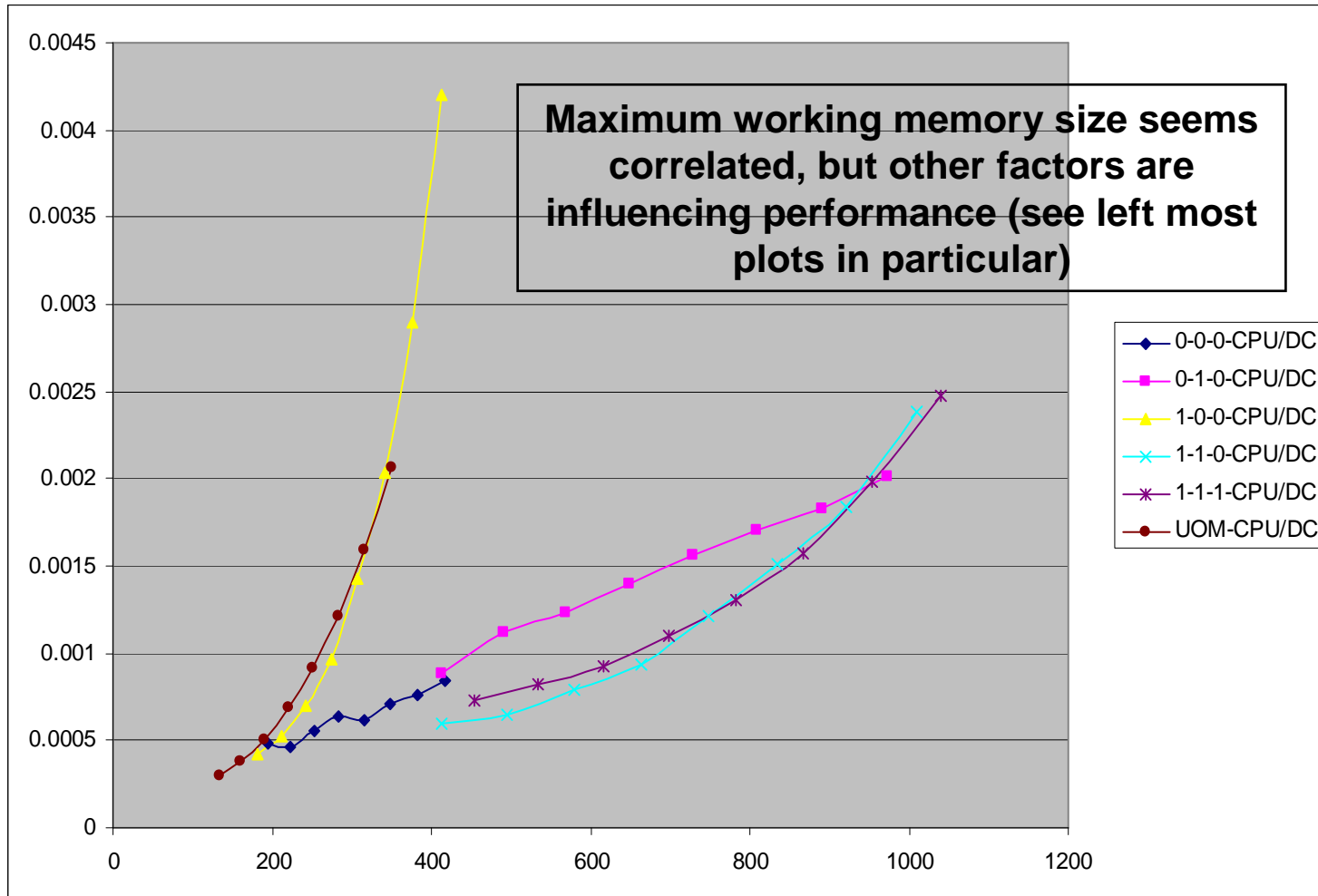




# Architecture Factors

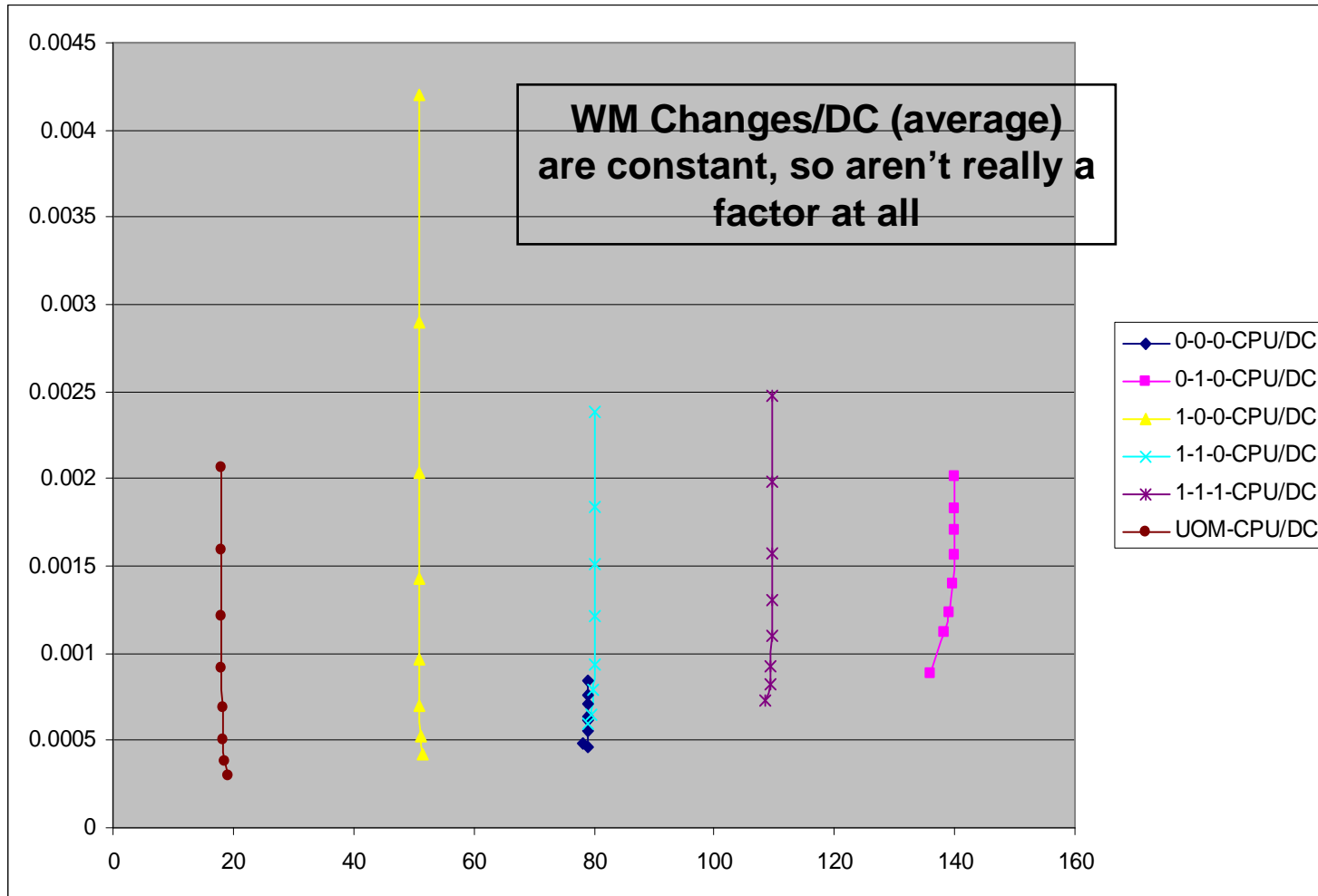
- Can we find the core factors of the architecture impacting the performance?
- Look at:
  - WM Size v. CPU/DC
  - WM Changes v. CPU/DC
  - Memories v. CPU/DC
  - Activations (RETE alpha/beta nodes) v. CPU/DC

# Working Memory Max v. CPU/DC



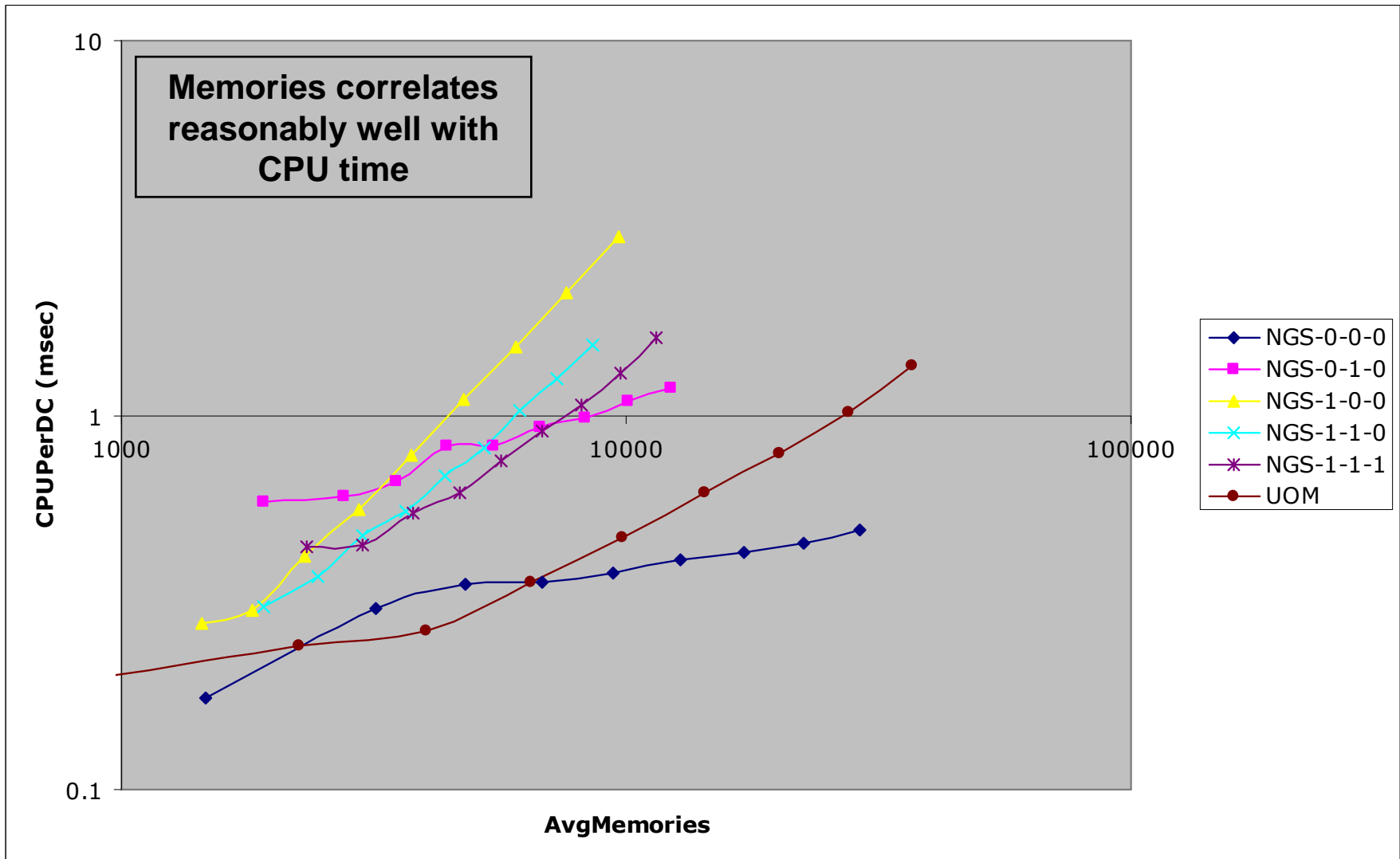
WM-MAX

# Working Memory Changes/DC v. CPU/DC

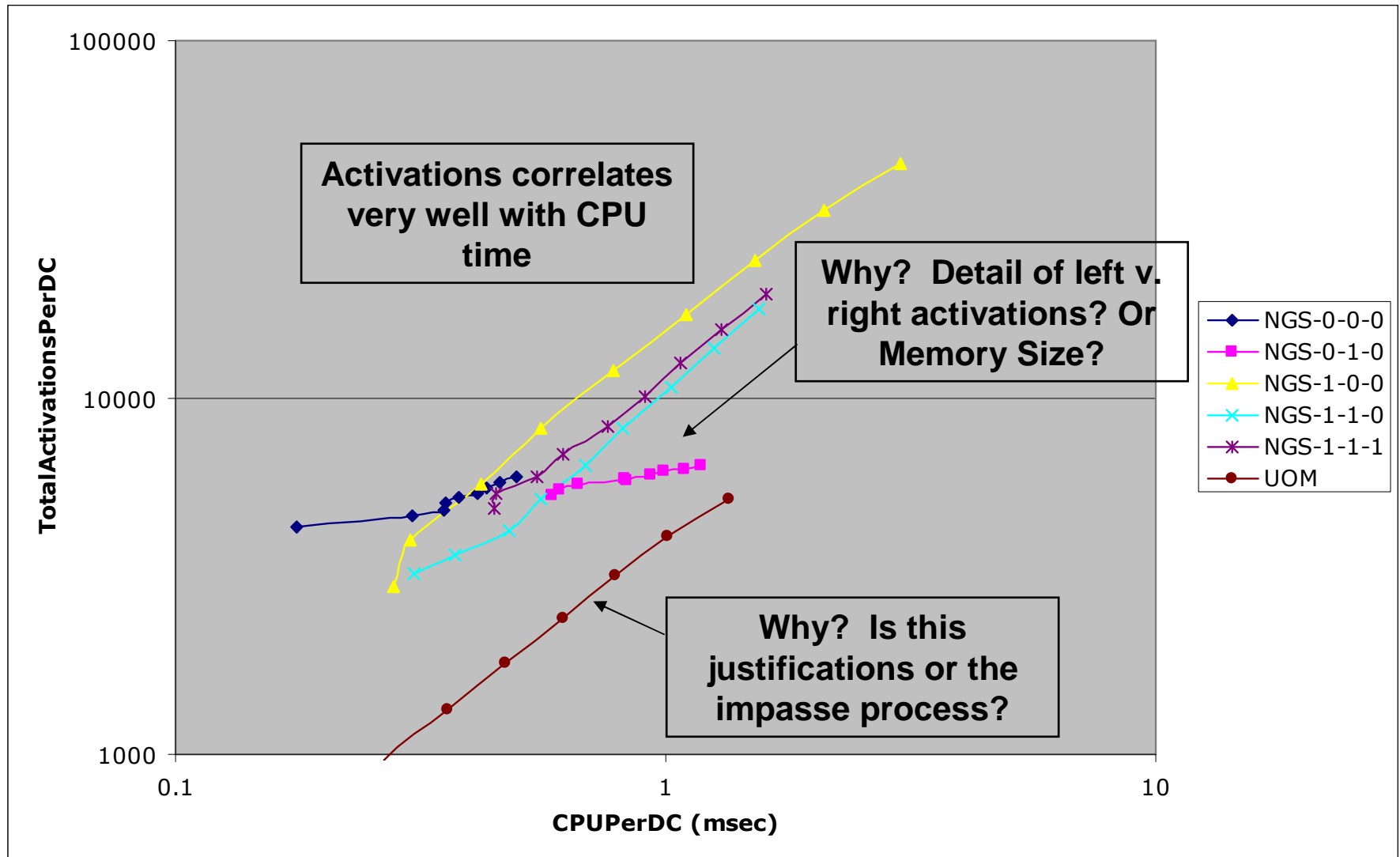


WM Changes/DC

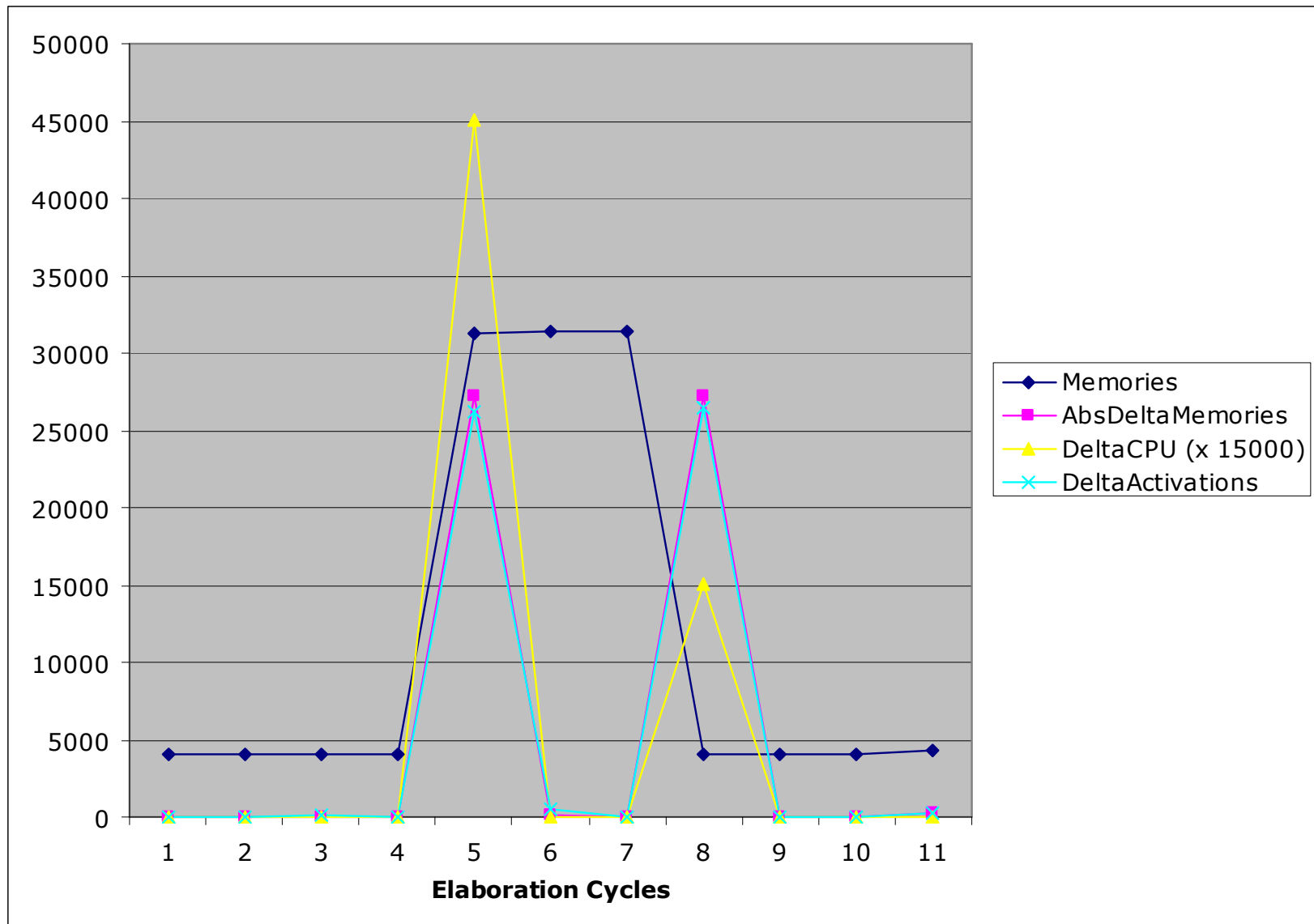
# Average Memories v. CPU/DC



# Activations v. CPU/DC



# Time Series Look (Micro-Details)



# Conclusions

- Comparison of Approaches (CPU)
  - NGS lite (Radical Randy) can be as fast or faster than UM approach
  - Adding significant numbers of named (declarative) conditions is a significant constant offset to CPU performance and adds significantly to memory size
  - Not sure goal approach is overriding factor in performance
- Leave named conditions for cases when you need the declarative aspects (e.g. communication and introspection)
- Rules of thumb for goals
  - Use i-supported goals if you can
  - Use :o-support w. achievement condition for faster o-support
  - Keep timeouts for goal deletion very short (0 if possible)

# Conclusions on Architecture-Level Factors

- **Rule of Thumb:** Soar systems scale with changes to working memory (ONLY PARTIALLY CORRECT)
- **Corrected Rule:** Soar systems scale with changes to RETE (activations) which are triggered by changes to WM but can be significantly larger than the WM change itself
- We already know that multi-value attributes are expensive, but lots of LHS conditions can also be expensive
  - These are actually related (multi-valued attributes impacts left and right activations and their cost)
- **Suggestions:**
  - Make operator applications simple (do work at/before proposal time)
  - Make substate construction simple (keep non-changing information on the top state)
  - In general, write smaller productions
  - Order your production logic so stuff most likely to change is at the bottom of the production LHS
- *Side Note:* Use variabilized attributes for multi-valued attributes
  - Create as `<set> ^<attr> <val> +) -- <attr> will be unique ID`
  - Test as `(<set> ^<attr> <val>) -- <attr> will bind to ID`

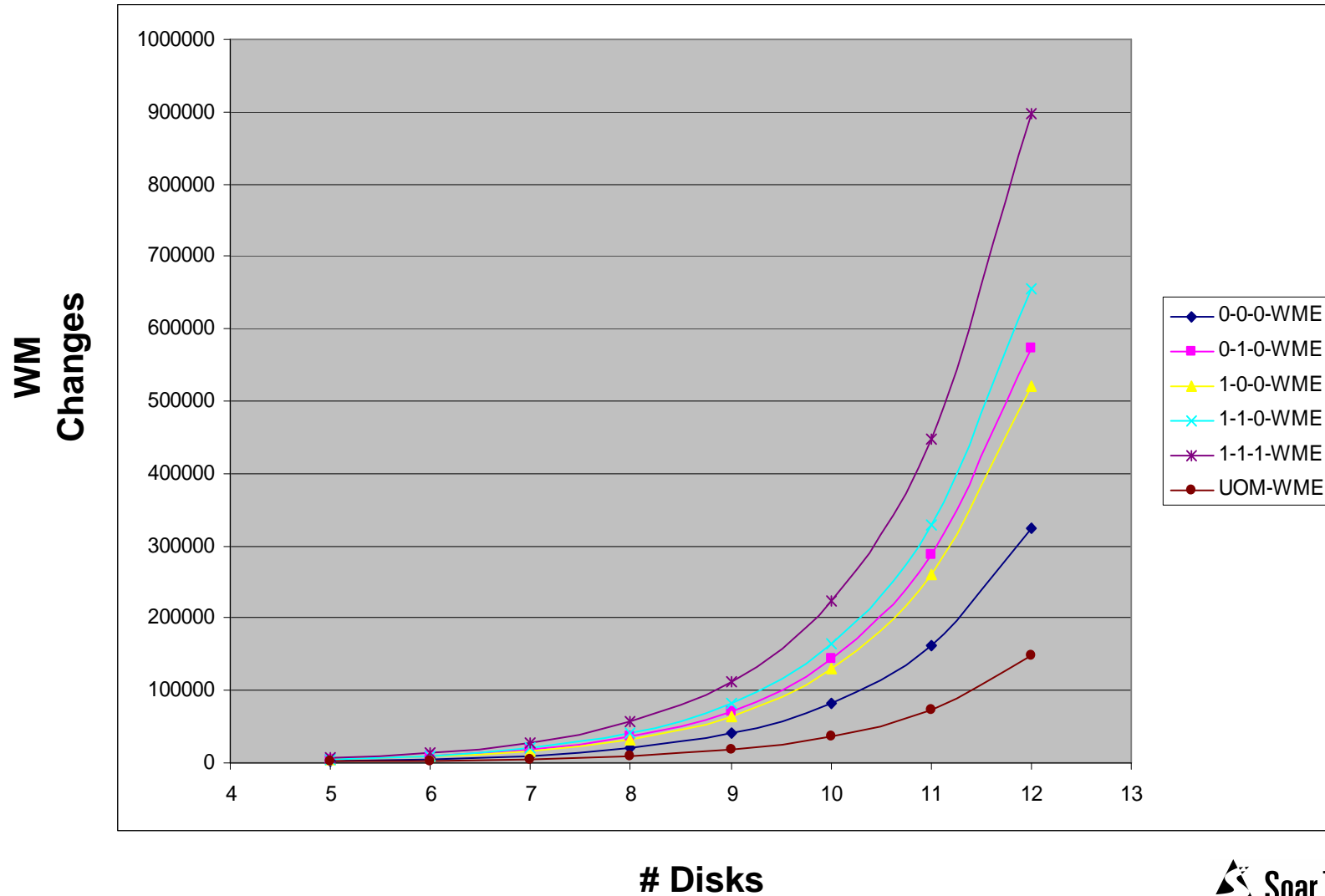


## Suggested Further Analysis/Work

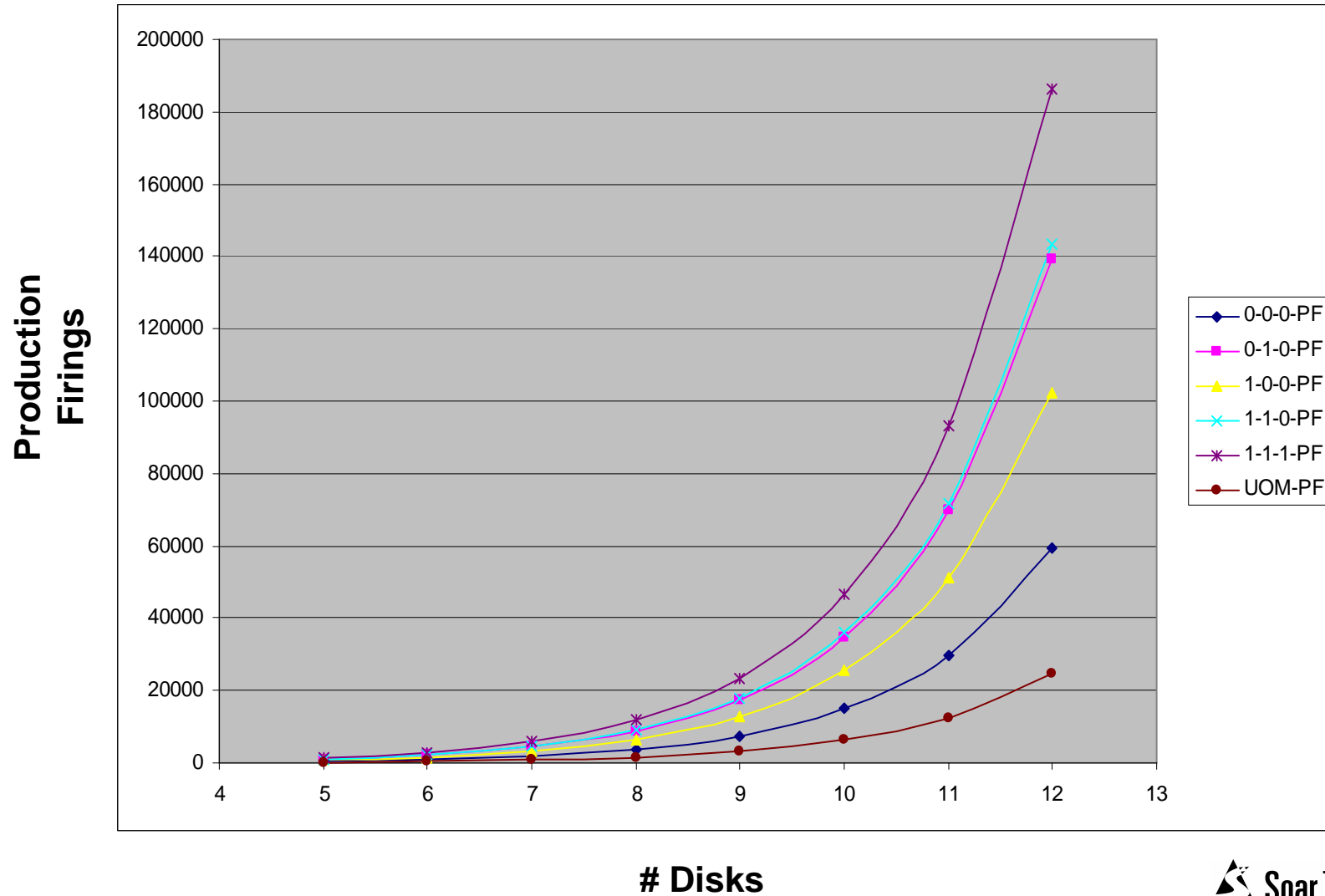
- What impact are justifications (and other aspects of the substate process) having?
- Details:
  - Can we say more about exactly how to organize LHS conditions?
  - How much of an impact does WM Size itself really have (and under what conditions)
  - Could we write preprocessors or runtime analysis tool to enable automatic ordering of LHS conditions?
- Suggestion:
  - Include Memories Changes to the available stats for productions (so developers can see which productions are expensive)

# Backup

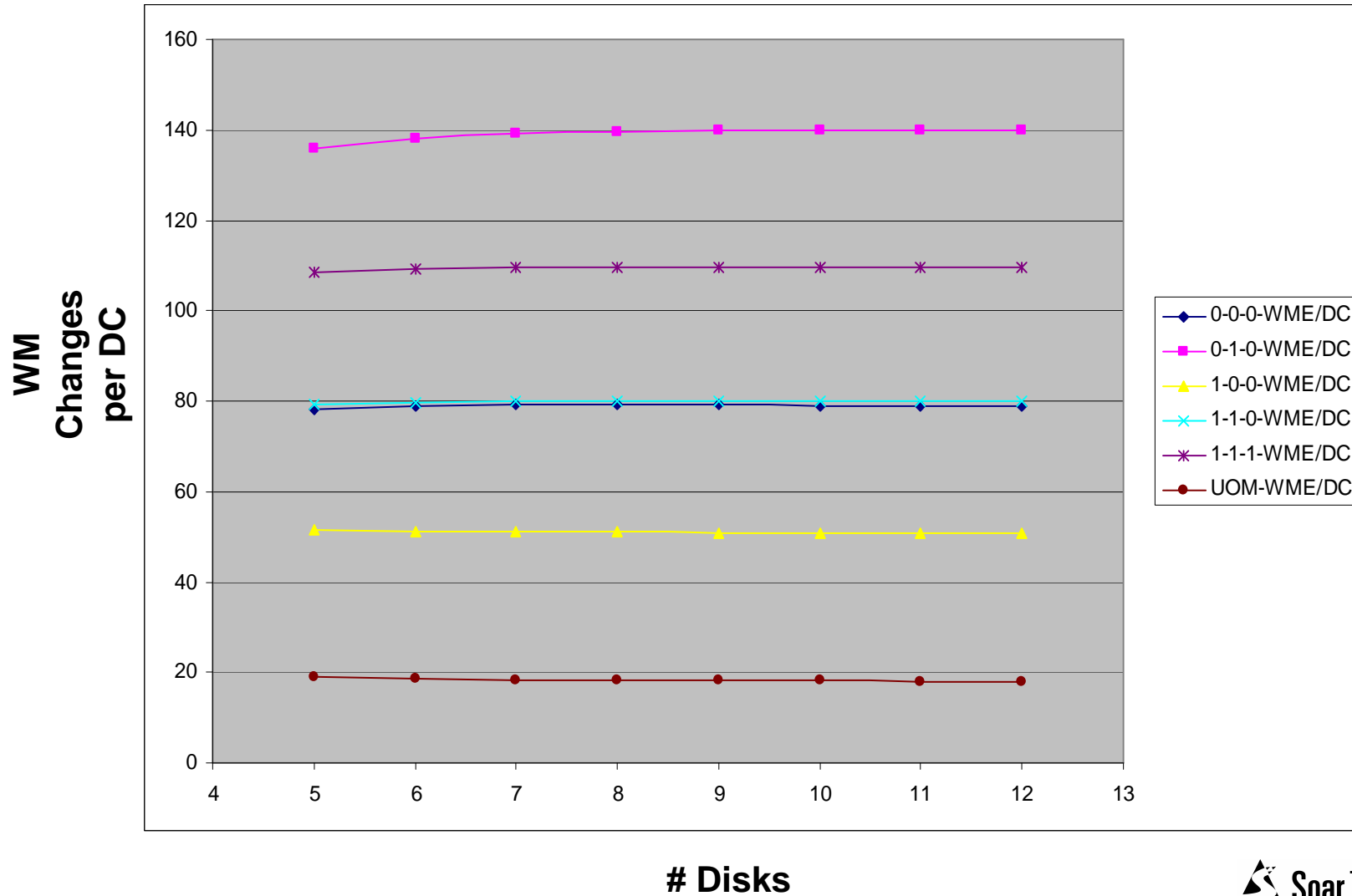
# WM changes



# Production Firings



# WM changes per DC



# CPU time, Timeout=0

