A segmentation-based algorithm for large-scale partially ordered monotonic regression

O. Sysoev\textsuperscript{a,}\textsuperscript{*}, O. Burdakov\textsuperscript{b}, A. Grimvall\textsuperscript{a}

\textsuperscript{a} Department of Computer and Information Sciences, Linköping University, SE-58183, Linköping, Sweden
\textsuperscript{b} Department of Mathematics, Linköping University, SE-58183, Linköping, Sweden

\textbf{A R T I C L E  I N F O}

Article history:
Received 15 February 2010
Received in revised form 7 March 2011
Accepted 7 March 2011
Available online 13 March 2011

Keywords:
Quadratic programming
Large-scale optimization
Least distance problem
Monotonic regression
Partially ordered data set
Pool-adjacent-violators algorithm

\textbf{A B S T R A C T}

Monotonic regression (MR) is an efficient tool for estimating functions that are monotonic with respect to input variables. A fast and highly accurate approximate algorithm called the GPAV was recently developed for efficient solving large-scale multivariate MR problems. When such problems are too large, the GPAV becomes too demanding in terms of computational time and memory. An approach, that extend the application area of the GPAV to encompass much larger MR problems, is presented. It is based on segmentation of a large-scale MR problem into a set of moderate-scale MR problems, each solved by the GPAV. The major contribution is the development of a computationally efficient strategy that produces a monotonic response using the local solutions. A theoretically motivated trend-following technique is introduced to ensure higher accuracy of the solution. The presented results of extensive simulations on very large data sets demonstrate the high efficiency of the new algorithm.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

The monotonic regression (MR) problem often originates from applications in which observations are composed of a vector of \( p \) explanatory variables \( x \in \mathbb{R}^p \) and a response variable \( y \in \mathbb{R} \). The values of \( x \) and \( y \) for each observation \( i = 1, \ldots, n \) are denoted as \( X_i \) and \( Y_i \), respectively, and they are collected in the input data set \( D = \{(X_i, Y_i), \ i = 1, \ldots, n\} \), which is frequently used as an input to applied MR problems.

It is assumed that the unknown true response function \( f(x) \) is monotonic, i.e., increasing with respect to some variables and decreasing with respect to others. For simplicity, we assume that \( f \) is isotonic, i.e., that it increases with respect to each component of \( x = (x_1, \ldots, x_p) \). This means that

\[ f(x') \leq f(x''), \quad \forall x', x'' \in \mathbb{R}^p \text{ such that } x' \prec x'', \]

where \( x' \prec x'' \) means that \( x'_i \leq x''_i \) for each \( i = 1, \ldots, p \). In real-life applications, the data sets are seldom monotonic, and hence \( X_i \prec X_j \) does not necessarily imply \( Y_i \leq Y_j \). This is due to some observation errors, \( y = f(x) + \varepsilon \).

Here, we focus on the case of \( p > 1 \), for which the set of observed explanatory variables \( X_1, \ldots, X_p \) is usually partially ordered. This means that some pairs \((X_i, X_j)\) may not be comparable, i.e., neither \( X_i \prec X_j \) nor \( X_j \prec X_i \) holds. The original function \( f(x) \) is not available and cannot be restored from the data set \( D \), and it is even impossible to restore the function values \( f(X_i) \) for \( i = 1, \ldots, n \). Nevertheless, by using the knowledge of monotonicity and data set \( D \), these function values can be approximated by solving the corresponding MR problem whose solution \( f^* \in \mathbb{R}^n \) is an approximation of the vector...
(f(X_1), \ldots, f(X_n)). Moreover, the components of f^* are consistent with the monotonicity and are as close as possible to the observed response values Y_i. In the weighted L_2-norm, the MR problem is formulated as

\begin{align}
\min & \sum_{i=1}^{n} w_i(f_i - Y_i)^2
\quad \text{s.t.} \quad f_i \leq f_j \quad \text{iff} \ X_i \prec X_j, \ i, j = 1, \ldots, n,
\end{align}

where w_k > 0, k = 1, \ldots, n are given weights.

The MR problem has numerous applications in operations research, statistics, biology, signal processing, and other areas; see Barlow et al. (1972), Robertson et al. (1988) and Oh and Dong (2011). The most challenging of the practical problems in this context are characterized by a large n value. Various techniques for solving problem (1) have been developed in recent decades. The pioneering work of Ayer et al. (1955) led to introduction of the Pool-Adjacent-Violators (PAV) algorithm. This algorithm is used to solve a special case of the MR problem, typically for p = 1, where the observations are completely (linearly) ordered, i.e., either X_i < X_j or X_j < X_i holds for each pair of observations (X_i, X_j). A profound framework for MR theory was developed by Barlow et al. (1972) and Robertson et al. (1988). At present, the most widely known exact algorithms for solving the general MR problems with a partially ordered input data set D include the following: the minimum lower set algorithm of Brunk (1955), the min–max algorithm described by Lee (1983), the network-based algorithm of Maxwell and Muchstadt (1985), and the IBC algorithm introduced by Block et al. (1994). Unfortunately, these exact algorithms can only solve problems that contain a relatively small number of observations, and thus they cannot provide satisfactory results within a reasonable amount of time when addressing medium- or large-scale problems.

In our recent papers (Burdakov et al., 2006a,b), we presented a generalized PAV algorithm called the GPAV, with which high-accuracy solutions of large-scale multivariate MR problems can be obtained for a partially ordered D. Here, we present a segmentation-based algorithm (designated SB), which extends the area of its applications to much larger MR problems, that are too demanding for the original GPAV and all other well-known MR algorithms in terms of computational time and memory. The SB splits a large-scale MR problem into a number of medium-size problems and solves them using the GPAV. The fitted values are monotonic only locally within each segment, but the monotonicity may be violated on the boundary of the neighbor segments. The SB offers a special, computationally efficient strategy that produces an overall monotonic fit on the basis of the local solutions.

The remainder of this paper is organized as follows. In Section 2, we introduce and discuss an alternative formulation of problem (1). The SB algorithm is presented in Section 3, and in Section 4 we consider a trend-following order aimed at making this algorithm more efficient. In Section 5, we justify the correctness of the SB and study the theoretical properties of this algorithm. In Section 6, the worst-case complexity of the SB is estimated. In Section 7, we analyze the computational performance of the SB and compare it with performance of the GPAV. Section 8 contains conclusions and comments.

2. Alternative formulation and the GPAV algorithm

The problem formulation (1) is natural for most practical applications. In computational science, as well as in our previous papers (Burdakov et al., 2006a,b), an alternative formulation is used in which a graph G = G(N, E) with a set of nodes N and a set of edges E is supposed to be given. Each node i ∈ N = {1, \ldots, n} is associated with the observation i, and each edge (i, j) ∈ E is associated with the relation X_i < X_j. The graph is obviously acyclic.

Here, we use the following definitions and notations. A node i ∈ N is a predecessor of node j ∈ N if there is a directed path in the graph from i to j. A block is a connected subset B ⊂ N such that if there is a directed path between two nodes in B, then all the nodes in the path belong to B. The MR problem admits the following graph formulation:

\begin{align}
\min & \sum_{i \in N} w_i(f_i - Y_i)^2
\quad \text{s.t.} \quad f_i \leq f_j \quad \text{for all} \ (i, j) \in E,
\end{align}

or, equivalently,

\begin{align}
\min & \sum_{i \in N} w_i(f_i - Y_i)^2
\quad \text{s.t.} \quad f_i \leq f_j \quad \text{for all} \ (i, j) : A_{ij} = 1,
\end{align}

where A is an n × n adjacency matrix (Cormen et al., 2001) with the components

\[ A_{ij} = \begin{cases} 1, & \text{if} (i, j) \in E, \\ 0, & \text{otherwise}. \end{cases} \]

Thus, for any given partially ordered input data set D, one can construct a corresponding acyclic graph G (or equivalently an adjacency matrix A). In other words, formulation (1) implies (2) for some G, or it implies (3) for some A. It is not difficult to show that the reverse is also true, i.e. given an acyclic graph G and an associated observed response Y, it is possible...
to construct a data set $D$ whose problem formulation (1) is equivalent to (2). For this purpose, we can define the vectors $X_i \in \mathbb{R}^n$, $i = 1, \ldots, n$ component-wise as

$$
(X_i)_j = \begin{cases} 
1, & \text{if } i \text{ is a predecessor of } j \text{ or } i = j \\
0, & \text{otherwise}
\end{cases}
$$

In our algorithms, the data are processed in accordance with an order that must be a topological sort. Recall that a topological sort (see Cormen et al., 2001) of a directed acyclic graph $G = (N, E)$ is a linear ordering $T = (t_1, \ldots, t_n)$ of all its nodes such that if $G$ contains an edge $(t_i, t_j)$, then $i < j$. Usually, the number of topological orders is very large for a given graph. In MR, any ordering of the nodes $N$ induces an ordering of the corresponding observations. In the case of the topological order $T = (t_1, \ldots, t_n)$, $X_{t_i} < X_{t_j}$ implies $i < j$. Conversely, if $i < j$, then either $X_{t_i} < X_{t_j}$ or $X_{t_i}$ and $X_{t_j}$ are incomparable. The partial order defined by the relation $<$ on the set $\{X_1, \ldots, X_n\}$ allows us to produce any topological order directly from the observations without constructing $G$.

The most straightforward way of producing the set of edges $E$ is based on the comparison of all pairs in $X_1, \ldots, X_n$. This yields

$$
E = \{(i, j) : X_i < X_j, \ i, j \in N\}.
$$

The resulting graph may have plenty of redundant edges. We call edge $(i, j) \in E$ redundant if there exists $k \in N$ such that $(i, k) \in E$ and $(k, j) \in E$. The presence of the redundant edges does not affect the resulting solution to problem (2) or the $O(n^2)$ complexity of the GPAV algorithm, but it does increase the overall processing time and the memory required for storing $E$.

The redundant edges are removed in Algorithm 1 consecutively while constructing $E$. The vectors $X_1, \ldots, X_n$ are assumed to have been sorted topologically. The algorithm generates successively for $k = 1, 2, \ldots, n$ the subset of all those non-redundant edges in (4) that connect the nodes $1, 2, \ldots, k$.

**Algorithm 1** (Construction of Non-redundant Edges).

*Input*: Topologically sorted $X_1, \ldots, X_n$.

*Output*: $E$.

Set $E = \emptyset$.

For $k = 1, 2, \ldots, n$ do:

- set $P = \{i : X_i < X_k, \ 1 \leq i \leq k - 1\}$,
- set $P_+ = \{j : (j, i) \in E, \ i \in P\}$,
- set $E = E \cup \{(i, k) : i \in P \setminus P_+\}$.

One can easily show that the generated set $E$ contains no redundant edges, and it correctly presents the partial order defined by the set $\{X_1, \ldots, X_n\}$.

The optimal solution to (2) has the following form (Robertson et al., 1988). The nodes of the graph $G$ are partitioned into disjoint blocks, and each block $B_j$ is characterized by the weight and the weighted average

$$
w(B_j) = \sum_{k \in B_j} w_k, 
$$

$$
\text{av}(B_j) = \frac{\sum_{k \in B_j} w_k Y_k}{w(B_j)}.
$$

If node $i \in B_j$, the corresponding component of the optimal solution is

$$
f^*_i = \text{av}(B_j).
$$

We call $\text{av}(B_j)$ the block common value of $B_j$.

The GPAV algorithm solves (2) approximately and produces a vector that satisfies the monotonicity constraints. The output of this algorithm can be represented as a set of blocks with the corresponding weighted averages defined by (6). The GPAV solution is not necessarily exact, because some of the generated blocks may differ from those in the optimal solution. Nevertheless, in practice it does ensure reasonably high accuracy; see Burdakov et al. (2006a,b).

The basic idea of the GPAV algorithm can be described as follows. The nodes of the graph $G$ are treated sequentially in the order given by $T$. At each step of the algorithm, a block is built by merging the current node $i$ with some other nodes. Initially, the block is a singleton set $\{i\}$, and then it can subsequently be enlarged by merging it with one of its adjacent blocks, if the monotonicity between the two blocks is violated. We say that two blocks $B_j$ and $B_i$ violate the monotonicity if there exists $l_0 \in B_j$ and $l_0 \in B_i$ such that $(l_0, l_0) \in E$ and the inequality $\text{av}(B_j) < \text{av}(B_i)$ is violated. If there is more than one block that violates monotonicity with the current block, then the block $B_j$ is to be chosen for merging with $B_i$ must have the largest violation value $\text{av}(B_j) - \text{av}(B_i)$. Once node $i$ has been treated, there are no violations among the produced blocks. After treating all the nodes, the GPAV algorithm produces the fitted value $\hat{f}_i = \text{av}(B_j)$ for each block $B_j$ and each node $i \in B_j$.
The blocks produced by the GPAV algorithm satisfy the following monotonicity relation:

\[(i_0 \in B_i) \& (j_0 \in B_j) \& (X_{i_0} < X_{j_0}) \implies av(B_i) < av(B_j).\]  \hspace{1cm} (8)

The input and output for the algorithm GPAV are as follows.

\textbf{Input:} Graph \(G\) with \(n\) nodes, observed response vector \(Y = (Y_1, \ldots, Y_n)\), vector of weights \(W = (w_1, \ldots, w_n)\), topological order \(T\). Each \(Y_i\) is associated with node \(i\).

\textbf{Output:} Set of blocks \(R = \{B_1, \ldots, B_m\}\), fitted values \(\hat{f}_1, \ldots, \hat{f}_n\).

3. The segmentation-based monotonic regression algorithm

The conventional MR algorithms, including the GPAV, can solve problem (2) if either the graph \(G\) or, equivalently, the adjacency matrix \(A\) is given. In the case of problem (1), neither of these is available. In order to obtain \(G\) or \(A\), it is necessary to perform a preprocessing of the input data set \(D\), and Algorithm 1 can be used for this purpose. Here, the combination of such preprocessing and the GPAV is referred to as PGPAV.

For large values of \(n\), the preprocessing is computationally very demanding, especially with regard to CPU time. The other issue is memory usage which increases faster than linearly with increasing \(n\). Indeed, the adjacency matrix has \(d \cdot n\) nonzero elements where \(d\) is the average number of predecessors. Although \(d\) is bounded above by \(n\), in practice it grows with \(n\). If the input data set is sufficiently large, the computer time or memory required for generating problems (2) or (3) may become prohibitively large.

Here, we suggest an alternative that allows direct solution of (1). Note that problem (1) exploits only the data set \(D\) with the memory size \(p \cdot n\). Since typically \(d \gg p\) for sufficiently large \(n\), it is reasonable to solve (1) directly in cases where a prohibitively large amount of memory is required to solve problem (2). We present a segmentation-based algorithm, which in practice uses little extra memory beyond the storage space required by \(D\).

The algorithm that we call SB consists of two stages: segmentation and assembly. At the segmentation stage, we choose a topological order \(T = (t_1, \ldots, t_n)\) for the partial order defined by the input data set \(D\), and then we choose the number of segments \(s\). Although we assume, for simplicity, that each segment has the same number of nodes, \(r = n/s\), our results can easily be extended to the case when \(n\) is not a multiple of \(s\). We partition the topological order \(T\) into \(s\) segments \(S_1 = \{t_1, \ldots, t_r\}, S_2 = \{t_{r+1}, \ldots, t_{2r}\}, \ldots, S_s = \{t_{(s-1)r+1}, \ldots, t_n\}\). The segmentation results in partitioning of the data set \(D\) into the subsets \(D_1, \ldots, D_s\), and the vector of weights \(W = (w_1, \ldots, w_n)\) into \(W_1, \ldots, W_s\).

Then, for each subset \(D_j\), a separate MR problem of the form (1) with the weights from \(W_j\) is solved using the PGPAV algorithm, and this results in the set of blocks \(R_j\). Recall that, in general, there are many possible topological orders. In previous work (Burdakov et al., 2006a), we studied how the accuracy of the solution produced by the GPAV algorithm was affected by the topological order chosen. In our preliminary experiments using the SB, we noticed that the choice of topological order can have a dramatic effect on the accuracy of the solution. In Section 4, we introduce a trend-following segmentation that seems most appropriate from both a theoretical and a practical point of view. The segment size \(r\) is another important factor that influences the computational complexity and accuracy. Possible choices of \(r\) are discussed in Section 7.

At the assembly stage, we formulate and solve a new MR problem of the form (2) where a new directed graph \(G_{N_{B}} = (N_{B}, E_{B})\) is used instead of \(G\). The sets of nodes \(N_{B}\) and edges \(E_{B}\) are constructed as follows. Each node in \(N_{B}\) is a block produced at the segmentation stage, hence, the set \(N_{B}\) is composed of all blocks in \(R = R_1 \cup \cdots \cup R_s\).

When constructing directed edges between the nodes in \(N_{B}\), we use this definition: observation \(i\) is called a \textit{predecessor} of observation \(j\) if \(X_i < X_j\), and this is denoted as \(i \prec j\). Observe that if graph \(G\) is available, \(i \prec j\) means that either node \(i\) is a predecessor of node \(j\), or there is a directed path from \(i\) to \(j\). In this connection, we will use the same notation \(i \prec j\) for the nodes in \(G_{B}\). The set of edges \(E_{B}\) is constructed as shown in Algorithm 2.

\section{Algorithm 2 (Edge Creation)}

Set \(E_{B} = \emptyset\).

For all pairs \(B_i, B_j \in N_{B}\),

\begin{itemize}
  \item if there are nodes \(i_0 \in B_i, j_0 \in B_j\) such that \(i_0 \prec j_0\), then add edge \((B_i, B_j)\) to \(E_{B}\).
\end{itemize}

Remove redundant edges.

Note that during the construction of \(E_{B}\) the redundant edges can be removed in the same way as in Algorithm 1 provided the blocks are sorted topologically. The simplest approach is to sort them segment-by-segment, and within each segment to sort them in increasing order of the available block common values. The idea underlying this sorting originates from (8).

If \((B_i, B_j) \in E_{B}\), we call block \(B_i\) a \textit{predecessor} of block \(B_j\). As shown in Lemma 3, for the blocks generated by the GPAV, if there exist \(i_0 \in B_i\) and \(j_0 \in B_j\) such that \(i_0 \prec j_0\), then there is no pair \(i_1 \in B_i, j_1 \in B_j\) such that \(j_1 \prec i_1\). To construct the set of edges \(E_{B}\), it is necessary to examine the blocks in \(N_{B}\) and ascertain, which one are predecessors of other blocks. The straightforward (but not the best!) way of accomplishing this task is to check each pair of blocks to determine whether there is any observation in one block that is a predecessor of an observation the other block. However, a strategy that is computationally much more efficient is to use Algorithm 3, the correctness of which is justified by Theorem 4. For this
Algorithm, we define minimum and maximum nodes: \( i \in B_j \) is called a minimum node of the block \( B_j \) if there is no \( k \in B_j \) such that \( k < i \); \( i \in B_j \) is called a maximum node of the block \( B_j \) if there is no \( k \in B_j \) such that \( i < k \).

**Algorithm 3** *(The Min–Max Algorithm)*.

1. Find all maximum and minimum nodes in the blocks \( B_i \) and \( B_j \).
2. Compare each minimum in one block with each maximum in another block.
3. Then:
   a. if \( j_{\text{min}} < j_{\text{max}} \) for some \( j_{\text{min}} \in B_i, j_{\text{max}} \in B_j \), add \( (B_i, B_j) \) to \( E_B \);
   b. if \( j_{\text{min}} < j_{\text{max}} \) for some \( j_{\text{min}} \in B_j, j_{\text{max}} \in B_i \), add \( (B_j, B_i) \) to \( E_B \);
   c. otherwise, blocks are unrelated and the edge is not added.

Algorithm 3 implies that it is unnecessary to compare each pair of observations in the two blocks. Instead, a relatively small fraction of observations (minima or maxima) in the blocks can be used to ascertain the block relation. Moreover, the algorithm can be terminated as soon as any edge is added to \( E_B \).

In **Theorem 5**, we show that the graph \( G_B \) is acyclic, i.e., the solution obtained at the segmentation stage is a feasible input for the assembly stage. This allows us to formulate a new problem of the type (2), where we set \( w_i := W(B_i) \), \( Y_i := av(B_i) \), and \( G := G_B \). Here \( B_i \) are blocks from the set \( R \). This problem is solved using the GPAV, which returns a set of blocks \( \hat{R} \). The fitted response \( \hat{f}_i \) is computed for each observation \( i \) by finding a block \( B_j \in \hat{R} \) such that \( i \in B_j \), and then setting \( \hat{f}_i = av(B_j) \). The fitted response will obviously be monotonic, since the blocks in \( \hat{R} \) were obtained using the GPAV and hence satisfy (8).

A general framework for the segmentation-based MR algorithm is presented in **Algorithm 4**.

**Algorithm 4** *(The Segmentation-based Monotonic Regression Algorithm)*.

Given a data set \( D = \{ (X_i, Y_i) : 1 \leq i \leq n \} \), number of segments \( s \), where \( n = sr, r \) is an integer

**Segmentation stage**

1. For \( D \), find any topological order \( T = (t_1, \ldots, t_n) \) *(the trend-following order is recommended, Algorithm 5)*
2. Set \( S_1 = \{ t_1, \ldots, t_r \}, S_2 = \{ t_{r+1}, \ldots, t_{2r} \}, \ldots, S_s = \{ t_{(s-1)r+1}, \ldots, t_n \} \)
3. For \( j = 1, \ldots, s \), create the data subset \( D_j = \{ (X_i, Y_i) : i \in S_j \} \) and the set of weights \( W_j = \{ w_i : i \in S_j \} \)
4. For each \( D_j \), create a problem of the form (2) with the weights from \( W_j \) and solve it using the GPAV that returns the set of blocks \( R_j \)

**Assembly stage**

1. Unite \( R_1, \ldots, R_s \) into the set \( R \)
2. Create a new graph \( G_B \):
   a. Set \( N_B = R \)
   b. Compute \( E_B \) by using Algorithms 2 and 3
3. Find any topological order \( T_B \) in \( G_B \).
4. Use \( T_B \) in the GPAV to solve a problem of the type (2)
   for \( G := G_B \) with \( w_i := W(B_i) \) and \( Y_i := av(B_i) \) defined \( \forall B_i \in R \),
   this returns the set of blocks \( \hat{R} \).
5. For \( i = 1, \ldots, n \), compute the solution components \( \hat{f}_i = av(B_i) \), where \( B_j \in \hat{R} \) is such that \( i \in B_j \)

Thus, the SB is characterized by

**Input:** Input data set \( D \), number of segments \( s \).

**Output:** Set of blocks \( \hat{R} \) and fitted values \( \hat{f}_1, \ldots, \hat{f}_n \).

4. **Trend-following order**

As already pointed out, the same partial order can admit several topological orders. The splitting into segments and, consequently, the fitted response depend on the topological order. According to our preliminary simulations, improper choice of a topological order may have a pronounced impact on accuracy.

Let us assume that an optimal solution to (2) is known, and a topological order \( T^* \) enumerates the observations by the increase of the optimal fitted values. In our previous paper (Burdakov et al., 2006a), we proved that if \( T^* \) is selected as the topological order in the GPAV, then the optimal solution will be returned by this algorithm.

Taking into account that the optimal fitted response is a good approximation of the true response \( f(X_i) \), it is natural to assume that if a topological order \( \hat{T} \) enumerates the observations by the increase of \( f(X_i) \) (i.e., in a trend-following order),
the fitted response obtained by the GPAV with \( \tilde{T} \) will be close to optimal. Our earlier simulations (Burdakov et al., 2006a) confirmed that the GPAV solution is highly accurate when a topological order is similar to \( \tilde{T} \). Consequently, it is reasonable to use \( \tilde{T} \) in the SB as well. In this case, the observations within any segment will be processed by the GPAV in a trend-following order, and the obtained components of the local solutions will be close to the optimal components. There is also a good chance that the global response acquired by combining the local fitted responses will be close to the optimal response. Since \( f(X_i) \) is not usually available, in Algorithm 5 we try to mimic the trend-following order. This algorithm begins with sorting \( Y_1, Y_2, \ldots, Y_n \) in increasing order of the \( Y_i \)'s. Assume that the result is

\[
Y_{p_01} \leq Y_{p_02} \leq \cdots \leq Y_{p_on},
\]

(9)

Let \( P_0 = \{p_{01}, p_{02}, \ldots, p_{0n}\} \) denote the corresponding sequence of indexes. Clearly, any subsequence \( P \) of \( P_0 \) inherits the increasing order of the \( Y_i \)'s, which implies

\[
Y_{p(1)} \leq Y_i, \quad \forall i \in P,
\]

(10)

where \( P(1) \) denotes the first element of \( P \).

A recursive procedure \( \text{LowerY}(P) \) forms the basis of the algorithm. For any subsequence \( P \) of \( P_0 \), the procedure returns a permutation of \( P \) that corresponds to a topological order of the observations in \( P \). Consequently, the output of \( \text{LowerY}(P_0) \) provides our algorithm with the desired topological order of \( P_0 \). The procedure is recursively applied to \( P_0 \) and its subsequences as follows. If \( P \) is empty, \( \text{LowerY}(P) \) will return an empty sequence; otherwise, procedure \( \text{LowerY}(P) \) will consecutively build an ordered sequence of indexes \( T \subseteq P \) starting from \( T = \emptyset \). Let \( \text{Pred}(i, P) \) denote the subsequence of \( P \) composed of all observations in \( P \) that are predecessors of \( i \). At the beginning, the procedure computes for \( i = P(1) \) the predecessors \( P' = \text{Pred}(i, P) \) that are subsequently permuted by the recursive computation \( P'' = \text{LowerY}(P') \). Then \( T \) is updated by adding to its end \( P'' \) followed by \( i \). Thereafter, \( P \) is updated by removing both \( i \) and \( P'' \) from this sequence. While \( P \neq \emptyset \), the same sequence of operations is consecutively applied to the updated \( T \) and \( P \). When \( P \) becomes empty, the procedure returns \( T \).

Suppose that \( i_k = P(1), P'_k = \text{Pred}(i_k, P), \) and \( P''_k = \text{LowerY}(P'_k) \) are produced at stage \( k \) of the described process of successive updating \( T \) and \( P \). Also assume that \( m \) updates of this kind are performed before the \( \text{LowerY} \) returns \( T \) of the following structure:

\[
T = [T_1, T_2, \ldots, T_m],
\]

(11)

where, \( T_k = [P''_k, i_k], k = 1, 2, \ldots, m \). Note that each \( P''_k \) is of the same kind of nested structure as \( T \). In Section 5, this structure is exploited to prove that the \( \text{LowerY} \) returns a topological order of observations.

Since (10) holds for each of the updated \( P \), we have \( Y_{i_1} \leq Y_{i_2} \leq \cdots \leq Y_{i_m} \). Furthermore, each nested subsequence \( P''_k \), being produced by \( \text{LowerY} \) at the lower level of the recursion, is also characterized by the increasing values of \( Y_i \)'s for certain \( i \)'s. This allows us to view the principle underlying the algorithm as being trend-following in nature.

**Algorithm 5 (Trend-Following Order).**

Given input data set \( D \), sort the observations in \( D \) by the value of \( Y_i \) and obtain the order (in general, not topological) \( P_0 = \{p_{01}, \ldots, p_{0n}\} \) such that (9) holds.

The topological order is produced by \( \text{LowerY}(P_0) \), where

**PROCEDURE** \( \text{LowerY} \)

**Input:** \( P \).

**Output:** order.

1. Set \( T = \emptyset \).
2. While \( P \neq \emptyset \):
   2.1 set \( i = P(1) \) and \( P' = \text{Pred}(i, P) \);
   2.2 compute \( P'' = \text{LowerY}(P') \);
   2.3 set \( T = [T, P'', i] \);
   2.4 update \( P \) by removing from this sequence both \( i \) and all \( k \in P' \).
   end_while
3. Return order = \( T \).

In our preliminary experiments, the MR solutions provided by the trend-following order were much more accurate than those offered by the topological orders considered in our earlier work (Burdakov et al., 2006b). The numerical results presented in Section 7 reinforce these benefits of using the trend-following order.

5. Theoretical justification

In this section, we formulate and prove the theoretical results that were referred to in previous sections. We begin by studying Algorithm 5.

**Theorem 1.** The order produced by Algorithm 5 is topological.
**Theorem 4** If there exist blocks $B_i$ and $B_j$ in $E_B$, then

(1) If $B_i$ and $B_j$ are from the same segment, then $av(B_i) < av(B_j)$;

(2) If $B_i \subset S_k$, $B_j \subset S_l$, then $k \leq l$.

**Proof.** The first statement is merely a reformulation of the monotonicity of the fitted response produced by the GPAV.

Considering the second statement, $k = l$ simply means that the blocks are from the same segment. If blocks are from different segments, then, by the definition of edge in $E_B$, the relation $(B_i, B_j) \in E_B$ implies $i_0 < j_0$ for some $i_0 \in B_i$ and $j_0 \in B_j$. Therefore, $i_0 \in S_k$, $j_0 \in S_l$, and, by the definition of segment, we conclude that $k < l$. □

**Lemma 3.** All blocks obtained at the segmentation stage are non-contradicting.

**Proof.** Suppose blocks $B_i$ and $B_j$ belong to the same segment. If we assume that the blocks are contradicting, this would mean the existence of $i_0$, $i_1 \in B_i$, and $j_0$, $j_1 \in B_j$ such that $i_0 < j_0$ and $j_1 < i_1$. The monotonicity of the GPAV solution (8) implies $av(B_i) < av(B_j)$ for the first relation and $av(B_j) < av(B_i)$ for the second one, and this is obviously an incongruity, which proves that these blocks cannot be contradicting. Suppose $B_i$ and $B_j$ are from different segments, say, $B_i$ belongs to a segment that in the topological order precedes the segment that contains $B_j$. Then, by the definition of a segment, there cannot exist $i_1 \in B_i$ and $j_1 \in B_j$ such that $j_1 < i_1$. Therefore, these two blocks are non-contradicting. □

Since all blocks are non-contradicting, **Algorithm 2** is correctly defined for any pair of blocks $(B_i, B_j)$. This means that one and only one of the following three alternatives holds: $(B_i, B_j) \in E_B$; $(B_i, B_j) \in E_B$; $B_i$ and $B_j$ are not connected.

**Theorem 4 (The Min–Max Theorem).** $(B_i, B_j) \in E_B$ if and only if there exist a minimum node $i_{\text{min}} \in B_i$ and a maximum node $j_{\text{max}} \in B_j$ such that $i_{\text{min}}$ is a predecessor of $j_{\text{max}}$.

**Proof.** If there exist $i_{\text{min}} \in B_i$ and $j_{\text{max}} \in B_j$ such that $i_{\text{min}} < j_{\text{max}}$ then $(B_i, B_j) \in E_B$ by the construction of $E_B$. On the other hand, if $(B_i, B_j) \in E_B$ then there exist $i_0 \in B_i$ and $j_0 \in B_j$ such that $i_0 < j_0$. Considering $i_0$, either it has a predecessor $i_{\text{min}}$ which is a minimum in $B_i$, or $i_0$ is a minimum node, $i_{\text{min}} = i_0$. Similarly, there exists a maximum node $j_{\text{max}} \in B_j$ such that $j_0 < j_{\text{max}}$. Thus, $(B_i, B_j) \in E_B$ implies $i_{\text{min}} < j_{\text{max}}$. □

**Theorem 4** provides a procedure for establishing relations between any pair of blocks $B_i$ and $B_j$ (see **Algorithm 3**). Hence, the edges in $E_B$ can be constructed by simply comparing the minimum and maximum nodes rather than comparing all possible pairs of nodes. In practice, the minimum and maximum nodes constitute only a very small fraction of their block. Therefore, **Algorithm 3**, which is implemented in conjunction with **Theorem 4**, allows us to significantly decrease the number of pairwise node comparisons.

**Theorem 5.** The graph $G_n$ created by **Algorithm 4** is acyclic.

**Proof.** In contrast to the statement of the theorem, assume that $G_n$ has a cycle. By statement 2 of **Lemma 2**, the segment number corresponding to blocks cannot decrease when moving from one block in the cycle to its successor. Therefore, the only possibility is that all blocks in the cycle belong to the same segment. However, by statement 1 of **Lemma 2**, the block common value must increase when moving in the cycle from one block to its successor, which is impossible due to the finite number of blocks in the cycle. □
6. Computational complexity

Here we will consider how the complexities of the SB and the PGPAV depend on the number of observations \( n \) assuming that the dimensionality \( p \) does not change with \( n \). This is motivated by the fact that \( p \ll n \) in the applied MR problems, and also that \( p \) appears as a multiplier in the estimates below.

According to our earlier work (Burdakov et al., 2006b), the computational complexity of the GPAV algorithm is \( O(n^2) \). This estimate assumes the availability of a graph \( G = (N, E) \) presenting the partial order, which is the case in some applied problems. Recall that if \( G \) is not available as an input to the GPAV, we apply the PGPAV to solve MR problem (1).

We begin by studying the complexity of the PGPAV, which is important not only in its own right, but also because the same combination is used repeatedly in step 4 of the segmentation stage. Consider the preprocessing based on \( \text{Algorithm 1} \). In the input to this algorithm, the vectors \( X_1, \ldots, X_n \) must be sorted topologically, which, for instance, can be done by applying any \( O(n \log n) \) algorithm that can sort the observed \( X \)-vectors in increasing order of one of their components. At iteration \( k \) of \( \text{Algorithm 1} \), first, the set of the predecessors \( P \) of observation \( k \) is obtained by pairwise comparisons with observations \( 1, \ldots, k - 1 \), and the number of associated operations grows in proportion to \( p \cdot k \), meaning that the complexity of this step is \( O(k) \). Thereafter, \( E \) is used to construct the list of immediate predecessors \( P \) for the observations in \( P \) (at most, \( O(k^2) \) operations). The subsequent construction of the set \( P \setminus P \) for adding new edges to \( E \) requires also a maximum of \( O(k^2) \) operations. Since \( k \) varies from 1 to \( n \), the overall complexity of \( \text{Algorithm 1} \) is \( O(n^3) \).

Taking into account the \( O(n^2) \) complexity of the GPAV, the overall complexity of solving MR problem (1) by the use of the preprocessing (\( \text{Algorithm 1} \)) and the GPAV can be estimated as \( O(n^3) \). If, instead of \( \text{Algorithm 1} \), the preprocessing is based on (4) without removing redundant edges, the overall complexity of solving the same problem becomes \( O(n^4) \), but the amount of memory required in this case may become prohibitively large.

The complexity estimation for the SB is not straightforward, because it depends largely on the number of blocks produced by the PGPAV algorithm at the segmentation stage, the number of observations, and the fractions of minimum and maximum elements in each of these blocks, as well as other characteristics of the solution process. However, it is possible to derive a worst-case complexity estimate, although for most practical problems this complexity will be higher than the actual complexity.

First, consider the segmentation stage. The first step in \( \text{Algorithm 4} \) is to compute a topological order, and, if the trend-following order is used, the complexity of this step can be estimated as follows. The procedure \( \text{LowerY} \) employs a sequence \( P \) which is initially set to \( P_0 \). Finding the predecessors \( P' \) of a node \( i \) in sequence \( P \) (step 2.1) can be done by performing element-by-element examination of \( P \), which at most will require \( O(n) \) operations. Step 2.3 entails adding elements to a sequence (a maximum of \( O(n) \) operations). Removing \( P' \) and \( i \) from \( P \) (step 2.4) is also an \( O(n) \) procedure. Step 2.3 implies execution of the same procedure for a sequence shorter than \( P \). Therefore, the overall complexity of \( \text{LowerY}(P) \) can be estimated as the total number of the ‘while’-iterations in all nested recursive calls multiplied by the sum of the complexities of steps 2.1, 2.3, and 2.4. Since each such iteration is associated with a distinctive number \( i \) in step 2.1, and any \( i = 1, \ldots, n \) is treated only once, the worst-case complexity of \( \text{Algorithm 5} \) is finally estimated to be \( O(n^2) \). The splitting of the data into subsets in steps 2–3 of the segmentation stage obviously requires \( O(n) \) operations. For each of the \( s \) segments, step 4 involves the preprocessing and solving of the local MR problem by the use of the GPAV; the total complexity is \( s \cdot O(r^3) = O(n \cdot r^3) \).

Now consider the assembly stage. Step 1 requires \( O(n) \) operations. Recall that in step 2, the Min–Max algorithm (\( \text{Algorithm 3} \)) calls for pairwise comparisons of some observations. Assuming that all possible pairwise comparisons are performed, the worst-case estimate for step 2 is \( O(n^2) \). In \( \text{Algorithm 2} \), the complexity is \( O(n^3) \) after the removal of redundant edges. In this estimate, the worst case occurs when each of the blocks produced at the segmentation stage consists of a single element (i.e., when the original response is monotonic), but this case is of no practical interest. The complexity of step 3 depends on the choice of the topological order \( T_b \). If topological order enumerates the nodes according to the number of all predecessors (Burdakov et al., 2006a), the complexity of step 3 is \( O(n \log n) \). Step 4 involves running the GPAV for the graph \( G_b \), and this requires a maximum of \( O(n^2) \) operations. Finally, in step 5 \( O(n) \) operations are required to produce the solution components from the blocks.

The overall worst-case complexity becomes \( O(n^3) \) due to step 2 in the assembly stage. The actual number of operations in the SB can approach the worst-case complexity estimate if the original response is almost monotonic. In any case, this complexity is lower than the complexity of the exact algorithms, since the best known estimate for such algorithms is \( O(n^4) \) (Maxwell and Muchstadt, 1985).

7. Numerical results

In this section, we compare PGPAV with the SB, because the GPAV is the only one of the available algorithms that is capable of solving large-scale MR problems (2) and ensures the highest accuracy of the fitted response (Burdakov et al., 2006a,b). It is impossible to compare the SB with exact algorithms, because the computational burden of the exact algorithms grows too rapidly with increasing size of the problem. To evaluate the actual accuracy of the PGPAV and the SB, in Section 7.4 we suggest an approach that makes it possible to generate large-scale MR problems and provide their exact solutions. Here, we use MATLAB 7.7 to implement the algorithms, and the numerical results that are presented are obtained on a PC with a Pentium 4 processor (2.8 GHz, 3 GB RAM) and running Windows XP.
7.1. Test models

In our simulations, we generate data sets that are used as the inputs to the SB and the PGPAV. Both these algorithms are based on the trend-following topological order. The data sets are generated for a few linear models of the type
\[ y(x) = \sum_{i=1}^{p} \alpha_i x_i + \varepsilon \]
for \( p = 2, 3 \) and one nonlinear model for \( p = 2 \):
\[ y(x) = g(x_1) - g(-x_2) + \varepsilon, \tag{12} \]
where
\[ g(t) = \begin{cases} \sqrt{t}, & t \leq 0, \\ t^3, & t > 0. \end{cases} \]
This choice of the linear test problems is inspired by the following observation. The fitted values \( f_i^* \) that correspond to a local area of values of \( x \) depend primarily on the local behavior of the \( y(x) \). Due to the block structure of the vector of fitted values \( f_i^* \), these local values of \( f_i^* \) typically do not change if the values of \( y(x) \) are perturbed in distant areas. Therefore, we assume that the local behavior can be well imitated by linear models. By choosing proper values of \( \alpha_i \), we can model a slower and a faster monotonic increase. The nonlinear function combines the types of behavior that are under consideration. In addition, depending on the side from which \( x_1 \) or \( x_2 \) approaches zero, the function value changes either sharply or remarkably slowly.

In our extensive experiments, the following combinations of model parameters are chosen. First, we consider \( p = 2 \). We use various types of the error terms by choosing \( \varepsilon \sim N(0, 1) \) and \( \varepsilon \sim \text{D.E.}(1) \), where ‘N’ and ‘D.E.’ signify normal and double-exponential distributions, respectively. Inasmuch as our preliminary experiments had shown that the performance of both algorithms is only weakly dependent on the distribution of \( x \), here we present results only for \( x_i \sim N(0, 1), \ i = 1, \ldots, p \). Various combinations of \( (\alpha_1, \alpha_2) \) are selected to create models for which the slope is higher or lower than the error term variance, and also models showing different rates of increase along the axes. For this purpose, we choose the combinations \( (0.2, 0.2), (0.2, 2), \) and \( (2, 2) \), and we omit \( (2, 0.2) \) due to the symmetry. The weight components in the MR problems are set as \( w_i = 1, i = 1, \ldots, n \).

We also perform simulations for \( p = 3 \). The distributions of \( x \) and \( \varepsilon \) are the same as above, and results are presented only for \( \alpha_1, \alpha_2, \alpha_3 \), all equal to 0.2 or 2, i.e., the extreme cases. The combinations representing different rates of increase in different directions are omitted from the respective table because they do not provide any new information, as indicated by the observation that the performance characteristics of these combinations are within the bounds defined by the characteristics corresponding to the two extreme cases.

To reduce the influence of the random variation in the performance characteristics (e.g., CPU time, memory, and relative error), the values presented here are averaged over a certain number of data sets generated for the same combination of model parameters. This number, denoted by \( l \), varies depending on \( n \) and \( p \).

7.2. Performance evaluation based on time, memory, and relative deviation

For each model described in Section 7.1, we generate input data sets for various values of \( n \). Since the SB is designed to solve for large-scale problems, there is no reason to consider \( n < 10,000 \) where the PGPAV performs quite well. We run the SB and the PGPAV on the input data sets generated for \( n = 10,000, n = 50,000 \) and each combination of model parameters. The SB is also applied to data sets of the size \( n = 500,000 \) which are too large to allow the preprocessing. The segment size \( r = 2000 \) does not change with \( n \) in any runs of the SB (see Section 7.3 for an explanation).

To demonstrate that the SB can process very large data sets, we performed a single run of this algorithm for a set with \( n = 10^6 \) observations and a medium slope. More specifically, the settings are \( p = 2, y(x) = \alpha_1 x_1 + \alpha_2 x_2 + \varepsilon, (\alpha_1, \alpha_2) = (1, 1) \), and the values of \( \varepsilon \) and \( x \) are normally distributed with mean zero and variance one. The results obtained for \( n = 10^6 \) are described and discussed below.

For each combination of model parameters, we generated \( l = 50 \) data sets for \( n = 10,000, l = 5 \) for \( n = 50,000 \) and \( l = 3 \) for \( n = 500,000 \). Tables 1 and 2 present performance characteristics averaged over \( l \) data sets for each of these cases. The errors are normally distributed. In the process of running the PGPAV and SB, we registered the maximum memory size as the number of KBytes required by the algorithms, and the CPU time was measured in seconds. The sign ‘−’ corresponds to the cases where the preprocessing failed to produce any result due to the limited memory resources. One of the performance characteristics is the relative deviation computed by the formula:
\[ \text{dev} = \frac{\delta_{SB} - \delta_{PGPAV}}{\delta_{PGPAV}} \cdot 100\%, \]
where \( \delta_{SB} \) and \( \delta_{PGPAV} \) are the objective function values in (1) produced by the SB and PGPAV algorithms, respectively, for the same data set. The averaged value of this characteristic is presented in the row \( \text{dev} \).

In the same cases as presented in the columns in Tables 1 and 2 for the normally distributed errors, the values of \( \text{dev} \) for the double-exponentially distributed errors remain within the interval 1%–4%. The other characteristics are close to those presented in the other rows of the same tables. For \( p = 3 \) and the double-exponentially distributed errors, the deviation is
within 1\%–7\%. All of the results we obtained show that the accuracy of the SB is only slightly lower than that provided by the PGPAV. It should be noted that the total CPU time of the PGPAV consists of the preprocessing time and the running time for the GPAV alone. Our tests indicate that, compared to the GPAV, the preprocessing is much more time consuming, and it also requires much more memory.

Table 1 shows that the CPU times of both PGPAV and SB increase with \( n \). Compared to the PGPAV, the SB is approximately three times faster for \( n = 10,000 \) and 10 times faster for \( n = 50,000 \). The corresponding rates in the case \( p = 3 \) (Table 2) are lower, although they still increase with \( n \). When \( p = 2 \), the CPU time of the SB for \( n = 500,000 \) is of the same order as the CPU time of the PGPAV for \( n = 50,000 \). The computational time of the SB also increases with the slope, because the blocks are smaller when the slopes are higher. For the higher slopes, there are more memory blocks at the end of the segmentation stage, and therefore, it takes longer to construct the graph \( G_b \).

The maximum amount of memory consumed by the SB is determined by two factors: first, the maximum size of all graphs generated individually for each segment (this, in turn, depends on the number of observations in the respective segments whose size is defined by \( r \)); second, the size of \( G_b \), which is strongly dependent on \( n \) and the slope. For \( p = 2 \) and \( n = 10,000 \) or \( n = 50,000 \), the first factor predominates over the second, and therefore, a maximum of 12 KBytes of memory is needed in either of these cases. However, when the second factor predominates, an increase in memory usage is observed for higher slope and \( n \) (e.g., consider \( p = 2 \) and \( n = 500,000 \)). It is apparent that \( m_{PGPAV} \) grows much more rapidly with \( n \) than \( m_{SB} \). For both \( p = 2 \) and \( p = 3 \), the SB consumes about 10 times less memory than the PGPAV does, and in some cases even less.

Here, the SB algorithm was able to solve a very large MR problem comprising \( n = 10^6 \) observations, and this required \( t_{SB} = 4597 \) s and \( m_{SB} = 88,874 \) KBytes.

### 7.3. Choice of the segmentation

The performance of our algorithm SB depends on the segmentation strategy that is chosen. To avoid accumulation of the errors in the boundaries of the segments, it seems the most reasonable to produce only one segment including all observations. However, the limitations on the computational time and memory do not allow this to be done for very large problems. Instead, it is generally best to produce as small a number of segments as possible so that the SB will require acceptable amounts of running time and memory. Regardless of how many segments there are, and whether they are of the same size or not, the most important aspect is that the accuracy of the final solution is acceptable.
In further consideration of the segmentation process, for the sake of simplicity we assume that all segments are of the same size $r$. The CPU time depends on $r$, or, equivalently, the number of segments $s = n/r$. The CPU time of the segmentation stage is the sum of CPU times for each segment. Each of these is a running time of the PGPAV, which, as shown in Section 6, is bounded above by $O(r^k)$. In practice, it is typical that the running time of the PGPAV increases less rapidly with $r$. Assume that, for a given class of MR problems, it increases in proportion to $r^k$, where $1 < k \leq 3$. In this case, the total segmentation time grows in proportion to

$$s \cdot r^k = n \cdot r^{k-1}.$$  

The CPU time of the assembly stage increases as the number of segments $s$ grows, which means that it decreases with increasing $r$. The time increase occurs because the more segments there are, the larger the number of monotonicity violations to be treated at the assembly stage. Since the CPU time of the segmentation stage increases with $r$, and the assembly time decreases with $r$, we can anticipate U-shaped dependence of the total CPU time of the SB, $t_{SB}$, on the segment size $r$. The number of segments should affect the accuracy of the solution as well. It is natural to expect that, in general, the increase in the amount of segments will lead to an increase in the total error in the boundaries between the segments, and, hence will also lower the accuracy of the solution.

For the simulations here, we used the same set of models as given in Section 7.1. We considered linear models with normal and double-exponential error and the following settings: number of observations $n = 10,000$ or $n = 100,000$, low or high slopes, $p = 2$. For each model with $n = 10,000$, we generated $l = 50$ data sets. For $n = 100,000$, there were $l = 5$ data sets. We tested also several models for $p = 3$. The tested segment sizes were $r = 10$: $100$: $1000$: $2000$: $10,000$: and also $r = 5$ if $n = 10,000$. The results confirm that, for each model, the dependence of the CPU time on $r$ is U-shaped, and the objective function value $\delta$ tends to decrease when $r$ increases (although the total decrease is only around 2%). Fig. 1 presents the mentioned dependences on $r$ for a single set of model parameters. No other graphs plotted for any other combinations of parameters are shown here, because their shapes are very similar to those given in the picture. A common feature of all graphs of the CPU time versus $r$ is that there is an optimal segment size that provides the lowest CPU time. The optimal $r$ varies from one case to another, but we observed that for all cases the CPU times for $r = 1000$ and $r = 2000$ were close to the corresponding lowest values. Considering accuracy, it is apparent that the larger the $r$, the better the results, which is why we choose $r = 2000$ for testing the SB in the current study.

### 7.4. Accuracy evaluation

It should be remembered that the accuracy of PGPAV in solving problem (1) is actually the accuracy of GPAV for solving the corresponding problem (2). To study the accuracy of the GPAV and the SB, we use the relative deviation:

$$dev^* = \frac{\delta_{alg} - \delta^*}{\delta^*} \cdot 100\%,$$

where $\delta_{alg}$ is the objective function value produced by a given algorithm, and $\delta^*$ is the optimal value.

In our previous paper (Burdakov et al., 2006b), the GPAV did offer sufficiently high accuracy, but the $dev^*$ was calculated only for the problems with $n \leq 400$. This can be explained by the fact that, when $n$ is large, the available exact algorithms producing $\delta_{exact}$ cannot be used because of their high computational burden. However, it is very important for the accuracy evaluation to have the value of $dev^*$ available in the case of large data sets.

Below, we present an approach that, for any given true monotonic response function $f(x)$, makes it possible to generate an input data set $(X_i, Y_i) : j = 1, \ldots, n$ along with the exact solution $f^* = (f_1^*, \ldots, f_n^*)^T$ to the corresponding MR problem (1). First, we generate $X_1, \ldots, X_n$, then we use the given $f(x)$ we construct $f^*$ and, finally, we calculate the values $Y_1, \ldots, Y_n$. The final stage of this process is based on the following properties of the exact solution.

---

**Fig. 1.** The SB’s processing time $\text{Time}$ (left) and objective function values $\delta$ (right) versus logarithmic segment size $\log_{10} r$. Settings: $p = 2$, $n = 10,000$, slope = $(2, 2)$, $x_i \sim N(0, 1)$, $\epsilon \sim N(0, 1)$. The $\text{Time}$ and $\delta$ are averages based on $l = 50$ sets.
Let the number of edges in the graph $G(N, E)$ be equal to $m$, and let $M \in \mathbb{R}^{m \times n}$ be the incidence matrix (see Cormen et al., 2001) for this graph. Denote:

$$f = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}, \quad Y = \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}, \quad w = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}, \quad \|v\|_w^2 = \sum_{i=1}^n w_i v_i^2.$$ 

Using these notations, the constraints $f_i - f_j \geq 0, \ \forall (i, j) \in E$ can be written as $Mf \geq 0$, and then problem (2) can be presented equivalently as

$$\min \frac{1}{2} \|f - Y\|_w^2 \quad \text{s.t.} \quad Mf \geq 0.$$ 

(14)

According to the Karush–Kuhn–Tucker optimality conditions (Bazaraa et al., 1993), there should exist a non-negative vector of Lagrange multipliers $\lambda \in \mathbb{R}^n$, such that the optimal solution $f^*$ to problem (14) obeys

$$\text{diag}(w)(f^* - Y) - M^T \lambda = 0.$$ 

(15)

Consider the special case in which the solution to problem (14) constitutes one block composed of all nodes $N$. This means that

$$f^* = (c, \ldots, c)^T,$$ 

(16)

where $c$ is the block common value computed by (7). The key observation is that we can generate all possible values of $Y$ that yield the given optimal solution (16) by generating all non-negative vectors $\lambda$. Each value of $Y$ is obtained from (15) after substituting $f^*$ and $\lambda$ in this equation.

Note that any value of $Y$ obtained as suggested does not violate the relation $c = av(N)$. This can be easily shown by summing all equations in (15) and taking into account the special structure of the matrix $M$, namely, that each row of $M$ is composed entirely of zeros except for two elements whose values are 1 and $-1$. Issues related to implementation of our test generator are discussed below, and the basic steps are presented in Algorithm 6.

**Algorithm 6.** Given weights $w_1, \ldots, w_n$, a monotonic function $f(x)$ and random generators for $X$ and $\epsilon$.

1. Generate a set of observed values $\chi = \{X_1, \ldots, X_n\}$.
2. Create a set of non-intersecting cubes $Q(\tilde{X}_1), \ldots, Q(\tilde{X}_n)$ centered in $\tilde{X}_1, \ldots, \tilde{X}_n$ such that they are of equal size, cover the set $\chi$, and each of them contains at least one observation from $\chi$.
3. For each $j = 1, \ldots, \tilde{n}$, compute $\bar{w}_j$ equal to the sum of $w_i$ over all $X_i \in Q(\tilde{X}_j)$.
4. For each $j = 1, \ldots, \tilde{n}$, generate $\tilde{Y}_j = f(\tilde{X}_j) + \epsilon_j$.
5. Find $\bar{f}_1, \ldots, \bar{f}_\tilde{n}$ that solve the MR problem of the form (1) defined by the set of data and weights $\{(X_i, \tilde{Y}_j), \bar{w}_j : j = 1, \ldots, \tilde{n}\}$.
6. Find $Y_1, \ldots, Y_n$ for which the solution to the MR problem (1) is defined by the values $\tilde{f}_j = \bar{f}_j, i = 1, \ldots, n$, where $j$ is such that $X_i \in Q(\tilde{X}_j)$.

In the implementation of line 2 of this algorithm, we use a set of discrete points in $\mathbb{R}^n$. The components of these points have the form $kh$, where $h > 0$ is the discretization step and $k$ may have any integer value $0, \pm 1, \pm 2, \ldots$. Denoting $\bar{h} = (h/2, \ldots, h/2)^T \in \mathbb{R}^n$, we associate the cube

$$Q(\bar{X}) = \{x \in \mathbb{R}^n : \bar{X} - \bar{h} \leq x < \bar{X} + \bar{h}\}$$

with each discrete point $\bar{X}$. These cubes do not intersect and they cover $\mathbb{R}^n$. To identify the cube that covers $X_i$, we properly scale the components of this vector, round the scaled components, and apply the inverse scaling to the rounded components. This yields one of the discrete points that determines the cube covering $X_i$. The aim of this discretization is to diminish the number of observations in the original data set in order to make the reduced size of the new MR problem affordable for solving it with any available exact algorithm.

Line 5 produces the exact solution to the reduced MR problem. The solution is actually a partitioning of the set $\tilde{\chi} = \{\tilde{X}_1, \ldots, \tilde{X}_n\}$ on blocks with monotonic block common values. In our implementation of line 6, these blocks induce a partitioning of the set $\chi$ on the same number of subsets as follows. Let $\tilde{B}_i \subset \tilde{\chi}$ be a block produced in line 5. Then the corresponding subset is defined as

$$B_i = \bigcup_{\tilde{X} \in \tilde{B}_i} \{X_i : X_i \in Q(\tilde{X})\}.$$ 

(17)

and the block common value $av(\tilde{B}_i)$ is assigned to $av(B_i)$. 
It is easy to prove that if $X', X'' \in B_i$, and if there exists $X \in \chi$ such that $X' \prec X \prec X''$, then $X \in B_i$. This means that each subset defined by (17) is a block for the partial order induced by the points in $\chi$. It is also apparent that if $X_i \in B_i$ and $X_j \in B_j$ are such that $X_i \prec X_j$, then $av(B_i) \leq av(B_j)$. This implies that the inherited block common values $av(B_i)$ meet the monotonicity requirement.

In line 6, each block $B_i$ produces the respective subset of $Y_1, \ldots, Y_n$, namely, those $Y_j$'s for which $X_j \in B_i$. Eq. (15) is formulated for each $B_i$ in terms of its specific vector of weights, block common value $av(B_i)$ and the matrix $M$ that represents the partial order induced by the points in $B_i$. The components of the vector $\lambda$ are generated randomly.

We performed numerical experiments to study the relation between the distribution of the error terms $\Sigma = \{(Y_1 - f(X_1)), \ldots, (Y_n - f(X_n))\}$ and the distribution of $\lambda$ in (15). We used the same functions $f(x)$ as considered in Section 7.1 and $w_i = 1$, $i = 1, \ldots, n$. The components of $\lambda$ were all uniformly distributed from zero to one. The histograms and the normal plots of $\Sigma$ suggest that the error terms are close to normal but have slightly heavier tails and the error variance is close to one. We can conclude that the data sets produced by Algorithm 6 mimic well those produced earlier for the models described in Section 7.1.

For the accuracy evaluation, we again use the same set of models for $p = 2$ or $p = 3$ and the following combinations of values for $n$ and the number of generated data sets $l$: $n = 10,000, l = 50$; $n = 50,000, l = 5$; $n = 500,000, l = 3$. The value of $dev^*$ is computed individually for each data set using formula (13), and is subsequently averaged over $l$ data sets. The resulting values are presented in Tables 3 and 4.

We observe that the trend-following order that is introduced here allows the GPAV to achieve a much greater accuracy than is possible with other orders we applied in our previous studies (Burdakov et al., 2006a,b). It is notable that the difference between the SB and the GPAV deviation agrees well with the values presented in Tables 1 and 2. It can be seen that the SB and the GPAV provided substantial accuracy for most of the models, and the solutions are more accurate for those with low slope. The accuracy is somewhat lower for the high slope and $p = 3$, but is still quite acceptable considering there is a lack of alternative algorithms that can solve such large problems with similar accuracy. As expected, the SB offered a somewhat lower accuracy than the GPAV did, but this drawback can be regarded as ‘payment’ for the ability to solve very large problems. It should be emphasized that, in our numerical experiments, the variance of the error $\epsilon$ was of the same order as the range of the generated function values $f(X_i)$. We chose such a high level of noise deliberately to be able to monitor the performance of the algorithms in this difficult case.

8. Conclusions and discussion

By segmenting a large-scale problem into smaller problems, we were able to significantly decrease both the processing time and memory consumption. Our algorithm required only 1.5 h to successfully process a data set containing $10^6$ two-dimensional observations. Performance characteristics such as time and memory depend on the average block size at the segmentation stage which, in turn, is determined primarily by the dimensionality $p$ and the slope. In cases when the average block is very small (i.e., the observed response is highly consistent with the monotonicity), our SB algorithm offers no advantages over the PGPAV.

The segmentation approach is quite general in the sense that it permits the use of any MR algorithm other than the GPAV. Inasmuch as the problem is solved separately for each segment, parallel processing can be applied.

The results of our numerical experiments demonstrate that the SB solution is only slightly less accurate than the PGPAV solution for the nonlinear model and linear test models with different slopes, large numbers of observations, and quite
different types of error distributions (normal and double-exponential). For similar models with known exact solutions, our simulations showed that both the PGPAV and SB algorithms give reasonably high accuracy. In the light of the representative sample of models we used (the choice is motivated in Section 7.1), we believe that the SB algorithm that is based on the GPAV will prove efficient in solving other MR problems as well.

Unfortunately, we are currently unable to provide any theoretical estimate of the accuracy of the SB that would explain its success in our numerical experiments. However, due to the growing demand for solving very large problems and the present lack of alternative algorithms for handling such problems, we believe that the SB algorithm will prove to be of practical importance.

We have also reported some less important results in this paper which are nonetheless worth mentioning. The trend-following topological order we introduced here substantially improves the accuracy of the GPAV. Moreover, we developed an approach that makes it possible to simultaneously generate large data sets for MR problems and their exact solutions.

References


