Offline Design Tuning for Hierarchies of Forecast Models

Ulrike Fischer, Matthias Boehm, Wolfgang Lehner

TU Dresden; Database Technology Group

Abstract: Forecasting of time series data is crucial for decision-making processes in many domains as it allows the prediction of future behavior. In this context, a model is fit to the observed data points of the time series by estimating the model parameters. The computed parameters are then utilized to forecast future points in time. Existing approaches integrate forecasting into traditional relational query processing, where a forecast query requests the creation of a forecast model. Models of continued interest should be deployed only once and used many times afterwards. This however leads to additional maintenance costs as models need to be kept up-to-date. Costs can be reduced by choosing a well-defined subset of models and answering queries using derivation schemes. In contrast to materialized view selection, model selection opens a whole new problem area as results are approximate. A derivation schema might increase or decrease the accuracy of a forecast query. Thus, a two-dimensional optimization problem of minimizing the model cost and model usage error is introduced in this paper. Our solution consists of a greedy enumeration approach that empirically evaluates different configurations of forecast models. In our experimental evaluation, with data sets from different domains, we show the superiority of our approach over traditional approaches from forecasting literature.

1 Introduction

In many domains, gathered data constitutes *time series*, e.g., sales per month, system load per hour, energy supply per minute. This is especially valid in data warehouse systems, where the time dimension is virtually guaranteed to be present [KR02]. This data is often used as a basis of decision-making processes. Forecasting is a fundamental prerequisite for such decisions, otherwise all decisions rely on the history only and might not be valid. One important use case is forecasting of energy supply. Many renewable energy sources (e.g., solar panels) pose the challenge that production depends on external factors (e.g., amount of sunlight). Hence, available power can only be predicted but not planned, which makes it rather difficult for energy distributors to efficiently include renewable energy sources into their daily schedules. This problem addresses the MIRACLE project [BBD$^+$10] that predicts the energy demand and supply of customers and suppliers and balances accordingly. This poses the challenge of high query and update intervals (15-minutes or less), where forecast queries need to be answered as fast and accurate as possible.

There are existing approaches that integrate time series forecasting into traditional relational query processing in DBMS [DB07, Ora10, Pre10]. In contrast to exporting the data to an external statistical program, these approaches allow for joint query processing and

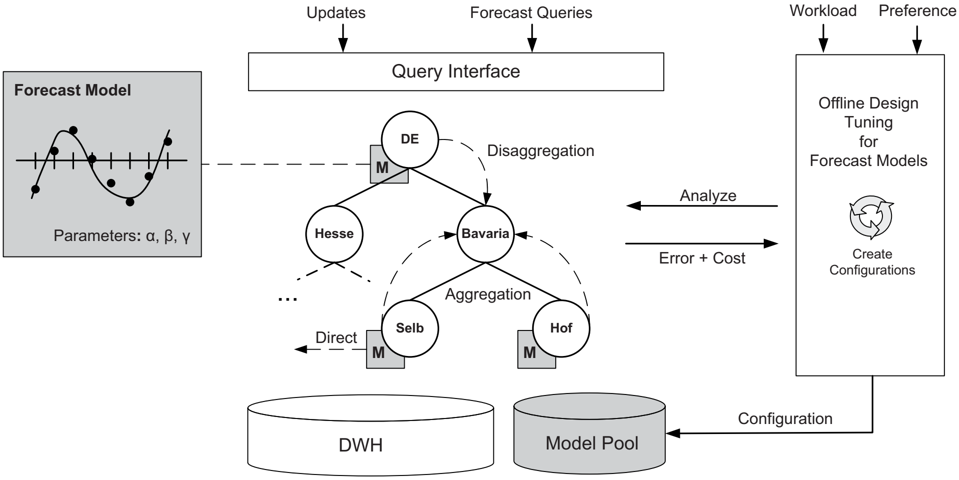


Figure 1: System Overview

exploit database specific optimization techniques. In this context, a *forecast query* is specified like a traditional query extended with a forecast horizon, which specifies the number of values to forecast or a future point in time [DB07, FRBL10]. A forecast query uses a model of the time series at hand to calculate the expected future behavior of the time series. In general, model-based forecasting involves two phases, *model creation* and *model usage*. Model creation tries to fit a model (e.g., represented by the parameters of a continuous function) to the observed data points of the original time series (Figure 1 left). Model creation is typically computationally expensive, often involving numerical optimization schemes to estimate the d model parameters that span a d-dimensional search space. In contrast, model usage utilizes the parameters calculated in the first step to forecast future points of the observed time series. It is cheap as only a few simple operations are necessary. Due to the high model creation costs, query processing can be sped up if models are only built once and kept in a *model pool*. Subsequent queries are answered by choosing a fittable model from this pool [FRBL10, ACL$^+$10, GZ08].

However, as a data-warehouse might contain a high number of individual time series, building a model for each single time series is expensive. In addition, time series characteristics change over time, requiring maintenance in form of parameter re-estimation. Only a few forecast methods however allow updating the parameters analytically by using just the new time series values, most approaches require access to the complete historical time series for parameter re-estimation. Parameter re-estimation might be as expensive as model creation. Therefore, in domains like energy supply, where minute-by-minute data is stored and forecasted, we do not have enough time to keep all models up-to-date until the next query arrives. One solution to this problem is to choose a well-defined subset of forecast models. Forecast queries can then be answered by specific derivation schemes.

There are two main derivation schemes in the area of forecasting – aggregation and disaggregation [Fli01]. *Aggregation* calculates the forecast values of a time series by using forecast values of subset time series, while *disaggregation* uses the forecast values of another time series representing a superset, e.g., by using the historical fraction. In the center of Figure 1 a simplified hierarchy for forecasting energy supply in Germany (abbr. DE) is presented. Here, the energy supply of single cities is recorded at level one. The supply is then aggregated according to different regions (level two) and to the supply over the whole country (level three). Now, the forecast values for the region Bavaria could be either calculated by disaggregation from the forecast values of the total time series over Germany or by aggregation over the forecast values of the single cities Selb and Hof.

However, the accuracy of a forecast value calculated from a model specifically created for a given query might be different from the accuracy of a forecast value derived from models at different aggregation levels of the time series. Interestingly, some derivation schemes might even improve the accuracy. Therefore, in addition to model usage and maintenance cost, we need to minimize the forecast error of queries using this model, resulting in a two-dimensional optimization problem. However, the error of a model cannot be determined without actually building the concerning model [DWD76, ACL$^+$10]. As a result, any solution to this problem requires the empirical comparison of alternative configurations.

In general, this problem seems similar to materialized view selection and usage. However, model selection requires a second metric, the forecast accuracy, while materialized view selection focuses on minimizing query and maintenance cost only. For model usage, forecast queries always output approximated tuples, so we can exploit additional derivation schemes, i.e., disaggregation. On the other hand, forecast models are always created at instance level of the data, so we can not apply compensation queries, e.g., it is impossible to calculate a selection on top of a model while this is a valid usage of materialized views.

To summarize, our offline design tuning algorithm takes as input a workload and a user preference regarding execution time and accuracy. It creates different configurations of models and empirically analyzes their forecast error and maintenance cost (Figure 1 right). As a result, the best configuration for the given workload is provided to the model pool. This configuration leads to a reduction of query processing times as models are already present in the database, a possible higher forecast accuracy as aggregation dependencies are taken into account and less maintenance costs as only necessary models are stored.

Contributions and Outline In summary, we make the following contributions:

- First, we introduce the fundamentals of physical design of forecast models in a data-warehouse schema with multiple defined hierarchies (Section 2.1 and 2.2).
- We then define our two-dimensional optimization problem, i.e., increase forecast accuracy and reduce maintenance cost (Section 2.3).
- Third, we present our greedy enumeration approach to reduce the space of possible configurations (Section 3.1) and present heuristics, which might reduce the number of forecast models considered (Section 3.2).
- Finally, we show the applicability of our approach in an experimental evaluation (Section 4).

We will finish with related work in Section 5 and conclude in Section 6.

2 Multi-Hierarchical Forecasting

In this section, we first sketch the basics of a multi-hierarchical forecasting system. Subsequently, we explain the notion of physical design in such a context. We finish by discussing conflicting optimization challenges and by formulating our general optimization goal.

2.1 Multi-Hierarchical Forecasting System

Hierarchical forecasting is based on grouping time series into groups, group families and so on [Fli01]. Each level results from the aggregation of the child elements one level below. The top level is the total aggregate of all elementary time series. If we transfer the hierarchical forecasting approach to a data-warehouse environment, where multiple hierarchies exists in parallel, merged by foreign keys to dimensions in the fact table, we get a *multi-hierarchical forecasting system*. This generalization also contains different levels of aggregation, where a parent element is calculated by aggregation of corresponding child elements on an arbitrary level below. However, as we have multiple hierarchies, a child might contribute to several parents. In addition, it might be possible to calculate parents' values from multiple sets of child nodes on the same level.

Definition 1 Multi-Hierarchical System: *In a multi-hierarchical system, the value $S_{ij}(t)$ at time t of the time series S_{ij} with index j at level i is calculated as follows:*

$$S_{ij}(t) = \mathit{AGG}_{l \in G_{pij}} S_{(i-1)l}(t),$$

where G_{pij} contains a list of child indexes of group p at level $i-1$, which contribute to the aggregate of element S_{ij}. AGG is an aggregation function, e.g., SUM. Elements at level 1 are the elementary time series, while the element at level h is the total aggregate *over all time series. The total number of elements $|S|$ is calculated by the Cartesian product over the number of elements per hierarchy.*

Example 1 *An example multi-hierarchical forecasting systems is shown in Figure 2. Recall our running example of forecasting energy supply. The energy supply can also be distinguished according to different energy sources, e.g. solar energy. Therefore, in addition to the location hierarchy (Figure 2 left), the energy supply can be aggregated according to different products (Figure 2 right). In the center of Figure 2, these two hierarchies are combined into a multi-hierarchical structure, where a single element represents an instance from both hierarchies (e.g., the supply of wind energy in Hof). Child elements are additionally annotated with an index p (at the top), where elements at level i with the same index can be used to calculate the corresponding parent. For example, the element $S_{31} = R_1$ can be either calculated by aggregating elements $S_{21} = C_1$ and $S_{24} = C_2$ or by aggregating elements $S_{22} = R_1 P_1$ and $S_{23} = R_1 P_2$.*

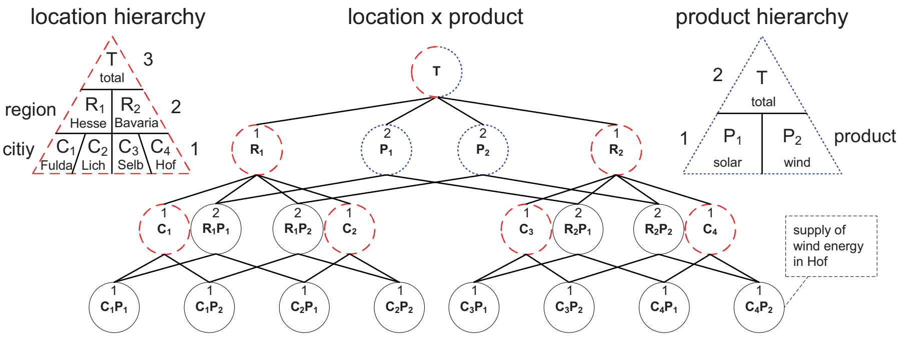


Figure 2: Multi-Hierarchical Forecasting Structure

Note that the multi-hierarchical structure is based on the instance-level of the data. In contrast to the aggregation lattice on attribute level, we include additional functional dependencies that might be given by information assurances (ORACLE) or derived from the underlying data. For example, in Figure 2 instances of the grouping (R, C, P) are not considered, as R directly depends on C and therefore (R, C, P) is equal to (C, P).

Forecasting Having this structure in mind, for each element S_{ij}, we can calculate the forecast value of the corresponding time series by three different ways:

Forecast Model: First, we can create a forecast model M_{ij} directly from the time series S_{ij}. We can then use this model to directly calculate a forecast value for element S_{ij}.

Aggregation: Second, we can create forecast models for all child elements with the same group index p at level k, where $k < i$. We then forecast using each model in the group and aggregate the forecast values to get the forecast value of the parent element. We denote this strategy as $A_{ij(kp)}$. In addition, we can aggregate recursively.

Disaggregation: Third, we can create a forecast model for one parent element p at level k and disaggregate the forecast value. The disaggregation strategy requires the calculation of a disaggregation key $D_{ij(kp)}$. A simple, but quite successful disaggregation strategy (assuming SUM as aggregation method), is an average over the fraction of the child series and the parent series: $D_{ij(kp)} = 1/n \cdot \sum_{t=1}^{n} S_{ij}(t)/S_{kp}(t)$ [GS90]. Then, the forecast values of the series element S_{ij} are the product of the disaggregation key $D_{ij(kp)}$ and the forecast value of S_{kp}.

Each forecasting strategy allows different underlying methods. For example, we could use exponential smoothing as forecast method, the summation as aggregation method and an average over the fraction of child and parent series as disaggregation method. While choosing a concrete method is independent from our approach, it might have a high impact on the resulting physical design.

2.2 Physical Design

Conceptually, the user expects a forecast model for every time series queried. However, based on the three possibilities to calculate the forecast values for a single element S_{ij}, we can have different physical designs in a multi-hierarchical system. The benefit of a physical design depends on the workload of the system. For example, a model at a higher level might support long-term forecasts as the general trend of the data is captured. We therefore consider a workload trace of forecast queries for a given period of time. A workload W consists of elements S_{ij}, their relative frequency f and the corresponding forecast horizon h: $W = \{S_{ij}, f, h\}$. An element S_{ij} might be either described by point queries or pure conjunctive queries, i.e., addressing multiple hierarchies. However, queries might also address several elements S_{ij} (e.g., disjunctive, join or group-by queries). In that case, we only store the individual elements in our workload model.

Definition 2 Configuration: *For a given workload W, a configuration C_W is a valid assignment of forecast models to individual elements. An assignment is valid if each element S_{ij} in W can be calculated by either a forecast model, aggregation or disaggregation.*

In order to make sure that we can calculate forecast values for each element S_{ij} in W, we always include the *least common parent* in our configuration, i.e., the element (might not be part of W) at the smallest level that is parent of all (other) elements in W.

Example 2 *Recall Example 1 and consider the workload W=$\{$($R_1, 1/5, 1$), ($C_1, 1/5, 1$), ($C_2, 1/5, 1$), ($C_1P_1, 1/5, 1$), ($C_1P_2, 1/5, 1$)$\}$, where the one-step ahead forecast for the supply in the region Hesse (R_1), the supply in the city Fulda (C_1), the supply in the city Lich (C_2), the supply of solar energy in Fulda (C_1P_1) and of wind energy in Fulda (C_1P_2) is requested. A first valid configuration is to create a model over the supply in the region Hesse $\{R_1\}$ and calculate all workload elements by disaggregation. A second possible configuration is $\{C_1, C_2\}$ that calculates elements C_1P_1 and C_1P_2 by disaggregation and element R_1 by aggregation. Many more possibilities exist.*

It is obvious that the number of possible configurations is exponential with the number of possible models $|C_W|$. For each element, we can decide if we create an individual forecast model and we can choose an arbitrary combination of models.

2.3 Optimization Problem

Our goal is to find the best configuration for a given workload. Each configuration can be described by two metrics, configuration error and configuration maintenance cost.

Intuitively one would think that a forecast value directly calculated from a model is always superior than disaggregation from a higher level model. However, many studies in mathematical and forecasting literature have shown that disaggregation (or aggregation) can be superior to individual time series models [DWD76, Fli99]. Therefore, the forecast error is not monotonic (related to a hierarchical system), as adding a new forecast model does not automatically imply a lower forecast error.

Definition 3 Configuration Error: *Given a configuration C_W, the error E_W is calculated by the total error over all accessed elements in W using the best strategy:*

$$E_W = \sum_{(S_{ij},f,h)\in W} f \cdot \min_{\forall k_1>i, k_2<i} \left(\mathbb{e}(M_{ij}, h), \mathbb{e}(D_{ij(k_1)}, h), \mathbb{e}(A_{ij(k_2)}, h) \right) \qquad (1)$$

where M_{ij} denotes a forecast model, $D_{ij(k_1)}$ denotes the disaggregation and $A_{ij(k_2)}$ denotes the aggregation strategy. We use the time series of length $|S_{ij}| - h$ to train the model M_{ij} and calculate the disaggregation key D_{ij}. Then, $\mathbb{e}(strategy, h)$ calculates the error for the given strategy over the h-step-ahead forecast over the remaining time series.

The forecast error is calculated using the absolute value of the symmetric mean absolute percentage error (SMAPE), which is a scale-independent accuracy measure. For both, aggregation and disaggregation, the forecast error is determined by choosing the set of child elements or the parent with the minimal forecast error. Note that we do not support direction changes, i.e., a disaggregate cannot be calculated from an aggregate as this might lead to an arbitrary result, not following the characteristics of the time series.

Maintenance of forecast models requires mainly parameter re-estimation, as maintaining the state of a model (e.g., smoothing constants) is cheap. The costs of parameter re-estimation depends on factors like frequency of re-estimation, time series length and runtime of the parameter estimation algorithm. However, as we are in an offline context, we assume a fixed estimation method and strategy leading to equal costs for each element. In addition, we can assume that the time series are about the same length as aggregate series at higher levels need all their child series to be of the same length. However, if a series has a missing value, this value might even have a meaning (e.g., zero products sold this month). Finally, we do not include the maintenance of disaggregation keys as these are, similar to the state of the model, significantly cheaper than parameter re-estimation of models. Following these considerations, we explicitly use a very simplified model to estimate the maintenance cost of a configuration by using the number of created models.

Definition 4 Maintenance Cost: *Given a configuration C_W, the maintenance cost B_W are calculated by the number of forecast models in C_W:*

$$B_W = |M_{ij} \in C_W|\,. \qquad (2)$$

Now, our optimization goal is twofold: First, we want to reduce the forecast error. Second, we want to do as less maintenance as possible.

Definition 5 Optimization Goal: *Our optimization goal is to find the configuration C_W, which is minimal according to the configuration error E_W and maintenance costs B_W:*

$$\min_{C_W} \left(\alpha \cdot \frac{E_W}{E_T} + (1-\alpha) \cdot \frac{B_W - 1}{B_{max} - 1} \right) \textit{ with } B_{max} > 1 \textit{ and } \alpha \in [0,1], \qquad (3)$$

where B_{max} as well as E_T are used as normalization constants. B_{max} is the maximum number of models in C_W and E_T is the forecast error for the configuration, where only a model for the common parent is used. For normalization purposes as well, we subtract 1 *in the second part of the equation.*

With parameter α we can weight the importance of both dimensions. If we set $\alpha = 0.5$, we give equal weight to maintenance cost and forecast error. If we set $\alpha = 1$ we try to find the best configuration according to the forecast error, without regarding maintenance. This generalizes the problem of finding the best hierarchical structure in forecasting literature. As the forecast error is not monotonic with respect to the number of models, the minimum forecast error can result for any configuration. On the other hand, with $\alpha = 0$, as we do not consider disaggregation costs, we get the configuration where only one model is used for the common parent and all other elements use the disaggregation strategy.

3 Offline Design Approach

In order to solve the optimization problem of Definition 5, our general approach works as follows:

1. Create a forecast model for each element in the workload and the common parent.
2. Enumerate all valid configurations according to Definition 2.
3. Evaluate each configuration regarding the configuration error (Definition 3) and maintenance cost (Definition 4).
4. Choose a configuration according to Definition 5 and drop all models, which are not part of the solution.

However, this naive algorithm has unacceptable runtime, because we first need to create a forecast model for every element considered in the workload. This is necessary to calculate the model, aggregation and disaggregation benefit for each element. However, model creation is expensive as parameters need to be estimated. Second, it is infeasible to enumerate all possibilities as the number of possible configurations is exponential with the number of elements (on instance-level). Therefore, we present an approach to reduce the number of configurations enumerated (Section 3.1). Then, we discuss some heuristics to reduce the number of forecast models considered (Section 3.2).

3.1 Greedy Enumeration

As explained before, maintenance cost increase monotonically with each additional forecast model while the behavior of the forecast error is unknown. A new model might lead to an improvement but the error could also stay similar or get even worse. Thus, in the

Algorithm 1 Greedy Enumeration

```
Require: $elementsConsidered$
 1: $bestConf \leftarrow$ concat(1,repeat(0, $numElements - 1$))
 2: $bestEval \leftarrow \alpha$
 3: repeat
 4:    $stop \leftarrow$ true
 5:    $currentConf \leftarrow bestConf$
 6:    for $i$ in $elementsConsidered$ do
 7:       $currentConf[i] \leftarrow 1$
 8:       if ($currentEval =$ evaluateConfiguration($currentConf$)) $< bestEval$ then
 9:          $bestConf \leftarrow currentConf$
10:          $bestEval \leftarrow currentEval$
11:          $newModel \leftarrow i$
12:          $stop \leftarrow$ false
13:       $currentConf[i] \leftarrow 0$
14:    $elementsConsidered \leftarrow$ remove($elementsConsidered, newModel$)
15: until $stop =$ true
```

second case, we would never use this model as only maintenance increases. Therefore, we propose a greedy enumeration approach, where we start with the configuration where only a model for the top element is used. Thus, this is the valid configuration with minimal maintenance cost. We then try to add additional models step-by-step, using most promising models first. As each model is described with equal maintenance costs, we always add the model next, which results in the lowest overall configuration error.

Algorithm 1 outlines our greedy enumeration approach. First, our algorithm requires the elements, which actually can qualify for a model. This could contain all elements in the multi-hierarchical structure. However, the workload will reduce these elements significantly to only the queried elements. In addition, the heuristics introduced in the next section will reduce the number of models considered further. To represent a configuration, we use a vector where 1 stands for forecast model and 0 for no model. The entries are the consecutive elements in the multi-hierarchy starting at the top and going down from left to right. Therefore, in line 1 we create the configuration, where only a model is created for the top element. According to Definition 5, the evaluation for our start configuration is always α, which we use as our current best evaluation (line 2). Then, for each element in our list of considered elements, we create a configuration where a model is used for this element (line 7). We evaluate this configuration with Definition 5 and if it is better than the current one, we store it (line 8-10). Then, we reset the current configuration (line 13). Therefore, in each step we iterate over all elements and output a new configuration with one additional element. In the end, we remove the new element from the list of considered elements (line 14). We stop when we find no better configuration than the current one. Note that we need to check the benefit of each single element in each iteration as a new model might have an impact on every other element in the workload.

Example 3 *To illustrate the greedy approach, an example is shown in Figure 3. This example is created for the workload described in Example 2 with* $\alpha = 0.5$*. A gray box shows*

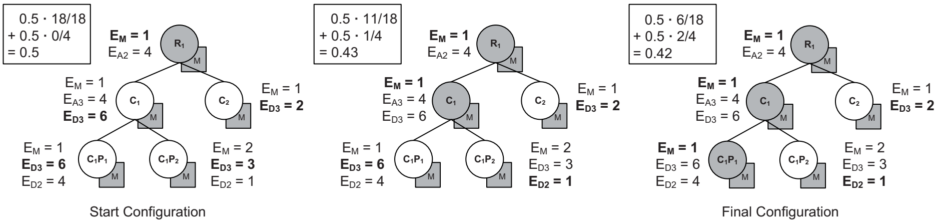
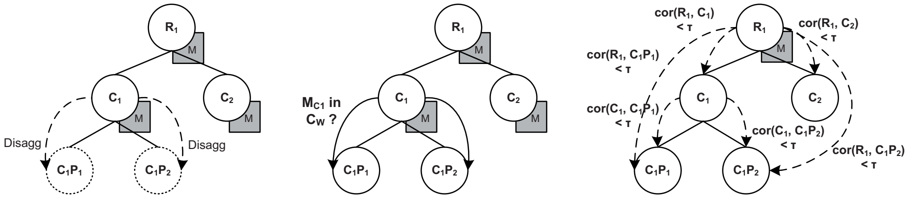


Figure 3: Greedy Enumeration

that an evaluation model was created for the corresponding element, while a gray colored node implies that the model is actually used and maintained in the current best configuration. Each node is annotated with the errors when using a model E_M*, aggregation* E_{Ak} *and disaggregation* E_{Dk}*, where* k *is level from which the forecast values are aggregated or disaggregated. In the left part of Figure 3, the start configuration is shown in which only a model for the top element is used. As explained before, the start evaluation equals always* α*, which is shown in the box in the upper left corner (Definition 5). Note that we use the total error to describe the configuration error as all elements in the workload have the same frequency. Now, our greedy approach sequentially checks the benefit of a model for each element. Elements* C_1 *and* C_1P_1 *have the highest disaggregation error of* 6.0 *and would both profit from a model equally as the model error is only* 1.0*. However, element* C_1P_2 *would also profit from a model at element* C_1 *as the disaggregation error using level two is lower. Therefore, we would decide to use a model for element* C_1 *(Figure 3 center). The new evaluation drops down to* 0.43*. Second, the greedy approach would decide to use a model for the element* C_1P_1 *as we get the best improvement regarding the forecast error and the evaluation drops to* 0.42*. Although, the element* C_2 *would also lead to a decrease of the error, the algorithm is finished now. The improved error does not compensate the increased maintenance costs. The evaluation would be* 0.51*.*

For ease of illustration, we used a single hierarchy in this example. However, the only difference in a multi-hierarchy is that a child node might benefit from the disaggregation from several parents and a parent from the aggregation of several sets of child elements.

This approach might result in local sub-optima for two reasons. The first reason is that models are never removed, although they might not be necessary anymore as the element can be calculated by aggregation. Therefore we additionally analyze aggregation benefits. Thus, whenever we evaluate a configuration, we also check if we can remove models, which can now be calculated by aggregation. If so, we also evaluate the configuration with the removed aggregate model. Second, we might miss optimal configurations because we only consider single models with each step. However, a whole group might improve the result as elements at higher levels can be calculated by aggregation. Therefore, we include a second optimization, where we analyze groups in each step as well. In our case, a group is a complete set p of elements at one level, which contribute to the same aggregate one level above. Different strategies to build groups are possible. However, this can have the drawback that groups are added too early in the process resulting in a different local sub-optima. In addition, we still end in local sub-optima if adding more than one model (but



(a) Decomposition (b) Recursive (c) Correlation

Figure 4: Model Creation Heuristics

not a whole group) would allow the removal of an aggregation model and lead to a better overall forecast error. However, in most cases, aggregation benefits are less important as they require many child models to be build.

The complexity of the greedy enumeration approach is $O(n + n^2)$ in the worst case. First, we create n forecast models. Then, we evaluate for each element the benefit of a model and add the element with the most benefit. We stop when we can not find a beneficial model anymore resulting in a worst case evaluation of $\sum_{i=1}^{n} i \approx n^2$ configurations. In the best case, we stop after one run as we did not find a better configuration, so we result in a linear behavior. Note that the creation of a forecast model might be much more expensive than the evaluation of one configuration.

3.2 Heuristics

In this section, we present heuristics to reduce the number of forecast models considered. Each heuristic only creates a subset of possible models. If there is no model created for an element, our greedy enumeration approach does never consider a configuration where a model for this element is used (Algorithm 1, line 6). All heuristics can be used by themselves in addition to the greedy enumeration approach, but an arbitrary combination of these heuristics is also possible.

Decomposition The decomposition approach assumes that if we find a good strategy for each single hierarchy, the combination of hierarchies, i.e., conjunctive queries, will also result in a low error. Therefore, only forecast models are considered that address elements from single hierarchies. For all combinations of hierarchies we always use the disaggregation strategy (Figure 4(a)). This heuristic reduces the number of considered elements to the sum over the number of elements in each single hierarchy.

Recursive The recursive approach has the underlying assumption that if disaggregation is best for one level, it is also best for all underlying child elements. Initially, we only create forecast models for the top element and the level underneath. Every time we decide to use a forecast model, we create all models one level below (Figure 4(b)). In the best case, this approach only creates models for the top two levels of the hierarchy. However, in the worst case, all models are created. If we combine the recursive heuristic with the decomposition heuristic, we might even get a higher reduction of forecast models created, as each child node is only reachable through exactly one parent node.

Time Series Characteristics This heuristic analyzes the characteristics and the relation of the time series to filter out forecast models to be created and considered. For this, we use two different characteristics, correlation and disaggregation error. With both approaches, forecast models are only created if a certain characteristic is fulfilled.

Correlation The core observation is that high correlated time series follow the same pattern and thus could be calculated by the same model. Therefore, we calculate the correlation between parent-child relations. For this, we also consider parent-child relations over more than one level (Figure 4(c)). Then, we only create models for child series if the correlation is below a threshold τ. The threshold τ defines the aggressiveness of this approach. A low τ might filter out many models, but might also miss good configurations. In contrast, a high τ is safer, but might only filter out a few models.

Disaggregation Error As we always create the top model, we can analyze the disaggregation error of each element in the workload. This approach only creates a model for an element if the disaggregation error is above ω. To calculate ω, we use the median of the disaggregation error $\texttt{median}(\texttt{e}(D_{ij(11)}, h))$ over the elements in the workload and the weight α of the configuration error. Then, ω is calculated by $\texttt{median}(\texttt{e}(D_{ij(11)}, h))/(\alpha + 0.5)$. Therefore, if we give equal weight to the configuration error and maintenance cost ($\alpha = 0.5$), we create all models higher than the medium disaggregation error. This works as we assume equal maintenance cost in this paper. If we give low weight to the configuration error, we create less models as ω increases and vice versa. Therefore, we adjust the number of created models according to the weight of the configuration error as a lower weight would result in a configuration with less models anyway.

4 Experimental Evaluation

We conducted an experimental study in order to evaluate (1) the performance of our approach on three data sets from different domains with respect to traditional approaches from forecasting literature, (2) the performance and scalability of the proposed heuristics and (3) the adaptability to different user requirements.

4.1 Experimental Setting

To implement the described offline design tuning approach we used the statistical computing software environment R. It provides efficient build-in forecast methods and parameter estimation approaches, which we used to build the individual forecast models. All experiments were executed on an IBM Blade (Suse Linux, 64bit) with two processors (each a Dual Core Intel Xeon at 2.80 GHz) and 4 GB RAM.

In order to show the general applicability of our offline design approach, we use the fol-

	electricity	tourism	energy
#elements level 1	1,568	32	86
#elements level 2	273	12	1
#elements level 3	14	1	-
#elements level 4	1	-	-
series length	28	25	5,808

Table 1: Sizes of Data Sets

lowing three data sets:

- *Electricity: Worldwide Electricity Generation* The first data set was obtained from the US Energy Information Administration and is public available at [US110]. This data set includes metered world-wide electricity generation. It consists of two hierarchies. The first hierarchy contains regional information from world-wide over continental to individual countries. In the second hierarchy different categories of electricity sources such as renewable or nuclear are distinguished. After merging both hierarchies, we result in multi-dimensional hierarchy with four levels, similar to Figure 2. The data is available from 1980 until 2008 in an annual resolution.
- *Tourism: Australian domestic tourism* The second data set consists of quarterly observations on the number of visitor nights for the Australian domestic tourism, which is an indicator of tourism activity. The sample begins with the first quarter of 2004 and ends with the first quarter of 2010. The series are obtained from the National Visitor Survey, which is managed by Tourism Research Australia [TRA10]. The data consists of two hierarchies, purpose of visit and state, resulting in three levels of the final multi-hierarchical structure.
- *Energy: EnBW MEREGIO Energy Demand* The third data set was provided by a partner from the MIRACLE project [BBD$^+$10] and was obtained during the MEREGIO project [MER10]. This data set contains energy demand from 86 customers ranging from November 1st 2009 to June 30th 2010 in a 1 h resolution. It therefore consists of a single hierarchy with two levels, i.e., level 1 contains the individual customer demand while level 2 contains the total aggregate over all customers.

Table 1 shows the sizes of the different levels and time series lengths for each data set.

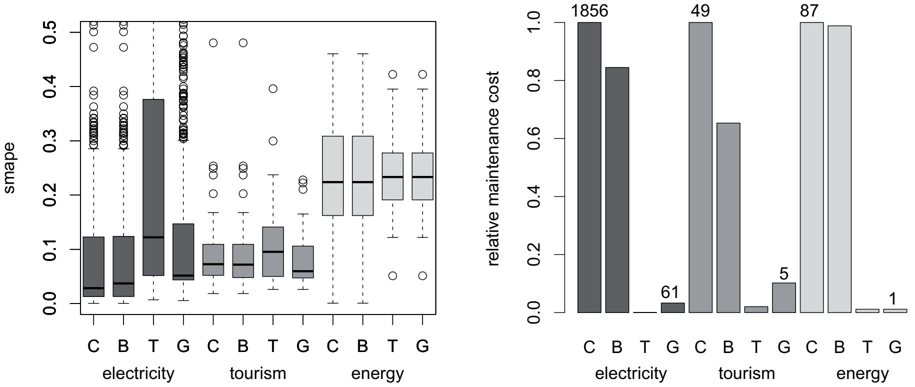
For all three data sets we fixed the forecast, aggregation and disaggregation method. As forecast method we use triple exponential smoothing. The class of exponential smoothing methods is widely used in hierarchical forecasting and has proven to be very robust and applicable in an automated fashion to a large set of time series [Cha00]. If we would use a more complex forecast method, we would save even more execution time with our approach as parameter estimation gets more expensive. For the aggregation method, we use summation. Gross and Sohl analyzed 21 disaggregation methods [GS90] and concluded that a simple average of the elements' proportion of the parent element over the entire historical period worked best compared to other, partly more complex, methods. Therefore, we will use this as disaggregation strategy.

In addition, we fix the workload. We assume that each element in the hierarchy is queried once requesting a one-step ahead forecast, so $W = \{(S_{ij}, 1/|S|, 1)\}$ for *i=1 ... #levels*, *j=1 ... #elements of level i* and $|S|$ is the total number of elements in the whole multi-hierarchical structure. Therefore, we analyze the worst case, where every element is considered equally. A different workload would only reduce the search space.

4.2 Performance Comparison

In the following, we compare our greedy approach with the three traditional approaches ”bottom-up”, ”top-down” [Fli01] and ”complete”. The bottom-up approach creates forecast models for all time series at level one and calculates all other forecasts by aggregation of elementary forecasts. The top-down approach creates only one forecast model for the top element and calculates all other forecasts by disaggregation of the top forecast. Finally, the complete approach creates forecast models for all elements in the hierarchy and calculates all forecasts using directly the model for the concerning element. In this experiment, we use 80% of the data sets to learn the models for all approaches and to learn the configuration for our greedy approach. We set $\alpha = 0.3$, so we give more weight to maintenance time. Then, we use the remaining 20% to produce one-step ahead forecasts, where we reestimate the model parameters after each forecast. Note that we also could use a more sophisticated model maintenance method, where we trigger parameter reestimation time- or threshold-based. However, this is not the scope of this paper and the effect would be similar for all approaches. Figure 5 summarizes the results for all three data sets and approaches. In Figure 5(a), the average forecast error (using the accuracy measure SMAPE) of each element over the evaluation period is illustrated. Here, the maximum forecast error equals 1. In Figure 5(b), the relative maintenance cost compared to the complete approach as well as the total number of used models is shown.

Our goal is to create as less models as possible while reaching a low forecast error. If we take a rough view at the results, we see that our greedy algorithm produces the best result considering both dimensions. Let us analyze each data set a bit more closely. For the electricity consumption, the complete (C) and bottom-up (B) approach have a much better forecast error than the top-down (T) approach. Therefore, our greedy approach (G) creates 61 forecast models reducing the forecast error a lot but adding only little additional maintenance time. In order to reduce the error even more and beat the complete approach, we need to choose a higher value of α. We will analyze the effect of α more closely in Subsection 4.4. For the tourism activity, the forecast error of the complete and bottom-up approach is slightly better than the top-down approach. However, the forecast error can actually be reduced a lot, if a few forecast models are created at level two and the forecasts at level one are created by disaggregation from level two. Therefore, our greedy approach decides to create some forecast models at level two, reducing the forecast error even more than the complete approach but resulting in much less number of models to maintain. For the energy demand, the top-down approach is similar to the complete and bottom-approach. Therefore, our greedy approach decides to use the top-down approach, creating no additional forecast models where we save a lot of maintenance cost compared



(a) Distribution of Forecast Error

(b) Maintenance Cost

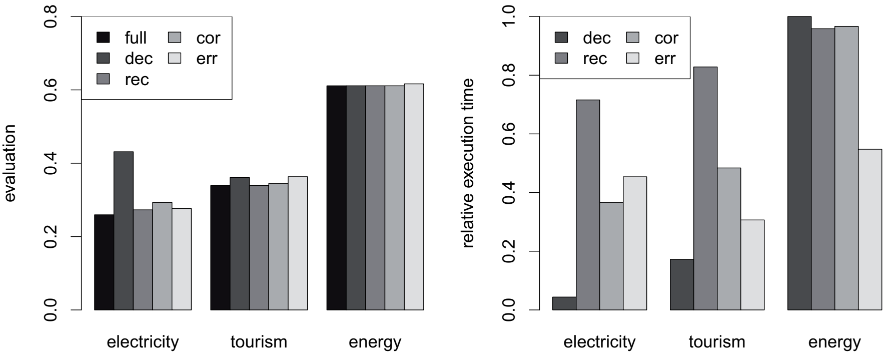
Figure 5: Comparison of Different Approaches

to the complete or bottom-up approach (one instead of 87 models).

To summarize, on the one hand our greedy approach can find a better configuration than the traditional approaches, even with a low value of α. On the other hand, maintenance time is reduced by using only necessary models. The speed up we can achieve strongly depends on the used hierarchy, i.e., the number of elements, and on the relationship of the time series, i.e., weather bottom-up or top-down is superior in general. In terms of total execution time, the benefit strongly depends on the used forecast method and the time series length. For example, for the energy data set, we save about half a minute in each maintenance step if we use triple exponential smoothing. The total maintenance time is 3.4 h for the complete approach and 2.3 minutes for the greedy approach. If we use an AR(12) model, which is an instance of the widely used class of ARIMA models, we save 26 minutes in each *single* maintenance step. As we are in an offline context, we did not explicitly measure the speed up of forecast queries. However, a low number of forecast models implies lower maintenance cost and thus, a lower system load or lower query processing times if deferred maintenance is used.

4.3 Comparison of Different Heuristics

In a second experiment, we take a closer look at the different heuristics introduced in Subsection 3.2. For each heuristic, we might get a slightly worse configuration compared to the full greedy approach but we might save execution time. Figure 6(a) shows the *evaluation* (= value of objective function according to Definition 5) of all heuristics compared to the evaluation of the greedy approach. As all data sets exhibit totally different execution times, Figure 6(b) shows only the relative execution time decrease to the full greedy approach. In this experiment, we set $\alpha = 0.75$ so that the impact of the heuristics are higher



(a) Evaluation

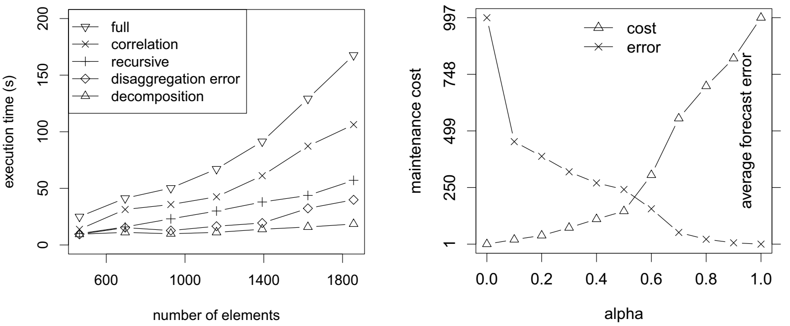
(b) Relative Execution Time

Figure 6: Comparison of Different Heuristics

as more forecast models are created.

The energy data set contains only a single hierarchy. Therefore, the decomposition heuristic (dec) leads to the same accuracy and execution time as the full greedy approach (full). For the other two data sets, it shows the lowest execution time but also increases the evaluation most for electricity and tourism data set. Therefore, it should only by used if very little time is available. The recursive heuristics (rec) increases the evaluation only slightly for the electricity data set but not at all for the other two data sets. However, it shows the smallest improvement in terms of execution time. Therefore, it can be safely used to save some of the execution time of the offline design approach. For the correlation heuristic (cor), we set the correlation threshold to 0.75. This heuristic takes the third rank in terms of execution time and only increases the evaluation slightly for the electricity and tourism data set. As a result, it should be preferred to the recursive heuristic in order to save more execution time. The disaggregation error (err) heuristics shows the second best execution time and only increases the evaluation slightly for all three data sets. To summarize, besides the decomposition heuristic, all heuristics only increase the evaluation slightly compared to the full greedy approach. Most execution time can be saved when the heuristics are used, which take time series characteristics into account.

In order to examine the scalability of our greedy approach and the different heuristics, we vary the size of the electricity data set. For this, we increase the number of electricity sources from one to seven and combine the resulting electricity source hierarchy with the region hierarchy. If we use only one electricity source, we result in a single hierarchical structure, which consists of 464 elements. With each additional electricity source, we add 232 elements. The experimental results are displayed in Figure 7(a). The full greedy approach (using no heuristic) has the longest runtime and shows a super linear behavior with increasing number of elements. In general, all heuristics also increase the number of considered models with increasing data size resulting in a super linear behavior as well. However, the concrete outcome strongly depends on the data itself, e.g., how many time



(a) Scalability (b) Adaptability

Figure 7: Scalability and Adaptability to User Requirements

series fall below the correlation threshold.

4.4 Adaptability to User Requirements

In this section, we take a closer look at the outcome of our greedy approach while varying the parameter α. Recall that an $\alpha = 0$ results in the top-down approach as this is the configuration with the minimal maintenance cost. In contrast, setting $\alpha = 1$ leads to the configuration with the best overall forecast error regardless of the maintenance cost. In this experiment, we execute our greedy approach using the whole electricity data set while increasing α from 0 to 1. Figure 7(b) presents the average forecast error and the maintenance cost of the final configuration. In the beginning, with only little additional maintenance cost we get a high improvement of the average forecast error. The reason is that our greedy approach always adds those models first that lead to the highest improvement. As α gets higher the number of additional models increases but the error improvement decreases. To conclude, with a small value of α we get the best improvement of the forecast error with low additional maintenance cost. A very high α leads to the best overall forecast error, however it might not be worse the maintenance cost. Nevertheless, for the energy data set, a value of $\alpha = 1$ leads to the creation of about 57% of the models where the average forecast error beats the bottom-up and complete approach.

In addition, we examined different kind of workloads. For this, we varied the number of distinct elements addressed by the workload, the forecast horizon and the frequency of forecast queries. The higher the number of distinct elements, the larger the search space. Therefore, with a higher number of distinct elements, we tend to a higher number of models but also to a higher accuracy as we can exploit more disaggregation and aggregation possibilities. An increase of the forecast horizon of the queries leads to a higher average error and decreasing maintenance costs for all data sets. Longer forecast horizons are harder to predict at single time series level. Due to more robustness, the greedy algorithm favors models at higher aggregation levels leading to lower maintenance costs. Last, we varied the distribution of query frequencies by increasing the parameter z of a zipf distribution. A high parameter implies a high frequency of only a few elements in the workload (high skew) while a low parameter results in equal distribution of the frequencies (low skew). With a high skew, we strongly prefer a few workload elements leading to the creation of models favoring only those. However, the overall forecast error stays roughly constant as the error is weighted with the workload frequencies (Definition 3).

5 Related Work

Related work can be found in three main areas: (1) existing approaches to integrate forecasting in database management systems, (2) hierarchical forecasting studies in forecasting and economic literature and (3) materialized view selection.

Forecasting in DBMS Forecasting has already been successfully integrated into DBMS. For example, within the Fa system [DB07] an incremental approach is proposed to build models for a multidimensional time-series in which more attributes are added to the model in successive iterations. Furthermore, the skip-list approach for efficient forecast query processing [GZ08] proposes an I/O-conscious skip list data structure for very large time series in order to enable the determination of a suitable history length for model building. However, all approaches investigate how to efficiently find the best forecast model for one specific forecast query using database techniques. We consider the problem of efficient forecast query processing from a different point of view by addressing the interaction of queries, which allows the reduction of the forecast error and maintenance cost by reusing models. Agarwal et al. address the problem of forecasting high-dimensional data over trillions of attribute combinations [ACL$^+$10]. They propose to store and forecast only a sub-set of attribute combinations and compute other combinations from those using high-dimensional attribute correlation models. However, they select the sub-set manually for historical importance and seasonality, while we propose an automatic approach to determine the optimal set of models to store. Then, their correlation models can be used as disaggregation method in our approach.

Hierarchical Forecasting In contrast to our work, hierarchical forecasting considers only a single hierarchy. In this context, the majority of the literature has focused on comparing the performance of bottom-up forecasting (individual item forecasts are made directly and aggregated) and top-down forecasting (forecasts are made at aggregate level and then allocated to individual items). Some favor the bottom-up approach [ZT00, DWD76], other the top-down approach [Fli99, NMB94] and some found no method to be superior for their specific data set [FM92]. In addition, influencing factors of the superiority of one approach over the other were investigated, e.g., quality of forecast method, correlation between variables and forecast errors [Bar80]. In a recent work, bottom-up versus top-down was investigated when the subaggregate series follows a first-order univariate moving average MA(1) process [WVP09]. They found no significant difference in the accuracy of the two strategies when the correlation between the subaggregate series is small or moderate. However, all research empirically analyzes one specific data set and conclude for one of the two methods. In contrast, we propose an approach that quantifies different solutions and also allows for mixed solutions. In addition, existing approaches focus on accuracy only, while we take maintenance cost into account as well. This allows for faster query processing and a lower system load as maintenance time of models is reduced.

Materialized View Selection There is a high amount of work in the area of materialized view selection [ACN00, BPT97, SDN98]. For example, in [BPT97] two techniques are proposed, which reduce the number of views to consider for materialization (based on query benefit, dependent views and the size of the materialized view). The general problem is the same – for a given workload find the optimal set of materialized views with minimal query and maintenance cost. However, the model selection problem strongly differs from the materialized view selection problem, as a second dimension – the forecast accuracy – is introduced. In addition, different derivation schemes are used in model selection, i.e., disaggregation. Therefore, we cannot apply existing techniques directly.

6 Conclusion and Future Work

In this paper, we introduced the problem of physical design of time series forecast models in a multi-hierarchical data-warehouse scenario. We defined the two-dimensional optimization problem of minimizing the forecast accuracy and maintenance cost. On the one hand, we generalized the problem of finding the best hierarchical structure addressed in forecasting literature. On the other hand, we additionally include maintenance cost addressing evolving time series. Our solution consists of a greedy enumeration approach and different heuristics, which might reduce the time consumption of the offline design algorithm. In our experimental evaluation, we used three data sets to show that we can find the configuration, which reaches a high accuracy while using as less models as possible.

This paper is part of an on-going research about physical design of forecast models inside a database. We started to analyze the physical design from an offline point of view. However, an important question is how this structure should be maintained when time series characteristics change, i.e., online physical design. In detail, we need to consider three different types of maintenance. First, maintenance of the model state and disaggregation keys, which is cheap and can be done incrementally every time a new tuple arrives. Second, maintenance of model parameters, which is expensive and therefore could be trigged either time or threshold based. Third, maintenance of the model pool itself. For the last case, we need to monitor the real workload, maintenance cost and accuracy of different forecast models. Depending on these statistics, we might decide to drop a model, which has not been worthwhile or to create a new model to improve forecast accuracy.

Acknowledgment

The work presented in this paper was partially funded by the EU within the MIRACLE project under the grant agreement number 248195.

References

[ACL$^+$10] Deepak Agarwal, Datong Chen, Long-ji Lin, Jayavel Shanmugasundaram, and Erik Vee. Forecasting high-dimensional data. In *SIGMOD Conference*, 2010.

[ACN00] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated Selection of Materialized Views and Indexes in SQL Databases. In *VLDB*, 2000.

[Bar80] Amir Barnea. An analysis of the usefulness of disaggregated accounting data for forecasts of corporate performance. *Decision Sciences*, 11:17–26, 1980.

[BBD$^+$10] H. Berthold, M. Böhm, L. Dannecker, F.-J. Rumph, T. B. Pedersen, C. Nychtis, H. Frey, Z. Marinsek, B. Filipic, and S. Tselepis. Exploiting renewables by request-based balancing of energy demand and supply. In *IAEE*, 2010.

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized Views Selection in a Multidimensional Database. In *VLDB*, 1997.

[Cha00] Chris Chatfield. *Time-Series Forecasting*. Chapman & Hall, 2000.

[DB07] Songyun Duan and Shivanath Babu. Processing Forecasting Queries. In *VLDB*, 2007.

[DWD76] D.M. Dunn, W.H. Williams, and T.L. DeChaine. Aggregate Versus Subaggregate Models in Local Area Forecasting. *Journal of the American Statistical Association*, 71:68–71, 1976.

[Fli99] Gene Fliedner. An investigation of aggregate variable time series forecast strategies with specific subaggregate time series statistical correlation. *Computers & Operations Research*, 26:1133–1149, 1999.

[Fli01] Gene Fliedner. Hierarichal forecasting issues and use guidelines. *Industrial Management & Data Systems*, 101:5–12, 2001.

[FM92] Eugene B. Fliedner and Vincent A. Mabert. Constrained Forecasting: Some Implementation Guidelines. *Decision Sciences*, 23:1143–1161, 1992.

[FRBL10] Ulrike Fischer, Frank Rosenthal, Matthias Boehm, and Wolfgang Lehner. Indexing Forecast Models for Matching and Maintenance. In *IDEAS*, 2010.

[GS90] Charles W. Gross and Jeffrey E. Sohl. Disaggregation methods to expedite product line forecasting. *Journal of Forecasting*, 9:233–254, 1990.

[GZ08] Tingjian Ge and Stan Zdonik. A skip-list approach for efficiently processing forecasting queries. *VLDB*, 2008.

[KR02] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit*. Wiley, 2002.

[MER10] *The MeRegio Project*, 2010. `http://www.meregio.de/en/`.

[NMB94] S.L. Narasimhan, D.W. McLeavey, and P. Billington. *Production Planning and Inventory Control*. Allyn & Bacon, 2 edition, 1994.

[Ora10] Oracle. Oracle OLAP DML Reference: FORECAST - DML Statement, 2010.

[Pre10] PredictTimeSeries – Microsoft SQL Server 2008 Books Online. `http://msdn.microsoft.com/en-us/library/ms132167.aspx`, 2010.

[SDN98] Amit Shukla, Prasad Deshpande, and Jeffrey F. Naughton. Materialized View Selection for Multidimensional Datasets. In *VLDB*, 1998.

[TRA10] *Tourism Research Australia - National Visitor Survey*, 2010. `http://www.ret.gov.au/tourism/tra/domestic/national/Pages/default.aspx`.

[US110] *US EIA - International Energy Statistics*, 2010. `http://tonto.eia.doe.gov/cfapps/ipdbproject/IEDIndex3.cfm?tid=2&pid=2&aid=2`.

[WVP09] Handik Widiarta, S. Viswanathan, and Rajesh Piplani. Forecasting aggregate demand: An analytical evaluation of top-down versus bottom-up forecasting in a production planning framework. *International Journal of Production Economics*, 118:87–94, 2009.

[ZT00] Arnold Zellner and Justin Tobias. A Note on Aggregation, Disaggregation and Forecasting Performance. *Journal of Forecasting*, 19:457–469, 2000.