Online Appendix to: Extended Wavelets for Multiple Measures

ANTONIOS DELIGIANNAKIS
University of Athens
MINOS GAROFALAKIS
Intel Research
and
NICK ROUSSOPOULOS
University of Maryland

A. ANALYSIS OF THE GREEDYL2 ALGORITHM

We now present the detailed proof of Theorem 1, as well as a useful corollary of our analysis that is employed in the discussion of the time/space complexity of GreedyL2.

Proof of Theorem 1. Our proof bears similarities to the corresponding proof for the 0-1 Knapsack problem. A significant observation is that whenever we select a set $PickedSet$ of coefficient values from a combined coefficient *Coeff* for storage, any candidate set *subOpt* of *Coeff* that will later be inserted into the max-heap for consideration cannot have a larger per-space benefit than that of *PickedSet*. We will prove this by contradiction. Assume that *subOpt* has a larger per-space benefit than $PickedSet$. By the way the candidate sets are formed, the following observations hold:

(1) The sets *PickedSet* and *subOpt* cannot share any coefficient values.
(2) The largest benefit of a coefficient value of *subOpt* cannot be larger than the smallest benefit of a coefficient value of *PickedSet*.
(3) The space overhead of *subOpt* does not include the size of the header, while for *PickedSet*, this depends on whether it is the first set of *Coeff* selected for storage.

If we depict the benefits of the coefficient values of $PickedSet$ as $v_1, v_2, \ldots, v_k$ and the benefits of the coefficient values of *subOpt* as $v'_1, v'_2, \ldots, v'_p$, then the per-space benefits of the two sets are, correspondingly, $\frac{\sum_{i=1}^{k} v_i}{\delta \times H + k}$ and $\frac{\sum_{i=1}^{p} v'_i}{p}$, where δ has a value of 1 or 0, depending on whether *PickedSet* is the first set of *Coeff* selected for storage. Since, by hypothesis, *subOpt* has a larger per-space benefit than *PickedSet*:

$$\frac{\sum_{i=1}^{p} v'_i}{p} > \frac{\sum_{i=1}^{k} v_i}{\delta \times H + k} \Longrightarrow \sum_{i=1}^{p} v'_i \times (\delta \times H + k) > \sum_{i=1}^{k} v_i \times p. \qquad (1)$$

At the time *PickedSet* was selected, a candidate set *notPicked* of *Coeff* with $k+p$ subitems existed, having a benefit at least equal to the set union containing all subitems in *PickedSet* and *subOpt*. Comparing the per-space benefits of *notPicked* and *PickedSet*, we have

$$\begin{aligned} &\textit{benefit(notPicked)} - \textit{benefit(PickedSet)} \geq \textit{benefit(union)} - \textit{benefit(PickedSet)} \\ &= \frac{\sum_{i=1}^{k} v_i + \sum_{i=1}^{p} v'_i}{\delta \times H + (k+p)} - \frac{\sum_{i=1}^{k} v_i}{\delta \times H + k} = \frac{\sum_{i=1}^{p} v'_i \times (\delta \times H + k) - \sum_{i=1}^{k} v_i \times p}{(\delta \times H + (k+p)) \times \delta \times H + k} > 0. \end{aligned} \tag{2}$$

The last part of formula (2) follows immediately from the inequality of formula (1). At this point, we have reached a contradiction, since *PickedSet* should not have a smaller per-space benefit than the set *notPicked*. Therefore, *subOpt* cannot have a larger per space benefit than *PickedSet*.

The previous observation also implies that each candidate set inserted in the max-heap after the first set selection made by the algorithm cannot have a larger per-space benefit than those which have already been selected. To prove this, consider any such set *nowInserted* that is inserted into the max-heap following the selection for storage of another set *nowStored*, of the same candidate combined coefficient *Coeff*, and consider the following two observations:

(1) Following the preceding proof, the set *nowInserted* cannot have a larger per-space benefit than any set already picked for storage from the same candidate combined coefficient *Coeff*.
(2) Consider any candidate set *otherSet* already selected for storage which corresponds to a candidate combined coefficient other than *Coeff*. At the moment *otherSet* was selected for storage, the candidate set of *Coeff* with the largest per-space benefit which was at that time in the max-heap could not have a larger per-space benefit than *otherSet*, since it would have been selected for storage instead, and cannot have a smaller per-space benefit than *nowStored*.

Now, consider that GreedyL2 solution has selected to store the sets $S_1, S_2, \ldots, S_l$, and that S_{l+1} is that set with the largest per-space benefit, which cannot be stored due to space constraints.[1] Let $BenStored = \sum_{i=1}^{l} S_i.psb \times S_i.space$ denote the sum of benefits of the l sets included in the solution (using the notation of Figure 4), $BenFraction = S_{l+1}.psb \times S_{l+1}.space$ denote the benefit of set S_{l+1}, and *BenOptimal* denote the benefit of the optimal solution. If the remaining storage space at this point of our algorithm is *SpaceLeft*, it can easily be shown[2] that the optimal solution has, at most, benefit equal to $BenStored + BenFraction \times \frac{SpaceLeft}{S_{l+1}.space}$.

Obviously, $S_l.psb \leq \frac{BenStored}{B - SpaceLeft} \leq \frac{BenStored}{\max\{H+1, B-(H+M)\}}$ (since the space of the stored sets must be at least $H+1$ and at least equal to $B-(H+M)$), and, since the per-space benefit of S_{l+1} cannot be larger than that of S_l, $S_{l+1}.psb \leq S_l.psb \Rightarrow BenFraction \times \frac{SpaceLeft}{S_{l+1}.space} \leq (H+M) \times S_l.psb \leq \frac{BenStored \times (H+M)}{\max\{H+1, B-(H+M)\}}$.

[1]For simplicity, consider that S_{l+1} could fit by itself within the original storage constraint.
[2]The proof is identical to the optimal proof for the fractional Knapsack problem.

Therefore,

$$\begin{aligned} \mathit{BenOptimal} &\le \mathit{BenStored} + \mathit{BenFraction} \times \frac{\mathit{SpaceLeft}}{S_{l+1}.\mathit{space}} \\ &\le \mathit{BenStored} \times \left(1 + \frac{H+M}{\max\{H+1, B-(H+M)\}}\right). \end{aligned}$$

For $B \ge 2H + M + 1$, the proof of the approximation ratio bound is complete. For $B < 2H + M + 1$, the solution Z = $\max\{\mathit{BenFraction}, \mathit{BenStored}\}$ has at least half the benefit of the optimal solution, since

$$\mathit{BenOptimal} \le \mathit{BenFraction} + \mathit{BenStored} \le 2 \times \mathit{max}\{\mathit{BenFraction}, \mathit{BenStored}\}.$$

□

COROLLARY 1. *Let $B - (H + M) < B_{used} \le B$ denote the space occupied by the* GreedyL2 *algorithm's solution at the time the first* PickedSet *(that requires space larger than* SpaceLeft $= B - B_{used}$*) is selected. The* GreedyL2 *algorithm always selects the optimal solution for the space constraint B_{used}.*

PROOF. Using the notation of the previous proof, for a space constraint B_{used}, there is no unused space (i.e., $\mathit{SpaceLeft}' = 0$). From the preceding proof, we can then infer that $\mathit{BenFraction} = 0$, resulting in $\mathit{BenOptimal} \le \mathit{BenStored} + \mathit{BenFraction} = \mathit{BenStored}$. □

B. PRACTICAL CONSIDERATIONS

Providing fairness and error guarantees. While the optimization problems of Sections 3 and 4.2 might be desirable objectives in many problems, certain cases may arise when both of our corresponding (for each minimization problem) greedy and dynamic programming algorithms will significantly favor certain measures at the expense of others. This usually happens when two or more measures with significantly different magnitudes of value occur within the same dataset. In such cases, measures with the largest data values typically exhibit not only significantly larger (in magnitude) coefficient values, but also larger error in their approximations. Note that in such datasets, both the DynProgL2 and GreedyL2 algorithms will almost exclusively store coefficient values corresponding to the measure with the largest coefficient values. A similar behavior will be observed if we slightly modify our PODPRel and GreedyRel algorithms to minimize the maximum absolute-error of the approximation.[3] Finally, a similar behavior can be observed, even for the maximum relative-error metric, if the value of the sanity bound S is chosen to be the same for all measures. Note that in this case, those two measures whose data (and therefore, whose coefficient) values are linearly correlated, and thus equally hard to approximate, may exhibit significantly different maximum relative-errors due to the common value of the sanity bound.

[3]According to the corresponding algorithms in Garofalakis and Gibbons [2004], this just requires a small modification to Eq. (1).

A plausible solution to this problem would be to normalize the values of all measures such that all measures have the same energy. The *energy* of a measure is defined to be the sum of its squared values. A natural way to normalize two or more measures is to subtract from each measure's data values the measure's average value, and then properly scale the produced data values so that all measures have the same energy. Alternative techniques to achieve a similar normalization process would be to either divide the weighted benefit of each coefficient value by the energy of the corresponding measure (for the minimization problem of Section 3) or to select the sanity bound of each measure (for the minimization problem of Section 4.2) based on the magnitude of its actual data values (i.e., select the 5%-quantile of the measure's data values).

Another solution involves adapting our proposed algorithms to provide certain guarantees on the quality of the produced solution. It turns out that all of our algorithms can be modified such that the resulting weighted benefit (or maximum squared NSE) for each measure is at least equal to the one produced by the Individual algorithm when the storage allocated to each measure is proportional to the measure's weight (which is set equal to 1 for the maximum relative-error minimization problem). This requires first computing the actual solution that the Individual algorithm would produce by using the aforementioned space allocation policy among measures. Then, the solution produced by our algorithms can be tested to verify whether for each measure the benefit (or maximum squared NSE) criterion has been met. If this is not the case for all measures, the algorithm can proceed by splitting these measures into two sets: those where the benefit (or maximum squared NSE) criterion has been satisfied, and those where it has not. The algorithms can then be called recursively for each of these two sets, with the storage constraint allocated to each set being proportional to the weights of its measures. If at some point an input set contains just one measure, then the algorithm may store the coefficients in the same format as does the Individual algorithm. Thus, in the worst case, for any measure we can guarantee that the benefit (or maximum squared NSE) we achieve is at least the same as the one produced by the Individual algorithm.

Improving space utilization. The space utilization of our algorithms can be further improved at the expense of query response time. For a dataset with M measures, we can split the produced coefficients into $M + 2$ groups of coefficients. One group will be created for each measure and include all extended wavelet coefficients that have stored a coefficient value only for the corresponding measure. Another group will contain the extended coefficients that have stored coefficient values for all M measures, while the final group will include extended coefficients that have stored from 2 to $M - 1$ coefficient values. From these $M + 2$ groups, the bitmap is necessary only for the last group. In the other groups, we can simply store the coefficients in the same way that the Individual and Combined algorithms would, without the bitmap. The proposed algorithms then only require a slight modification when calculating the size needed to store a coefficient value (for the DynProgL2 algorithm) or a set of coefficient values (for the remaining algorithms). A query involving X measures would then have to probe $X + 2$ groups of coefficients in search for coefficient values that influence the query result. This overhead in response time is, in most cases, negligible, given the small response times that queries exhibit when using wavelet synopses [Chakrabarti et al. 2001].

C. EXTENSIONS TO MULTIDIMENSIONAL WAVELETS

We now discuss the key ideas for extending our relative-error synopsis construction techniques to multidimensional data.

Extending PODPRel. Our PODPRel algorithm for multidimensional datasets generalizes the corresponding multidimensional MinRelVar strategy in Garofalakis and Gibbons [2004] in a way analogous to the one-dimensional case. In a nutshell, PODPRel needs to consider, at each internal node of the error tree, the optimal allocation of space to the $\leq 2^D - 1$ wavelet coefficients of the node and its $\leq 2^D$ child subtrees. The extension of PODPRel to multidimensional datasets is therefore a fairly simple adaptation of the multidimensional MinRelVar algorithm. However, as discussed in Section 4.3, PODPRel needs to maintain, for each node i and each possible space allotment B, a collection $\mathcal{R}[i, B]$ of incomparable solutions. This requirement, once again, makes the time/space requirements of PODPRel significantly higher than those of MinRelVar.

Extending GreedyRel. The first modification involved in extending our GreedyRel algorithm to multidimensional datasets has to do with the computation of $G[i, j]$, which now involves examining the estimated NSE^2 values over $\leq 2^D$ child subtrees and maintaining the maximum such estimate. Let $\mathcal{S}(i)$ denote the set of $\leq 2^D - 1$ coefficients of node i, and let $i_1, \dots, i_p$ be the indexes of i's child nodes in the error tree. Then,

$$G[i, j] = \begin{cases} \max \begin{cases} \sum_{c_k \in \mathcal{S}(i)} \frac{\text{Var}(c_{kj}, y_{kj})}{\text{Norm}(i_1, j)} + G[i_1, j] \\ \dots \\ \sum_{c_k \in \mathcal{S}(i)} \frac{\text{Var}(c_{kj}, y_{ij})}{\text{Norm}(i_p, j)} + G[i_p, j] \end{cases} & i < N \\ 0 & i \geq N \end{cases}.$$

The only other necessary modification involves the estimation of marginal error-gains at each node. In Section 4.4, we consider a total of three possible choices for forming potSet$[i, j]$ for each (node, measure) combination. Now, each node has up to 2^D child subtrees, resulting in a total of $2^D + 1$ possible choices of forming potSet$[i, j]$. The first choice is to increase the retention probability for measure j of one of the $\leq 2^D - 1$ coefficients in node i. In this case, we simply include in potSet$[i, j]$ the coefficient in node i that is expected to exhibit the largest marginal gain for measure j. For each of the remaining 2^D possible choices of forming potSet$[i, j]$, the kth choice ($1 \leq k \leq 2^D$) considers the marginal gain of increasing the retention probabilities in the child subtrees through which the k maximum NSE^2 values occur, as estimated in the righthand-side of the preceding equation for $G[i, j]$. At each node, the computation of $G_{\text{pot}}[i, j]$, potSpace$[i, j]$, and choice$_{ij}$ incurs a worst-case time cost of $O(D \times 2^D)$ due to the possible ways of forming potSet$[i, j]$, and the required sorting operation of 2^D quantities. Again, let N denote the total number of cells in the multidimensional data array and

N_{max} denote the maximum domain size of any dimension. Then, the running time complexity of GreedyRel becomes $O(D \times 2^D \times (NM + BM\mathrm{q}\log N_{max}))$. Note, of course, that in most real-life scenarios using wavelet-based data reduction, the number of dimensions is typically a small constant (e.g., four–six).

Improving the complexity of GreedyRel. In the wavelet decomposition process of a multidimensional dataset, the number of nonzero coefficients produced may be significantly larger than the number N_z of nonzero data values. In Garofalakis and Gibbons [2004], the authors proposed an adaptive coefficient thresholding procedure that retains, at most, N_z wavelet coefficients *without* introducing any reconstruction bias. Using this procedure, the authors in Garofalakis and Gibbons [2004] demonstrated how the MinRelVar algorithm can be modified so that its running time and space complexities have a dependency on N_z, and not on N (i.e., the total number of cells in the multidimensional data array). It would thus be desirable if the GreedyRel algorithm could be modified in a similar way, in order to decrease its running time and space requirements.

Let N_z denote the number of error-tree nodes that contain nonzero coefficient values, possibly after the aforementioned thresholding process. We will first illustrate that for any node in the error tree containing zero coefficient values, and which has (at most) one node in its subtree that contains nonzero coefficient values, no computation is needed. Equivalently, our algorithm will need to compute G, G_{pot} values in only: (i) nodes containing nonzero coefficient values; or (ii) nodes that contain zero coefficient values, but which are the least common ancestor of at least two nonzero tree nodes beneath them in the error tree.

Let k be a node that is the only node in its subtree with nonzero coefficient values. Obviously, we do not need to consider the G, G_{pot} values in the descendant nodes of k, since they will be zero. An important observation is that for any ancestor of k which contains just a single nonzero error tree beneath it (which is certainly the subtree of node k), no computation is necessary, since the G, G_{pot} values of k can always be used instead. The only additional computation is needed in any node n with zero-coefficients that has at least two nonzero error-tree nodes beneath it in the error tree (in different subtrees). In this case, the G, G_{pot} values of node n need to be calculated, using as input the G, G_{pot} values of its nonzero descendant tree nodes. It is easy to demonstrate that at most $N_z - 1$ such nodes may exist. Thus, the GreedyRel algorithm will need to calculate the G, G_{pot} values in at most $O(2N_z - 1) = O(N_z)$ nodes, thus yielding running time and space complexities of $O(D \times 2^D \times (N_z M + BM\mathrm{q}\log N_{max}))$ and $O(N_z M)$, respectively. We here need to note that in order to implement our algorithm as described here, we need to sort the N_z coefficients based on their postorder numbering in the error tree. This requires additional $O(N_z \log N_z)$ time for the sorting process. However, this running time is often significantly smaller than the benefits of having running time and space dependencies based on N_z, rather than on N.

Table I contains a synopsis of the running time and space complexities of both our GreedyRel and the MinRelVar algorithm of Garofalakis and Gibbons [2004].

Table I. GreedyRel and MinRelVar Complexities

Algorithm	Space	Runtime
GreedyRel	$O(N_z M)$	$O(D2^D \times (N_z M + BM\text{q}\log N_{max}))$
MinRelVar	$O(N_z MB2^D\text{q})$	$O(N_z BM2^D\text{q}(\text{q}\log(\text{q}B) + D2^D))$

D. COMPARING GREEDYREL AND GREEDYL2

When comparing the GreedyL2 and GreedyRel algorithms, we can observe that while the two algorithms share some common characteristics, there are distinct differences in the way they operate. Both algorithms operate on all measures of the dataset simultaneously and utilize the notion of extended wavelets in order to achieve better storage utilization, and thus, increased accuracy. Moreover, both algorithms allocate, at each step, space to a *group* of coefficient values. The GreedyL2 algorithm, at each step, stores a group of coefficient values corresponding to the same coefficient coordinates, since this grouped space allocation policy often results in better per-space benefits than storing individual coefficient values one-by-one. On the other hand, the GreedyRel algorithm increases, at each step, the retention probabilities of a group of coefficient values corresponding to the same measure. The intuition behind this allocation policy is that a group assignment may result in a larger per-space decrease of the maximum squared NSE value than increasing the retention probabilities of individual coefficient values, especially in cases where two or more subtrees have similar maximum NSE values. The storage dependencies among coefficient values are taken into account when calculating the per-space benefits of each space assignment on each node. We thus expect that the improvements in accuracy of the obtained approximation will be larger in the case of minimizing the weighted sum-squared error, since the intracoefficient storage dependencies are more directly taken into account. While nonzero coefficient values always have some benefit in reducing the weighted sum-squared error of the approximation, this is often not true when trying to minimize the maximum relative-error of any data value, since these nonzero coefficient values need to lie on root-to-leaf paths with high NSE values in order for the algorithm to increase their retention probabilities. Finally, both algorithms naturally extend to multidimensional data, either directly (GreedyL2) or through simple modifications to account for the more complex error-tree structure (GreedyRel).

E. ADDITIONAL EXPERIMENTS

E.1 Weighted Sum-Squared Error Algorithms

Variance in weights. We varied the weight of the first measure from 0.5 to 4 to identify the impact on accuracy of the produced result. Figures 1 and 2 present the results. GreedyL2 exhibits weighted absolute errors that are consistently about 1.5 times smaller than those of the closest competitor (97.36 versus 147.15 for weight = 3.5). For average weighted sum-squared errors, the GreedyL2 algorithm consistently provides a three-fold improvement, and as high as 3.23 (71,530.3 versus 22,148.4 for weight = 2). It is interesting to note that the GreedyL2 and IndSorted methods exhibit the greatest improvement in

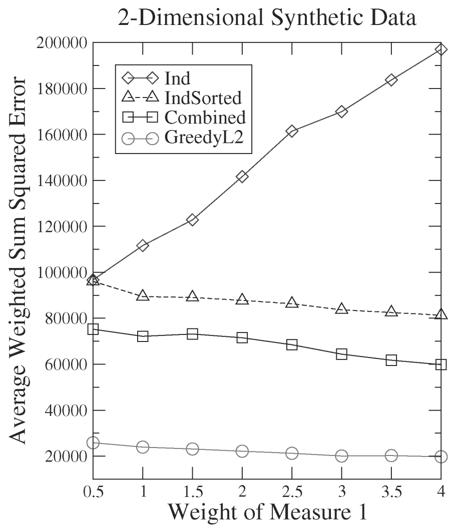

Fig. 1. Sensitivity to weights: sum-squared error.

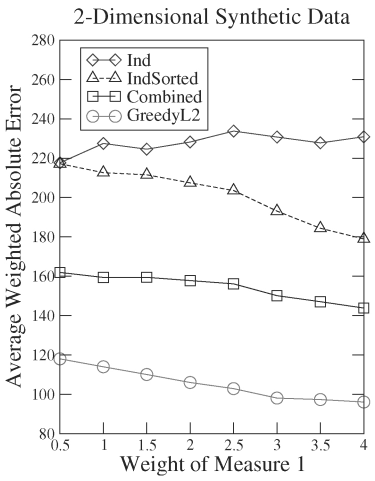

Fig. 2. Sensitivity to weights: absolute error.

accuracy when the weights are varied significantly, both reducing their errors by about 19%.

It is interesting to see for this experiment how well each measure is approximated by the different algorithms. Figure 3 presents the average absolute error for each measure, for the case when the first measure is assigned a weight value of 4. To calculate the average weighted error for all measures, the error of Measure 1 ($M1$) needs to be multiplied by a factor of 4, and the resulting quantity divided by the value 9, which is the sum of the measures' weights. As Figure 3 shows, the Ind algorithm exhibits the smallest error for the measure with the largest weight, about half of the error that GreedyL2 achieves, while the Combined algorithm performs the worst for this measure. However, GreedyL2

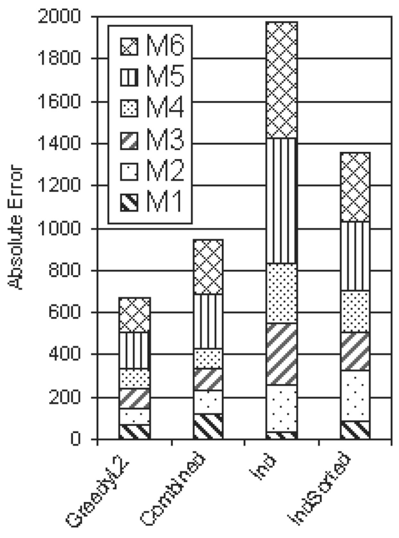

Fig. 3. Errors for different measures.

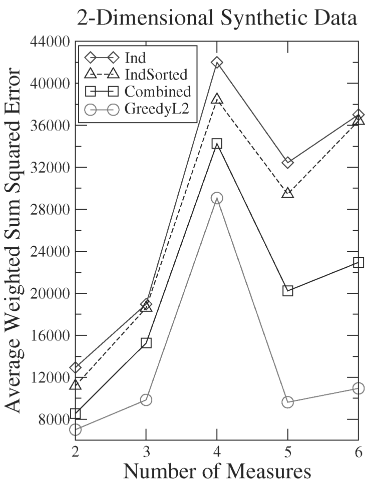

Fig. 4. Sensitivity to number of measures.

achieves the lowest errors for the remaining five measures, thus displaying that even though it can adjust its choices in cases of measures with large weights, it does so without severely impacting the accuracy of the remaining measures. Another interesting observation is that the second measure, which follows a distribution similar to the heavily weighted Measure 1, benefited significantly in the GreedyL2 algorithm, a behavior not observed in the other algorithms.

Number of measures. In Figures 4 and 5, we present the average weighted sum-squared and absolute errors as the number of measures is varied from two to six. The initial two measures are those with distributions Normal and Middle, and the measures that are later added are: PipeOrgan, Altered-Normal, Altered-PipeOrgan, and Altered-Middle. As the number of measures

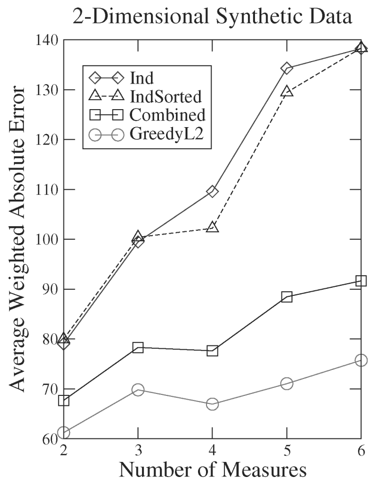

Fig. 5. Sensitivity to number of measures.

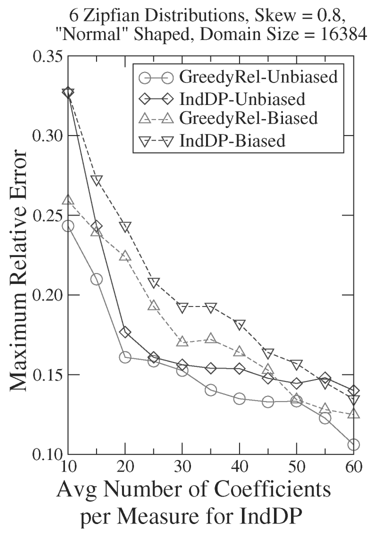

Fig. 6. Skew 0.2, AllNormal.

increases, the improvement on accuracy of GreedyL2 over competitive methods increases.

E.2 Maximum Relative-Error Algorithms

In Figure 6, we plot the maximum relative-errors for both biased and unbiased versions of the GreedyRel and IndDP algorithms as the average number of coefficient values which the IndDP algorithm uses per measure is varied from 10 to 60, and for the AllNormal selection of Zipfian distribution shapes with a skew parameter of 0.2. In Figures 7 and 8, we show the corresponding results for the AllNoPerm and AllNormal combinations of used data distributions and for

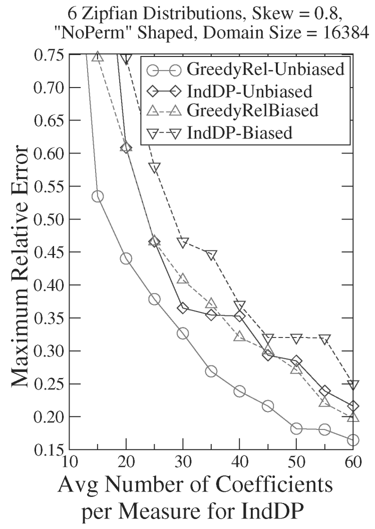

Fig. 7. Skew 0.8, AllNoPerm.

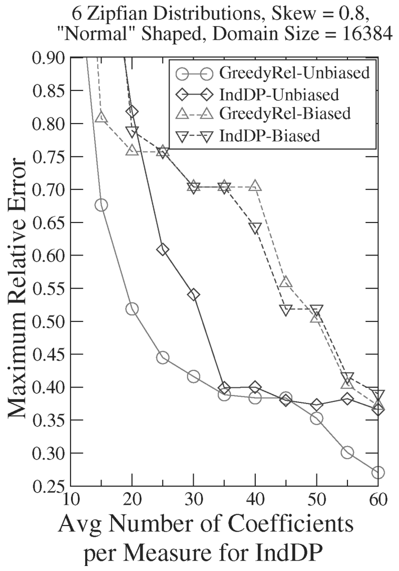

Fig. 8. Skew 0.8, AllNormal.

a skew parameter of 0.8. In all cases, the benefits achieved by the GreedyRel algorithm over the IndDP algorithm are significant.