

UCLA

Department of Statistics Papers

Title

Active Set Methods for Isotone Optimization

Permalink

<https://escholarship.org/uc/item/8rk7h4g3>

Author

de Leeuw, Jan

Publication Date

2008-10-09

ACTIVE SET METHODS FOR ISOTONE OPTIMIZATION

JAN DE LEEUW

ABSTRACT. Isotone optimization is formulated as a convex programming problem with simple linear constraints. A $\mathbb{R}$ implementation of a particular active set strategy is discussed, and applied to various isotone optimization problems important in statistics. The implementation is user-extendable, and handles a great many convex loss functions and partial orders.

1. INTRODUCTION

Suppose $\mathcal{I}_n = \{1, 2, \dots, n\}$ and $\succeq$ is a partial order on $\mathcal{I}_n$. A vector $x \in \mathbb{R}^n$ is $\succeq$ -isotone if $x_i \geq x_j$ for all index pairs with $i \succeq j$. In this paper we study the problem $\mathcal{P}(f, \succeq)$ of minimizing a closed proper convex function $f : \mathbb{R}^n \Rightarrow \mathbb{R}$ over all $\succeq$ -isotone vectors. Note that because x is totally ordered, all $\succeq$ -isotone vectors define linear extensions of the partial order $\succeq$. To prevent various kinds of problems that are irrelevant for our purposes anyway, we assume that f is continuous and bounded below by zero.

The inequalities defining isotonicity can be written in matrix form as $Ax \geq 0$, where A is a matrix in which each row corresponds with an index pair (i, j) such that $i \succeq j$. Such a row has element i equal to $+1$, element j equal to -1 , and the rest of the elements equal to zero.

In order to eliminate redundancies we include a row for a pair (i, j) if and only if i covers j , which means that $i \succeq j$ and there is no $k \neq i, j$ such that $i \succeq k \succeq j$. Thus the rows are taken from the *cover graph* (or the *Hasse diagram*) of the partial order. Figure 1 gives some examples.

Insert Figure 1 about here

Date: September 30, 2008 — 19h 41min — Typeset in TIMES ROMAN.

2000 Mathematics Subject Classification. 00A00.

Key words and phrases. Binomials, Normals, $\LaTeX$.

The isotone programming problem $\mathcal{P}(f, \succeq)$ can thus also be written as a convex programming problem $\mathcal{P}(f, A)$ with linear inequality constraints.

2. CONDITIONS FOR A MINIMUM

2.1. Kuhn-Tucker Vectors. A convex function f is minimized on a convex set $\{x \mid Ax \geq 0\}$ at $\hat{x}$ if and only if there exist a vector of Lagrange multipliers $\hat{\lambda}$ (also known as a *Kuhn-Tucker vector*) such that [Rockafellar, 1970, Chapter 28]

$$A'\hat{\lambda} \in \partial f(\hat{x}), \quad A\hat{x} \geq 0, \quad \hat{\lambda} \geq 0, \quad \hat{\lambda}'A\hat{x} = 0.$$

Here $\partial f(\hat{x})$ is the *subdifferential* of f at $\hat{x}$. The subdifferential at x is the set of all *subgradients* of f at x , where y is a subgradient at x if

$$f(z) \geq f(x) + (z - x)'y \quad \forall z.$$

In general, the subdifferential is a convex compact set. If f is differentiable at x there is a unique subgradient, the gradient $\nabla f(x)$. Thus the necessary and sufficient conditions for a $\hat{x}$ to be a minimizer in the differentiable case are existence of a Kuhn-Tucker vector $\hat{\lambda}$ such that

$$\nabla f(\hat{x}) = A'\hat{\lambda}, \quad A\hat{x} \geq 0, \quad \hat{\lambda} \geq 0, \quad \hat{\lambda}'A\hat{x} = 0.$$

2.2. Auxiliary Problems. We now define a number of related problems, all for a given f and a given $m \times n$ matrix A . Problem $\mathcal{P}$ is to minimize f over $Ax \geq 0$. The minimum is $\hat{f}$ and the minimizer is $\hat{x}$.

Write I for subsets of the index set $\mathcal{J} = \{1, 2, \dots, m\}$. Then $A(I)$ is the corresponding $\text{card}(I) \times n$ submatrix of A , and $A(\bar{I})$ is the $(m - \text{card}(I)) \times n$ complementary submatrix. The *active constraints* at x , which we write as $I(x)$, are the indices i for which $a_i'x = 0$.

Problem $\mathcal{P}_+(I)$ is to minimize f over $A(I)x = 0$ and $A(\bar{I})x > 0$. The solutions is $\hat{x}_+(I)$, and the minimum value is $\hat{f}_+(I) = f(\hat{x}_+(I))$. We have $\hat{f}_+(I) \geq \hat{f}$ for all $I \subseteq \mathcal{J}$. Because the partitioning into equality and strict inequality constraints partitions the constraint set $\{x \mid Ax \geq 0\}$ into 2^m faces, some of which may be empty, we also have $\hat{f} = \min_{I \subseteq \mathcal{J}} \hat{f}_+(I)$. Solution $\hat{x}_+(I)$ is optimal for $\mathcal{P}_+(I)$ if and only if there exist a Kuhn-Tucker vector $\hat{\lambda}_+(I)$ such that

$$A(I)'\hat{\lambda}_+(I) \in \partial f(\hat{x}_+(I)), \quad A(I)\hat{x}_+(I) = 0, \quad A(\bar{I})\hat{x}_+(I) > 0.$$

It follows that if $\hat{\lambda}_+(I) \geq 0$ then actually $\hat{x}_+(I)$ is optimal for $\mathcal{P}$. Conversely, if $I(\hat{x})$ are the indices of the active constraints at the solution $\hat{x}$ of $\mathcal{P}$, then $\hat{x}$ also solves $\mathcal{P}_+(I(\hat{x}))$.

We also define the 2^m problems $\mathcal{P}(I)$, which is to minimize f over $A(I)x = 0$, with minimum value $\hat{f}(I)$ and solution $\hat{x}(I)$. Now $\hat{f}(I) \leq \hat{f}_+(I)$ and $\hat{f}(I) \leq \hat{f}$ for all $I \subseteq \mathcal{I}$. $\hat{x}(I)$ is a solution if and only if there exists a Kuhn-Tucker vector $\hat{\lambda}(I)$ such that

$$A(I)' \hat{\lambda}(I) \in \partial f(\hat{x}(I)), \quad A(I) \hat{x}(I) = 0.$$

It follows that if $A(I) \hat{x}(I) \geq 0$ and $\hat{\lambda}(I) \geq 0$ then $\hat{x}(I)$ solves problem $\mathcal{P}$. Conversely $\hat{x}$ solves $\mathcal{P}(I(\hat{x}))$ and $\hat{f} = \min_{I \subseteq \mathcal{I}} \hat{f}(I)$, with the minimum attained for $I(\hat{x})$. Thus if we knew $I(\hat{x})$ we could solve $\mathcal{P}$ by solving $\mathcal{P}(I)$.

Because the problems $\mathcal{P}(I)$ play an important part in the manifold suboptimization algorithm given below, we discuss an equivalent formulation. The constraints $A(I)x = 0$ define a relation $\approx_I$ on $\{1, 2, \dots, n\}$, with $i \approx_I k$ if there is a row j of $A(I)$ in which both a_{ji} and a_{jk} are non-zero. The reflexive and transitive closure $\bar{\approx}_I$ of $\approx_I$ is an equivalence relation, which can be coded as an *indicator matrix* $G(I)$, i.e. a binary matrix in which all n rows have exactly one element equal to one. We can construct $G(I)$ from the adjacency matrix of $\bar{\approx}_I$ by selecting unique columns. Note that $G(I)$ is of full column-rank, even if $A(I)$ is singular. We write $r(I)$ for the number of equivalence classes of $\bar{\approx}_I$. Thus $G(I)$ is an $n \times r(I)$ matrix satisfying $A(I)G(I) = 0$, in fact $G(I)$ is a basis for the null space of $A(I)$. Moreover $D(I) \triangleq G(I)'G(I)$ is diagonal and indicates the number of elements in each of the equivalence classes.

Problem $\mathcal{P}(I)$ can be written as minimization of $f(G(I)\xi)$ over $\xi \in \mathbb{R}^{r(I)}$, which is a convex unconstrained problem. A vector $\hat{\xi}(I)$ is a solution if and only if $0 \in G(I)' \partial f(G(I)\hat{\xi}(I))$. Then $\hat{x}(I) = G(I)\hat{\xi}(I)$ solves $\mathcal{P}(I)$. If $0 \in G(I)' \partial f(G(I)\hat{\xi}(I))$ it follows that there is a non-empty intersection of the subgradient $\partial f(G(I)\hat{\xi}(I))$ and the row-space of $A(I)$, i.e. there is a Kuhn-Tucker vector $A(I)' \hat{\lambda}(I) \in \partial f(G(I)\hat{\xi}(I))$.

3. ALGORITHM

To solve the problem we use an *active set strategy*-[Gill et al., 1981, Chapter 5.2], in particular the *manifold optimization strategy* described in Zangwill [1967] and again in Zangwill [1969, Chapter 8]. We solve a finite sequence of subproblems

$\mathcal{P}(I)$, that minimize $f(x)$ over x satisfying $A(I)x$. After solving each of the problems we change the active set I , either by adding or by dropping a constraint. The algorithm can be expected to be efficient if minimizing f under simple equality constraints, or equivalently minimizing $f(G\xi)$ over ξ , can be done quickly and reliably.

ST: Suppose $x^{(s-1)}$ is a candidate solution in iteration $s-1$. It defines the index sets $I^{(s-1)} = I(x^{(s-1)})$ and $\bar{I}^{(s-1)} = \mathcal{J} - I(x^{(s-1)})$.

EQ: Suppose $y^{(s-1)}$ is a solution of $\mathcal{P}(I^{(s-1)})$. Then $y^{(s-1)}$ can be either feasible or infeasible for $\mathcal{P}$, depending on if $A(\bar{I}^{(s-1)})y^{(s-1)} \geq 0$ or not.

IN: If $y^{(s-1)}$ is *infeasible* we choose $x^{(s)}$ on the line between $x^{(s-1)}$ and $y^{(s-1)}$, where it crosses the boundary of the feasible region. This defines a new and larger set of active constraints $I(x^{(s)})$. Go back to step **EQ**.

FS: If $y^{(s-1)}$ is *feasible* we determine the corresponding Lagrange multipliers $\lambda^{(s-1)}$ for $\mathcal{P}(I^{(s-1)})$. If $\lambda^{(s-1)} \geq 0$ we have solved $\mathcal{P}$. If $\min \lambda^{(s-1)} < 0$, we find the most negative Lagrange multiplier and drop the corresponding equality constraint from $I^{(s-1)}$ to define a new and smaller set of active constraints $I^{(s)}$. Go back to step **EQ**.

Convergence of the algorithm follows from the fact that f decreases in each step and an index set $\mathcal{J}$ is never repeated. For details we refer to the publications by Zangwill we mentioned above.

In step **IN** we solve

$$\begin{aligned} \max_{\alpha} & x^{(s-1)} + \alpha(y^{(s-1)} - x^{(s-1)}) \\ \text{over} & \min_{i \in A(\bar{I}^{(s-1)})} a'_i x^{(s-1)} + \alpha(a'_i y^{(s-1)} - a'_i x^{(s-1)}) \geq 0. \end{aligned}$$

Finding the smallest Lagrange multiplier in step **FS** is straightforward to implement in the differentiable case. We have to solve $A(I^{(s-1)})'\lambda = \nabla f(y^{(s-1)})$, and because $G(I^{(s-1)})'\nabla f(y^{(s-1)}) = 0$ and $A(I^{(s-1)})$ is of full row-rank, there is a unique solution $\lambda^{(s-1)}$. In the convex non-differentiable case, which we discuss in the next section, matters are more complicated.

4. SOME NONSMOOTH CASES

If f is convex, but not differentiable, we have to deal with the fact that in general $\partial f(x)$ may not be a singleton. It is possible to develop a general theory for active

set methods in this case [Panier, 1987], but we will just look at some important special cases.

4.1. The ℓ_∞ (or weighted Chebyshev) norm. In the ℓ_∞ case we must minimize

$$f(\xi) = \|h(\xi)\|_\infty = \max_{i=1}^n |w_i h_i(\xi)|,$$

where $h(\xi) = z - G\xi$ are the *residuals*. We assume, without loss of generality, that $w_i > 0$ for all i .

The minimization can be done for each of the r columns of the indicator matrix G separately, and the solution $\hat{\xi}_j$ is the corresponding *weighted mid-range*. More specifically, let $I(j) = \{i \mid g_{ij} = 1\}$. Then

$$f_j(\hat{\xi}_j) = \min_{\xi_j} \max_{i \in I(j)} |z_i - \xi_j| = \max_{i,k \in I(j)} \frac{w_i w_k}{w_i + w_k} |z_i - z_k|.$$

If the (not necessarily unique) maximum over $(i, k) \in I(j)$ is attained at $(i(j), k(j))$, then the minimum of f over ξ is attained at

$$\hat{\xi}_j = \frac{w_{i(j)} z_{i(j)} + w_{k(j)} z_{k(j)}}{w_{i(j)} + w_{k(j)}},$$

where we choose the order within the pair $(i(j), k(j))$ such that $z_{i(j)} \leq \hat{\xi}_j \leq z_{k(j)}$. Now

$$\min_{\xi} f(\xi) = \max_{j=1}^r f_j(\hat{\xi}_j).$$

These results also applies if $I(j)$ is a singleton $\{i\}$, in which case $\hat{\xi}_j = z_i$ and $f_j(\hat{\xi}_j) = 0$. Set $\hat{x} = G\hat{\xi}$.

Next we must compute a subgradient in $\partial f(\hat{x})$ orthogonal to G . Suppose the e_i is a unit weight vectors, i.e. a vector with all elements equal to zero, except element i which is equal to either plus or minus w_i . Consider the set $\mathcal{E}$ of the $2n$ unit weight vectors. Then $f(\xi) = \max_{e_i \in \mathcal{E}} e_i' h(\xi)$. Let $\mathcal{E}(\xi) = \{e_i \mid e_i' h(\xi) = f(\xi)\}$. Then, by the formula for the subdifferential of the pointwise maximum of a finite number of convex functions (also known as Danskin's Theorem [Danskin, 1966]), we have $\partial f(\xi) = \text{conv}(\mathcal{E}(\xi))$, with $\text{conv}()$ the convex hull.

Choose any j for which $f_j(\hat{\xi}_j)$ is maximal. Such a j may not be unique in general. The index pair $(i(j), k(j))$ corresponds with the two unit weight vectors with non-zero elements $-w_{i(j)}$ and $+w_{k(j)}$. The subgradient we choose is the convex combination which has element -1 at position $i(j)$ and element $+1$ at position

$k(j)$. It is orthogonal to G , and thus we can find a corresponding Kuhn-Tucker vector.

4.2. The ℓ_1 (or weighted absolute value) norm. For the ℓ_1 norm we find the optimum $\hat{\xi}$ by computing *weighted medians* instead of weighted mid-ranges. Uniqueness problems, and the subdifferentials, will generally be much less smaller than in the ℓ_∞ . For ℓ_1 we define $\mathcal{E}$ to be the set of 2^n vectors $(\pm w_1, \pm w_2, \dots, \pm w_n)$. The subdifferential is the convex hull of the vectors $e \in \mathcal{E}$ for which $e'_i h(\xi) = \min_{\xi} f(\xi)$. If $h_i(\xi) \neq 0$ then $e_i = \text{sign}(h_i(\xi))w_i$, but if $h_i(\xi) = 0$ element e_i can be any number in $[-w_i, +w_i]$. Thus the subdifferential is a multidimensional rectangle. If the medians are not equal to the observations the loss function is differentiable. If $h_i(\xi) = 0$ for some i in $I(j)$ then we select the corresponding element in the subgradient in such a way that they add up to zero over all $i \in I(j)$.

5. IMPLEMENTATION NOTES

5.1. Computing Indicators. We compute $G(I)$ from $A(I)$ in two steps. We first make the adjacency matrix of $\approx_I$ and add the identity to make it reflexive. We then apply Warshall's Algorithm [Warshall, 1962] to replace the adjacency matrix by that of the transitive closure $\approx_I$, which is an equivalence relation. Thus the transitive adjacency matrix has blocks of ones for the equivalence classes. We then use the `unique()` function in R to select the unique rows of the transitive adjacency matrix, and transpose to get $G(I)$.

5.2. Packaged Loss Functions. There are three types of loss functions that actually are implemented in the package. Many more could be added with very little extra effort.

5.2.1. Differentiable Convex Functions. Since solving $\mathcal{P}(I)$ is an unconstrained convex problem, we can simply use the `optim()` in R to minimize $f(G(I)(\xi))$ over ξ , and then set $\hat{x} = G(I)\hat{\xi}$. We have implemented this for the differentiable case, using the BFGS option of `optim()`. This guarantees (if the optimum is found with sufficient precision) that the gradient at $\hat{x}$ is orthogonal to the indicator matrix $G(I)$, and consequently that Lagrange multipliers can be computed. By making sure that $A(I)$ has full row-rank, the Kuhn-Tucker vector is actually unique. The routine `fSolver()` takes the arguments `fobj()` and `gobj()`, which are functions

returning the loss function value and the gradient. Because of the way `optim()` is written it is also possible to tackle problems with non-differentiable loss functions, or even non-convex ones. But we are not responsible if this gets you into trouble.

5.2.2. Special Problems. In some cases solving $\mathcal{P}(I)$, i.e. minimizing $f(G(I)(\xi))$ over ξ , can be done more efficiently because of the structure of the problem. This is true, in particular, for least squares, least absolute value, Chebyshev, and Quantile regression. The solvers are, respectively, `lsSolver()`, `dSolver()`, `mSolver()` and `pSolver()`. We have added `lfSolver()`, which solves $\mathcal{P}(I)$ for the more general least squares problem $f(x) = (z-x)'W(z-x)$, where W is a not necessarily diagonal positive semi-definite matrix of order n .

5.2.3. Hybrids. In addition we have some differentiable solvers which internally computes loss function value and gradient, and then calls `fSolver()`. So they do not need to be given `fobj()` and `gobj()`. The first is `oSolver()`, which minimizes $f(x) = \sum_{i=1}^n w_i |z_i - x_i|^p$ for some $p > 1$. The solver `aSolver()` does asymmetric least squares, as in Efron [1991]. `eSolver()` minimizes the familiar ℓ_1 approximation $f(x) = \sum_{i=1}^n w_i \sqrt{(z_i - x_i)^2 + \epsilon}$. `sSolver()` minimizes negative Poisson likelihood, `hSolver()` does Huber-loss as in Huber [1981], and `iSolver()` does SILF-loss [Chu et al., 2004]. With little extra effort various other fashionable SVM and lasso isotone regressions could be added.

5.3. User-defined Functions. Since the driver function `activeSet()` has a separate R function to solve $\mathcal{P}(I)$ as one of its parameters, users can implement their own isotone regression methods. For differentiable convex function they can use `optim()`, or they can write their own subroutines. Note that it is not at all necessary that the problems are of the regression or projection type, i.e. minimize some norm $\|z - x\|$. In fact, the driver can be easily modified to deal with general convex optimization problems with linear inequality constraints which are not necessarily of the isotone type.

5.4. Computing Lagrange Multipliers. In each step of the algorithm we have selected a subgradient such that $A(I)'\lambda = \nabla f(x(I))$ is solvable, because we have made sure that $G(I)'\nabla f(x(I)) = 0$. In R we use `qr.coef(qr())` to actually compute the Kuhn-Tucker vector λ . By making sure that $A(I)$ has full row-rank, this Kuhn-Tucker vector is actually unique (for a given choice of the subgradient).

REFERENCES

- W. Chu, S.S. Keerthi, and C.J. Ong. Bayesian Support Vector Regression Using a Unified Loss Function. *IEEE Transactions on Neural Networks*, 15:29–44, 2004.
- J.M. Danskin. The Theory of Max-Min, with Applications. *SIAM Journal on Applied Mathematics*, 14:641–664, 1966.
- B. Efron. Regression Percentiles using Asymmetric Squared Error Loss. *Statistica Sinica*, 1:93–125, 1991.
- P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, New York, N.Y., 1981.
- P. Huber. *Robust Regression*. Wiley, New York, NY, 1981.
- E.R. Panier. An Active Set Method for Solving Linearly Constrained Nonsmooth Optimization Problems. *Mathematical Programming*, 37:269–292, 1987.
- R.T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.
- S. Warshall. A Theorem on Boolean Matrices. *Journal of the Association of Computer Machinery*, 9:11–12, 1962.
- W. I. Zangwill. *Nonlinear Programming: a Unified Approach*. Prentice-Hall, Englewood-Cliffs, N.J., 1969.
- W.I. Zangwill. A Decomposable Nonlinear Programming Approach. *Operations Research*, 15:1068–1087, 1967.

APPENDIX A. CODE

A.1. Programs.

```
1  #
2  #   activeSet package
3  #   Copyright (C) 2008 Jan de Leeuw <deleeuw@stat.ucla.edu>
4  #   UCLA Department of Statistics, Box 951554,
5  #   Los Angeles, CA 90095-1554
6  #
7  #   This program is free software; you can redistribute it
8  #   and/or modify it under the terms of the GNU General Public
9  #   License as published by the Free Software Foundation;
10 #   either version 2 of the License, or (at your option)
11 #   any later version.
12 #
13 #   This program is distributed in the hope that it will be
14 #   useful, but WITHOUT ANY WARRANTY; without even the implied
15 #   warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
16 #   PURPOSE. See the GNU General Public License for more
17 #   details.
18 #
19 #   You should have received a copy of the GNU General Public
20 #   License along with this program; if not, write to the
21 #   Free Software Foundation, Inc., 675 Mass Ave, Cambridge,
22 #   MA 02139, USA.
23 #####
24 #
25 # version 0.0.1, 2008-09-24, initial
26 # version 0.0.2, 2008-09-24, squashed a buggy
27 # version 0.1.0, 2008-09-25, replaced null space algorithm
28 # version 0.1.1, 2008-09-25, fSolver now works for iso=TRUE
29 # version 0.2.0, 2008-09-25, added dSolver, pSolver, mSolver
30 # version 0.2.1, 2008-09-25, corrected weightedMidRange
31 # version 0.2.2, 2008-09-26, improved qSolver
32 # version 1.0.0, 2008-09-26, throw out all iso=FALSE stuff
33 # version 1.0.1, 2008-09-27, additional xSolvers
34 # version 1.0.2, 2008-09-27, many buggies squashed
```

```

35 # version 1.0.3, 2008-09-27, check for optimality
36 # version 1.0.4, 2008-09-27, other rule to add to active set
37 # version 1.0.5, 2008-09-27, added SILF loss
38 # version 1.0.6, 2008-09-28, corrected mSolver
39 #
40 # To do (maybe):
41 #
42 # -- bound constraints
43 # -- regression constraints, as in  $(x-Zb)'W(x-Zb)$  with  $AZb \geq 0$ 
44 #
45
46 activeSet<-function(a,x,mySolver=lsSolver,ups=1e-12,check=FALSE
  ,...) {
47   extra<-list(...); n<-length(x)
48   xold<-x; ax<-aTx(a,xold)
49   ia<-is.active(ax,ups=ups)
50   repeat {
51     if (length(ia)==0) aia<-NULL
52     else aia<-a[ia,]
53     yl<-mySolver(xold,aia,extra)
54     y<-yl$y; lbd<-yl$lbd; fy<-yl$f; gy<-yl$gy
55     ay<-aTx(a,y)
56     iy<-which.min(ay); my<-ay[iy]
57     if (length(lbd)==0) ml<-Inf
58     else {
59       il<-which.min(lbd)
60       ml<-lbd[il]
61     }
62     if (is.pos(my,ups)) {
63       if (is.pos(ml,ups)) break()
64       xnew<-y; ax<-ay
65       ia<-ia[-il]
66     }
67     else {
68       k<-which((ax>0) & (ay<0))
69       rat<-ay[k]/(ax[k]-ay[k])
70       ir<-which.max(rat); alw<-rat[ir]
71       xnew<-y+alw*(xold-y)
72       ax<-aTx(a,xnew)

```

```

73         ia<-sort(c(ia,k[ir]))
74     }
75     xold<-xnew
76 }
77 lup<-rep(0,length(ay)); lup[ia]<-lbd; hl<-taTx(a,lup,n)
78 if (check) ck<-checkSol(y,gy,a,ay,hl,lup,ups)
79     else ck<-NULL
80 return(list(x=y,lbd=lup,f=fy,ay=ay,hl=hl,gy=gy,ck=ck))
81 }
82
83 # least squares with diagonal weights
84
85 lsSolver<-function(x,a,extra) {
86     w<-extra$w; z<-extra$z; n<-length(z)
87     if (length(a)==0) return(list(y=z,l=0,f=0))
88     if (is.vector(a)) a<-matrix(a,1,length(a))
89     indi<-mkIndi(a,n)
90     h<-crossprod(indi,w*indi); r<-drop(crossprod(indi,w*z))
91     b<-solve(h,r); y<-drop(indi**b); gy<-2*w*(y-z)
92     lbd<-mkLagrange(a,gy)
93     f<-sum(w*(y-z)^2)
94     return(list(y=y,lbd=lbd,f=f,gy=gy))
95 }
96
97 # least squares with non-diagonal weights
98
99 lfSolver<-function(x,a,extra) {
100     w<-extra$w; z<-extra$z; n<-length(z)
101     if (length(a)==0) return(list(y=z,l=0,f=0))
102     if (is.vector(a)) a<-matrix(a,1,length(a))
103     indi<-mkIndi(a,n)
104     h<-crossprod(indi,w**indi); r<-drop(crossprod(indi,w**z))
105     b<-solve(h,r); y<-drop(indi**b); gy<-2*drop(w** (y-z))
106     lbd<-mkLagrange(a,gy)
107     f<-sum(w*outer(y-z,y-z))
108     return(list(y=y,lbd=lbd,f=f,gy=gy))
109 }
110
111 # least absolute value

```

```

112
113 dSolver<-function(x,a,extra) {
114     w<-extra$w; z<-extra$z; n<-length(z)
115     if (length(a)==0) return(list(y=z,l=0,f=0))
116     if (is.vector(a)) a<-matrix(a,1,2)
117     indi<-mkIndi(a,n)
118     m<-ncol(indi); h<-rep(0,m)
119     for (j in 1:m) {
120         ij<-which(indi[,j]==1)
121         zj<-z[ij]; wj<-w[ij]
122         h[j]<-weightedMedian(zj,wj)
123     }
124     y<-drop(indi%%h); f<-sum(w*abs(z-y)); gy<-w*sign(y-z)
125     lbd<-mkLagrange(a,gy)
126     return(list(y=y,lbd=lbd,f=f,gy=gy))
127 }
128
129 # quantile loss function
130
131 pSolver<-function(x,a,extra) {
132     w<-extra$w; z<-extra$z; aw<-extra$aw; bw<-extra$bw; n
133         <-length(z)
134     if (length(a)==0) return(list(y=z,l=0,f=0))
135     if (is.vector(a)) a<-matrix(a,1,2)
136     indi<-mkIndi(a,n)
137     m<-ncol(indi); h<-rep(0,m)
138     for (j in 1:m) {
139         ij<-which(indi[,j]==1)
140         zj<-z[ij]; wj<-w[ij]
141         h[j]<-weightedFractile(zj,wj,aw,bw)
142     }
143     y<-drop(indi%%h); dv<-ifelse(y<=z,w*aw*(z-y),w*bw*(y-z))
144     f<-sum(dv); gy<-ifelse(y<=z,-w*aw,w*bw)
145     lbd<-mkLagrange(a,gy)
146     return(list(y=y,lbd=lbd,f=f,gy=gy))
147 }
148 # Chebyshev norm
149

```

```

150 mSolver<-function(x,a,extra) {
151     w<-extra$w; z<-extra$z; n<-length(z)
152     if (length(a)==0) return(list(y=z,l=0,f=0))
153     if (is.vector(a)) a<-matrix(a,1,2)
154     indi<-mkIndi(a,n)
155     m<-ncol(indi); h<-rep(0,m)
156     for (j in 1:m) {
157         ij<-which(indi[,j]==1)
158         zj<-z[ij]; wj<-w[ij]
159         h[j]<-weightedMidRange(zj,wj)
160     }
161     y<-drop(indi**h); dv<-w*(y-z)
162     i1<-which.max(dv); i2<-which.min(dv)
163     f<-max(abs(dv))
164     gy1<-rep(0,n); gy1[i1]<-w[i1]
165     lbd1<-mkLagrange(a,gy1)
166     gy2<-rep(0,n); gy2[i2]<-w[i2]
167     lbd2<-mkLagrange(a,gy2)
168     lbd<-(w[i2]*lbd1+w[i1]*lbd2)/(w[i1]+w[i2])
169     gy<-(w[i2]*gy1+w[i1]*gy2)/(w[i1]+w[i2])
170     return(list(y=y,lbd=lbd,f=f,gy=gy))
171 }
172
173 # arbitrary differentiable function
174
175 fSolver<-function(x,a,extra) {
176     fobj<-extra$fobj; gobj<-extra$gobj; n<-length(x)
177     if (length(a)==0) indi<-diag(n)
178     else {
179         if (is.vector(a)) a<-matrix(a,1,2)
180         indi<-mkIndi(a,n)
181     }
182     z<-drop(crossprod(indi,x))
183     p<-optim(z,
184         fn=function(u) fobj(drop(indi**u)),
185         gr=function(u) drop(crossprod(indi,gobj(drop(indi**u))
186             )),
187         method="BFGS")
188     y<-drop(indi**p$par); f<-p$value; gy<-gobj(y)

```

```

188     if (length(a)==0) lbd<-0
189         else lbd<-mkLagrange(a,gy)
190     return(list(y=y,lbd=lbd,f=f,gy=gy))
191 }
192
193 # Power Norms
194
195 oSolver<-function(x,a,extra) {
196     w<-extra$w; z<-extra$z; pow<-extra$p
197     fobj<-function(x) sum(w*(abs(x-z)^pow))
198     gobj<-function(x) pow*w*sign(x-z)*abs(x-z)^(pow-1)
199     return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
200 }
201
202 # Asymmetric Least Squares
203
204 aSolver<-function(x,a,extra) {
205     w<-extra$w; z<-extra$z; aw<-extra$aw; bw<-extra$bw
206     fobj<-function(x) sum(w*(x-z)^2*ifelse(x<z,aw,bw))
207     gobj<-function(x) 2*w*(x-z)*ifelse(x<z,aw,bw)
208     return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
209 }
210
211 # Approximate l_1
212
213 eSolver<-function(x,a,extra) {
214     w<-extra$w; z<-extra$z; eps<-extra$eps
215     fobj<-function(x) sum(w*sqrt((x-z)^2+eps))
216     gobj<-function(x) w*(x-z)/sqrt((x-z)^2+eps)
217     return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
218 }
219
220 # Poisson Likelihood
221
222 sSolver<-function(x,a,extra) {
223     z<-extra$z
224     fobj<-function(x) sum(x-z*log(x))
225     gobj<-function(x) 1-z/x
226     return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))

```

```

227 }
228
229 # Huber Loss
230
231 hSolver<-function(x,a,extra) {
232     w<-extra$w; z<-extra$z; eps<-extra$eps
233     fobj<-function(x) sum(w*ifelse(abs(x-z)<2*eps, ((x-z)^2)/(4
234         *eps), abs(x-z)-eps))
235     gobj<-function(x) w*ifelse(abs(x-z)<2*eps, ((x-z)/(2*eps),
236         sign(x-z))
237     return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
238 }
239
240 # SILF Loss
241
242 iSolver<-function(x,a,extra) {
243     w<-extra$w; z<-extra$z; eps<-extra$eps; beta<-extra$beta
244     fobj<-function(x) {
245         y<-abs(x-z)
246         g<-((y-(1-beta)*eps)^2)/(4*beta*eps)
247         g[which(y < (1-beta)*eps)]<-0
248         ii<-which(y > (1+beta)*eps)
249         g[ii]<-y[ii]-eps
250         return(sum(w*g))
251     }
252     gobj<-function(x) {
253         y<-x-z
254         g<-rep(0,length(y))
255         g[which(y < -(1+beta)*eps)]<-1
256         ii<-which((y > -(1+beta)*eps) & (y < -(1-beta)*eps))
257         g[ii]<- (y[ii]+(1-beta)*eps)/(2*beta*eps)
258         ii<-which((y > (1-beta)*eps) & (y < (1+beta)*eps))
259         g[ii]<- (y[ii]-(1-beta)*eps)/(2*beta*eps)
260         g[which(y > (1+beta)*eps)]<-1
261         return(w*g)
262     }
263     return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
264 }
265

```

```

264 aTx<-function(a,x) {
265     if (is.vector(x)) return(x[a[,1]]-x[a[,2]])
266     return(x[a[,1],]-x[a[,2],])
267 }
268
269 xT<-function(x) {
270     if (is.vector(x)) return(as.matrix(x))
271     else return(t(x))
272 }
273
274 taTx<-function(a,x,n) {
275     m<-nrow(a); h<-rep(0,n)
276     for (i in 1:m) {
277         h[a[i,1]]<-h[a[i,1]]+x[i]
278         h[a[i,2]]<-h[a[i,2]]-x[i]
279     }
280     return(h)
281 }
282
283 b2a<-function(b,n) {
284     m<-nrow(b)
285     a<-matrix(0,m,n)
286     for (i in 1:m) {
287         a[i,b[i,1]]<-1
288         a[i,b[i,2]]<- -1
289     }
290     return(a)
291 }
292
293 warshall<-function(a) {
294     n<-nrow(a)
295     for (j in 1:n) {
296         for(i in 1:n) {
297             if (a[i,j]==1) a[i,]<-pmax(a[i,],a[j,])
298         }
299     }
300     return(a)
301 }
302

```

```

303 mkIndi<-function(a,n) {
304   im<-matrix(0,n,n); m<-nrow(a)
305   for (i in 1:m) im[a[i,1],a[i,2]]<-im[a[i,2],a[i,1]]<-1
306   im<-im+diag(n)
307   return(t(unique(warshall(im))))
308 }
309
310 mkLagrange<-function(b,g) {
311   ta<-t(b2a(b,length(g)))
312   qa<-qr(ta)
313   return(qr.coef(qr(ta),g))
314 }
315
316 checkSol<-function(y,gy,a,ay,hl,lbd,ups) {
317   ckFeasibility<-min(ay)
318   ckLagrange<-min(lbd)
319   ckCompSlack<-sum(ay*lbd)
320   ckGrad<-max(abs(gy-hl))
321   return(c(ckFeasibility,ckLagrange,ckCompSlack,ckGrad))
322 }
323
324 weightedMedian<-function(x,w=rep(1,length(x))) {
325   ox<-order(x); x<-x[ox]; w<-w[ox]; k<-1
326   low<-cumsum(c(0,w)); up<-sum(w)-low; df<-low-up
327   repeat{
328     if (df[k] < 0) k<-k+1
329     else if (df[k] == 0) return((w[k]*x[k]+w[k-1]*x[k-1])/
330                                (w[k]+w[k-1]))
330     else return(x[k-1])
331   }
332 }
333
334 weightedFractile<-function(x,w=rep(1,length(x)),a=1,b=1) {
335   ox<-order(x); x<-x[ox]; w<-w[ox]; k<-1
336   low<-cumsum(c(0,w)); up<-sum(w)-low; df<-a*low-b*up
337   repeat{
338     if (df[k] < 0) k<-k+1
339     else if (df[k] == 0) return((w[k]*x[k]+w[k-1]*x[k-1])/
340                                (w[k]+w[k-1]))

```

```

340         else return(x[k-1])
341     }
342 }
343
344 weightedMidRange<-function(x,w=rep(1,length(x))) {
345     s<=0; n<=length(x)
346     if (n==1) return(x)
347     for (i in 1:(n-1)) for(j in (i+1):n) {
348         t<=w[i]*w[j]*abs(x[i]-x[j])/ (w[i]+w[j])
349         if (t > s) {
350             s<=t; i0<=i; j0<=j
351         }
352     }
353     return((w[i0]*x[i0]+w[j0]*x[j0])/ (w[i0]+w[j0]))
354 }
355
356 is.active<-function(f,ups=1e-12) which(abs(f) < ups)
357
358 is.pos<-function(x,ups=1e-12) x > -ups
359
360 is.neg<-function(x,ups=1e-12) x < ups

```

A.2. Examples.

```

1  set.seed(12345)
2  z<=rnorm(9)
3  wu<=rep(1,9)
4  ww<=1:9
5  wf<=crossprod(matrix(rnorm(81),9,9))/9
6  x0<=9:1
7  btota<=cbind(1:8,2:9)
8  btree<=matrix(c(1,1,2,2,2,3,3,8,2,3,4,5,6,7,8,9),8,2)
9  bprim<=cbind(
10     c(rep(1,3),rep(2,3),rep(3,3),rep(4,3),rep(5,3),rep(6,3)),
11     c(rep(c(4,5,6),3),rep(c(7,8,9),3)))
12 bloop<=matrix(c(1,2,3,3,4,5,6,6,7,8,3,3,4,5,6,6,7,8,9,9),10,2)
13
14 compPava<-function() {
15     cat("Comparison with gpava\n")
16     library(pava)

```

```

17   for (i in 1:100) {
18       z<-rnorm(9)
19       u<-gpava(x0,z)$yfit
20       h<-activeSet(btota,x0,lsSolver,w=wu,z=z)$x
21       k<-activeSet(btota,x0,fSolver,fobj=function(x) sum(wu*(
           x-z)^2),gobj=function(x) 2*drop(wu*(x-z)) )$x
22       print(max(apply(cbind(u,h,k),1,var)))
23   }
24 }
25
26 otherWeights<-function() {
27     cat("Diagonal Weights\n")
28     print(activeSet(btota,x0,lsSolver,check=TRUE,w=ww,z=z))
29     cat("Nondiagonal weights\n")
30     print(activeSet(btota,x0,lfSolver,check=TRUE,x0,w=wf,z=z))
31 }
32
33 otherNorms<-function() {
34     cat("Approximate l_1 with eps\n")
35     print(activeSet(btota,x0,eSolver,check=TRUE,z=z,w=wu,eps=1e
           -4))
36     cat("Approximate l_1 with power\n")
37     print(activeSet(btota,x0,oSolver,check=TRUE,z=z,w=wu,p=1.2)
           )
38     cat("Exact l_1\n")
39     print(activeSet(btota,x0,dSolver,check=TRUE,w=wu,z=z))
40     cat("Approximate l_infty with power\n")
41     print(activeSet(btota,x0,oSolver,check=TRUE,z=z,w=wu,p=7))
42     cat("Exact l_infty\n")
43     print(activeSet(btota,x0,mSolver,check=TRUE,w=wu,z=z))
44     cat("Poisson likelihood\n")
45     z<-rpois(9,5)
46     print(activeSet(btota,x0,sSolver,check=TRUE,w=wu,z=z))
47     cat("Asymmetric Least Squares\n")
48     print(activeSet(btota,x0,aSolver,check=TRUE,z=z,w=wu,aw=2,
           bw=1))
49     cat("Huber Norm\n")
50     print(activeSet(btota,x0,hSolver,check=TRUE,z=z,w=wu,eps=1)
           )

```

```
51   cat("SILF Norm\n")
52   print(activeSet(btota,x0,lsSolver,check=TRUE,z=z,w=wu,beta=
      .8,eps=.2))
53 }
54
55 otherOrders<-function() {
56   cat("Tree Order\n")
57   print(activeSet(btrees,x0,lsSolver,check=TRUE,w=wu,z=z))
58   cat("Block Order\n")
59   print(activeSet(bprim,x0,lsSolver,check=TRUE,w=wu,z=z))
60   cat("Loop Order\n")
61   print(activeSet(bloop,x0,lsSolver,check=TRUE,w=wu,z=z))
62 }
```

DEPARTMENT OF STATISTICS, UNIVERSITY OF CALIFORNIA, LOS ANGELES, CA 90095-1554

E-mail address, Jan de Leeuw: `deleeuw@stat.ucla.edu`

URL, Jan de Leeuw: `http://gifi.stat.ucla.edu`

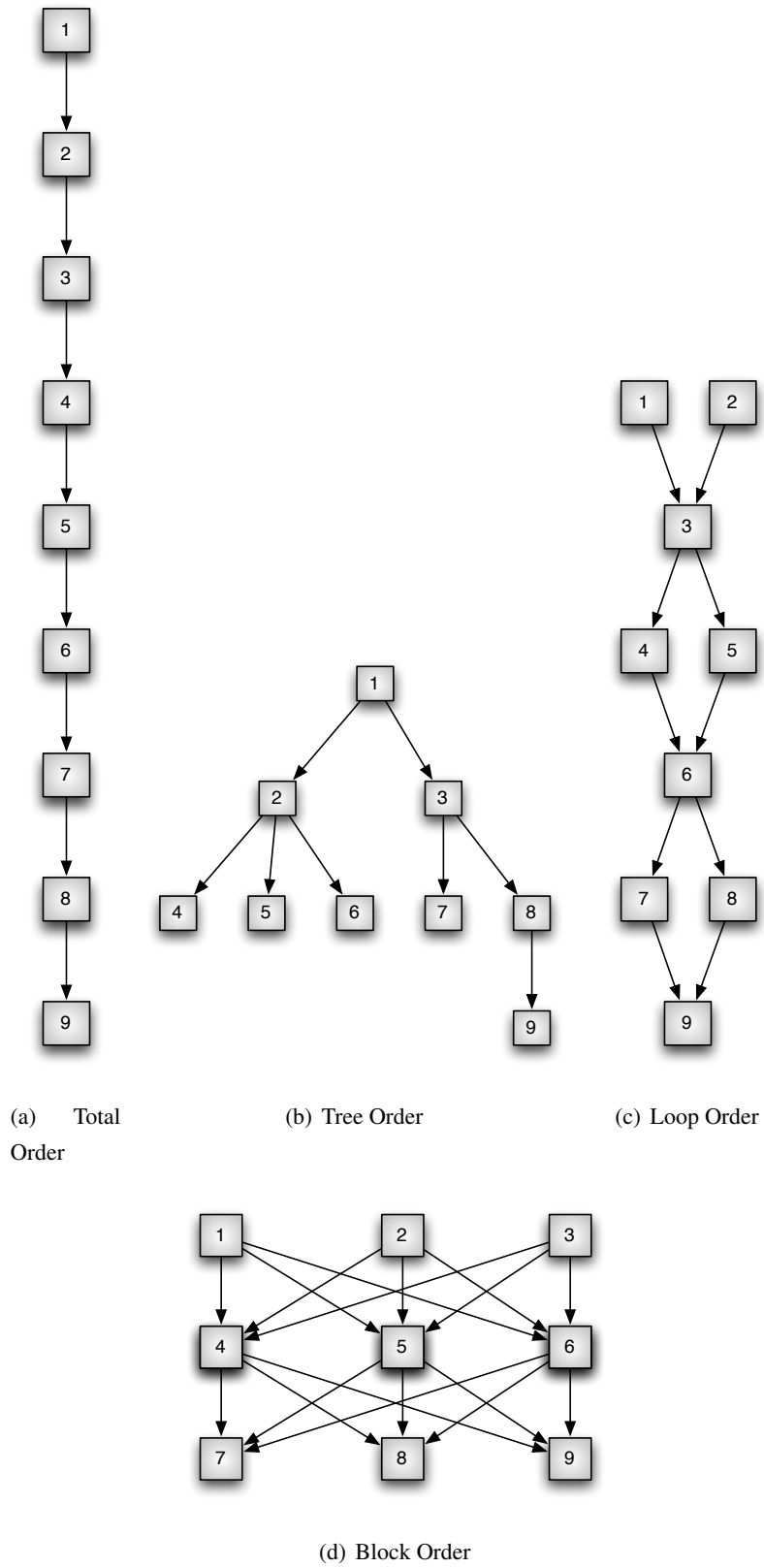


FIGURE 1. Some Partial Orders