An Empirical Analysis of Algorithms for Bayesian Sparse Linear Regression

Christopher Nell and Gustavo Lacerda
{cnell, gusl}@cs.ubc.ca

CPSC 536H Project Report
The University of British Coumbia
Vancouver, BC, Canada

April 10, 2009

Abstract

Two conceptually similar algorithms for performing sparse linear regression under a Bayesian framework are presented. A novel implementation of a third algorithm is adapted from the log-linear regression task and applied to the linear regression model. The algorithms differ in terms of the search strategy used to explore the space of possible active variables; one uses a greedy approach, while the others use variants of stochastic local search. The algorithms are evaluated on a variety of simulated data sets, and the relative strengths and weaknesses of each approach are discussed.

1 Introduction

Consider the problem of performing linear regression when the number of observed features is large relative to the size of the sample. If only a small subset of the features is actually relevant, this problem is known as sparse linear regression. Formally, the regression model under consideration is

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{z} \tag{1}$$

where $\mathbf{X}$ is a $m \times n$ regressor matrix (or design matrix), $\mathbf{y}$ is a $m \times 1$ vector of noisy observed values, $\mathbf{z}$ is a $m \times 1$ vector representing additive noise, and $\mathbf{w}$ is a $n \times 1$ parameter vector which is assumed to be of sparsity $p_1 \ll 1$. Given $\mathbf{y}$ and $\mathbf{X}$, the problem is to find the best estimate $\hat{\mathbf{w}}$ of $\mathbf{w}$ according to some quality measure.

A general Bayesian approach to solving the sparse linear regression problem begins by specifying the prior probability distributions underlying $\mathbf{X}$, $\mathbf{w}$, and $\mathbf{z}$, under some set of constraining assumptions. With these priors and the data, the posterior probability of any particular model $\mathbf{s}$ (choice of active variables) can be computed. The model with the highest posterior corresponds to the MAP estimate $\hat{\mathbf{s}}_{\text{MAP}}$, and determines $\hat{\mathbf{w}}(\hat{\mathbf{s}}_{\text{MAP}})$. In order to be "fully Bayesian", the full posterior can be approximated by averaging the set of models for which the individual posteriors are non-negligible. Thus, the problem becomes one of model selection – where each model is represented by a $n \times 1$ bit vector $\mathbf{s}$, the task is to find the models with largest posterior $p(\mathbf{s}|\mathbf{y})$.

Note that even if consideration is restricted to models with d active variables, there are still $C(n, d)$ points in the search space of $\mathbf{s}$. If for example $n = 512$ and $d = 20$, there are $> 10^{35}$ possible models to consider – far too many for an exhaustive approach. As such, algorithms for sparse linear regression are fundamentally dependent on the choice of search algorithm for exploring this space.

In this report, we present the results of an empirical analysis of three such algorithms: Fast Bayesian Matching Pursuit [2], Shotgun Stochastic Search [1], and Fast Mode-Oriented Stochastic Search (an adaptation of an existing algorithm for log-linear regression [5]). In particular, we focus on determining the effectiveness of their respective search strategies for finding good MAP estimates of $\mathbf{s}$. The remainder of the

paper is structured as follows: in Section 2 we describe the algorithms under consideration, and in Section 3 we present our empirical analysis. Finally, in Section 4 we draw conclusions based on our results, and suggest directions for further study.

2 Algorithms for Sparse Linear Regression

As discussed in Section 1, an algorithm for Bayesian sparse linear regression can be characterized by its regression model priors and its search strategy. We describe each of the tested algorithms in terms of these components.

2.1 Fast Bayesian Matching Pursuit

Fast Bayesian Matching Pursuit (FBMP) is a deterministic algorithm designed to find the estimate $\hat{\mathbf{w}}$ which minimizes mean-squared error (MSE) with respect to the true coefficient vector [2]. The nonzero elements of $\mathbf{w}$ and elements of $\mathbf{z}$ are assumed to be drawn from Normal distributions; in particular:

$$\mathbf{w}|\mathbf{s} \sim \mathcal{N}(\mathbf{0}, \sigma_s^2 \mathbf{I}_n) \tag{2}$$

$$s_i \sim \text{Bernoulli}(p_1) \tag{3}$$

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma_z^2 \mathbf{I}_m) \tag{4}$$

Further, the columns of $\mathbf{X}$ are assumed to be unit-normal. The algorithm takes p_1, σ_s^2, and σ_z^2 as parameters from the user.

The most recent implementation of FBMP runs an iterative greedy forward-selection search to construct $\hat{\mathbf{w}}$, stopping when the number of active variables exceeds some criteria (analytically determined by n and p_1) [3]. Thus, at each iteration FBMP evaluates $n - ||\mathbf{s}|| \in \mathcal{O}(n)$ models.[1] The greedy search is repeated a user-specified number of times, avoiding re-exploration of models considered in previous iterations. The log posterior probability $\log p(\mathbf{s}|\mathbf{y})$ is used as the objective function guiding search. The Matlab implementation of FBMP available online was augmented to record the incumbent model after each iteration of the search, along with the corresponding total number of model evaluations performed (n per bit activation), in order to permit the presented analyses.

The most significant innovation introduced in FBMP is a fast posterior update equation. Exploiting the sequential bit-activation behavior of the algorithm, the authors are able to evaluate the objective function for a new model in $\mathcal{O}(nm)$ multiplications, compared to $\mathcal{O}(nm^2)$ if standard matrix multiplication is used.

2.2 Shotgun Stochastic Search

Shotgun Stochastic Search (SSS) is a randomized approach to sparse linear regression which uses a slightly different prior than that of FBMP [1], and with the goal of finding the true model $\mathbf{s}$ (but not necessarily the coefficients $\mathbf{w}$ which minimize MSE). In particular, it does not require that the Normal distribution variances are known *a priori*. Instead, SSS assumes:

$$\mathbf{w}|\mathbf{s} \sim \mathcal{N}(\mathbf{0}, \tau^{-1}\sigma^2 \mathbf{I}_n) \tag{5}$$

$$\sigma^2|\mathbf{s} \sim \text{IG}((\delta + |\mathbf{s}|)/2, \tau/2) \tag{6}$$

$$s_i \sim \text{Bernoulli}(p_1) \tag{7}$$

In principle, τ, δ, and p_1 should be provided as parameters to the algorithm; however, only p_1 is exposed in the compiled code that is available. Fixed choices $\tau = 1$ and $\delta = 3$ are justified in the paper, and are appropriate for comparison to FBMP. Finally, SSS assumes both $\mathbf{y}$ and the rows of $\mathbf{X}$ are mean-centered with unit variance, and it omits the intercept term ($\mathbf{z} = \mathbf{0}$). One implication of this transformation of the data is that the coefficients of $\mathbf{w}$ will not general correspond to the minimum MSE estimate of the untransformed data.

[1] Since $||\mathbf{s}|| \ll n$, the FBMP implementation actually computes n models at each step, for the sake of simplicity.

SSS utilizes a local search strategy, again guided by log posterior probability $\log p(\mathbf{s}|\mathbf{y})$. The neighborhood $\{\mathbf{s}\}^*$ of $\mathbf{s}$ is constructed from three sub-neighborhoods:

$$\{\mathbf{s}^+\} \equiv \text{models constructed from } \mathbf{s} \text{ by a single variable activation}$$
$$\{\mathbf{s}^-\} \equiv \text{models constructed from } \mathbf{s} \text{ by a single variable deactivation}$$
$$\{\mathbf{s}^\circ\} \equiv \text{models constructed from } \mathbf{s} \text{ by a swap of activation between two variables}$$

It is straightforward to verify that the neighborhood is of size:

$$\begin{aligned} ||\{\mathbf{s}\}^*|| &= ||\{\mathbf{s}\}^+|| + ||\{\mathbf{s}\}^-|| + ||\{\mathbf{s}\}^\circ|| \\ &= (n - ||\mathbf{s}||) + ||\mathbf{s}|| + ||\mathbf{s}||(n - ||\mathbf{s}||) \\ &= n + ||\mathbf{s}||(n - ||\mathbf{s}||) \\ &\in \mathcal{O}(n||\mathbf{s}||) \end{aligned}$$

Note that the neighborhood size of SSS is asymptotically larger than that of FBMP. In addition, each neighbor evaluation takes asymptotically longer, as fast updates for SSS have not been implemented. These factors make SSS substantially slower than FBMP in practice.

At each iteration, all neighbors of the current model are evaluated, and one neighbor is sampled proportionally to posterior probability from each of the three sub-neighborhoods. One of these three samples is chosen at random to be the next model. The relative probabilities of the various sampling distributions are annealed over time according to four user-specified parameters.

2.3 Fast Mode-Oriented Stochastic Search

Mode-Oriented Stochastic Search (MOSS) is a stochastic local search strategy introduced by Dobra *et al* for log-linear regression models [5]. The MOSS strategy differs from SSS in two key respects:

1. Instead of keeping a single incumbent in memory and exploring only its neighbors, MOSS maintains a set S of the top models encountered in all previous iterations. At each iteration, the next model to be explored is sampled from S proportionally to the posterior probabilities, conditional on not having been previously explored.

2. Where the incumbent has posterior probability $p(\bar{\mathbf{s}}|\mathbf{y})$, all models with posterior $p(\mathbf{s}|\mathbf{y}) < c'p(\bar{\mathbf{s}}|\mathbf{y})$ are removed from consideration. As such, it is possible for MOSS to exhaust S and terminate.

There are also two key conceptual differences between FBMP and MOSS:

1. MOSS chooses candidates for exploration stochastically instead of greedily.

2. MOSS also considers "downdates" (deactivating variables from $\mathbf{s}$) in addition to updates.

Since we aim to evaluate MOSS in the context of sparse linear regression, we implemented a variant which uses the prior and fast updates of FBMP. We refer to this implementation as FMOSS (Fast MOSS). Motivations for implementing this particular algorithm include:

- Using the FBMP prior allows us to isolate the effect of search strategy on algorithm performance.

- MOSS is a more recent development from some coauthors of SSS, and is presented as an improvement. As such, it is a better representative of SLS algorithms.

- Access to source code allows reporting of incumbents at every iteration, which allows a finer analysis than is possible with closed-source SSS.

To the best of our knowledge, our implementation is novel in that it is the first to apply MOSS to the problem of linear regression using the FBMP prior.

Algorithm 1 Fast Mode-Oriented Stochastic Search

```
[incumbents,objective] =
fmoss(X, y, neighborhoodRelation, initialModels, cprime, maxIter, maxSizeS)
   mark all initialModels as unexplored and add them to S.
   compute the posterior for every initial model, and set incumbent and ...
fmax accordingly.
   for iteration := 1 to maxIter
     if there are unexplored models in S
       randomly pick an unexplored model m from S, and mark it as explored.
     else exit;
     for every m' in the neighborhood of m
       post := computePosterior(X,y,m');
       if post>cprime*fmax
         add m' to S, as unexplored.
         if (size(S) > maxSizeS), remove worst model from S.
         if post>fmax
           incumbent := m';
           fmax := post;
           remove from S all models whose posterior is < cprime * fmax.
         end if
       end if
     end for
     incumbents(iteration) = incumbent;
     objective(iteration) = fmax;
   end for
end function
```

2.3.1 Implementation details

FMOSS is implemented in Matlab. It was initially designed as a general optimization algorithm, and later the log posterior under the family of FBMP priors was specified as the objective. Direct computation of the posterior is slow because it involves inverting a large matrix $\mathbf{\Phi}$. Therefore, we use the fast update procedure to update this inverse between neighboring models. This involves storing a matrix $\mathbf{\Phi}^{-1}$ for each unexplored model in S. Using fast updates does not affect our plots or analysis, since we are measuring the number of model evaluations – it simply improves the runtime of our algorithm.

Pseudo-code for FMOSS is presented in Algorithm 1. There are a few key differences between our implementation and the algorithm described in [5].

- We artificially limit the size of S to $S_{\max}$ in order to conserve memory.
- We artificially limit the number of iterations performed to limit the maximum runtime, rather than waiting for S to be exhausted.
- We specify the neighborhood relation to be single bit flips, but not swaps; i.e.:

$$||\{\mathbf{s}\}^*|| = ||\{\mathbf{s}\}^+|| + ||\{\mathbf{s}\}^-|| = n$$

3 Empirical Evaluation

3.1 Data Sets

We evaluate the described algorithms on two classes of randomly-generated data. Since the true $\mathbf{w}$ is known for each instance, we are able to precisely specify the solution quality provided by each of the algorithms. In all cases, we choose $m = 128$, $n = 512$, as inspired by the FBMP paper [3]. Finally, as FBMP requires the columns of $\mathbf{X}$ to have unit norm, all $\mathbf{X}$ matrices were normalized before being used to generate $\mathbf{y}$.

We first generated 40 instances according to the procedure described in the FBMP paper; 20 each at sparsity $p_1 = 0.04$ and $p_1 = 0.12$. These values of sparsity were chosen in order to test the search strategy's performance as a function of the size of the search space. Specifically, $\mathbf{X}$ was generated by sampling each entry i.i.d. from $\mathcal{N}(0, 1)$ – since this corresponds to a diagonal covariance matrix, we refer to these as *Diagonal instances*. The entries of noise term $\mathbf{z}$ were sampled from $\mathcal{N}(0, \sigma_z^2 = 0.01)$, and the nonzero entries of the true $\mathbf{w}$ were sampled from $\mathcal{N}(0, \sigma_s^2 = 1)$. $\mathbf{y}$ was calculated from $\mathbf{X}$, $\mathbf{w}$, and $\mathbf{z}$.

In order to evaluate performance on data with correlated variables, we also generated 40 instances using the same parameters, but in which the covariance matrix generating $\mathbf{X}$ is upper-triangular Toeplitz instead of diagonal (we call these *Toeplitz instances*)[2]. This data choice is motivated by the study of Meinhausen *et. al.* [4].

3.2 Evaluation Metrics

We are ultimately interested in measuring the algorithms' solution quality as a function of runtime. However, as SSS is only available as a compiled C++ binary whereas FBMP and FMOSS are implemented in interpreted Matlab code, direct comparisons of CPU time between algorithms would be somewhat inappropriate. Furthermore, both FBMP and FMOSS utilize an efficient posterior update, while SSS does not. Since we are interested in comparing the search strategies (rather than posterior computation), and since we are interested in evaluating the algorithms rather than their implementations, we instead measure runtime in terms of the number of model evaluations performed. Runtime vs. number of model evaluations is plotted in Figure 1 for each of the algorithms; these plots clearly demonstrate that the quantities are linearly related, justifying our choice of metric. They also establish that the FBMP implementation is about 50 times faster per evaluation than SSS, and 500 times faster than FMOSS. We attribute this discrepancy to the lack of fast-updates in SSS, and to the unoptimized state of our implementation of FMOSS.

As a first solution quality measure, we consider the normalized mean squared error (NMSE) between the predicted $\hat{\mathbf{w}}$ and true $\mathbf{w}$, as per [3]:

$$NMSE(\mathbf{w}, \hat{\mathbf{w}}) \equiv \frac{||\hat{\mathbf{w}} - \mathbf{w}||_2^2}{||\mathbf{w}||_2^2} \tag{8}$$

However, as discussed in Section 2.2, SSS does not return minimum MSE coefficients, so this measure is unfair. Distance measures between bit vectors (such as Hamming distance) are insufficient, as they do not distinguish between relatively high- and low-coefficient variables being active. In order to eliminate the effect of the magnitude $||\hat{\mathbf{w}}||_2$ on our analysis while remaining sensitive to the specific activated variables, we also consider the cosine distance:

$$CD(\mathbf{w}, \hat{\mathbf{w}}) \equiv 1 - \frac{\mathbf{w}\hat{\mathbf{w}}}{||\mathbf{w}||_2||\hat{\mathbf{w}}||_2} \tag{9}$$

3.3 Experiments with FBMP

Parameters for all FBMP tests were $\sigma_s^2 = \mathbf{1}$, $\sigma_z^2 = 0.01$, and p_1 as appropriate. Each run was terminated after 10^7 model evaluations.

The performance of FBMP on one sample instance from each class (chosen to represent 'typical' behavior) is illustrated in the solution-quality/time (SQT) plots of Figure 3. Notice that the NMSE and cosine distance metrics are not strictly decreasing, but as expected the objective function is.[3] From these figures, we can see

[2] Specifically, the Toeplitz covariance matrix $\mathbf{C}$ has $C_{ij} = 0.9^{|i-j|}$

[3] The plotted 'objective' is actually $1 - \log(p(\mathbf{y}|\mathbf{s}_t))/\max_i \log(p(\mathbf{y}|\mathbf{s}_i))$

that NMSE and cosine distance are qualitatively very similar, as expected. On the other hand, the objective function is qualitatively quite different from cosine distance for Toeplitz instances; this suggests that FBMP's prior is not particularly robust to dependent variables.

Each individual-run SQT has an associated plot of the evolution of $\hat{\mathbf{w}}$ as the search progresses. The columns of this plot represent the incumbent after each iteration; the final two columns indicate the true $\mathbf{w}$ and corresponding $\mathbf{s}$. Intensity represents the magnitude of the coefficient assigned to each active variable, and variables which are (correctly) never activated are not drawn. It is interesting to note that in each case, progress slows considerably after approximately 10^4 evaluations (corresponding to the first greedy descent), and that on harder instances, sudden jumps between very different incumbent models commonly occur.

Finally, the performance of FBMP across all instances of a given class is summarized by the aggregate SQT plots of Figure 2. Qualified runtime distributions (QRTDs) are plotted in Figure 7, and solution quality distributions (SQDs) in Figure 9. These plots indicate that while FBMP performs very well on $p_1 = 0.04$ Diagonal instances (the class tested in the original paper), there is substantial room for improvement in the remaining classes. This suggests that any algorithm able to outperform FBMP will do so in one of these other classes.

3.4 Experiments with SSS

SSS is available only in compiled form, and does not output incumbents at each iteration; instead, it outputs details of the expanded candidate. It also outputs a list of the top models found, along with the iteration in which they were found. Thus, we know the incumbents for iterations starting from the earliest reported top model, but only have a lower-bound estimate of the incumbents for earlier iterations. As there is no way to modify the code, nor a way to set the random seed to repeat a run, we must make do with this data. In order to increase the number of iterations for which the incumbent is known, we configured the algorithm to output the top 10^4 models, instead of the default 10. The annealing rates of SSS were left at their default values, as suggested in the documentation. p_1 was set appropriately, and the algorithm was terminated after 1000 iterations.

Due to its relative slowness compared to FBMP, we initially ran SSS 5 times for each instance tested, for 1000 iterations per run. As the $p_1 = 0.04$ Toeplitz case proved the most interesting (see below), we performed an additional 5 runs per instance on this class only.

The performance of SSS in a sample run on each of the sample instances is illustrated in the SQT plots of Figure 4, and SQTs aggregating over all runs for these instances are illustrated in Figure 5. The dashed vertical line in these plots marks the position of the boundary between 'known' and 'lower bound' incumbents discussed above[4]. There are a few important observations to be made from these plots, when compared with Figure 3. First, SSS tends not to exhibit sudden changes to entirely different incumbents; its progress is more gradual. Second, the NMSE values are very much higher than FBMP's, even though the cosine distances are comparable – as predicted in Section 3.2. Finally, the objective function function behaves in a more similar manner to the cosine distance, suggesting that the SSS prior is better suited to dependent variables.

The performance of FBMP across all instances of a given class is summarized by the aggregate SQT plots of Figure 2. Based on these plots, a reasonable conclusion is that FBMP probabilistically weakly dominates SSS on Diagonal instance classes. A similar statement can be motivated for the $p_1 = 0.12$ Toeplitz class, although in this case it is more conceivable that SSS might overtake FBMP after $\gg 10^7$ evaluations. However, it is clear that something interesting is happening in the case of $p_1 = 0.04$ Toeplitz instances.

QRTDs for SSS are plotted in Figure 8. On comparison with Figure 7, it is evident that in all cases but $p_1 = 0.04$ Toeplitz, FBMP indeed reaches a given cosine distance faster than SSS does. In the $p_1 = 0.04$ Toeplitz case, however, SSS has a higher probability of finding high-quality $\hat{\mathbf{w}}$ estimates than FBMP after about 10^5 evaluations. This is consistent with the trend seen in Figure 2.

SQDs for SSS are plotted in Figure 10. Comparing these with Figure 9 leads to the same conclusions. These plots in particular illustrate how SSS is able to benefit from longer runs, whereas FBMP has made most of its progress before 10^5 evaluations. This behavior is typical of local versus greedy search.

Finally, a scatter plot comparing FBMP and SSS on all $p_1 = 0.04$ Toeplitz instances is presented in Figure 6. This plot confirms that after many evaluations, SSS achieves an equivalent or better cosine distance than FBMP on nearly every instance.

[4]For aggregate plots, the three lines represent the median and quartile positions of the individual-run boundaries.

3.5 Experiments with FMOSS

From the previous analyses we can conclude that in situations with dependent regressors and no runtime restrictions, SSS is preferred to FBMP. Unfortunately, it is unclear whether this is due to the prior distribution or search strategy of SSS; indeed, there is evidence to suggest that both are important factors. In order to clarify the situation, we evaluate FMOSS – an algorithm which combines the FBMP prior with a local search strategy.

We ran FMOSS 20 times per instance for both of the Toeplitz classes; each run was a maximum of 250 iterations long. Prior parameters were configured as discussed for FBMP, and we choose $c' = 0.02$ and $S_{\max} = 50$. All searches are initialized from the singleton set $S_0 = \{\mathbf{s}_0 = \mathbf{0}\}$.

At every iteration, we compute $\hat{\mathbf{w}}$ corresponding to the incumbent[5] and compare it to the true $\mathbf{w}$ in terms of cosine distance.

3.5.1 $\mathbf{p_1 = 0.04}$ Toeplitz Instances

The performance of FMOSS on four specific $p_1 = 0.04$ Toeplitz instances is summarized in Figure 12. As expected, the median cosine distance from $\mathbf{w_{true}}$ decreases in a very nearly monotonic fashion as more models are evaluated. However, in two instances the quartiles coincide with the median at every iteration. This suggests unexpected deterministic behavior. To investigate this possibility, we plotted the maximum and minimum cosine distance for every instance, and observed that only 12 of the 20 instances showed any variability across their 20 runs. One possible explanation for this is that FMOSS explored exactly the same models at every iteration (which could occur when the incumbent is always much more probable than the 2nd best model); another is that this choice of which models to explore did not affect the time when new incumbents were found. It is worthwhile to note that such deterministic behavior was not observed with SSS.

The first deterministic instance from Figure 12 is also plotted for FBMP in Figure 3. The shape of the respective curves is very similar. Up until 6000 evaluations, they seem to be identical, which suggests that they are taking the same update steps. After that, MOSS most likely finds a different new incumbent via a "downdate". It is unlikely that a different "update" is responsible; since FBMP always explores the most probable model, *some* of the 20 runs of FMOSS should have explored that model too.

In Figure 2, we see a summary of FMOSS's performance over all $p_1 = 0.04$ Toeplitz instances. We observe that FMOSS behaves very similarly to FBMP until about 10^4 model evaluations. Between 10^4 and 10^5 evaluations, FMOSS seems to perform better – for every quantile shown, FMOSS's SQT curves are to the left of FBMP's. Comparing the QRTDs and SQDs of Figure 11 to those of FBMP in Figures 7 and 9 respectively confirms this observation.

It also appears possible that FMOSS is dominating SSS – unfortunately, insufficient model evaluations were performed to confirm whether this is the case. This would be interesting to determine, as if true it would establish FMOSS as the dominant algorithm among the three tested for this class of instances.

3.5.2 $\mathbf{p_1 = 0.12}$ Toeplitz Instances

The performance of FMOSS on four $p_1 = 0.12$ Toeplitz instances is summarized in Figure 13. As before, deterministic behavior was observed; this time 19 out of the 20 instances appeared to behave deterministically across their 20 runs. The sole exceptional instance is shown in the plot.

In Figure 2, we see a summary of FMOSS's performance over all $p_1 = 0.12$ Toeplitz instances. Again here, the cosine distance from $\mathbf{w_{true}}$ decreases in a mostly monotonic fashion.

On the overall plot, we observe that FMOSS tends to track FBMP quite closely up until 25k iterations (which is longer than what we observe for Toeplitz .04 instances, consistent with the fact that these instances tend to be more "deterministic"). The curves look very similar, except near 100k, where the difference between the medians reaches about 0.05. However, this is not very significant since the inter-quartile range is around 0.15.

[5] $\hat{\mathbf{w}}$ is determined from $\hat{\mathbf{s}}$ by performing the linear regression specified in Eq. 17 of Schniter *et al* [2]

4 Conclusions

In this report, we have presented a thorough analysis of the performance characteristics of two recent algorithms for sparse linear regression in a Bayesian setting – Fast Bayesian Matching Pursuit, a greedy algorithm, and Shotgun Stochastic Search, which uses local search. We observed that for data sets with independent regressors and given an equivalent budget for model evaluations, FBMP dominates SSS. In terms of real-world implication, this result is even stronger – FBMP is also able to evaluate models 50 times faster than SSS, due to its use of fast posterior updates. However, in situations with dependent regressors and a relatively unrestricted time budget, we have also shown that SSS is tends to yield estimates that are closer to $\mathbf{w}_{true}$.

We have also introduced FMOSS, which combines the FBMP prior with a local search strategy. For the dependent-variable instance class on which SSS outperformed FBMP, we demonstrated that FMOSS also outperforms FBMP. This suggests that at least part of the long-run advantage seen in SSS is due to its search strategy. However, our analysis is not able to rule out the possibility that the SSS prior is more robust to this type of dependent data.

Future Work

There are two main directions in which this work could be extended. First, in terms of the empirical analysis, several simplifying assumptions were made which could in principle be relaxed. For example, runtime was measured in terms of model evaluations – while this method is justified, it does not capture the inherent overhead involved in implementing complex search strategies to select the models for evaluation. In practice, these details can have a large impact on overall performance.

Another simplification made in this analysis is that we focus our analysis on the MAP estimate, when in fact these algorithms output a Bayesian model average of many high-probability candidate models. While it is true that finding better MAP estimates usually leads to better averaged models, the two criteria are fundamentally different, and a more complete analysis would take this into account.

On a related point, an analysis of domination between two algorithms which share common prior should evaluate their search strategies in terms of the implemented objective function. Since FBMP and SSS have different priors, we have instead compared $\hat{\mathbf{w}}$ against $\mathbf{w}_{true}$. However, for FBMP and FMOSS, this indirect mode of analysis was unnecessary. It would be interesting to perform the same analyses, using posterior probability as the solution quality metric.

The second direction is in further development of FMOSS. Several limitations to the current implementation have been discussed – for example, the neighborhood definition is more restrictive than that of SSS, and the number of candidates on the search frontier is artificially limited. Second, its observed deterministic behavior is puzzling. It would be interesting to investigate why FMOSS seems to behave more deterministically on these harder instances; we suspect this is because the algorithm encounters more "obviously" active variables when searching. Finally, the implementation currently entirely unoptimized. It is expected that with a bit of engineering effort, FMOSS could be sped up by one to two orders of magnitude of CPU time.

Further, a more detailed analysis of the instance-level performance of the local search algorithm could suggest further algorithmic improvements. The local search strategies of FMOSS and SSS are both quite simple, and it seems likely that there are optimizations which could significantly improve their performance. For example, neither FMOSS nor SSS incorporate any sort of random-restart or random-walk strategy to escape bad regions of the search space; such techniques are often critical in the design of high-performance search algorithms.

Finally, both SSS and FMOSS have non-prior-related parameters (annealing rates for SSS, and c' and $S_{\max}$ for FMOSS). These parameters were tuned only coarsely before these experiments, and their effect was not studied in detail. It would be interesting to investigate how much improvement could be realized by adopting a more sophisticated parameter tuning methodology.

Acknowledgements

The authors would like to thank Kevin Murphy and M. Emtiyaz Khan for many discussions throughout the course of the work described here, and for providing code which was a useful reference during the generation of test data and the implementation of FMOSS.

References

[1] C. Hans, A. Dobra, M. West: Shotgun Stochastic Search for "Large p" Regression. *Journal of the American Statistical Association 2007, vol. 102, no478, pp. 507-516.*(2007)

[2] P. Schniter, L. C. Potter, J. Ziniel: Fast Bayesian Matching Pursuit. *Proc. Workshop on Information Theory and Applications.*(2008)

[3] P. Schniter, L. C. Potter, J. Ziniel: Fast Bayesian Matching Pursuit: Model Uncertainty and Parameter Estimation for Sparse Linear Models. http://www.ece.osu.edu/ zinielj/FBMP_TransSP.pdf (2009)

[4] N. Meinshausen, G. Rocha, and B. Yu: A tale of three cousins: Lasso, L2Boosting, and Danzig. *Annals of Statistics* (2007)

[5] A. Dobra, H. Massam: The mode oriented stochastic search (MOSS) algorithm for log-linear models with conjugate priors. http://www.csss.washington.edu/Papers/wp84.pdf *Submitted for publication.* (2008)

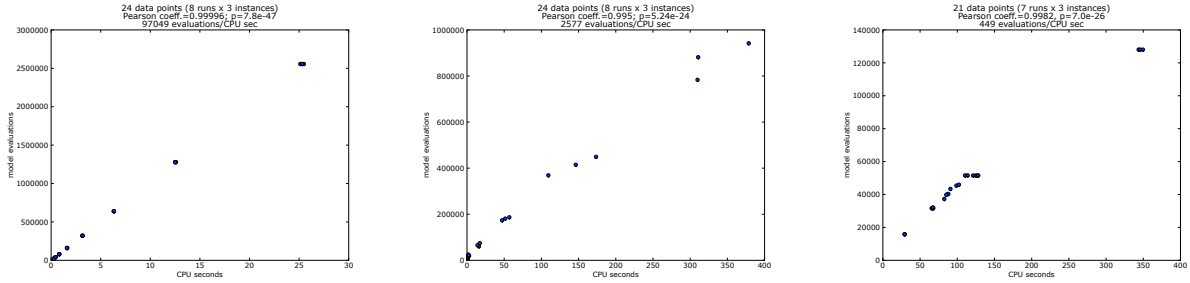

Figure 1: CPU time vs. model evaluations for FBMP, SSS, and FMOSS respectively. Timing runs were performed on a 2.4GHz Core 2 Duo computer with 2GB RAM, running OS-X 10.5.6.

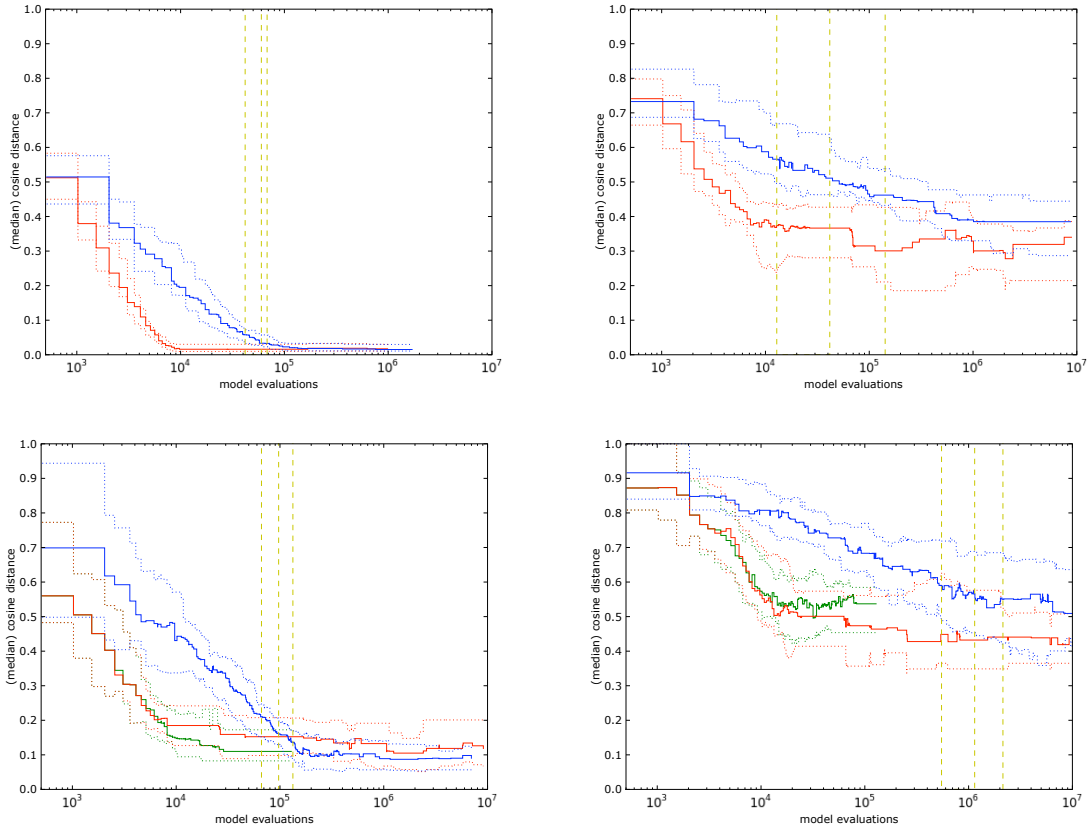

Figure 2: SQTs comparing FBMP (red), median SSS (blue), and median FMOSS (green) performance, aggregated across all Diagonal (top) and Toeplitz (bottom) instances. Instances on the left have $p_1 = 0.04$; on the right $p_1 = 0.12$. Solid lines indicate median values; dotted lines indicate first and third quartiles. Dashed vertical lines indicate the transition between exact and approximate SSS incumbent data.

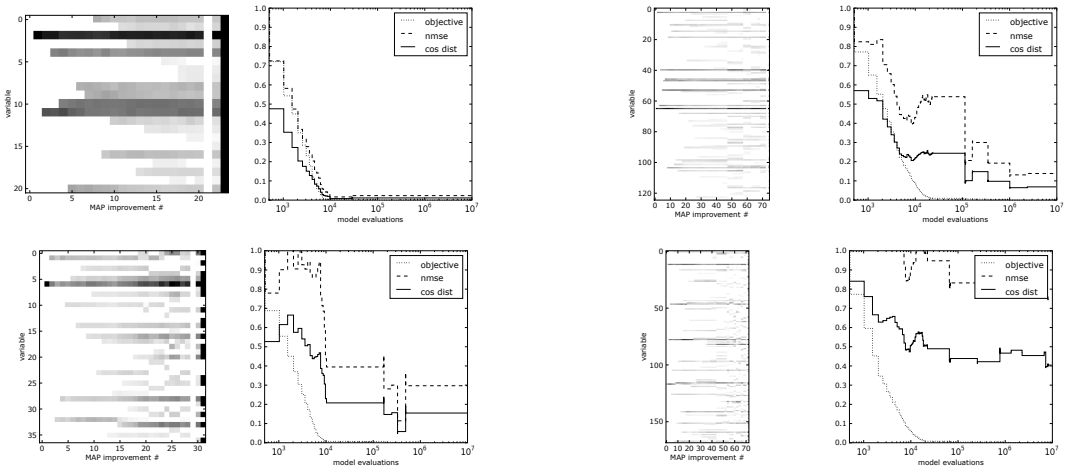

Figure 3: SQTs for FBMP on sample Diagonal (top) and Toeplitz (bottom) instances. Instances chosen exhibit behavior typical of their class. Instances on the left have $p_1 = 0.04$; on the right $p_1 = 0.12$.

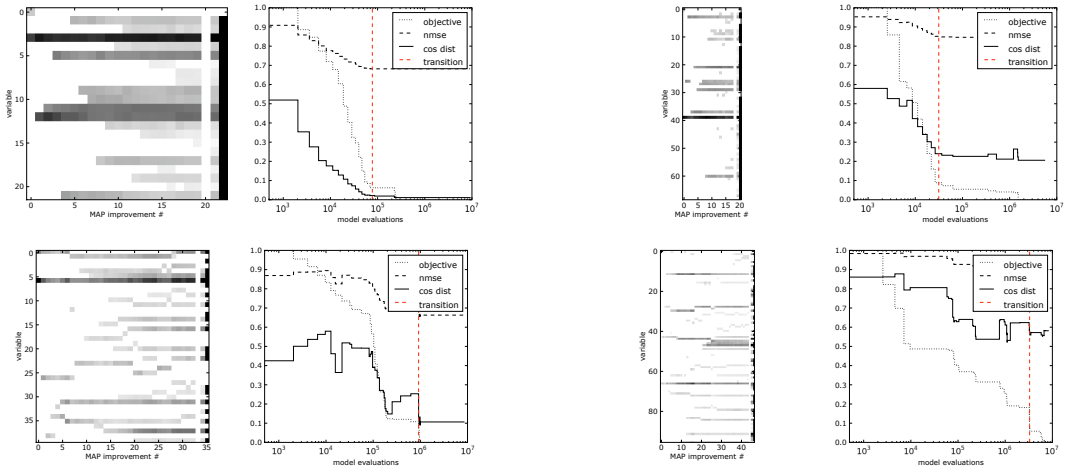

Figure 4: Typical single-run SQTs for SSS on sample Diagonal (top) and Toeplitz (bottom) instances. Instances are the same as those of Figure 3.

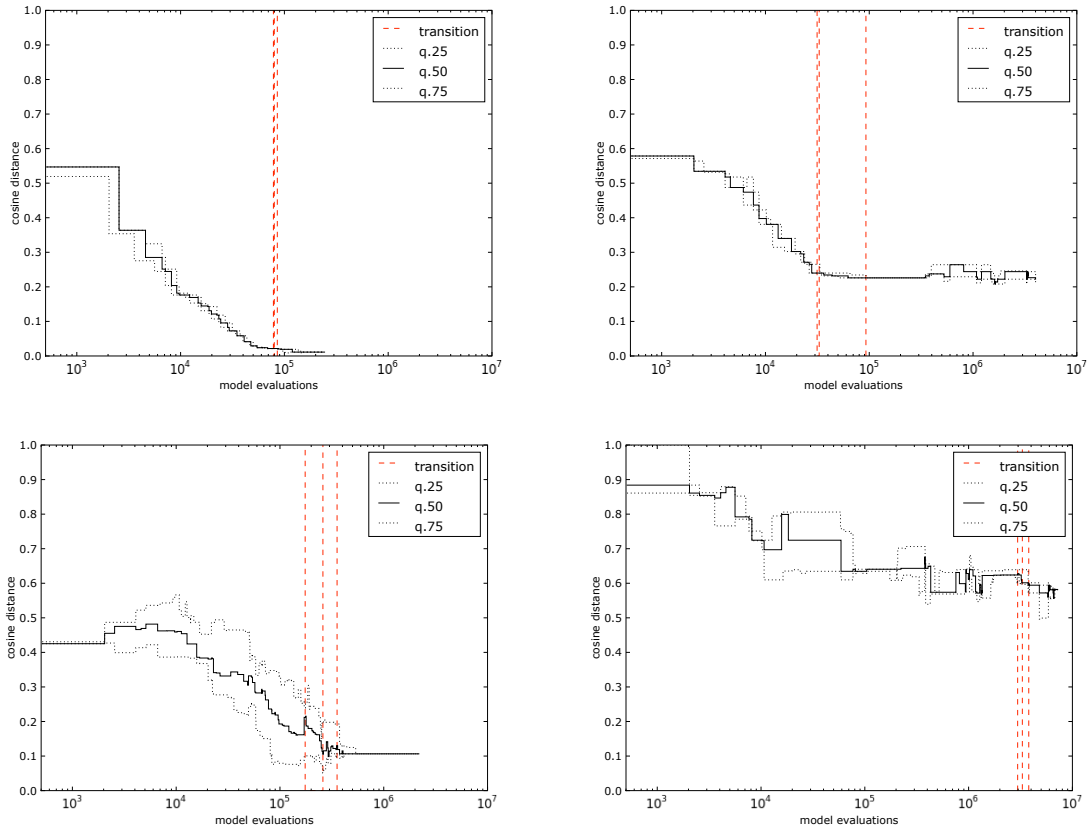

Figure 5: SQTs summarizing performance of SSS on sample Diagonal (top) and Toeplitz (bottom) instances. Median and quartile runtimes across all runs of the instance are plotted. Instances are the same as those of Figure 3.

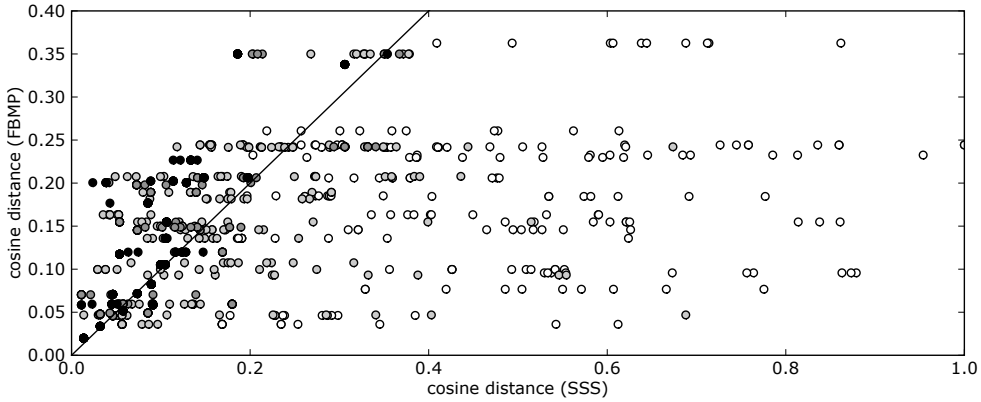

Figure 6: Scatter plot comparing FBMP and SSS on all $p_1 = 0.04$ Toeplitz instances at different run lengths. In order of decreasing darkness, lengths are 10^7, 10^6, 10^5, and 10^4 model evaluations.

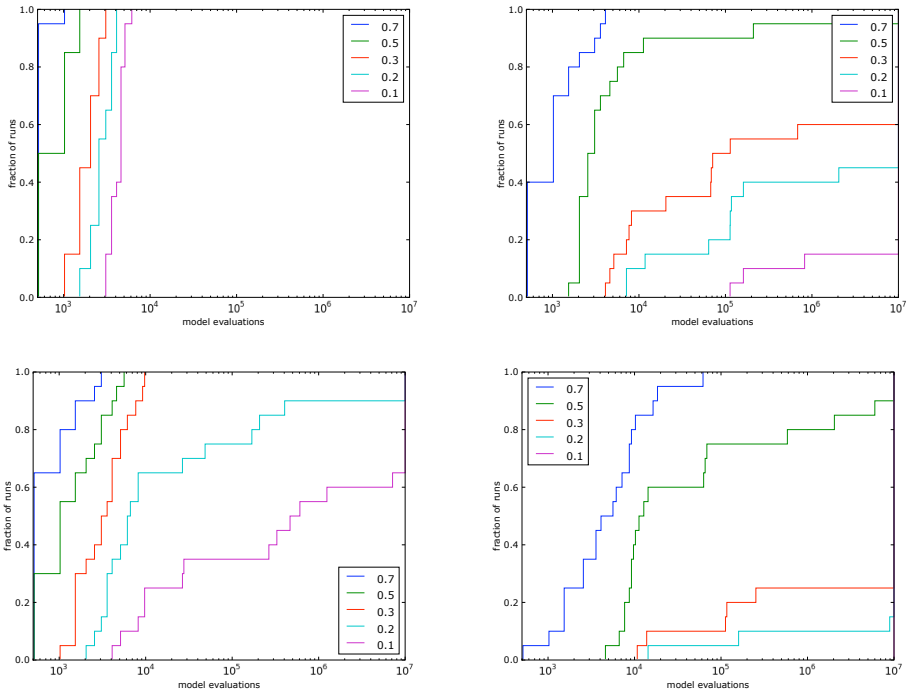

Figure 7: QRTDs plotting evaluations required to first achieve a fixed cosine distance, for FBMP across all Diagonal (top) and Toeplitz (bottom) instances. Instances on the left have $p_1 = 0.04$; on the right $p_1 = 0.12$.

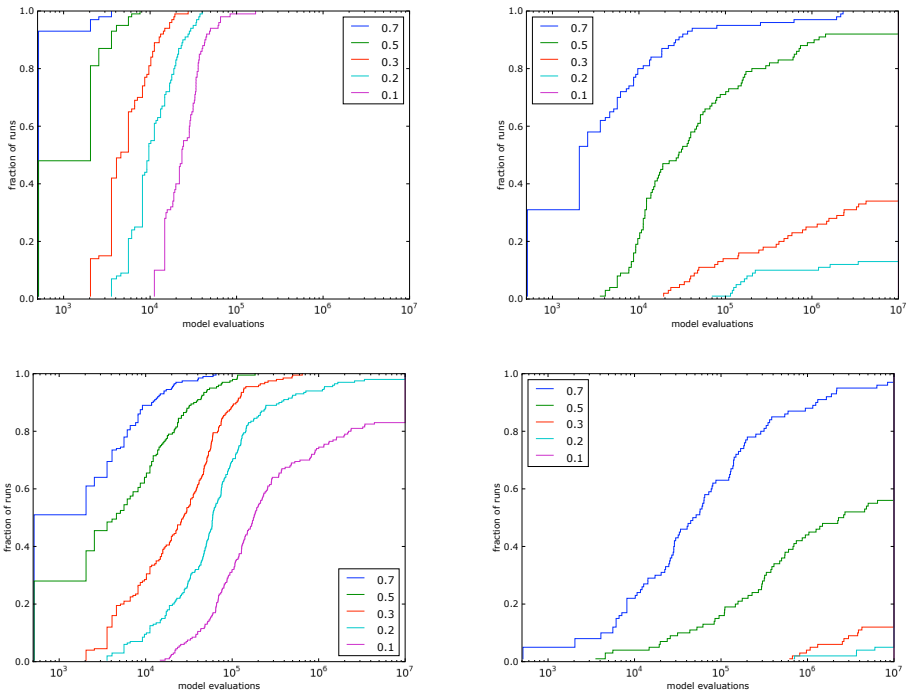

Figure 8: QRTDs plotting evaluations required to first achieve a fixed cosine distance, for SSS across all Diagonal (top) and Toeplitz (bottom) instances. Instances on the left have $p_1 = 0.04$; on the right $p_1 = 0.12$.

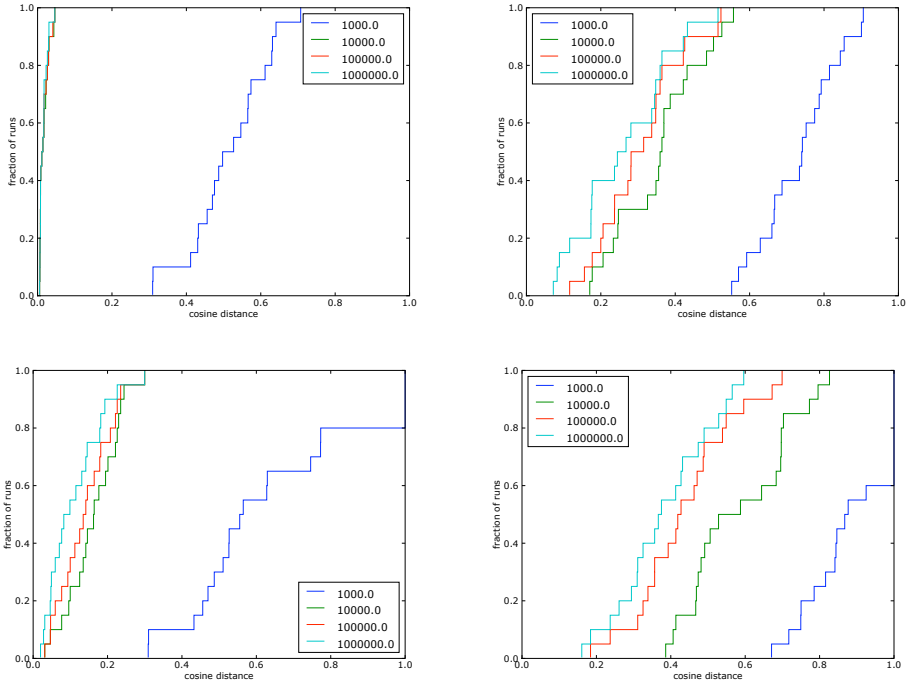

Figure 9: SQDs plotting best cosine distance achieved within a fixed number of evaluations, for FBMP across all Diagonal (top) and Toeplitz (bottom) instances. Instances on the left have $p_1 = 0.04$; on the right $p_1 = 0.12$.

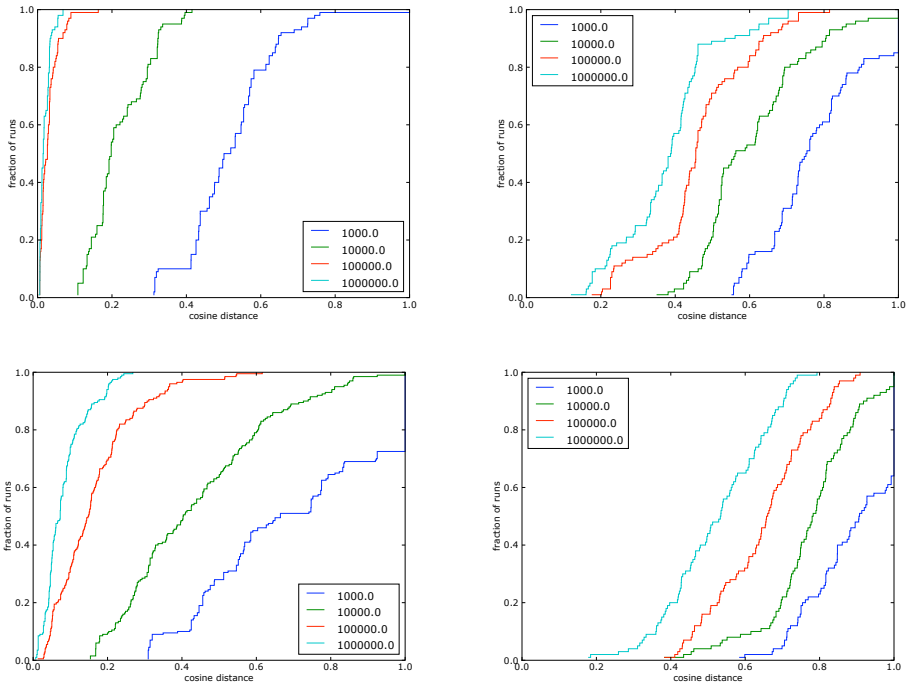

Figure 10: SQDs plotting best cosine distance achieved within a fixed number of evaluations, for SSS across all Diagonal (top) and Toeplitz (bottom) instances. Instances on the left have $p_1 = 0.04$; on the right $p_1 = 0.12$.

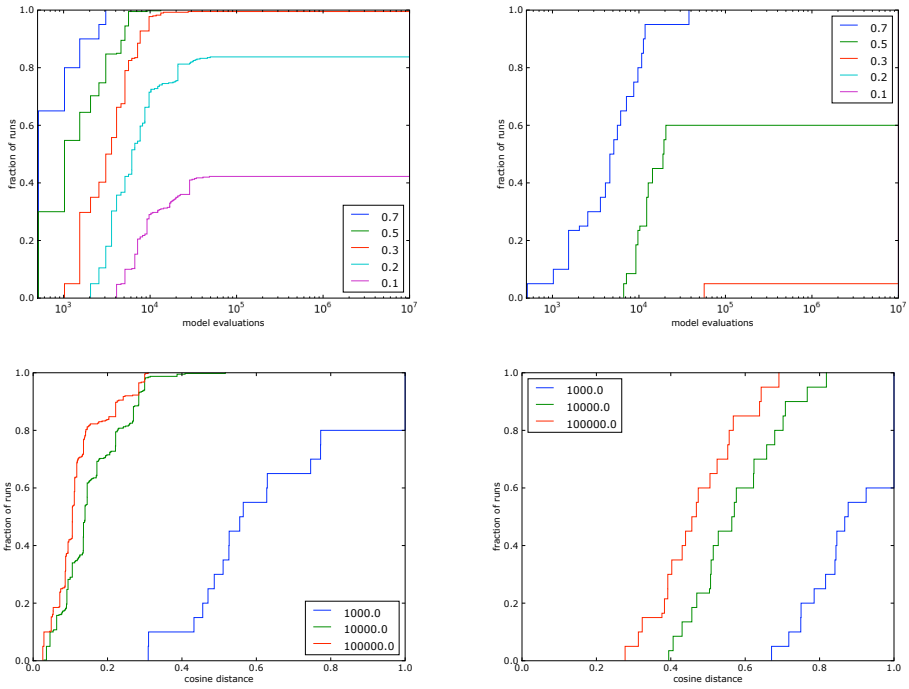

Figure 11: QRTDs (top) and SQDs (bottom) for FMOSS across all Toeplitz instances. Instances on the left have $p_1 = 0.04$; on the right $p_1 = 0.12$.

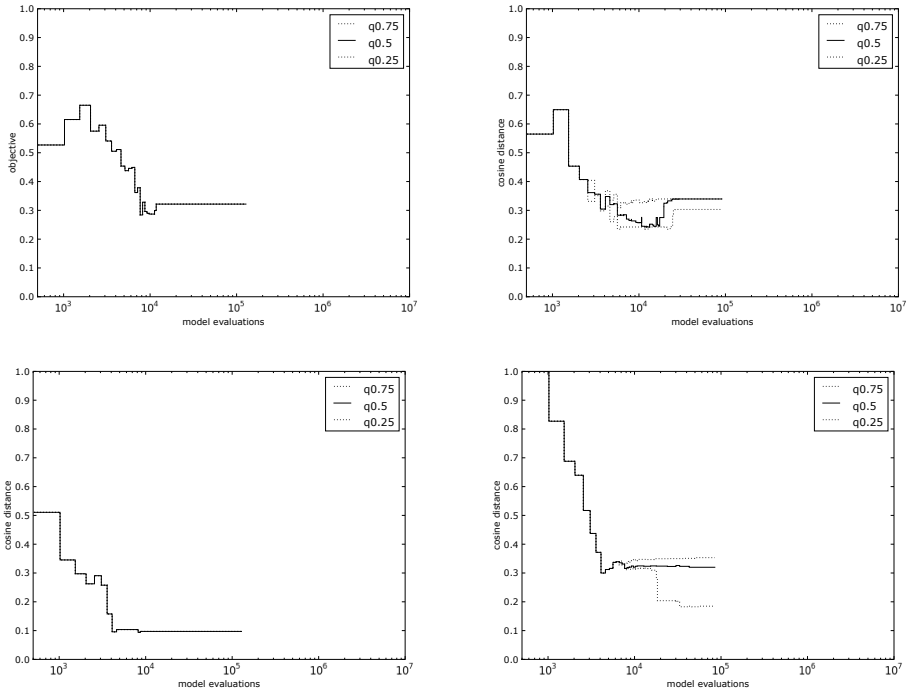

Figure 12: SQTs for FMOSS on sample $p_1 = 0.04$ Toeplitz instances. The first plot corresponds to the $p_1 = 0.04$ Toeplitz instance of Figure 3.

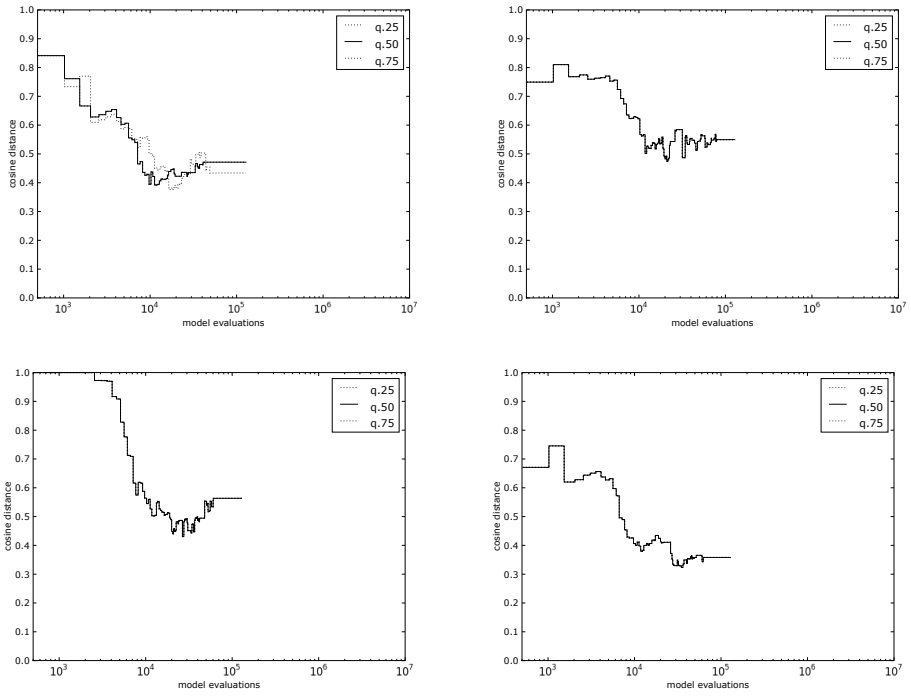

Figure 13: SQTs for FMOSS on sample $p_1 = 0.12$ Toeplitz instances. The first plot corresponds to the $p_1 = 0.12$ Toeplitz instance of Figure 3.