Projection onto A Nonnegative Max-Heap

Authored by:

Jieping Ye
Liang Sun
Jun Liu

Abstract

We consider the problem of computing the Euclidean projection of a vector of length p onto a non-negative max-heap—an ordered tree where the values of the nodes are all nonnegative and the value of any parent node is no less than the value(s) of its child node(s). This Euclidean projection plays a building block role in the optimization problem with a non-negative max-heap constraint. Such a constraint is desirable when the features follow an ordered tree structure, that is, a given feature is selected for the given regression/classification task only if its parent node is selected. In this paper, we show that such Euclidean projection problem admits an analytical solution and we develop a top-down algorithm where the key operation is to find the so-called emph{maximal root-tree} of the subtree rooted at each node. A naive approach for finding the maximal root-tree is to enumerate all the possible root-trees, which, however, does not scale well. We reveal several important properties of the maximal root-tree, based on which we design a bottom-up algorithm with merge for efficiently finding the maximal root-tree. The proposed algorithm has a (worst-case) linear time complexity for a sequential list, and $O(p2)$ for a general tree. We report simulation results showing the effectiveness of the max-heap for regression with an ordered tree structure. Empirical results show that the proposed algorithm has an expected linear time complexity for many special cases including a sequential list, a full binary tree, and a tree with depth 1.

1 Paper Body

In many regression/classification problems, the features exhibit certain hierarchical or structural relationships, the usage of which can yield an interpretable model with improved regression/classification performance [25]. Recently, there have been increasing interests on structured sparisty with various approaches for incorporating structures; see [7, 8, 9, 17, 24, 25] and references therein. In this paper, we consider an ordered tree structure: a given feature is selected for the given regression/classification task only if its parent node is selected. To incorporate such ordered tree structure, we assume that the model parameter x ? Rp follows the non-negative max-heap structure1 : P = {x ? 0, xi ? xj ?(xi , xj) ? E t },

(1)

where T t = (V t , E t) is a target tree with V t = {x1 , x2 , . . . , xp } containing all the nodes and E t all the edges. The constraint set P implies that if xi is the parent node of a child node xj then the value of xi is no less than the value of xj . In other words, if a parent node xi is 0, then any of its child nodes xj is also 0. Figure 1 illustrates three special tree structures: 1) a full binary tree, 2) a sequential list, and 3) a tree with depth 1. 1 To deal with the negative model parameters, one can make use of the technique employed in [24], which solves the scaled version of the least square estimate.

1

x1 x1 x2

x3

x1

x2

x3

x4

x5

x6

x7 x2

x4

x5

x6

x3

x4

x5

x6

x7

x7

(a) (b) (c) Figure 1: Illustration of a non-negative max-heap depicted in (1). Plots (a), (b), and (c) correspond to a full binary tree, a sequential list, and a tree with depth 1, respectively.

The set P defined in (1) induces the so-called ?heredity principle? [3, 6, 18, 24], which has been proven effective for high-dimensional variable selection. In a recent study [12], Li et al. conducted a meta-analysis of 113 data sets from published factorial experiments and concluded that an overwhelming majority of these real studies conform with the heredity principles. The ordered tree structure is a special case of the non-negative garrote discussed in [24] when the hierarchical relationship is depicted by a tree. Therefore, the asymptotic properties established in [24] are applicable to the ordered tree structrue. Several related approaches that can incorporate the ordered tree structure include the Wedge approach [17] and the hierarchical group Lasso [25]. The Wedge approach incorporates such ordering information P p x2 by designing a penalty for the model parameter x as ?(x—P) = inf t?P 21 i=1 (tii + ti), with tree being a sequential list. By imposing the mixed ?1 -?2 norm on each group formed by the nodes in the subtree of a parent node, the hierarchical group Lasso is able to incorporate such ordering information. The hierarchical group Lasso has been applied for multi-task learning in [11] with a tree structure, and the efficient computation was discussed in [10, 15]. Compared to Wedge and hierarchical group Lasso, the max-heap in (1) incorporates such ordering information in a direct way, and our simulation results show that the max-heap can achieve lower reconstruction error than both approaches. In estimating the model parameter satisfying the ordered tree structure, one needs to solve the following constrained optimization problem: min f (x) (2) x?P

for some convex function f (?). The problem (2) can be solved via many approaches including subgradient descent, cutting plane method, gradient descent, accelerated gradient descent, etc [19, 20]. In applying these approaches, a key building block is the so-called Euclidean projection of a vector v onto the convex set P : 1 ?P (v) = arg min kx ? vk22 , (3) x?P 2 which ensures that the solution belongs to the constraint set P . For some special set P (e.g., hyperplane, halfspace, and rectangle), the Euclidean projection admits a simple analytical solution, see [2]. In the literature, researchers have developed efficient Euclidean projection algorithms for the ?1 -ball [5, 14], the ?1 /?2 -ball [1], and the polyhedra [4, 22]. When P is induced by a sequential list, a linear time algorithm was recently proposed in [26]. Without the non-negative constraints, problem (3) is the so-called isotonic regression problem [16, 21]. Our major technical contribution in this paper is the efficient computation of (3) for the set P defined in (1). In Section 2, we show that the Euclidean projection admits an analytical solution, and we develop a top-down algorithm where the key operation is to find the so-called maximal root-tree of the subtree rooted at each node. In Section 3, we design a bottom-up algorithm with merge for efficiently finding the maximal root-tree by using its properties. We provide empirical results for the proposed algorithm in Section 4, and conclude this paper in Section 5.

2

Atda: A Top-Down Algorithm

In this section, we develop an algorithm in a top-down manner called Atda for solving (3). With the target tree T t = (V t , E t), we construct the input tree T = (V, E) with the input vector v, where V = {v1 , v2 , . . . , vp } and E = {(vi , vj)—(xi , xj) ? E t }. For the convenience of presenting our proposed algorithm, we begin with several definitions. We also provide some examples for elaborating the definitions in the supplementary file A.1. 2

Definition 1. For a non-empty tree T = (V, E), we define its root-tree as any non-empty ? that satisfies: 1) V? ? V , 2) E ? ? E, and 3) T? shares the same root as T . tree T? = (V? , E) Definition 2. For a non-empty tree T = (V, E), we define R(T) as the root-tree set containing all its root-trees. Definition 3. For a non-empty tree T = (V, E), we define

P vi ?V vi ,0 , m(T) = max —V —

(4)

which equals the mean of all the nodes in T if such mean is non-negative, and 0 otherwise. Definition 4. For a non-empty tree T = (V, E), we define its maximal root-tree as: Mmax (T) = arg where

max

? T? ?R(T),m(T?)=mmax (T) T? =(V? ,E):

mmax (T) = max m(T?) T? ?R(T)

—V? —,

(5)

(6)

is the maximal value of all the root-trees of the tree T . Note that if two root-trees share the same maximal value, (5) selects the one with the largest tree size. ? is a part of a ?larger? tree T = (V, E), i.e., V? ? V and E ? ? E, we When T? = (V? , E) ? ? can treat T as a ?super-node? of the tree T with value m(T). Thus, we have the following definition of a super-tree (note that a super-tree provides a disjoint partition of the given tree): Definition 5. For a non-empty tree T = (V, E), we define its super-tree as S = (VS , ES), which satisfies: 1) each node inTVS = {T1 , T2 , . . . , Tn } isSa non-empty tree with Ti = (Vi , Ei), n 2) Vi ? V and Ei ? E, 3) Vi Vj = ?, i 6= j and V = i=1 Vi , and 4) (Ti , Tj) ? ES if and only if there exists a node in Tj whose parent node is in Ti . 2.1

Proposed Algorithm

We present the pseudo code for solving (3) in Algorithm 1. The key idea of the proposed algorithm is that, in the i-th call, we find Ti = Mmax (T), the maximal root-tree of T , set x ? corresponding to the nodes of Ti to mi = mmax (T) = m(Ti), remove Ti from the tree T , and apply Atda to the resulting trees one by one recursively. Algorithm 1 A Top-Down Algorithm: Atda Input: the tree structure T = (V, E), i Output: x ? ? Rp 1: Set i = i + 1 2: Find the maximal root-tree of T , denoted by Ti = (Vi , Ei), and set mi = m(Ti) 3: if mi ¿ 0 then 4: Set x ?j = mi , ?vj ? Vi 5: Remove the root-tree Ti from T , denote the resulting trees as T?1 , T?2 , . . . , T?ri , and apply Atda(T?j ,i), ?j = 1, 2, . . . , ri 6: else 7: Set x ?j = mi , ?vj ? Vi 8: end if 2.2

Illustration & Justification

For a better illustration and justification of the proposed algorithm, we provide the analysis of Atda for a special case?the sequential list?in the supplementary file A.2. Let us analyze Algorithm 1 for the general tree. Figure 2 illustrates solving (3) via Algorithm 1 for a tree with depth 3. Plot (a) shows a target tree T t , and plot (b) denotes the input tree T . The dashed frame of plot (b) shows Mmax (T), the maximal root-tree of T , and 3

1

x1

x2

x5

x3

x6 x12

x7

x8

x13

5
x4
x9 x14
x10
x11
-1
-4 1
x15
3
2
1
3
1 1
5
-1 1
2 0
3
-4 0
0
1
-4
-1
-1
2
-4 1
2
-1
2
2 4
(b)
(c)
3
1
1
-1
2
0
0
5
2 4
1
0 0 0 -1 -4 2 2 2
-1
2
(a)
2

-4
2 3 4
(f)
1
1 1
-1 0
2
3
5
1
3
-4
0
0
0
0
2
0
1
1
0
-1
-4
2
-1
2
1
-1
1 0
0
1
1
1
(e)
0
2
4
2
(d)

Figure 2: Illustration of Algorithm 1 for solving (3) for a tree with depth 3. Plot (a) shows the target tree T t , and plots (b-e) illustrate Atda. Specifically, plot (b) denotes the input tree T , with the dashed frame displaying its maximal root-tree; plot (c) depicts the resulting trees after removing the maximal root-tree in plot (b); plot (d) shows the resulting super-tree (we treat each tree enclosed by the dashed frame as a super-node) of the algorithm; plot (e) gives the solution x ? ? R15 ; and the edges of plot (f) show the dual variables,

from which we can also obtain the optimal solution x ? (refer to the proof of Theorem 1).

we have Mmax (T) = 3. Thus, we set the corresponding entries of x ? to 3. Plot (c) depicts the resulting trees after removing the maximal root-tree in plot (b), and plot (d) shows the generated maximal root-trees (enclosed by dashed frame) by the algorithm. When treating each generated maximal root-tree as a super-node with the value defined in Definition 3, plot (d) is a super-tree of the input tree T . In addition, the super-tree is a max-heap, i.e., the value of the parent node is no less than the values of its child nodes. Plot (e) gives the ? ? R15 . The edges of plot (f) correspond to the values of the dual variables, from solution x ? ? R15 . Finally, we can observe that the which we can also obtain the optimal solution x non-zero entries of x ? constitute a cut of the original tree. We verify the correctness of Algorithm 1 for the general tree in the following theorem. We make use of the KKT conditions and variational inequality [20] in the proof. Theorem 1. x ? = Atda(T, 0) provides the unique optimal solution to (3). Proof: As the objective function of (3) is strictly convex and the constraints are affine, it admits a unique solution. After running Algorithm 1, we obtain the sequences {Ti }ki=1 and {mi }ki=1 , where k satisfies 1 ? k ? p. It is easy to verify that the trees Ti , i = 1, 2, . . . , k constitute a disjoint partition of the input tree T . With the sequences {Ti }ki=1 and {mi }ki=1 , we can construct a super-tree of the input tree T as follows: 1) we treat Ti as a super-node with value mi , and 2) we put an edge between Ti and Tj if there is an edge between the nodes of Ti and Tj in the input tree T . With Algorithm 1, we can verify that the resulting super-tree has the property that the value of the parent node is no less than its child nodes. Therefore, x ? = Atda(T, 0) satisfies x ? ? P. Let xl and vl denote a subset of x and v corresponding to the indices appearing in the subtree Tl , respectively. Denote P l = {xl : xl ? 0, xi ? xj , (vi , vj) ? El }, I1 = {l : ml ¿ 0}, I2 = {l : ml = 0}. Our proof is based on the following inequality: X X 1 1 1 min kx ? vk22 ? (7) min kxl ? vl k22 + min kxl ? vl k22 , x?P 2 xl ?P l 2 xl ?P l 2 l?I1

l?I2

which holds as the left hand side has the additional inequality constraints compared to the right hand side. Our methodology is to show that x ? = Atda(T, 0) provides the optimal solution to the right hand side of (7), i.e., x ?l = arg min

1 l kx ? vl k22 , ?l ? I1 , 2

(8)

x ?l = arg min

1 l kx ? vl k22 , ?l ? I2 , 2

(9)

xl ?P l

xl ?P l

4

which, together with the fact 12 k? ? ? P lead to our main x ? vk22 ? minx?P 21 kx ? vk22 , x argument. Next, we prove (8) by the KKT conditions, and prove (9) by the variational inequality [20]. Firstly, ?l ? I1 , we introduce the dual variable yij for the edge (vi , vj) ? El , and yii if vi ? Ll , where Ll contains all the leaf nodes of the tree Tl . Denote the root of Tl by vrl . For all vi ? Vl , vi 6= vrl , we denote its parent node by vji , and for the root vrl , we denote jrl = rl . We let Cil = {j—vj is a child node of vi in the tree Tl }. Ril = {j—vj is in the subtree of Tl rooted at vi }. ? To prove (8), we verify that the primal variable x ? = Atda(T, 0) and the dual variable y satisfy the following KKT conditions: ?(vi , vj) ? El , x ?i ? x ?j ?(vi , vj) ? El , (? xi ? x ?j)? yij ?vi ? Ll , y?ii x ?i X ?vi ? Vl , x ? i ? vi ? y?ij + y?ji i

? = =

0 0 0

(10) (11) (12)

=

0

(13)

?(vi , vj) ? El , y?ij ?vi ? Ll , y?ii

? ?

0 0,

(14) (15)

j?Cil

where y?jrl rl = 0 (Note that y?jrl rl is a dual variable, and it is introduced for the simplicity ? is set as: of presenting (12)), and the dual variable y

y?ji i

y?ii = 0, ?i ? Ll , X y?ij , ?vi ? Vl . = v i ? ml +

(16) (17)

j?Cil

According to Algorithm 1, x ?i = ml ¿ 0, ?vi ? Vl , l ? I1 . Thus, we have (10)-(12) and (15). It follows from (17) that (13) holds. According to (16) and (17), we have X y?ji i = vj ? —Ril —ml , ?vi ? Vl , (18) j?Ril

of Tl rooted at vi . From where —Ril — denotes the number of elements in Ril , the subtree P the nature of the maximal root-tree Tl , l ? I1 , we have j?Rl vj ? —Ril —ml . Otherwise, if i P l ? j?Ril vj ¡ —Ri —ml , we can construct from Tl a new root-tree Tl by removing the subtree of Tl rooted at vi , so that T?l achieves a larger value than Tl . This contradicts with the argument that Tl , l ? I1 is the maximal root-tree of the working tree T . Therefore, it follows from (18) that (14) holds. Secondly, we prove (9) by verifying the following optimality condition: hxl ? x ?l , x ?l ? vl i ? 0, ?xl ? P l , l ? I2 ,

(19)

l

which is the so-called variational inequality condition for x ? being the optimal solution to (9). According to Algorithm 1, if l ? I2 , we have x ?i = 0, ?vi ? Vl . Thus, (19) is equivalent to hxl , vl i ? 0, ?xl ? P l , l ? I2 .

(20)

For a given xl ? P l , if xi = 0, ?vi ? V l , (20) naturally holds. Next, we consider xl 6= 0. Denote by x ?l1 the minimal nonzero element in xl , and Tl1 = (Vl1 , El1) a tree constructed by removing the nodes corresponding to the indices in the set {i : xli = 0, vi ? VP l } from Tl . It is clear that Tl1 shares

the same root as Tl . It follows from Algorithm 1 that i:vi ?V 1 vi ? 0. l Thus, we have X X X hxl , vl i = x ?l1 vi + (xi ? x ?l1)vi ? (xi ? x ?l1)vi . i:vi ?Vl1
i:vi ?Vl1
i:vi ?Vl1
5
?lr the minimal If xli = x ?l1 , ?vi ? Vl1 , we arrive at (20). Otherwise, we set r = 2; denote by x Pr?1 l r?1 r r nonzero element in the set {xi ? j=1 x ?j : vi ? Vl }, and Tl = (Vl , Elr) a subtree of Pr?1 l r?1 Tl by removing those nodes with the indices in the set {i : xli ? j=1 x ?j = 0, vi ? Vlr?1 }. It is clear that TlrPshares the same root as Tlr?1 and Tl as well, so that it follows from Algorithm 1 that i:vi ?V r vi ? 0. Therefore, we have l
X
i:vi ?Vlr?1
(xi ?
r?1 X j=1
x ?lj)vi = x ?lr
X
vi +
i:vi ?Vlr
Repeating the above process until
Vlr
X
(xi ?
i:vi ?Vlr
r X j=1
x ?lj)vi ?
X
i:vi ?Vlr
(xi ?
r X
x ?lj)vi . (21)
j=1
is empty, we can verify that (20) holds.

For a better understanding of the proof, we make use of the edges of Figure 2 (f) to show the dual variables, where the edge connecting vi and vj corresponds to the dual variable y?ij , and the edge starting from the leaf node vi corresponds to the dual variable y?ii . With the dual variables, we can compute x ? via (13). We note that, for the maximal root-tree with a positive value, the associated dual variables are unique, but for the maximal root-tree with zero value, the associated dual variables may not be unique. For example, in Figure 2 (f), we set y?ii = 1 for i = 12, y?ii = 0 for i = 13, y?ij = 2 for i = 6, j = 12, and y?ij = 2 for i = 6, j = 13. It is easy to check that the dual variables can also be set as follows: y?ii = 0 for i = 12, y?ii = 1 for i = 13, y?ij = 1 for i = 6, j = 12, and y?ij = 3 for i = 6, j = 13.

3

Finding the Maximal Root-Tree

A key operation of Algorithm 1 is to find the maximal root-tree used in Step 2. A naive approach for finding the maximal root-tree of a tree T is to enumerate all possible roottrees in the root-tree set R(T), and identify the maximal root-tree via (5). We call such an approach Anae, which stands for a naive algorithm with enumeration. Although Anae is simple to describe, it has a very high time complexity (see the analysis given in supplementary file A.3). To this end, we develop Abuam (A Bottom-Up Algorithm with Merge). The underlying idea is to make use of the special structure of the maximal root-tree defined in (5) for avoiding the enumeration of all possible root-trees. We begin the discussion with some key properties of the maximal root-tree, and the proof is given in the supplementary file A.4. Lemma 1. For a non-empty tree T = (V, E), denote its maximal root-tree as Tmax = ? be a root-tree of Tmax . Assume that there are n nodes (Vmax , Emax). Let T? = (V? , E) / V? , 2) vij ? V , and 3) the parent node of vij is in vi1 , . . . , vin , which satisfy: 1) vij ? ? V . If n ? 1, we denote the subtree of T rooted at vij as T j = (V j , E j), j = 1, 2, . . . , n, j j j j Tmax = (Vmax , Emax) as the maximal root-trees of T j , and m ? = maxj=1,2,...,n m(Tmax). Then, the followings hold: (1) If n = 0, then Tmax = T? = T ; (2) If n ? 1, m(T?) = 0, and m ? = 0, then Tmax = T ; (3) If n ? 1, m(T?) ¿ 0, and m(T?) ¿ m, ? then Tmax = T?; (4) If j j n ? 1, m(T?) ¿ 0, and m(T?) ? m, ? then Vmax ? Vmax , Emax ? Emax and (vi0 , vij) ? Emax , j j j ? ? Vmax , Emax) = m; ? and (5) If n ? 1, m(T?) = 0, and m ? ¿ 0, then Vmax ?j : m(Tmax j ? Emax and (vi0 , vij) ? Emax , ?j : m(Tmax) = m. For the convenience of presenting our proposed algorithm, we define the operation ?merge? as follows: Definition 6. Let T = (V, E) be a non-empty tree, and T1 = (V 1 , E 1) and T2 = (V 2 , E 2) be two trees that satisfy: 1) they are composed of a subset of the nodes and edges of T , i.e., T V1T ? V , V 2 ? V , E 1 ? E, and E 2 ? E; 2) they do not overlap, i.e., V 1 V 2 = ?, and E 1 E 2 = ?; and 3) in the tree T , vi2 , the root node of T2 is a child of vi1 , a leaf node ? ? ? ? of T1 . We S define the operation S S?merge? as T = merge(T1 , T2 , T), where T = (V , E) with V = V1 V2 and E = E1 E2 {(vi1 , vi2)}. Next, we make use of Lemma 1 to efficiently compute the maximal root-tree, and present the pseudo code for Abuam in Algorithm 2. We provide the illustration of the proposed algorithm and the analysis of its computational cost in the supplementary file A.5 and A.6, respectively. 6

Algorithm 2 A Bottom-Up Algorithm with Merge: Abuam Input: the input tree T = (V, E) Output: the maximal root-tree Tmax = (Vmax , Emax) 1: Set T0 = (V0 , E0), where V0 = {xi0 } and E0 = ? 2: if vi0 does not have a child node in T then 3: Set Tmax = T0 , return 4: end if 5: while 1 do / V0 , 2) vij ? V , 6: Set m ? = 0, denote by vi1 , . . . , vin the n nodes that satisfy: 1) vij ? and 3) the parent node of vij is in V0 , and denote by T j = (V j , E j), j = 1, 2, . . . , n the subtree of T rooted at vij . 7: if n = 0 then 8: Set Tmax = T0 = T , return 9: end if 10: for j = 1 to n do j j), m) ? = Abuam(T j), and m ? = max(m(Tmax 11: Set Tmax 12: end for 13: if m(T0) = m ? = 0 then 14: Set Tmax = T , return 15: else if m(T?) ¿ 0 and m(T?) ¿ m ? then 16: Set Tmax = T0 , return 17: else j j)=m ? , T), ?j : m(Tmax 18: Set T0 =merge(T0 , Tmax 19: end if 20: end while

Making use of the fact that T0 is always a valid root-tree of Tmax , the maximal root-tree of T , we can easily prove the following result using Lemma 1. Theorem 2. Tmax returned by Algorithm 2 is the maximal root-tree of the input tree T .

4

Numerical Simulations

Effectiveness of the Max-Heap Structure We test the effectiveness of the max-heap structure for linear regression b = Ax, following the same experimental setting as in [17]. Specifically, the elements of A ? Rn?p are generated i.i.d. from the Gaussian distribution with zero mean and standard derivation and the columns of A are then normalized to have unit length. The regression vector x has p = 127 nonincreasing elements, where the first 10 elements are set as x?i = 11 ? i, i = 1, 2, . . . , 10 and the rest are zeros. We compared with the following three approaches: Lasso [23], Group Lasso [25], and Wedge [17]. Lasso makes no use of such ordering, while Wedge incorporates the structure by using an auxiliary ordered variable. For Group Lasso and Max-Heap, we try binary-tree grouping and list-tree grouping, where the associated trees are a full binary tree and a sequential list, respectively. The regression vector is put on the tree so that, the closer the node to the root, the larger the element is placed. In Group Lasso, the nodes appearing in the same subtree form a group. For the compared approaches, we use the implementations provided in [17]2 ; and for Max-Heap, we solve (2) with f (x) = 12 kAx?bk22 +?kxk1 for some small ? = r?kAT bk? (we set r = 10?4 , and 10?8 for the binary-tree grouping and list-tree grouping, respectively) and apply the accelerated gradient descent [19] approach with our proposed Euclidean projection. We compute the average model error kx ? x? k2 over 50 independent runs, and report the results with a varying number of sample size n in Figure 3 (a) & (b). As expected, GL-binary, MH-binary, Wedge, GL-list and MH-list outperform Lasso which does not incorporate such ordering information. MH-binary performs better than GL-binary, and MH-list performs better than Wedge and GL-list, due to the direct usage of such ordering information. In addition, the list-tree grouping performs better than the binary-tree grouping, as it makes better usage of such ordering information. 2

http://www.cs.ucl.ac.uk/staff/M.Pontil/software/sparsity.html

7

450

250 200 150

0

10 Computational Time

0

Computational Time

300

?1

10

?2

10

?3
10
100
Gaussian Distribution, Full Binary Tree
1
10
sequential list full binary tree tree of depth 1
10
350 Model error
Gaussian Distribution for v
1
10
Lasso GL?binary MH?binary
400
d=10 d=12 d=14 d=18 d=18 d=20
?1
10
?2
10
?3
10
?4
10
50 0 12
?5
15
18
20 25 Sample size
30
40
50
10
?4
4
5
10
10
6
10
10
(a)
(c)
120
0
10
10

Computational Time
40
Uniform Distribution, Full Binary Tree
1
sequential list full binary tree tree of depth 1 Computational Time
Model error
60
100
10
0
80
20 40 60 80 Random Initialization of v
(e)
Uniform Distribution for v
2
10
Wedge GL?list MH?list
100
?2
10
?4
10
d=10 d=12 d=14 d=18 d=18 d=20
?1
10
?2
10
?3
10
20 0 12
0
p
?6
15
18
20 25 Sample size
(b)
30
40
50
10
?4
4
5
10
10 p

(d)
6
10
10
0
20 40 60 80 Random Initialization of v
100
(f)

Figure 3: Simulation results. In plots (a) and (b), we show the average model error kx ? x? k2 over 50 independet runs by different approaches with the full binary-tree ordering and the list-tree ordering. In plots (c) and (d), we report the computational time (in seconds) of the proposed Atda (averaged over 100 runs) with different randomly initialized input v. In plots (e) and (f), we show the computational time of Atda over 100 runs.

Efficiency of the Proposed Projection We test the efficiency of the proposed Atda approach for solving the Euclidean projection onto the non-negative max-heap, equipped with our proposed Abuam approach for finding the maximal root-trees. In the experiments, we make use of the three tree structures as depicted in Figure 1, and try two different distributions: 1) Gaussian distribution with zero mean and standard derivation and 2) uniform distribution in [0, 1] for randomly and independently generating the entries of the input v ? Rp . In Figure 3 (c) & (d), we report the average computational time (in seconds) over 100 runs under different values of p = 2d+1 ? 1, where d = 10, 12, . . . , 20. We can observe that, the proposed algorithm scales linearly with the size of p. In Figure 3 (e) & (f), we report the computational time of Atda over 100 runs when the ordered tree structure is a full binary tree. The results show that the computational time of the proposed algorithm is relatively stable for different runs, especially for larger d or p. Note that, the source codes for our proposed algorithm have been included in the SLEP package [13].

5

Conclusion

In this paper, we have developed an efficient algorithm for the computation of the Euclidean projection onto a non-negative max-heap. The proposed algorithm has a (worst-case) linear time complexity for a sequential list, and O(p2) for a general tree. Empirical results show that: 1) the proposed approach deals with the ordering information better than existing approaches, and 2) the proposed algorithm has an expected linear time complexity for the sequential list, the full binary tree, and the tree of depth 1. It will be interesting to explore whether the proposed Abuam has a worst case linear (or linearithmic) time complexity for the binary tree. We plan to apply the proposed algorithms to real-world applications with an ordered tree structure. We also plan to extend our proposed approaches to the general hierarchical structure.
Acknowledgments This work was supported by NSF IIS-0812551, IIS-0953662, MCB-1026710, CCF-1025177, NGA HM1582-08-1-0016, and NSFC 60905035, 61035003.

8

2 References

[1] E. Berg, M. Schmidt, M. P. Friedlander, and K. Murphy. Group sparsity via linear-time projection. Tech. Rep. TR-2008-09, Department of Computer Science, University of British Columbia, Vancouver, July 2008. [2] S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge University Press, 2004. [3] N. Choi, W. Li, and J. Zhu. Variable selection with the strong heredity constraint and its oracle property. Journal of the American Statistical Association, 105:354?364, 2010. [4] Z. Dost? al. Box constrained quadratic programming with proportioning and projections. SIAM Journal on Optimization, 7(3):871?887, 1997. [5] J. Duchi, S. Shalev-Shwartz, Y. Singer, and C. Tushar. Efficient projection onto the ?1 -ball for learning in high dimensions. In International Conference on Machine Learning, 2008. [6] M. Hamada and C. Wu. Analysis of designed experiments with complex aliasing. Journal of Quality Technology, 24:130?137, 1992. [7] J. Huang, T. Zhang, and D. Metaxas. Learning with structured sparsity. In International Conference on Machine Learning. 2009. [8] L. Jacob, G. Obozinski, and J. Vert. Group lasso with overlap and graph lasso. In International Conference on Machine Learning, 2009. [9] R. Jenatton, J.-Y. Audibert, and F. Bach. Structured variable selection with sparsity-inducing norms. Technical report, arXiv:0904.3523v2, 2009. [10] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for sparse hierarchical dictionary learning. In International Conference on Machine Learning, 2010. [11] S. Kim and E. P. Xing. Tree-guided group lasso for multi-task regression with structured sparsity. In International Conference on Machine Learning, 2010. [12] X. Li, N. Sundarsanam, and D. Frey. Regularities in data from factorial experiments. Complexity, 11:32?45, 2006. [13] J. Liu, S. Ji, and J. Ye. SLEP: Sparse Learning with Efficient Projections. Arizona State University, 2009. [14] J. Liu and J. Ye. Efficient Euclidean projections in linear time. In International Conference on Machine Learning, 2009. [15] J. Liu and J. Ye. Moreau-yosida regularization for grouped tree structure learning. In Advances in Neural Information Processing Systems, 2010. [16] R. Luss, S. Rosset, and M. Shahar. Decomposing isotonic regression for efficiently solving large problems. In Advances in Neural Information Processing Systems, 2010. [17] C. Micchelli, J. Morales, and M. Pontil. A family of penalty functions for structured sparsity. In Advances in Neural Information Processing Systems 23, pages 1612?1623. 2010. [18] J. Nelder. The selection of terms in response-surface models?how strong is the weak-heredity principle? Annals of Applied Statistics, 52:315?318, 1998. [19] A. Nemirovski. Efficient methods in convex programming. Lecture Notes, 1994. [20] Y. Nesterov. Introductory Lectures on Convex Optimization: A Basic Course. Kluwer Academic Publishers, 2004. [21] P. M. Pardalos and G. Xue. Algorithms for a class of isotonic regression problems. Algorithmica, 23:211?222, 1999. [22] S. Shalev-Shwartz and Y. Singer. Efficient learning of label ranking by soft projections onto polyhedra. Journal of Machine Learning Research, 7:1567?1599, 2006. [23] R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society Series B, 58(1):267?288, 1996. [24] M. Yuan, V. R. Joseph, and H.

Zou. Structured variable selection and estimation. Annals of Applied Statistics, 3:1738?1757, 2009. [25] P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. Annals of Statistics, 37(6A):3468?3497, 2009. [26] L.W. Zhong and J.T. Kwok. Efficient sparse modeling with automatic feature grouping. In International Conference on Machine Learning, 2011.

9