An Empirical Analysis of Algorithms for Sparse Bayesian Linear Regression

Gustavo Lacerda and Chris Nell

Outline

- Discussion of optimization with different priors
- Our Problem: Bayesian Sparse Linear Regression
- FBMP
- SSS
- Choosing Evaluation Measures
- Experiments
- FMOSS
- More Experiments
- Conclusion / Future Work

MAP inference in a Bayesian setting

- When your sample size is small relative to the space of models, a maximum likelihood estimate will almost always overfit.

- Using Bayesian priors is one way to *regularize*, i.e. make the estimate more stable.

- Instead of MLE, we use Maximum A Posteriori (MAP)

Large Model Spaces

- Sometimes we deal with very large model spaces, and exhaustive searching is intractable.

- So we use heuristics!

Prior x Search Algorithm Pairs

- When using off-the-shell implementations, it can be difficult to disentangle prior from algorithm

- While a more controlled comparison is desirable, sometimes you have to work with what you've got.

- Q: So how do you evaluate them against each other?

	search X	search Y
prior B		Alg #2
prior A	Alg #1	

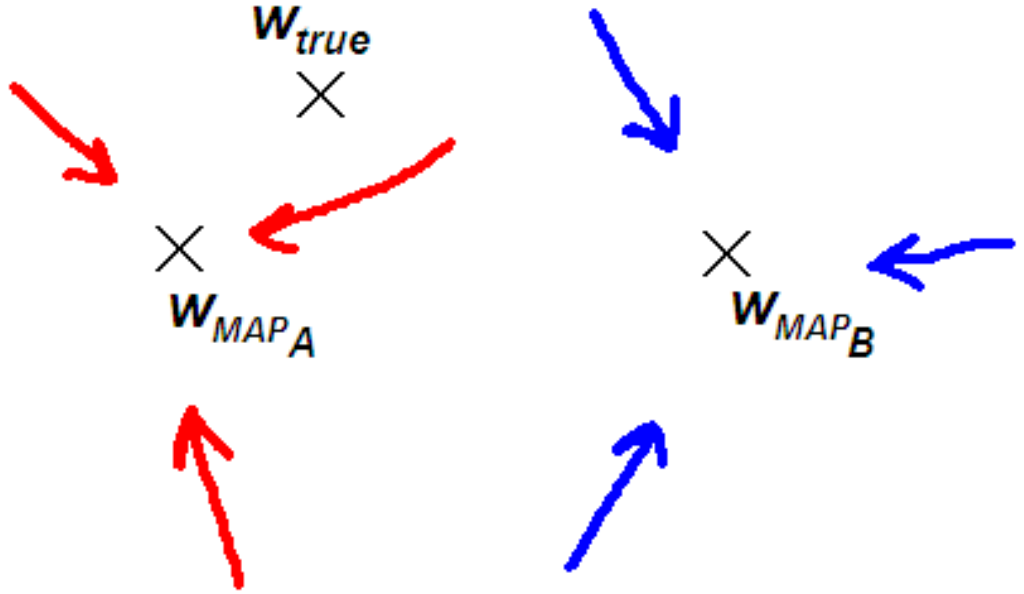
Different priors

- The prior you use makes a difference (especially when there's little data)

- As a result, for every prior, you're maximizing a different objective

- Q: So how do you evaluate them against each other?

- A: Compare their results against the truth.

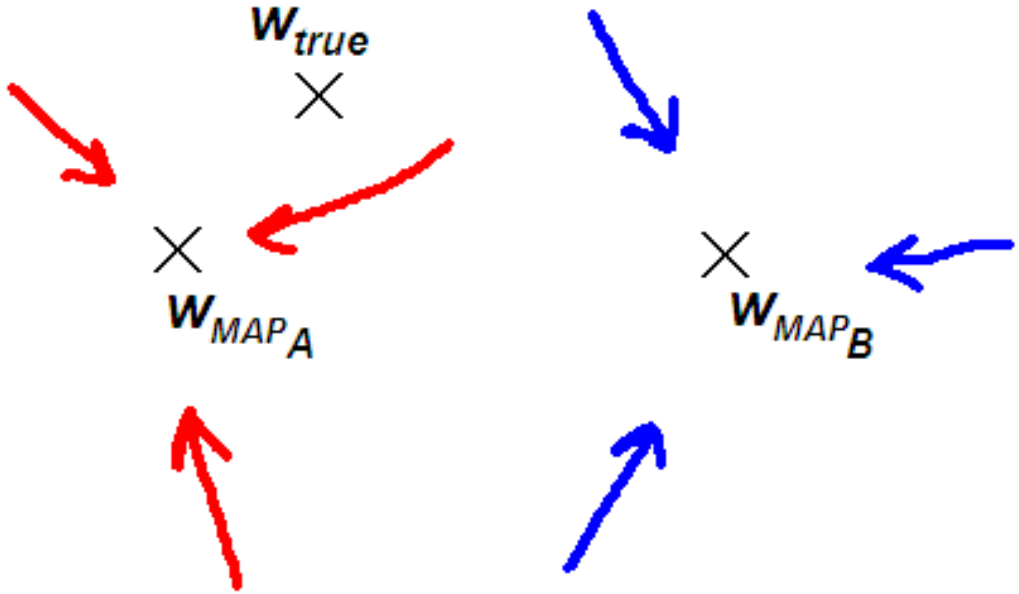


Prior-Algorithm Pairs

- Suppose Alg #1 gets closer to the truth most of the time.

- We still don't know if that's because search X is better than search Y, or prior A is better than prior B on this problem.

- Still, this is better than nothing



Prior-Algorithm Pairs: FBMP vs SSS

We will now see an instance of such a comparison

- FBMP is deterministic and uses a greedy heuristic
- SSS uses a stochastic heuristic
- They also use different priors.

	greedy deterministic search	stochastic search
prior B	***FBMP***	
prior A		***SSS***

The Problem: Bayesian Sparse Linear Regression

- Consider linear regression $\boldsymbol{y = Xw + z}$

- You believe that only a few features are relevant

- ***s*** is a bitvector representing the set of active (i.e. nonzero) variables.

- Priors encode the prior probability of any individual bit of ***s*** being on (i.e. any variable being active). This prior parameter is called ***p1***.

- Instead of doing model averaging with the several good models found, we will focus on the incumbent MAP estimate.

FBMP *(Schniter et al)*

- “Fast Bayesian Matching Pursuit”

- Neighborhood relation: bit additions (i.e. adding variables), never exceeds the number of features.

- It never removes features once they've been added. For this reason it's called a “tree-search”.

- Greedy, deterministic: always moves to the best neighbor.

- Invented a “Fast Update”, which uses a neighboring model to compute the score much faster than the naïve way (used by SSS).

SSS *(Dobra et al)*

- “Shotgun Stochastic Search”
- Neighborhood is bitflips AND “swaps” (as a result, it gets bigger as |s| gets bigger)
- Uses a different prior than FBMP
- Samples the next point from the set of all neighbors
- Does not report the incumbent at every iteration, so we have to estimate it

Creating Instances

- Chris and I needed to run experiments on different computers
- So we agreed on a set of random seeds (#s 1-20)
- … and modified the instance generation code to take seeds as input
- Two instance generators: Diagonal, Toeplitz

Choosing Evaluation Measures

Measuring Quality

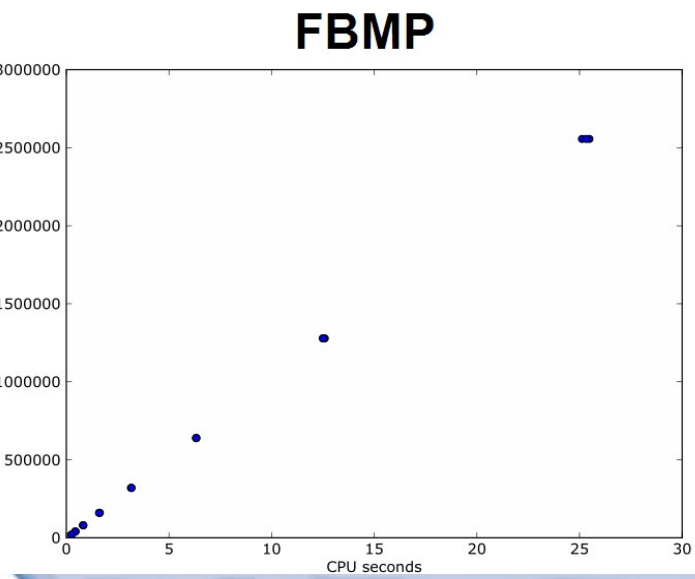
- Objectives (not comparable)
- Normalized Mean-Squared Error from $\boldsymbol{w}_{true}$
- Cosine Distance from $\boldsymbol{w}_{true}$ ✓

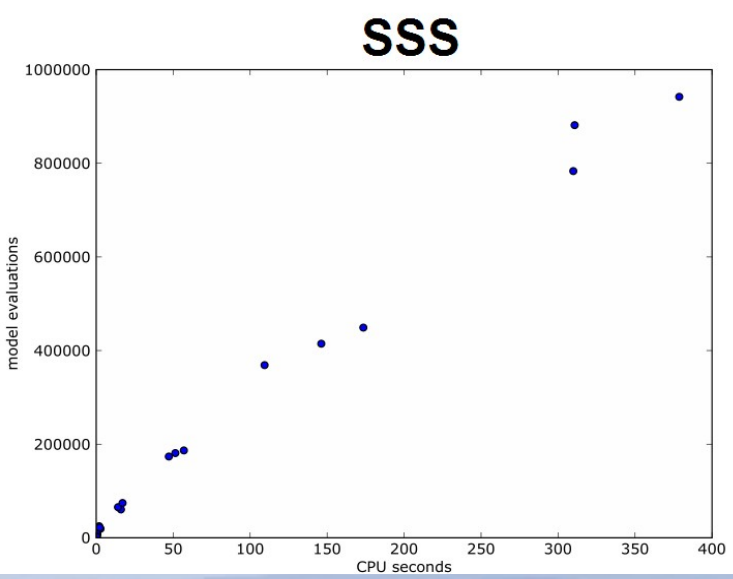
Measuring Runtime

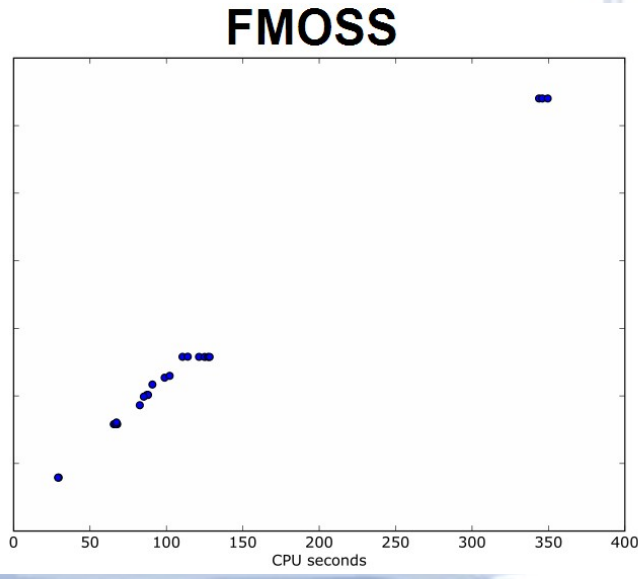
- CPU time
- Number of model evaluations ✓

Measuring Runtime with number of model evaluations

- Immune to optimization: language, fast updates, etc.
- We believe it to be the main factor determining practical runtime
- The plots below show that it strongly correlates with CPU time, within each algorithm.



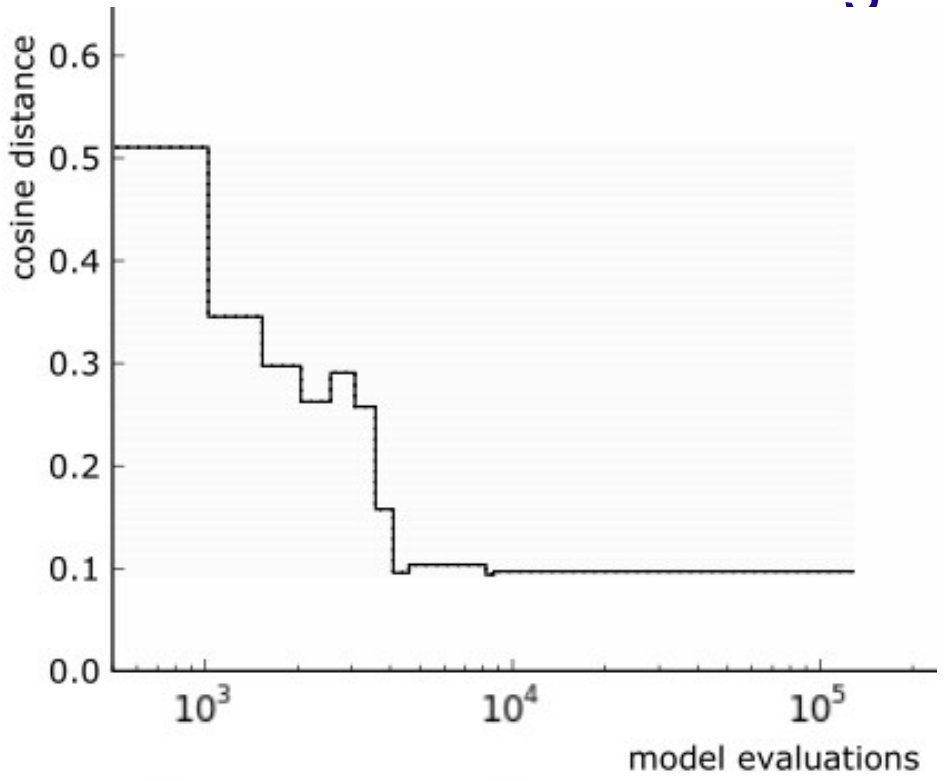




Interpreting “pseudo-SQT” plots

- They mostly decrease, but occasionally increase.

- Each step represents a new incumbent found (i.e. the objective is lower, even when cosine distance goes up).



Running experiments

- For each instance class ({0.04, 0.12} x {Diag, Toep}) 20 instances x some number of runs each (1 FBMP, 5 SSS, 20 FMOSS)

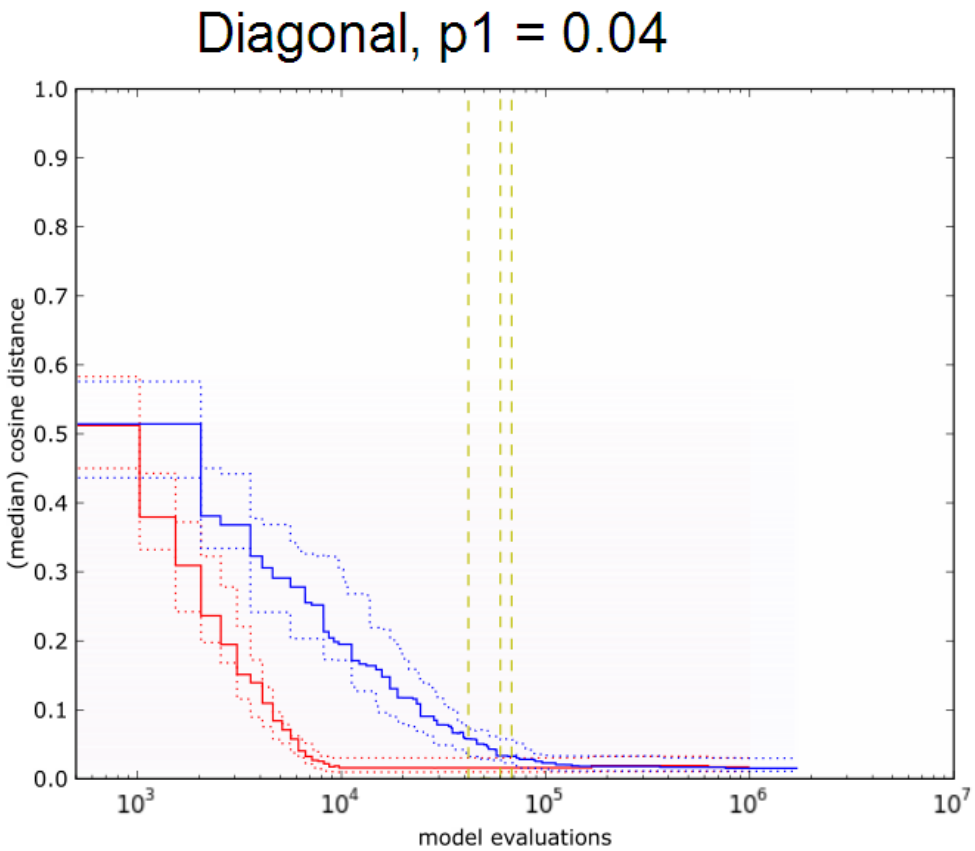
- Sparsity of 0.12 is harder than 0.04 because there are more active variables.

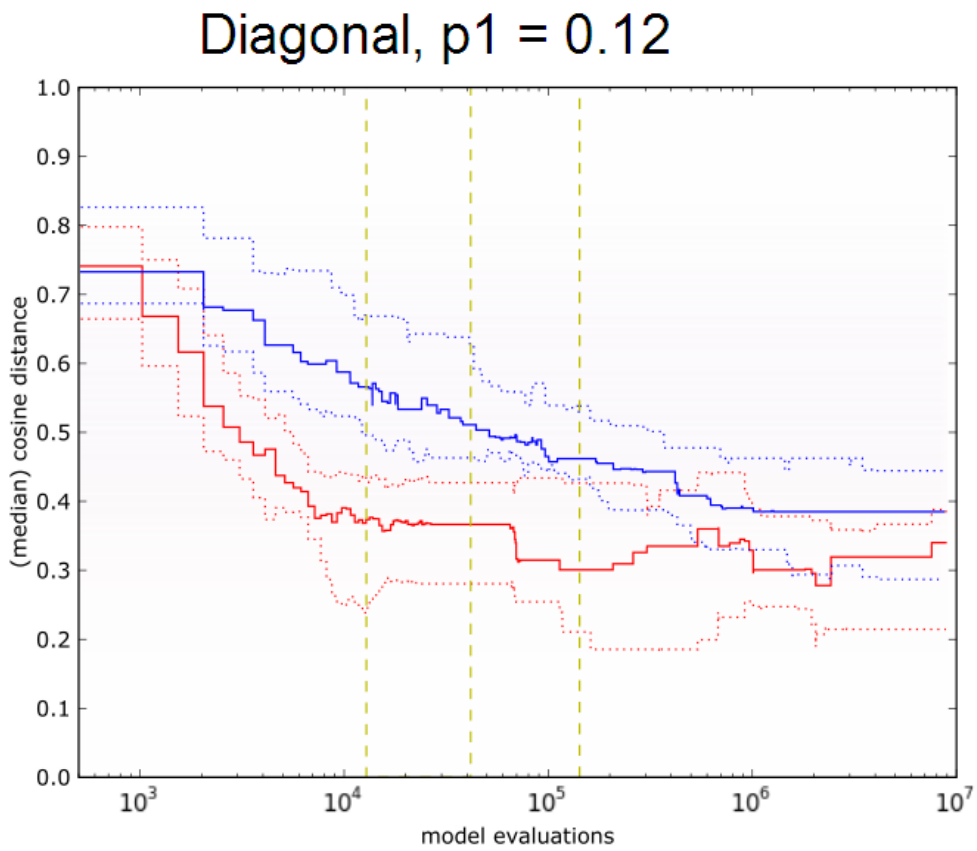
- Toeplitz is harder than Diagonal because variables are correlated.

- We always adjusted the prior parameters to the values used in generating the instances: sparsity ***p1***, noise s.d. σ_z, etc. .... and ran

Diagonal instances – FBMP wins

- FBMP dominates SSS everywhere



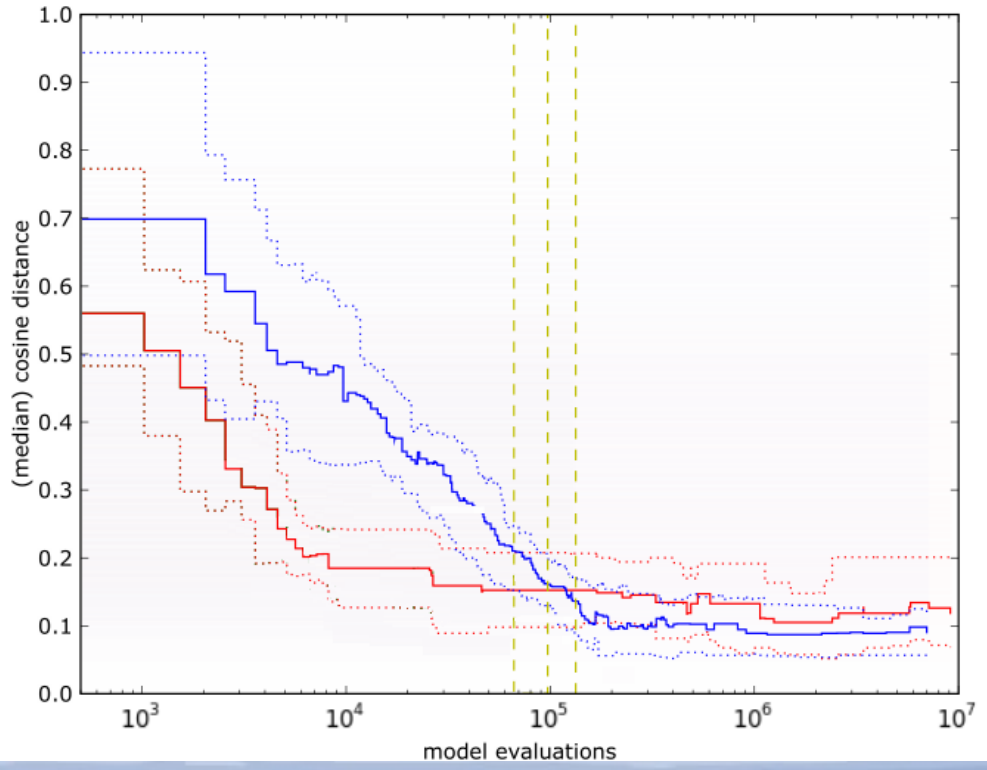


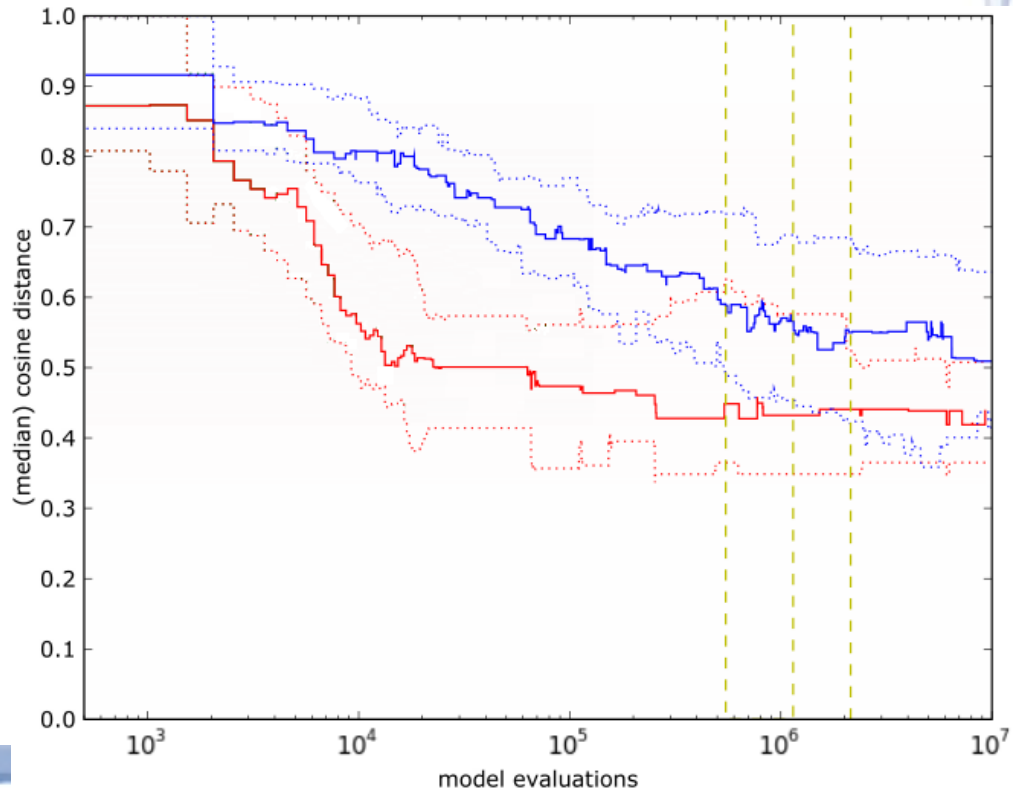
Toeplitz instances – it depends

- Toeplitz when p1 = 0.04:

after 100k evaluations, median SSS dominated median FBMP.

- When p1 = 0.12, we expected this class to be better suited to SSS, but we didn't run for enough time to know.



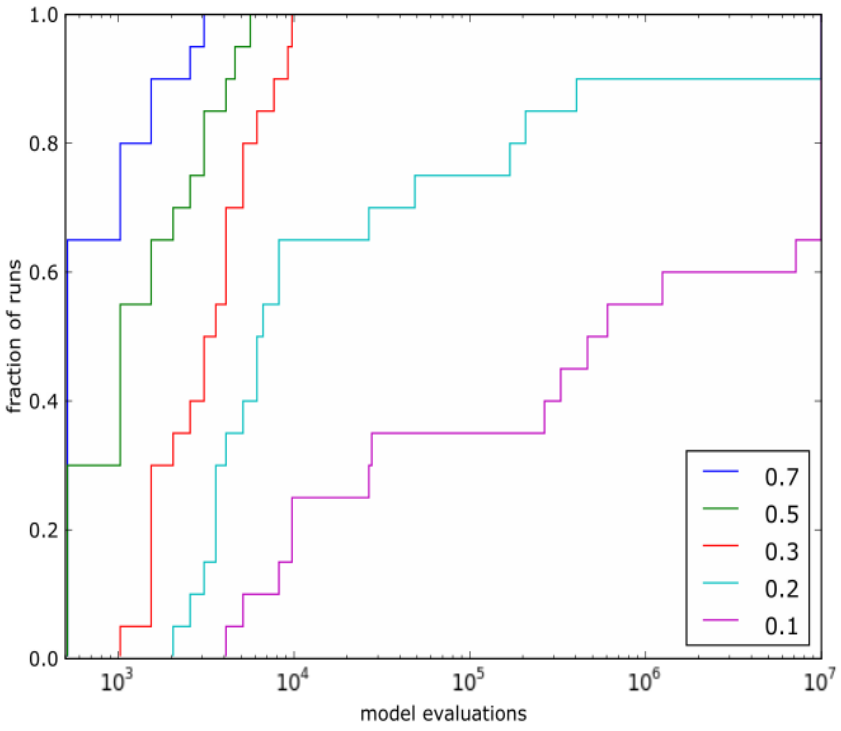


FBMP vs SSS on Toep 0.04 – QRTDs

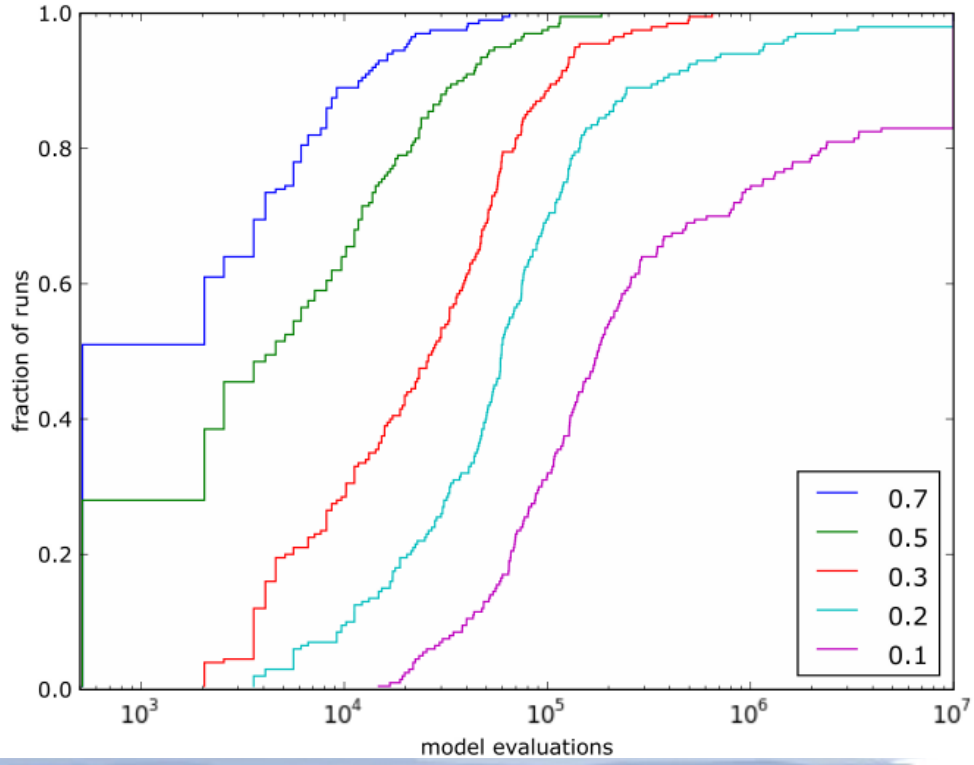
The QRTD shows how the RTD is affected by the Solution Quality standard.

FBMP does not dominate: if your goal is to reach 0.1 as often as possible with 300k iterations, SSS looks like the better choice.

FBMP



SSS

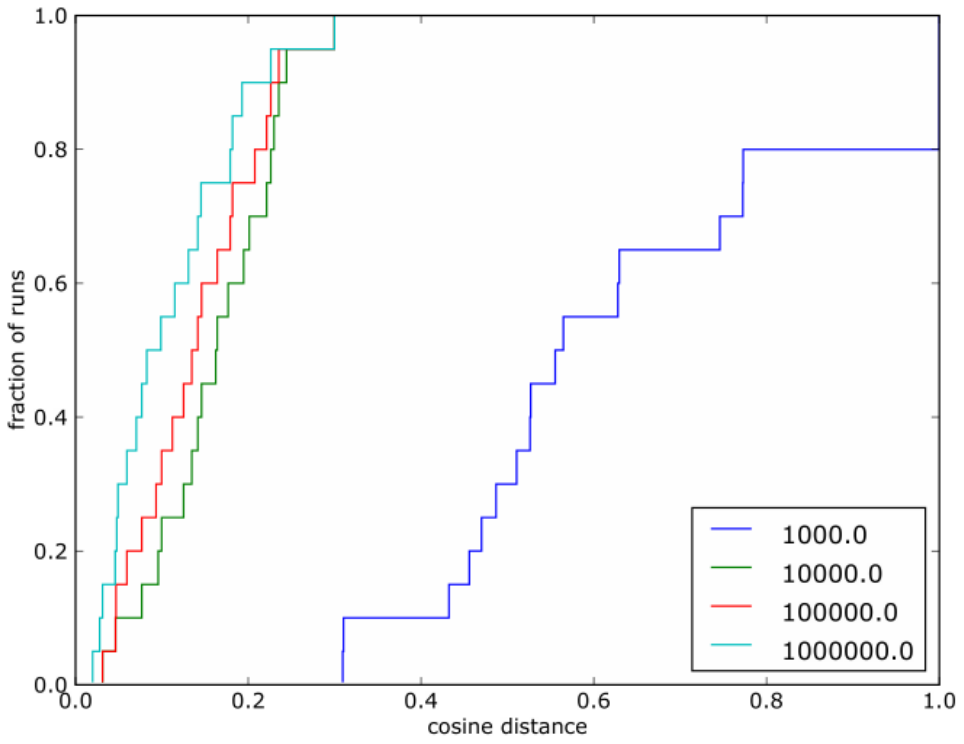


FBMP vs SSS on Toep 0.04 – SQDs

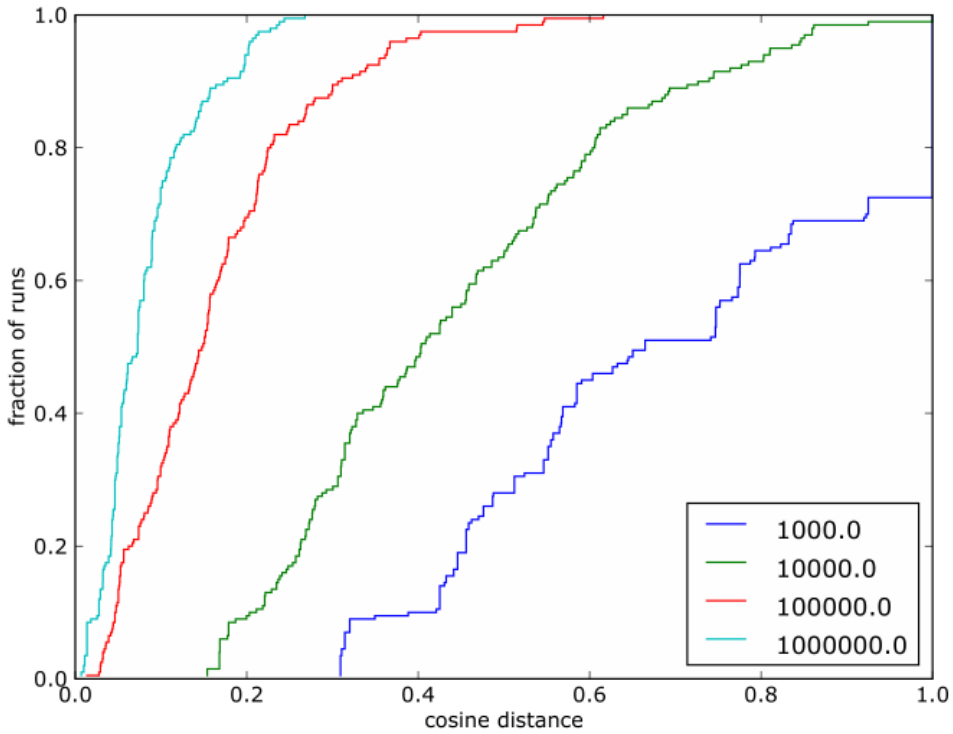
These plots show how the solution quality distribution is affected by a given Runtime investment.

- FBMP does better for 10k, but SSS does better for 1 million.
- FBMP's SQD doesn't improve much after 10k.

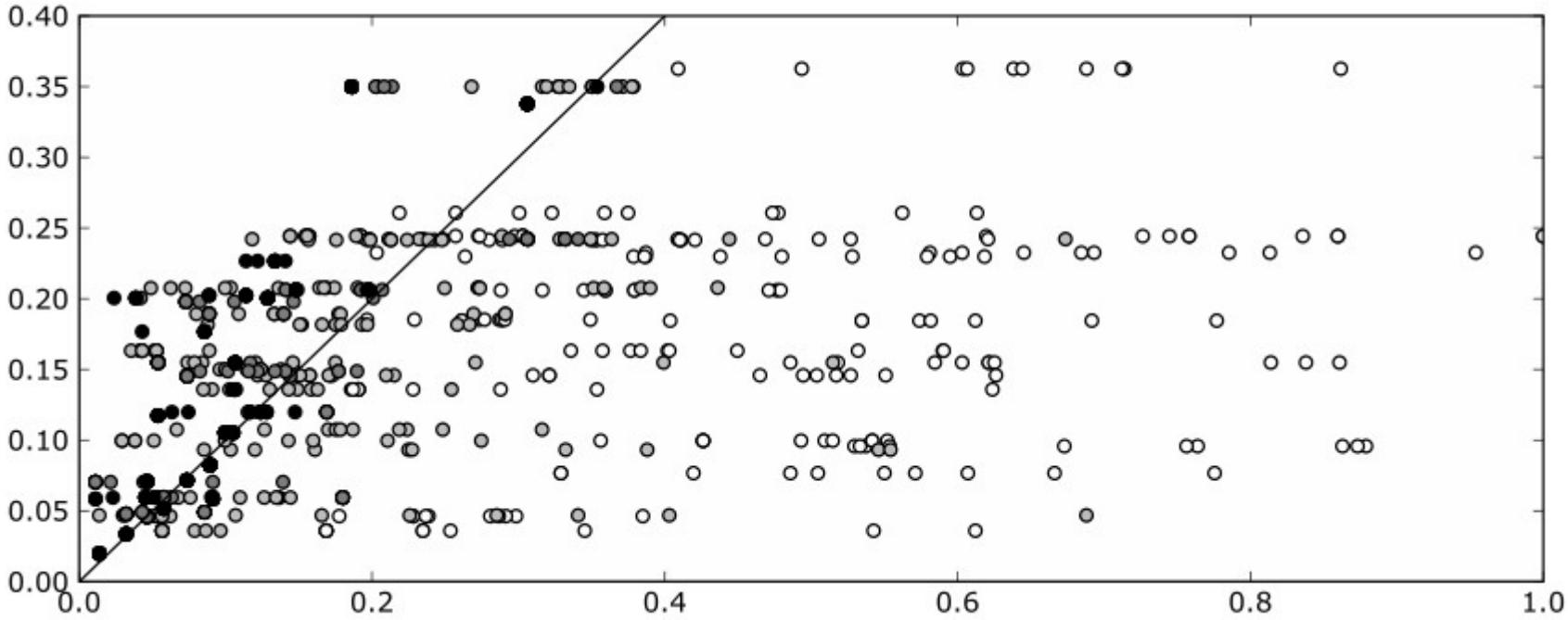
FBMP



SSS

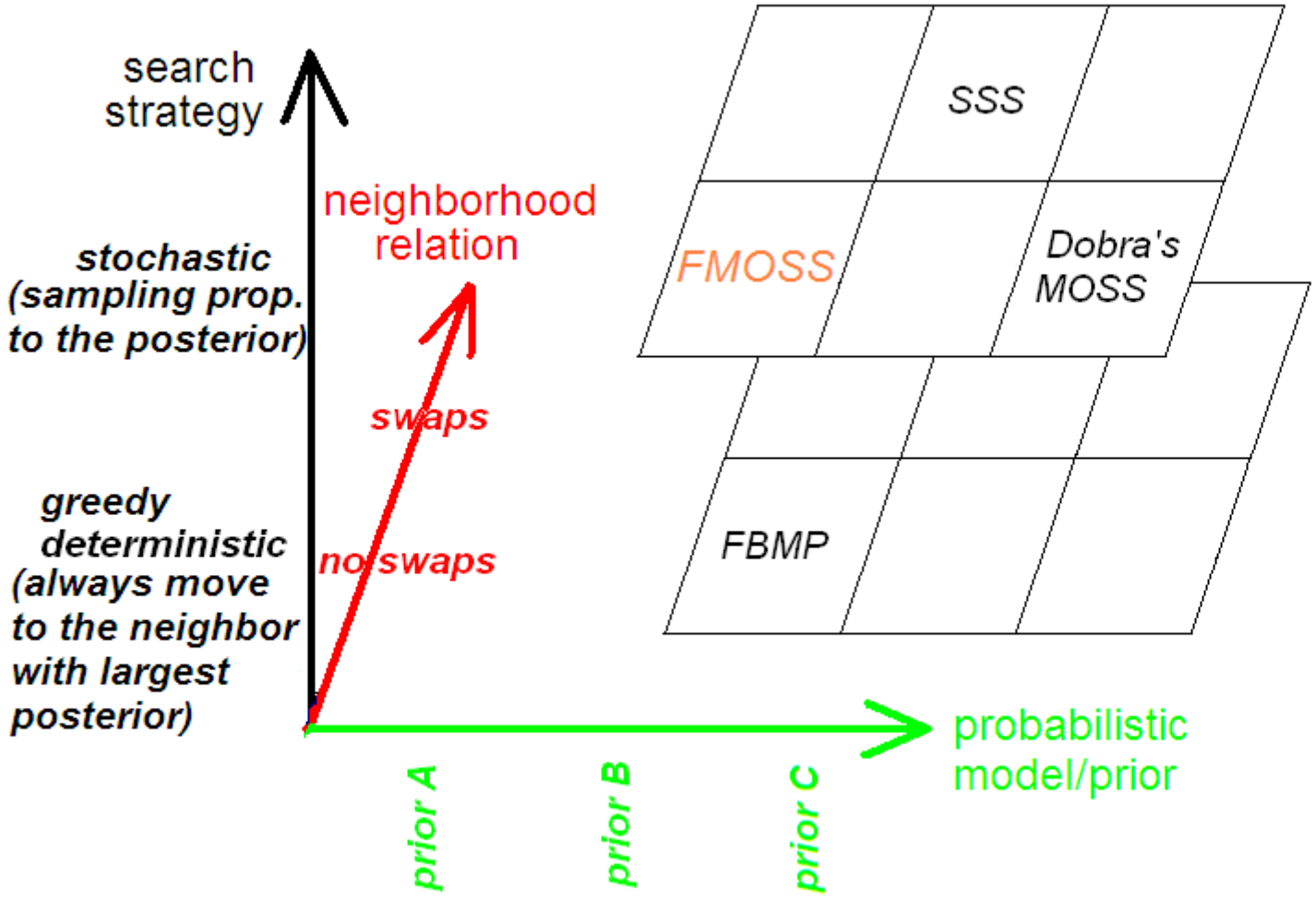


SSS vs FBMP on Toep 0.04 - Solution Quality



FMOSS

- First, a 3D version of our previous diagram



FMOSS

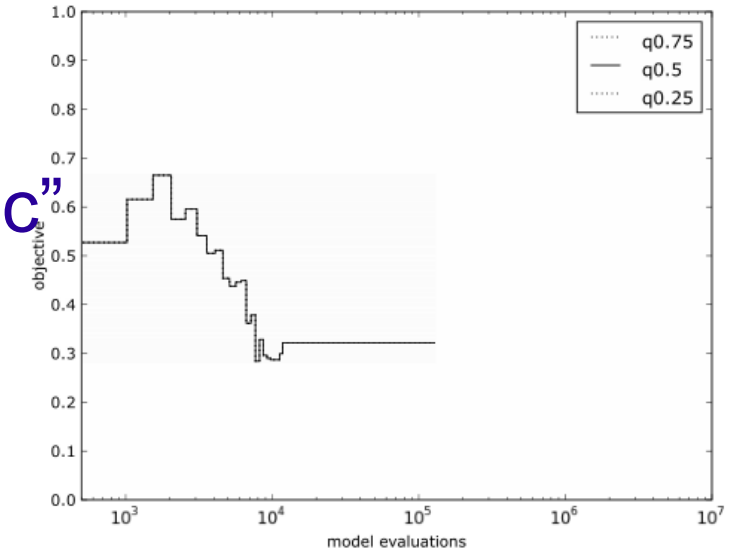
- FMOSS uses the search strategy from MOSS (Dobra et al), which is used for log-linear models.
- But the probabilistic model is taken from FBMP.
- Like FBMP, we use “fast updates”.
- Unlike FBMP, the neighborhood includes “downdates”, and we explore stochastically.
- Unlike SSS, the neighborhood does not include swaps.

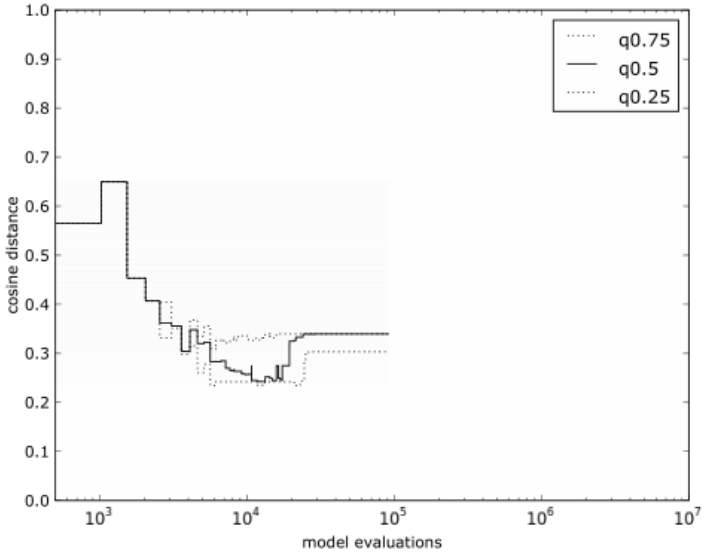
FMOSS – deterministic behavior

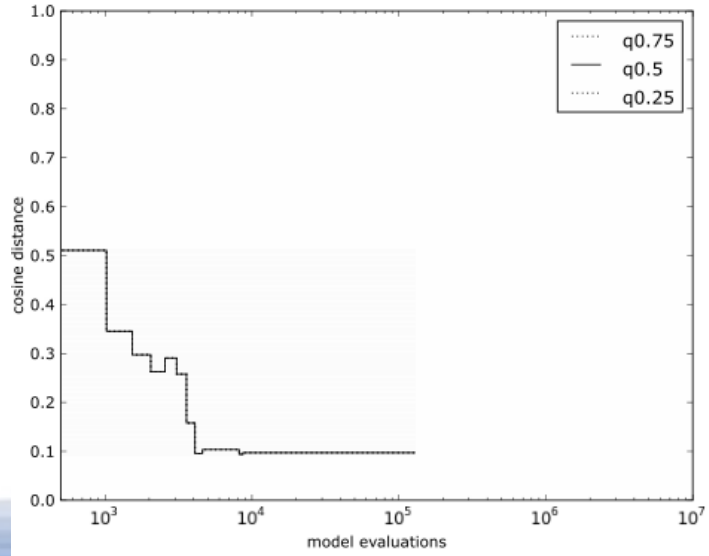
- On Toep 0.04, there were 8 instances in which FMOSS took the exact same path on all 20 runs.

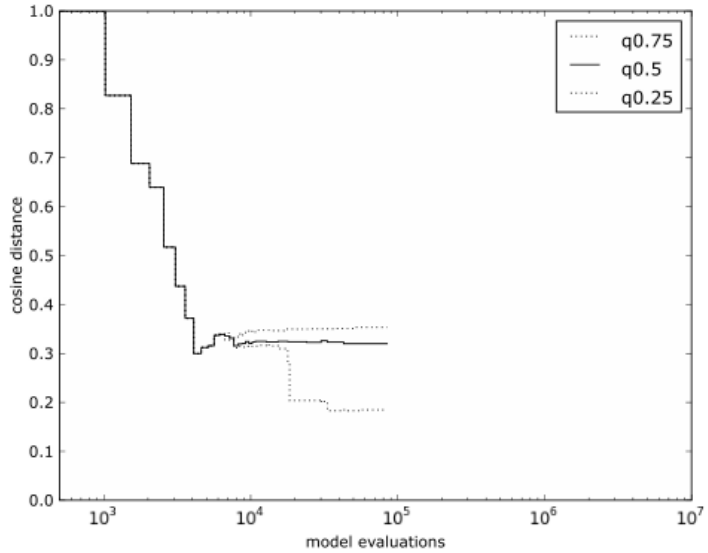
- Left: 2 “deterministic” instances

- Right: 2 “stochastic” instances





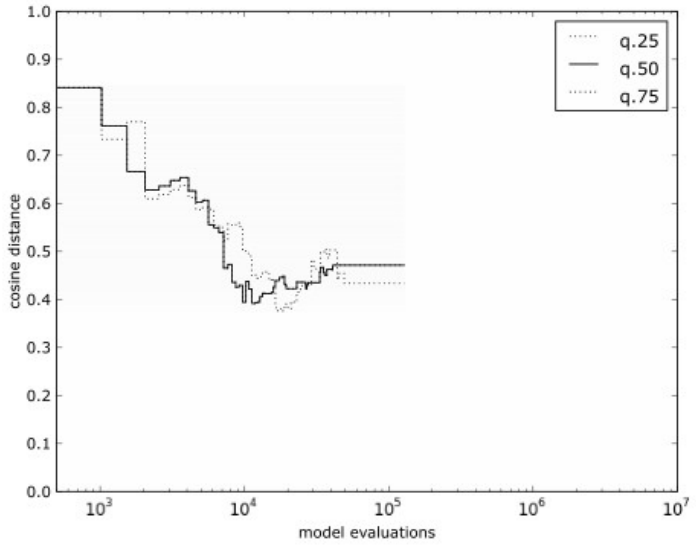


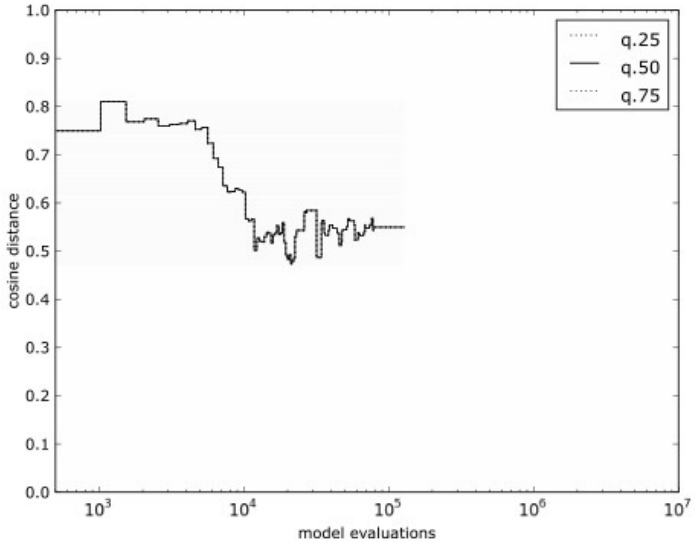


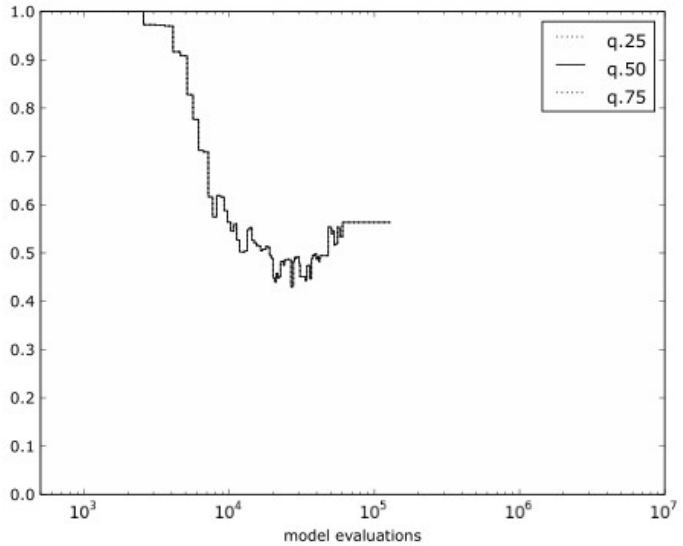
FMOSS – deterministic behavior

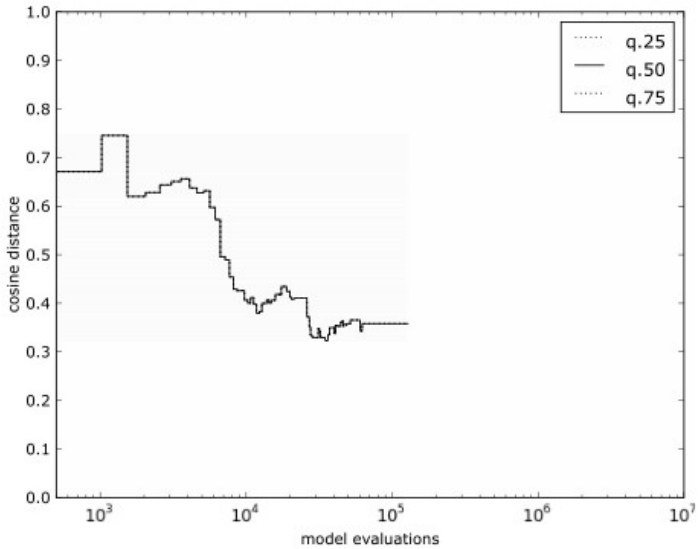
- On Toep 0.12, this happened in 19 instances!

The exception is shown on top-left







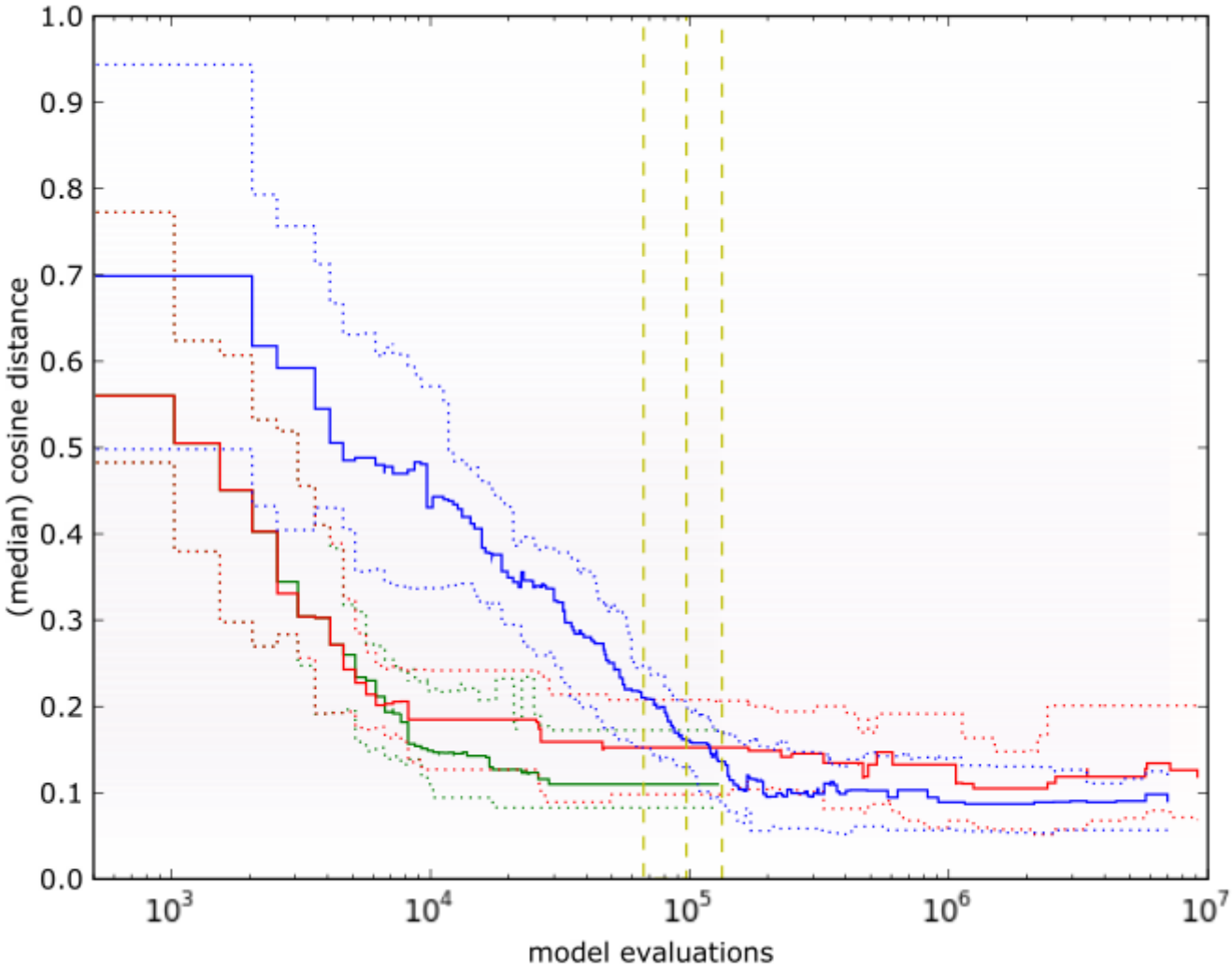


All 3 algorithms - Toeplitz 0.04

- After ~30k evaluations, the median of FMOSS reaches asymptotic performance

For SSS it takes ~100k evaluations.

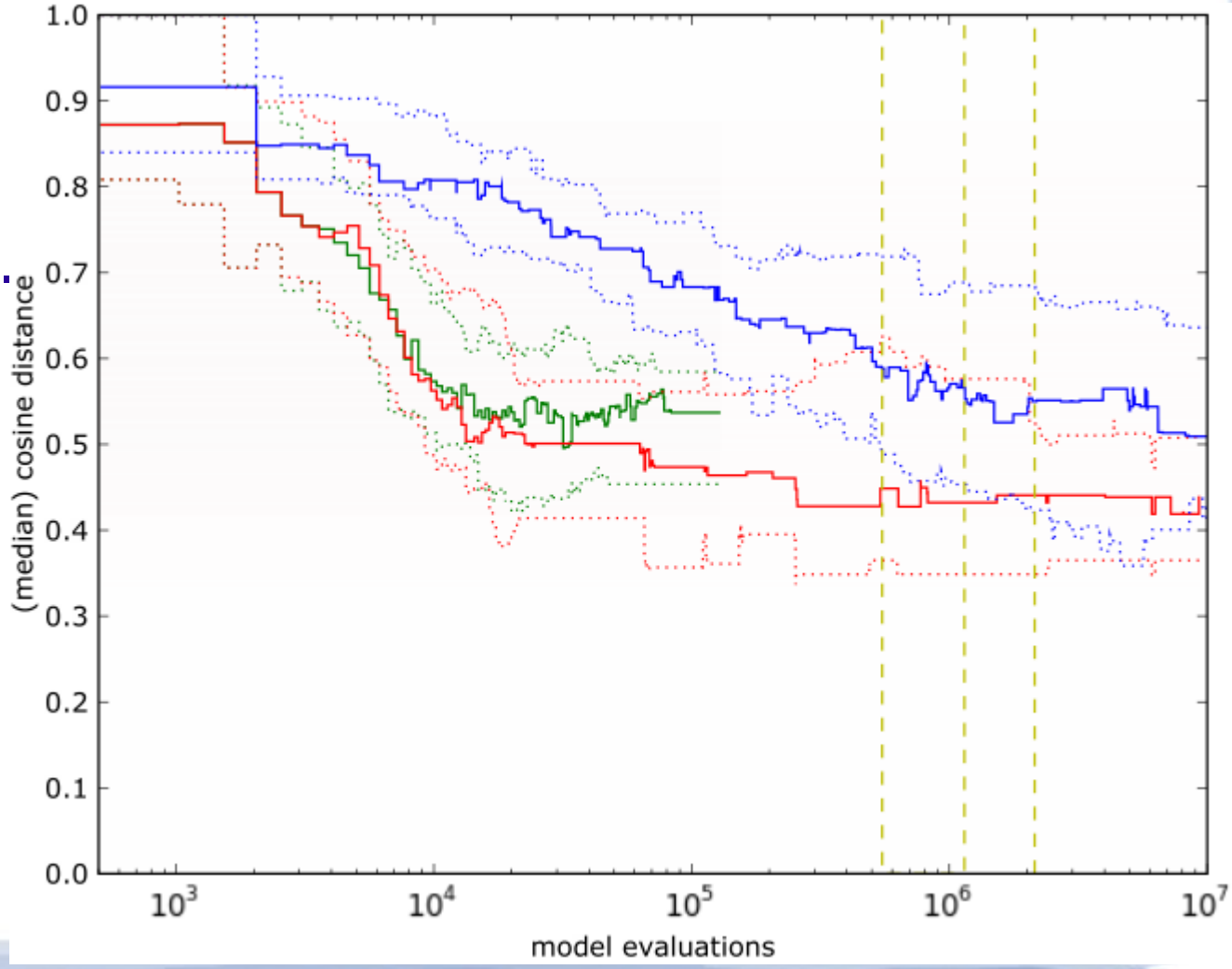
- FMOSS's median dominates FBMP's but still within FMOSS's quartiles



All 3 - Toeplitz instances 0.12

- FBMP seems to perform better than FMOSS.

- although the differences may appear significant, keep in mind that FMOSS's median within the quartiles of FBMP.



Conclusion

- On Toeplitz 0.04 instances, SSS outperforms FBMP after 100k model evaluations.

- FMOSS performs quite similarly to FBMP on both Toeplitz instance classes, but FMOSS appears to dominate between 10k and 100k evaluations.

- Harder instances see more deterministic behavior by FMOSS.

Future Work

- Compare FBMP vs FMOSS on the objective, rather than against the truth, since they use the same prior (and thus the same objective).

- Perform tests of statistical significance to evaluate claims of domination.

- Experiment with a more stochastic FMOSS, by making the distribution closer to uniform.

- Instance-level comparison of FMOSS vs FBMP (e.g. do a quantile plot for each instance)

- Explore other spots in algorithm space (e.g. greedy like FBMP but downdates like FMOSS)

- FMOSS has some overhead, so counting model evaluations is not exactly fair. We could instead use a cost model.

- Evaluate on prediction tasks, using Bayesian model averaging.

- Evaluate on real data?

FMOSS with uniform sampling – inter vs intra instance variability

- Showing 3 random instances x 5 runs.
- Some instances are consistently harder