WHInter: A Working set algorithm for High-dimensional sparse second order Interaction models.

Marine Le Morvan[1,2,3] and Jean-Philippe Vert[1,2,3,4]

[1]MINES ParisTech, PSL Research University, CBIO-Centre for Computational Biology, 75006 Paris, France
[2]Institut Curie, 75005 Paris, France
[3]INSERM, U900, 75005 Paris, France
[4]Ecole Normale Supérieure, Department of Mathematics and Applications, 75005 Paris, France

{marine.le_morvan,jean-philippe.vert}@mines-paristech.fr

February 19, 2018

Abstract

Learning sparse linear models with two-way interactions is desirable in many application domains such as genomics. ℓ_1-regularised linear models are popular to estimate sparse models, yet standard implementations fail to address specifically the quadratic explosion of candidate two-way interactions in high dimensions, and typically do not scale to genetic data with hundreds of thousands of features. Here we present WHInter, a working set algorithm to solve large ℓ_1-regularised problems with two-way interactions for binary design matrices. The novelty of WHInter stems from a new bound to efficiently identify working sets while avoiding to scan all features, and on fast computations inspired from solutions to the maximum inner product search problem. We apply WHInter to simulated and real genetic data and show that it is more scalable and two orders of magnitude faster than the state of the art.

1 Introduction

In application domains where the number of features exceeds the number of available samples, sparsity-inducing regularisers have a long history of success. Genomic prediction of complex phenotypes, biomedical imaging, astronomy or finance are a few examples. In particular the least squares with ℓ_1 regularisation, known as the LASSO (Tibshirani, 1996), has been extensively studied. It enjoys desirable statistical properties, since the number of samples required for exact support recovery of a sparse model scales as the logarithm of the number of features, under some assumptions (Wainwright, 2009). It also enjoys practical advantages, notably the interpretability of the learned models and the availability of fast solvers.

Indeed, a lot of research effort has been devoted to accelerating solvers for sparsity constrained problems in high dimension. A central idea is to exploit the sparsity of the solution to develop algorithms that do not spend too much time on optimising coefficients that will end up being 0. For example, safe screening rules identify features which are guaranteed to be inactive at the optimum so that their corresponding coefficients can be safely zeroed and set aside from the pool of coefficients to update (El Ghaoui et al., 2012; Xiang et al., 2011; Xiang and Ramadge, 2012; Fercoq et al.,

2015; Wang et al., 2013; Raj et al., 2016). Dynamic screening rules (Bonnefoy et al., 2015) such as the GAP safe rules (Fercoq et al., 2015) are particularly useful since more and more coefficients can be safely zeroed while the solver approaches the optimal solution. In spite of this, safe rules tend to be conservative, thereby limiting the potential speed-up. To remedy this drawback, new working set heuristics have been proposed. Working set algorithms iteratively solve subproblems, either problems restricted to a subset of features in the primal or to a subset of constraints in the dual, until convergence. Working set methods allow to focus coefficient updates on a set of features which can be significantly smaller than that yielded by safe rules. However this comes at a cost, that of checking the optimality conditions for all features at each iteration. BLITZ (Johnson and Guestrin, 2015) is a recently proposed working set algorithm that has been shown to have state-of-the-art performance for ℓ_1-regularised problems. Interestingly, the choice of the working sets in BLITZ can be seen as an aggressive use of the GAP safe rules (as noted in Massias et al., 2017) where the size of the working set is chosen to maximise the progress towards convergence. BLITZ can therefore be combined with the GAP safe rules (or the FLEX constraint elimination according to Johnson et al. terminology) at no cost. A direct comparison between BLITZ and the GAP safe rules by Ndiaye et al. (2017) illustrates the effectiveness of the working set approach. Further developments have also focused on coordinate descent (CD) to avoid wasteful coordinate updates, which represent most of the time spent by the solver (Fujiwara et al., 2016; Johnson and Guestrin, 2017).

The problem of fitting sparse linear models with two-way interactions has also attracted attention during the past decade. By two-way interactions we mean the entry-wise multiplication between two features; this is for example important in genomics to detect possible epistasis between genes. Surprisingly, very few of these works have links with the aforementioned literature. A majority of them focus on the design of sparsity-inducing penalties which enforce heredity assumptions and apply to moderate-dimensional settings ($p < 1,000$) (Radchenko and James, 2010; Bien et al., 2013; Lim and Hastie, 2015; Haris et al., 2016). Heredity assumptions state that an interaction can be included in the model only if one or both of its corresponding main effects are included. We note however that `glinternet` (Lim and Hastie, 2015) was applied to higher dimensional problems and in particular to a dataset with roughly $p = 27,000$ main effects, although the size of the learned model is not specified and the running time for the experiment is not reported by the authors. Interestingly, `glinternet` uses an active set strategy. Comparatively few works have been devoted to learning sparse regression models with interactions when the number of interactions is higher. Most of them are heuristics which start by selecting main effects and then incorporate interactions generated under the heredity constraint in a possibly iterative fashion. The simplest form of such heuristics consists in fitting a sparse linear model with the main effects only, and then fitting a second sparse linear model on all previously selected main effects and their interactions. This has been used in practice for example by Wu et al. (2009). Iterative refinements have been proposed where the LASSO is fit several times, and each time the set of candidate interactions considered is updated either by subsets, with the interactions between the K most relevant main effects selected at the previous fit (Bickel et al., 2010), or in a greedy fashion, where new interactions are included in the model as soon as a new main effect enters the LASSO path (Shah, 2016). In a similar vein, Hao and Zhang (2014) is based on a greedy model selection procedure instead of several LASSO fits. While these heuristics can deal with higher-dimensional problems than previous methods and enjoy some desirable statistical properties, they do not provide exact solutions and do not enjoy statistical properties as strong as those of the LASSO estimator.

An interesting link between the literature on interactions and that of solver acceleration with

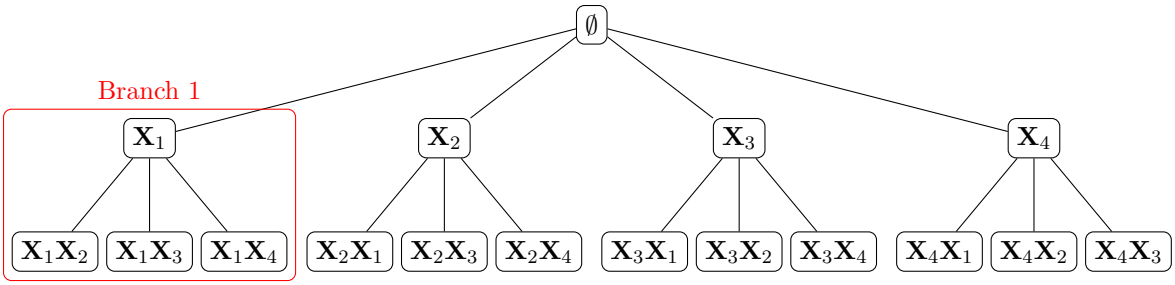

Figure 1 – Organisation of the main effects and interactions in a tree, depicted for 4 main effects.

sparsity inducing norms has been made recently by Nakagawa et al. (2016). In the case where variables are binary or with values in $[0, 1]$, they propose an approach called Safe Pattern Pruning (SPP) which is able to provide the optimal solution of the LASSO with two-way interactions for fairly high-dimensional problems, with no heredity constraint. Typically, for a problem with 1,000 samples and 10,000 main effects, SPP can provide solutions for a grid of regularisation parameters within one or two hours on a laptop with one core. SPP relies on the recently developed GAP safe screening rules. More precisely, the authors propose a safe pattern pruning criterion that can safely discard subsets of interactions from the model to speed up convergence. The performance of SPP is however hindered by several factors. One of them is that safe screening rules can be quite conservative even in the sequential setting. This property is inherited and amplified by the SPP criterion which can lead to heavy computations. Moreover, the GAP safe rules rely on a dual feasible point which is expensive to compute especially when the number of interactions is huge.

Inspired by SPP and the acceleration of solvers for sparsity constrained problems we propose a scalable algorithm, WHInter, to compute the optimal solution of ℓ_1-regularised linear problems with two-way interactions. WHInter is a working set method that efficiently delineates working sets among all interactions and main effects thanks to two contributions. First, we introduce a cheap and effective bound to rule out subsets of interactions that are guaranteed to be outside of the working set. Second, the identification of the working set among the remaining features is cast as a variant of the Maximum Inner Product Search (MIPS) problem to alleviate the afferent computational load. We find that WHInter is up to two orders of magnitude faster than SPP. For example, a problem with roughly 700 samples and 100,000 main effects can be solved for a grid of regularisation parameters in half an hour on a laptop with one core compared to more than 30 hours with SPP. This improvement in the scalability opens up new horizons in several application fields. The rest of the paper is organised as follows. In section 2, we present useful knowledge and notations used throughout the paper. In section 3 we describe in details our algorithm and our main contributions. In section 4, we evaluate WHInter on simulated datasets and finally in Section 5, we report results on a toxicogenomics prediction task.

2 Preliminaries

2.1 Setting and notations

For any integer $d \in \mathbb{N}$, we note $[\![d]\!] = \{1, \dots, d\}$ and $\mathbf{1}_d \in \mathbb{R}^d$ the d-dimensional vector of 1's. For any vector $\mathbf{u} = (\mathbf{u}_1, \dots, \mathbf{u}_d) \in \mathbb{R}^d$, we note $\|\mathbf{u}\|_1 = \sum_{i=1}^d |\mathbf{u}_i|$, $\|\mathbf{u}\|_2 = \left(\sum_{i=1}^d \mathbf{u}_i^2\right)^{1/2}$, $\text{supp}(\mathbf{u}) = \{i \in [\![d]\!] : \mathbf{u}_i \neq 0\}$ and $\|\mathbf{u}\|_0 = |\text{supp}(\mathbf{u})|$. For any two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, $\mathbf{u} \odot \mathbf{v}$ is the vector

Table 1 – **Summary of useful functions for the LASSO and logistic regression**: loss function f_i, its derivative f_i', its Fenchel-Legendre transform f_i^*.

	$f_i(u)$	$f_i'(u)$	$f_i^*(u)$
LASSO	$\frac{1}{2}(\mathbf{y}_i - u)^2$	$u - \mathbf{y}_i$	$\frac{1}{2}(\mathbf{y}_i + u)^2 - \frac{1}{2}\mathbf{y}_i^2$
Logistic regr.	$\log(1 + \exp(-\mathbf{y}_i u))$	$-\frac{u}{\mathbf{y}_i}\log(-\frac{u}{\mathbf{y}_i}) + (1 + \frac{u}{\mathbf{y}_i})\log(1 + \frac{u}{\mathbf{y}_i})$	$\frac{-\mathbf{y}_i}{1+\exp(\mathbf{y}_i u)}$

of entry-wise products, i.e., $(\mathbf{u} \odot \mathbf{v})_i := \mathbf{u}_i \mathbf{v}_i$ for $i = 1, \dots, d$. For any matrix $\mathbf{M}$, we denote by $\mathbf{M}_{i,j}$ its (i,j)-th entry, $\mathbf{M}_j$ its j-th column and by $\boldsymbol{m}_i$ its i-th row. For any $\mathbf{u} \in \mathbb{R}^d$ and $\mathcal{I} \subset [\![d]\!]$, $\mathbf{u}_\mathcal{I} = (\mathbf{u}_i)_{i \in \mathcal{I}}$, and similarly, if $\mathbf{M}$ is a matrix with d columns, $\mathbf{M}_\mathcal{I}$ is the sub-matrix with $|\mathcal{I}|$ columns $\mathbf{M}_\mathcal{I} = (\mathbf{M}_i)_{i \in \mathcal{I}}$.

Throughout the text we consider a design matrix $\mathbf{X} \in \{0,1\}^{n \times p}$ corresponding to n samples and p binary features, together with a response vector $\mathbf{y} \in \mathbb{R}^n$. We define an expanded design matrix $\mathbf{Z} \in \{0,1\}^{n \times D}$, with $D = p(p+1)/2$, which contains all p features from $\mathbf{X}$ plus the $p(p-1)/2$ interaction features. For clarity purposes, we define a symmetric indexing function $\tau : [\![p]\!]^2 \mapsto [\![D]\!]$ that uniquely assigns to every main effect and interaction an index in the expanded matrix $\mathbf{Z}$ such that $\mathbf{Z}_{\tau(j,k)} = \mathbf{Z}_{\tau(k,j)} := \mathbf{X}_j \odot \mathbf{X}_k$. In particular $\mathbf{Z}_{\tau(i,i)} = \mathbf{X}_i \odot \mathbf{X}_i = \mathbf{X}_i$ represents the i^{th} main effect. Since $\mathbf{X}$ is a binary matrix, the interaction feature $\mathbf{X}_j \odot \mathbf{X}_k$ corresponds to a logical AND between features $\mathbf{X}_i$ and $\mathbf{X}_j$. We organise the main effects and interactions in a simple tree as depicted in Figure 1 so as to reflect the property that $\forall (j,k) \in [\![p]\!]^2, \mathbf{Z}_{\tau(j,k)} \leq \mathbf{X}_j$ and $\mathbf{Z}_{\tau(j,k)} \leq \mathbf{X}_k$. In the sequel, the set composed of a main effect and its interactions with all other main effects will be referred to as *a branch* and for for any $j \in [\![p]\!]$, we note $\text{branch}(j) = \{\tau(j,k) : k \in [\![p]\!]\}$.

We consider the convex optimization problem:

$$\min_{(\mathbf{w},b) \in \mathbb{R}^D \times \mathbb{R}} P_{\mathbf{Z},\lambda}(\mathbf{w}, b) := F(\mathbf{Z}\mathbf{w} + b\mathbf{1}_n) + \lambda \|\mathbf{w}\|_1 := \sum_{i=1}^n f_i(\mathbf{z}_i \mathbf{w} + b) + \lambda \|\mathbf{w}\|_1 , \tag{1}$$

where $\lambda > 0$ is a regularisation parameter and, for any $i \in [\![n]\!]$, $f_i : \mathbb{R} \mapsto [-\infty, +\infty]$ is a loss function parametrised by $\mathbf{y}_i$ and assumed to be convex and differentiable. Table 1 provides examples of classical loss functions in classification and regression. A dual formulation of (1) reads:

$$\max_{\boldsymbol{\theta} \in \mathbb{R}^n} D_{\mathbf{Z},\lambda}(\boldsymbol{\theta}) := -\sum_{i=1}^n f_i^*(-\boldsymbol{\theta}_i) \quad \text{s.t.} \quad \begin{cases} |\mathbf{Z}_i^\top \boldsymbol{\theta}| \leq \lambda & \forall i \in [\![D]\!] , \\ \mathbf{1}_n^\top \boldsymbol{\theta} = 0 , \end{cases} \tag{2}$$

where f_i^* is the Fenchel-Legendre transform of the loss f_i, i.e., the function $f_i^* : \mathbb{R} \mapsto [-\infty, +\infty]$ defined by $f_i^*(u) = \sup_{v \in \mathbb{R}} uv - f_i(v)$. For the derivation of the dual problem, we refer the reader to Johnson and Guestrin (2015, Appendix E). The constraint $\mathbf{1}_n^\top \boldsymbol{\theta} = 0$ comes from the bias term $b\mathbf{1}_n$ in the primal problem (1). We denote by $(\mathbf{w}^*, b^*)$ and $\boldsymbol{\theta}^*$ a set of primal and dual optimal solutions to problems (1) and (2) respectively. Strong duality holds and therefore $(\mathbf{w}^*, b^*)$ and $\boldsymbol{\theta}^*$ satisfy Fermat's rules (Ndiaye et al., 2017):

$$\boldsymbol{\theta}^* = -\nabla F(\mathbf{Z}\mathbf{w}^* + b^* \mathbf{1}_n) , \tag{3}$$

and

$$\forall i \in [\![D]\!] , \quad \mathbf{Z}_i^\top \boldsymbol{\theta}^* \in \begin{cases} \{-\lambda, \lambda\} & \text{if } \mathbf{w}_i^* \neq 0 , \\ [-\lambda, \lambda] & \text{if } \mathbf{w}_i^* = 0 . \end{cases} \tag{4}$$

2.2 Basic working set algorithm

A general strategy to solve (1) is to follow a *working set* approach, as summarised in Algorithm 1. At each iteration, it solves (1) restricted to a small subset of features $\mathcal{W}$ called the working set. $\mathcal{W}$ is typically chosen as the set of features that violate the optimality condition (4) at the current iteration. In the sequel, we will call such features *violating features*, and the branches which contain at least one violating feature will be called *violating branches*. The algorithm converges when no violating feature remains, which occurs in a finite number of iterations as shown in Kowalski et al. (2011). When the number of interaction features runs into the billions, Algorithm 1 is not tractable since the delineation of the working set (line 3 in Alg. 1) requires $O(p^2 n)$ operations at each iteration.

Algorithm 1 Working set algorithm

Input: $\mathbf{Z} \in \{0,1\}^{n \times p}, \mathbf{y} \in \mathbb{R}^n, \lambda > 0$
Output: $\mathbf{w}^*, b^*$

1: Set $\boldsymbol{\theta} \leftarrow -\nabla F(\mathbf{0}_n)$, $\mathcal{W} = \emptyset$. ▷ Initialisation
2: **while** true **do**
3: $\quad \mathcal{W}' = \left\{ i \in [\![D]\!] : \left| \mathbf{Z}_i^\top \boldsymbol{\theta} \right| \geq \lambda \right\}$ ▷ Update the working set
4: $\quad$ **if** $\max_{i \in \mathcal{W}'} \left| \mathbf{Z}_i^\top \theta \right| \leq \lambda$ **then** Break **else** $\mathcal{W} \leftarrow \mathcal{W}'$
5: $\quad \mathbf{w}^*_{\mathcal{W}}, b^* \leftarrow \underset{\mathbf{w}_{\mathcal{W}}, b}{\operatorname{argmin}} P_{\mathbf{Z}_{\mathcal{W}}, \lambda}(\mathbf{w}_{\mathcal{W}}, b)$ ▷ Solve subproblem
6: $\quad \boldsymbol{\theta} \leftarrow -\nabla F(\mathbf{Z}_{\mathcal{W}} \mathbf{w}^*_{\mathcal{W}} + b^* \mathbf{1}_n)$.
7: **end while**

3 The WHInter algorithm

3.1 Overview

WHInter is a working set algorithm that follows the general scheme of Algorithm 1 but implements an efficient strategy to delineate the working set among all main effects and interactions. It is described in Algorithm 2. The identification of the working set (line 3 in Algorithm 1) corresponds to lines 11-18 in Algorithm 2. Instead of scanning through all features to build the working set, WHInter first identifies branches that are guaranteed to contain no violating feature. These branches are identified via the evaluation of a *branch bound* $\eta(\mathbf{X}_j, \boldsymbol{\Theta}_j^{ref}, \boldsymbol{\theta}, \mathbf{m}_j^{ref})$ (line 13) which is described in Section 3.2. The branch bound is cheap to evaluate since it solely depends on main effects and not on their numerous interactions. Moreover, it is designed to efficiently rule out branches thanks to the exploitation of the shared structure among features in a branch, as well as the correlation among dual variables for two sufficiently close points in the optimisation path. In cases where a branch cannot be ruled out, features in the branch are considered one by one to build the working set, which is very computationally expensive. In order to reduce this cost, we cast the problem as a variant of the Maximum Inner Product Search (MIPS) problem, which is described in Section 3.3. If no violating feature is identified then the algorithm has converged. Otherwise, a new candidate solution is obtained by solving problem (1) restricted to the features in the working set, and the process is repeated until no violating feature remains. While any solver can be used to solve the restricted problem, we implemented in WHInter a coordinate descent approach with safe pruning.

Algorithm 2 WHInter

Input: $\mathbf{X} \in \{0,1\}^{n \times p}$, $\mathbf{y} \in \mathbb{R}^n$, $\lambda_1 > \cdots > \lambda_T$.
Output: $(\mathcal{W}, \mathbf{w}^*_{\mathcal{W}}, b^*)_t$ for each λ_t

$\boldsymbol{\theta} \leftarrow -\nabla F(\mathbf{0}_n)$
for j in $[\![p]\!]$ **do**
  $\boldsymbol{\Theta}^{ref}_j \leftarrow \boldsymbol{\theta}$
end for
$\mathcal{W}, \mathbf{m}^{ref} \leftarrow$ `update_W`$(\mathbf{X}, \boldsymbol{\theta}, [\![p]\!], \lambda_1, \emptyset)$ ▷ See Section 3.3
for $t = 1$ to T **do**
  $\mathbf{w}^*_{\mathcal{W}}, b^* \leftarrow \underset{\mathbf{w}_{\mathcal{W}}, b}{\text{argmin}}\ P_{\mathbf{Z}_{\mathcal{W}}, \lambda_t}(\mathbf{w}_{\mathcal{W}}, b)$ ▷ Pre-Solve
  $\boldsymbol{\theta} \leftarrow -\nabla F(\mathbf{Z}_{\mathcal{W}} \mathbf{w}^*_{\mathcal{W}} + b^* \mathbf{1}_n)$.
  $\mathcal{W}, \boldsymbol{m}^{ref} \leftarrow$ `clean_W`$(\mathcal{W}, \lambda_t, \boldsymbol{\theta}, \boldsymbol{\Theta}^{ref}, \boldsymbol{m}^{ref})$
  while true **do**
    $\mathcal{V} \leftarrow \emptyset$ ▷ Identify violated branches
    for j in $[\![p]\!]$ **do**
      if $\eta(\mathbf{X}_j, \boldsymbol{\Theta}^{ref}_j, \boldsymbol{\theta}, \mathbf{m}^{ref}_j) > \lambda_t$ **then** ▷ See Section 3.2
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{j\}$
        $\boldsymbol{\Theta}^{ref}_j \leftarrow \boldsymbol{\theta}$
      end if
    end for
    $\mathcal{W}', \mathbf{m}^{ref}_{\mathcal{V}} \leftarrow$ `update_W`$(\mathbf{X}, \boldsymbol{\theta}, \mathcal{V}, \lambda_t, \mathcal{W})$ ▷ See Section 3.3
    if $\max_{i \in \mathcal{W}'} \left| \mathbf{Z}_i^\top \theta \right| \leq \lambda$ **then** Break **else** $\mathcal{W} \leftarrow \mathcal{W}'$
    $\mathbf{w}^*_{\mathcal{W}}, b^* \leftarrow \underset{\mathbf{w}_{\mathcal{W}}, b}{\text{argmin}}\ P_{\mathbf{Z}_{\mathcal{W}}, \lambda_t}(\mathbf{w}_{\mathcal{W}}, b)$ ▷ Solve subproblem
    $\boldsymbol{\theta} \leftarrow -\nabla F(\mathbf{Z}_{\mathcal{W}} \mathbf{w}^*_{\mathcal{W}} + b^* \mathbf{1}_n)$.
    $\mathcal{W}, \boldsymbol{m}^{ref} \leftarrow$ `clean_W`$(\mathcal{W}, \lambda_t, \boldsymbol{\theta}, \boldsymbol{\Theta}^{ref}, \boldsymbol{m}^{ref})$
  end while
  $(\mathcal{W}, \mathbf{w}^*_{\mathcal{W}}, b^*)_k \leftarrow (\mathcal{W}, \mathbf{w}^*_{\mathcal{W}}, b^*)$
end for

function `clean_W`$(\mathcal{W}, \lambda, \boldsymbol{\theta}, \boldsymbol{\Theta}^{ref}, \boldsymbol{m}^{ref})$
  for i in $\mathcal{W}$ **do**
    if $\left| \mathbf{Z}_i^\top \boldsymbol{\theta} \right| < \lambda$ **then**
      Remove $\{i\}$ from $\mathcal{W}$
      for b in branch(i) **do**
        if $\mathbf{m}^{ref}_b < \left| \mathbf{Z}_i^\top \boldsymbol{\Theta}^{ref}_b \right|$ **then** $\mathbf{m}^{ref}_b \leftarrow \left| \mathbf{Z}_i^\top \boldsymbol{\Theta}^{ref}_b \right|$
  return $\mathcal{W}$, $\boldsymbol{m}^{ref}$

3.2 The Branch bound η

As WHInter iterates, it produces candidate solutions $(\mathbf{w}^*, b^*)$ and corresponding dual variables $\boldsymbol{\theta}$ (lines 20 and 21 of Algorithm 2). For two sufficiently close iterations, or for two problems with sufficiently close regularisation parameters, the candidate solutions are likely to be *close* to one another, as well as the corresponding dual variables. WHInter exploits this intuition to speed up the identification of the working set from an iteration to another or from one problem to another. The following results relate the criteria used to identify the working set (line 3 of Algorithm 1) for two distinct dual variables.

Lemma 3.1. *For any* $\mathbf{X} \in \{0,1\}^{n\times p}$, $\mathbf{v} \in \mathbb{R}^n_+$, $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^n$, $j \in [\![p]\!]$, $\mathcal{I} \subset [\![p]\!]$ *and* $\alpha \in \mathbb{R}$, *the following holds:*

$$\max_{k\in\mathcal{I}} \left| \boldsymbol{\theta}_2^\top (\mathbf{v} \odot \mathbf{X}_k) \right| \leq |\alpha| \max_{k\in\mathcal{I}} \left| \boldsymbol{\theta}_1^\top (\mathbf{v} \odot \mathbf{X}_k) \right| + \zeta(\boldsymbol{\theta}_2 - \alpha\boldsymbol{\theta}_1, \mathbf{v}), \tag{5}$$

where

$$\forall (\mathbf{u}, \mathbf{v}) \in \mathbb{R}^n \times \mathbb{R}^n_+, \quad \zeta(\mathbf{u}, \mathbf{v}) = \max\left(\sum_{i:\mathbf{u}_i>0} \mathbf{u}_i\mathbf{v}_i, -\sum_{i:\mathbf{u}_i<0} \mathbf{u}_i\mathbf{v}_i \right).$$

The proof of Lemma 3.1 is postponed to Appendix A. It is based on the decomposition $\boldsymbol{\theta}_2 = \alpha\boldsymbol{\theta}_1 + (\boldsymbol{\theta}_2 - \alpha\boldsymbol{\theta}_1)$, and exploits the tree structure among features in a branch. To exploit Lemma 3.1 in WHInter, we define for $\alpha \in \mathbb{R}$ the function

$$\forall (\mathbf{v}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, m) \in \mathbb{R}^n_+ \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}, \quad \eta_\alpha(\mathbf{v}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, m) = |\alpha| m + \zeta(\boldsymbol{\theta}_2 - \alpha\boldsymbol{\theta}_1, \mathbf{v}), \tag{6}$$

and we maintain an active set $\mathcal{W} \subset [\![D]\!]$, a matrix $\boldsymbol{\Theta}^{ref} \in \mathbb{R}^{n\times p}$ that contains *reference dual variables* $\boldsymbol{\Theta}^{ref}_j \in \mathbb{R}^n$ for each branch $j \in [\![p]\!]$, and the vector $\mathbf{m}^{ref} \in \mathbb{R}^p$ defined by:

$$\forall j \in [\![p]\!], \quad \mathbf{m}^{ref}_j = \max_{k\in[\![p]\!]:\tau(j,k)\notin\mathcal{W}} \left| \mathbf{Z}^\top_{\tau(j,k)} \boldsymbol{\Theta}^{ref}_j \right|. \tag{7}$$

We now state our pruning theorem which allows to identify branches which are guaranteed to not contain any violating feature (line 13 of algorithm 2):

Theorem 3.1 (Branch pruning)**.** *For any* $\boldsymbol{\Theta}^{ref} \in \mathbb{R}^{n\times p}$, $\mathcal{W} \subset [\![p]\!]$, $j \in [\![p]\!]$, *let* $\mathbf{m}^{ref}_j \in \mathbb{R}_+$ *be given by (7). Then for any* $\boldsymbol{\theta} \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$ *and* $\lambda > 0$, *if*

$$\eta_\alpha\left(\mathbf{X}_j, \boldsymbol{\Theta}^{ref}_j, \boldsymbol{\theta}, \mathbf{m}^{ref}_j\right) < \lambda, \tag{8}$$

then any feature from branch j *that belongs to the working set* $\{i \in [\![D]\!] : \left|\mathbf{Z}_i^\top \boldsymbol{\theta}\right| \geq \lambda\}$ *is already in* $\mathcal{W}$. *This holds in particular if*

$$\eta_{min}\left(\mathbf{X}_j, \boldsymbol{\Theta}^{ref}_j, \boldsymbol{\theta}, \mathbf{m}^{ref}_j\right) := \min_{\alpha\in\mathbb{R}} \eta_\alpha\left(\mathbf{X}_j, \boldsymbol{\Theta}^{ref}_j, \boldsymbol{\theta}, \mathbf{m}^{ref}_j\right) < \lambda. \tag{9}$$

Proof. Take $\mathcal{I} = \{k \in [\![p]\!] : \tau(j,k) \notin \mathcal{W}\}$, $\mathbf{v} = X_j$, $\boldsymbol{\theta}_1 = \boldsymbol{\Theta}^{ref}_j$ and $\boldsymbol{\theta}_2 = \boldsymbol{\theta}$ in Lemma 3.1. Then if (8) holds, we deduce from (3.1) that

$$\max_{k\in[\![p]\!]:\tau(j,k)\notin\mathcal{W}} \left| \mathbf{Z}^\top_{\tau(j,k)} \boldsymbol{\theta} \right| < \lambda.$$

This shows that there is no feature i in branch j such that $\left|\mathbf{Z}_i^\top \boldsymbol{\theta}\right| \geq \lambda$ *and* i is not already in $\mathcal{W}$. The fact that for fixed arguments, the function $\alpha \to \eta_\alpha$ has a minimum $\alpha^* \in \mathbb{R}$ is shown in Appendix B, along with with an algorithm to compute it in $O\left(\|\mathbf{X}_j\|_0 \ln \|\mathbf{X}_j\|_0\right)$ operations. Since the statement is true for any α, it is *a fortiori* true for α^*. □

Theorem 3.1 provides criteria (8) and (9) that can be computed for each branch j, and which if satisfied allow to skip the search for violating variables in the branch. Importantly, the features that are already in the working set $\mathcal{W}$ are not taken into account to compute the criterion for a given branch. This subtlety allows to rule out branches even if they already contain features that were previously incorporated in the working set. Note that the reference dual variable for branch j, i.e, $\boldsymbol{\Theta}_j^{ref}$, is kept unchanged as long as branch j is pruned, and is otherwise updated to the latest dual variable (line 15 of Algorithm 2). As $\mathbf{m}_j^{ref}$ depends on the reference dual variable instead of the current one, it is solely reevaluated each time the reference residual is updated (line 18 of Algorithm 2) or when a feature from branch j leaves the working set (line 22 of Algorithm 2) .

Criterion (9) is the most stringent one, and therefore the most efficient one to prune branches, but it takes $O\left(\| \mathbf{X}_j \|_0 \ln \| \mathbf{X}_j \|_0\right)$ operations to compute. In order to balance computational complexity of the bound with its efficacy to prune branches, criterion (8) can be used as an alternative for a specific α value. One simple choice is to just take $\alpha = 1$, which leads to the criterion

$$\eta_1\left(\mathbf{X}_j, \boldsymbol{\Theta}_j^{ref}, \boldsymbol{\theta}, \mathbf{m}_j^{ref}\right) = \mathbf{m}_j^{ref} + \zeta\left(\boldsymbol{\Theta}_j^{ref} - \boldsymbol{\theta}, \mathbf{X}_j\right) < \lambda\,. \tag{10}$$

Alternatively, a simple heuristic to expect a more efficient pruning is to choose an α that minimises $\| \left(\boldsymbol{\theta} - \alpha\boldsymbol{\Theta}_j^{ref}\right) \odot \mathbf{X}_j \|_2$, i.e,

$$\alpha_{\ell_2} = \frac{\boldsymbol{\theta}^\top\left(\boldsymbol{\Theta}_j^{ref} \odot \mathbf{X}_j\right)}{\| \boldsymbol{\Theta}_j^{ref} \odot \mathbf{X}_j \|_2^2}\,. \tag{11}$$

$\eta_{\alpha_{\ell_2}}$ is expected to be more effective than η_1 since it is reasonable to expect that $\zeta\left(\boldsymbol{\theta} - \alpha_{\ell_2}\boldsymbol{\Theta}_j^{ref}, \mathbf{X}_j\right)$ is smaller than $\zeta\left(\boldsymbol{\theta} - \boldsymbol{\Theta}_j^{ref}, \mathbf{X}_j\right)$. Overall, computing criterion (9) for $\alpha = 1$ as in (10), or for $\alpha = \alpha_{\ell_2}$ as in (11), is an $O(\| \mathbf{X}_j \|)$ operation. Since computing $\zeta(\boldsymbol{\theta} - \alpha\boldsymbol{\Theta}_j^{ref}, \mathbf{X}_j)$ for a fixed α is also a $O(\| \mathbf{X}_j \|)$ computation, the total cost of identifying branch j as violated is $O(\| \mathbf{X}_j \|)$ for criterion (10), compared to $O\left(\| \mathbf{X}_j \|_0 \ln \| \mathbf{X}_j \|_0\right)$ for criterion (9). In Algorithm 2, the notation η refers to a user-defined function among $\eta_1, \eta_{\alpha_{\ell_2}}$ or η_{min}.

3.3 Updating the working set

When some branches $\mathcal{V} \subset [\![p]\!]$ cannot be pruned, the simultaneous updates of the working set $\mathcal{W}$ and of $\mathbf{m}_{\mathcal{V}}^{ref}$ requires scanning through all features in the branches $\mathcal{V}$ (lines 5 and 18 in Algorithm 2). In what follows we discuss strategies to make these updates efficient. For that purpose, let us first notice that:

$$\begin{aligned}
\forall j, k \in [\![p]\!], \left|\mathbf{Z}_{\tau(j,k)}^\top \boldsymbol{\theta}\right| &= \left|(\mathbf{X}_j \odot \mathbf{X}_k)^\top \boldsymbol{\theta}\right| \\
&= \left|(\mathbf{X}_j \odot \boldsymbol{\theta})^\top \mathbf{X}_k\right| \\
&= \left|\boldsymbol{Q}_j^\top \mathbf{X}_k\right|,
\end{aligned}$$

where for any $j \in [\![p]\!]$, $\boldsymbol{Q}_j = \boldsymbol{X}_j \odot \boldsymbol{\theta}$. This allows us to write the updates of $\mathcal{W}$ and $\boldsymbol{m}_{\mathcal{V}}^{ref}$ as:

$$\begin{cases}
\mathcal{W}' &= \mathcal{W} \cup \left\{\tau(j,k) : j \in \mathcal{V}, k \in [\![p]\!], \left|\boldsymbol{Q}_j^\top \boldsymbol{X}_k\right| \geq \lambda\right\}, \\
\boldsymbol{m}_j^{ref} &= \max_{k:\, |\boldsymbol{Q}_j^\top \boldsymbol{X}_k| < \lambda} \left|\boldsymbol{Q}_j^\top \boldsymbol{X}_k\right|, \forall j \in \mathcal{V}.
\end{cases} \tag{12}$$

This highlights the fact that the updates of the working set $\mathcal{W}$ and of $\boldsymbol{m}_{\mathcal{V}}^{ref}$ can be cast as particular variants of the Maximum Inner Product Search (MIPS) problem. MIPS aims at finding a vector in a database of probes which maximises the inner product with a given query vector. If we consider $\boldsymbol{X}$ as a set of probes, and $\boldsymbol{Q}_j$ as a query, then (12) is a variant of MIPS where (i) the set of probe vectors satisfies some constraints and is not known upfront and (ii) the problem is a maximum *absolute* inner product search. The update of $\mathcal{W}$ involves what is sometimes referred to as *above*-λ-MIPS problems where again, maximum *absolute* inner products are considered.

The interest of casting these updates as variants of MIPS problems is to exploit the ideas developed in the literature for solving these problems efficiently. Teflioudi and Gemulla (2016) and Fontoura et al. (2011) give good overviews of MIPS solvers developed for recommender systems and information retrieval applications respectively. In both cases, the proposed methods rely on two main ideas: (i) adequate indexing techniques or data structures and (ii) pruning criteria which allow to not compute all inner products entirely. Since none of these methods can directly be applied to problem (12) because of its specificities, we propose an appropriate algorithm based on a simple inverted index approach, which we will refer to as IL, and which exploits the sparsity of the problem. Another option would be to leverage pruning techniques. We detail such an attempt in Appendix C. However, since our preliminary results with the pruning technique were not conclusive compared to IL on the simulated and real data, we will only focus on the inverted index approach below.

Algorithm 3 `update_W`

Input: $\mathbf{X} \in \{0,1\}^{n \times p}$, $\boldsymbol{\theta} \in \mathbb{R}^n$, $\mathcal{Q} \subset [\![p]\!]$, $\lambda \in \mathbb{R}$, $\mathcal{W} \subset [\![D]\!]$
Output: $\mathcal{W}$, $\mathbf{m}^{ref}$

for $j \in \mathcal{Q}$ **do**
  Initialise an array $\mathbf{a}$ of size p to zero.
  for each i in supp($\mathbf{X}_j$) **do**
    for each k in supp($\boldsymbol{x}_i$) **do**
      $\mathbf{a}_k = \mathbf{a}_k + \boldsymbol{\theta}_i$
    end for
  end for
  for each k s.t. $\mathbf{a}_k \neq 0$ **do**
    if $\mathbf{m}_j^{ref} < \mathbf{a}_k < \lambda$ **then** set $\mathbf{m}_j^{ref} = \mathbf{a}_k$
    if $\mathbf{a}_k \geq \lambda$ and $\tau(j,k) \notin \mathcal{W}$ **then** add $\tau(j,k)$ to $\mathcal{W}$
  end for
end for
return $\mathcal{W}, \boldsymbol{m}^{ref}$

IL is detailed in Algorithm 3. The inverted indices consist of n lists, one for each dimension, where each list supp($\boldsymbol{x}_i$) records the indices of the features in $\mathbf{X}$ which have a non-zero element for the i^{th} dimension. These inverted lists can be computed once for all when WHInter starts and be reused for all MIPS problems, and therefore building the inverted lists requires a negligible additional computational cost. Algorithm (3) computes inner product following a *term-at-a-time* (TAAT) scheme (Fontoura et al., 2011), i.e, the inner products are accumulated simultaneously across probes and the contribution of the i^{th} dimension to the inner products is entirely processed before moving to the next one.

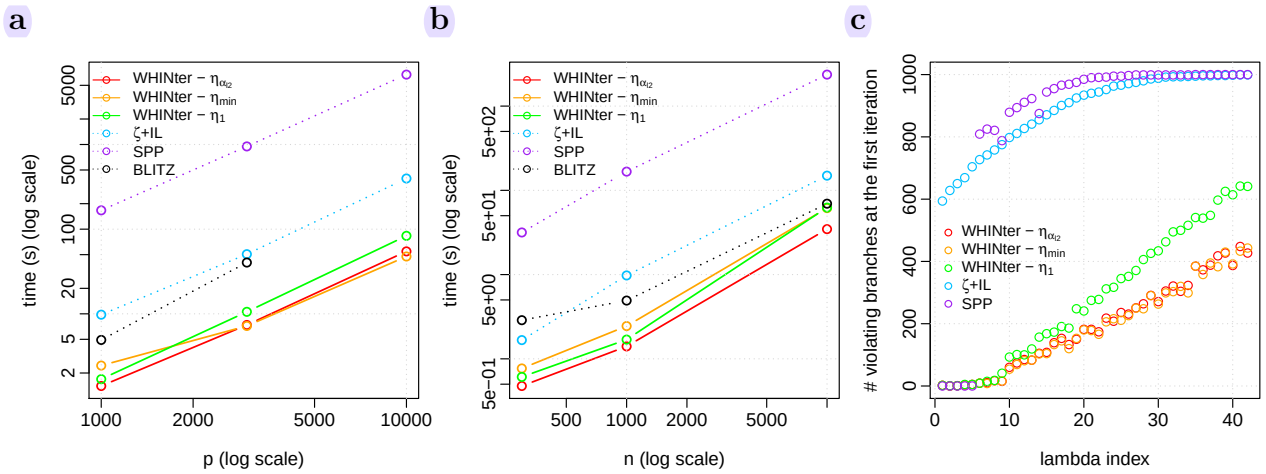

Figure 2 – **Performance comparison on simulated data for an entire regularisation path**. Comparison of WHInter with three branch pruning criteria $\eta \in \{\eta_{\alpha_2}, \eta_{min}, \eta_1\}$ to $\zeta + IL$, SPP and BLITZ. **(a)** Time in seconds for $n = 1 \times 10^3$ fixed and p varied. **(b)** Time in seconds for $p = 1 \times 10^3$ fixed and n varied. **(c)** number of branches that are *not* pruned at the first iteration, as a function of λ, for $n = p = 1 \times 10^3$.

4 Simulation study

We first test the performances of WHInter on synthetic LASSO datasets. We assess the performances of the different branch pruning bounds presented in 3.2, i.e, η_{min}, η_1 and $\eta_{\alpha_{\ell_2}}$, and further compare WHInter to a working set method that uses the bound $\zeta(\boldsymbol{\theta}, \mathbf{X}_j)$ instead of η_α, but is otherwise equivalent to WHInter. We refer to this method as $\zeta + IL$. It is expected to prune less branches than WHInter but does not require to maintain $\boldsymbol{m}^{ref}$. We also compare WHInter to SPP (Nakagawa et al., 2016) and BLITZ (Johnson and Guestrin, 2015). In our experiments, we use a slightly modified, more efficient version of the code provided by the authors of SPP (cf Appendix D). As for BLITZ, since the method is not tailored for interaction problems, we first compute the matrix $\boldsymbol{Z}$ which is fed as input to BLITZ. For this reason we could not solve problems when p is too large (e.g., $p = 1 \times 10^4$ in the simulations) since, even in sparse format, storing $\boldsymbol{Z}$ requires too much memory. Importantly, the performances reported for BLITZ do not include the time required to compute $\boldsymbol{Z}$ from $\boldsymbol{X}$, which clearly advantages BLITZ compared to the other methods.

We simulate five datasets $\mathbf{X} \in \{0,1\}^{n \times p}$ with varying number of features and samples: three datasets with $p = 1 \times 10^3$ fixed and $n \in \{3 \times 10^2, 1 \times 10^3, 1 \times 10^4\}$, and two more with $n = 1 \times 10^3$ fixed and $p \in \{3 \times 10^3, 1 \times 10^4\}$. The features are drawn from a Bernoulli distribution with parameter $q \in [0.1, 0.5]$ itself drawn from a uniform distribution $\mathcal{U}_{[0.1,0.5]}$. We then randomly pick a set $\mathcal{S}$ of 100 features among the main effects and interactions and compute the response as $\mathbf{y} = \mathbf{Z}_{\mathcal{S}} \mathbf{w}^*_{\mathcal{S}}$ where $\mathbf{w}^*_{\mathcal{S}} \sim \mathcal{N}(\mathbf{0}_{|\mathcal{S}|}, I_{|\mathcal{S}|})$. In all experiments, the LASSO is solved for a sequence $(\lambda_t)_{t \in [\![T]\!]}$, $T = 100$, logarithmically spaced between λ_{max} and $\max(0.01\lambda_{max}, \lambda')$ where λ_{max} is the largest value of λ for which at least one feature is selected, and λ' is the first λ_i for which 150 features or more are selected in the model. For all methods, the time to compute λ_{max} is included in the total time required to solve the regularisation path. In WHInter, λ_{max} can easily be deduced from the initialisation of $\mathbf{m}^{ref}$ since $\lambda_{max} = \max_{j \in [\![p]\!]} \boldsymbol{m}^{ref}_j$. All algorithms are implemented in C++ and compiled with the `-O3` optimisation flag. The experiments are run on a 64-bit machine with Intel Core i7 Processor 2.5 GHz, 16GB of memory and 6MB of cache.

Results are shown in Figure 2. For $n = 1 \times 10^3$ (Figure 2a), LASSO solutions are computed for

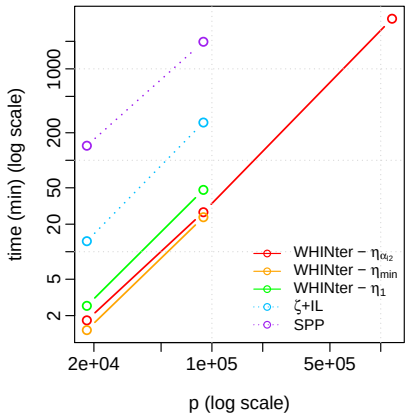

Figure 3 – Performance comparison on SNPs data for an entire regularisation path. The y-axis reports the total time (in minutes) required to compute the LASSO path for chromosome 22 (around 20,000 SNPs), chromosome 1 (around 90,000 SNPs) and the whole genome (around 1.2 million SNPs).

42, 32 and 28 values of λ for $p = 1 \times 10^3, p = 3 \times 10^3$ and $p = 1 \times 10^4$ respectively. In these cases smaller values of λ result in model sizes exceeding 150 features. For the remaining settings where $p = 1 \times 10^3$ and $n = 3 \times 10^2$ or $n = 1 \times 10^4$ (Figure 2b), LASSO solutions are computed for 34 and all 100 values of λ between λ_{max} and $0.01\lambda_{max}$, respectively. We checked that all methods return the exact same support.

In all settings, WHInter is the fastest method. Its better performance compared to $\zeta + IL$ highlights the benefit of using reference dual variables even if it implies to maintain $\boldsymbol{m}^{ref}$. The results also show the importance of α, since WHInter with η_{ℓ_2} is always better ($\times 1.2$ to $\times 1.8$) than WHInter with η_1 for example. Figure 2c confirms that the choice of α has an impact on the pruning efficiency and consequently on the performance. It shows, however, that on this experiment η_{min} does not allow to prune many more branches than η_{ℓ_2}. This explains why η_{ℓ_2} tends to outperform η_{min}, notably for large n, since the higher computational complexity of η_{min} does not sufficiently enhance the pruning. We also notice that SPP is the slowest algorithm, and in particular $\zeta + IL$ is $\times 17$ faster than SPP on average. This speed-up is mostly explained by the fact that $\zeta + IL$ relies on inverted lists to update the working set while SPP identifies the safe set naively. Overall, WHInter offers a signifiant speed-up of two orders of magnitude or more compared to its safe screening counterpart.

5 Results on real world data

We now illustrate the performance of the different algorithms on a real-world problem, where we want to predict the cytotoxic response of 884 lymphoblastoid cell lines split into a train ($n = 620$) and a test ($n = 264$) set, and characterized by about 1.2×10^6 single nucleotide polymorphisms (SNP) that represent their genotypes. The data was released as part of the Dialogue on Reverse Engineering Assessment and Methods 8 (DREAM 8) toxicogenetics challenge (Eduati et al., 2015). We encode the SNP data as a binary matrix were 1 stand for the presence of a minor allele on one or both copies of the chromosomes. As preprocessing we removed SNP with less than 5% of 1's and corrected the data for population structure as in Price et al. (2006). To focus on problems of increasing scales, we first considered the SNPs of the smallest chromosome only (chr. 22), then of the largest only (chr. 1) and finally of all chromosomes together. This leads to train matrices

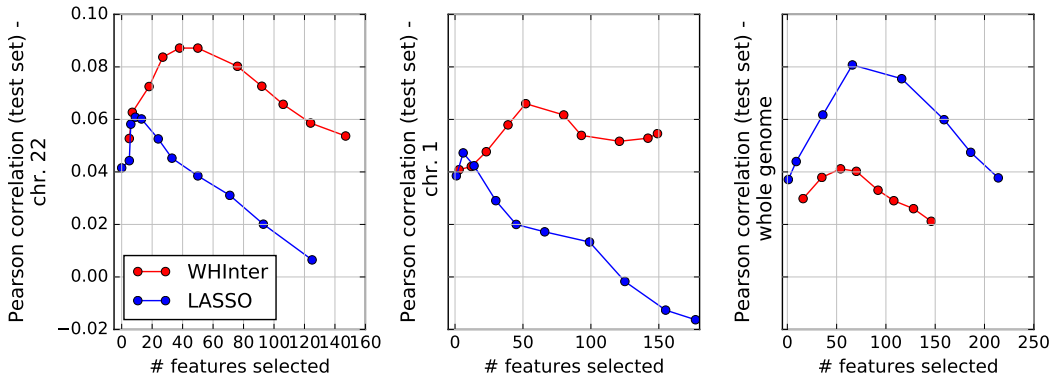

Figure 4 – Predictive performance on the test set. The y-axis reports the pearson correlation between the true and predicted response. The x-axis reports the number of selected features for the sequence of regularisation parameters tested.

with $n = 620$ and $p = 18,168$ SNPs for chromosome 22, $p = 89,027$ SNPs for chromosome 1 and $p = 1,166,836$ SNPs for the whole genome. We consider a sequence of 100 regularisation parameters λ logarithmically spaced between λ_{max} and $0.01\lambda_{max}$, and by default stop computations as soon as 150 features or more are selected. This occurs after the 12^{th}, the 11^{th} and the 9^{th} value of λ for chromosome 22, chromosome 1 and all chromosomes respectively. The time required to compute the regularisation paths are shown in Fig. 3.

The relative performances of the methods are the same as for the simulations. $\eta_{\alpha_{\ell_2}}$ provides a $\times 1.4$ (resp. $\times 1.8$) speed up compared to using η_1 for chromosome 22 (resp. chr. 1). and compared to SPP, there is a $\times 81$ (resp. $\times 73$) speed up for chromosome 22 (resp chr. 1). In the case of the whole genome, we only ran WHInter with $\eta_{\alpha_{\ell_2}}$ which takes two days and a half. While this can seem a lot, we recall that this corresponds to a problem with roughly 680 *billion* features. We did not run other methods on the whole genome since most of them are expected to take too long.

Out of curiosity, we also obtained preliminary results concerning the predictive performance of WHInter compared to a LASSO with no interactions on such high-dimensional problems. The results, presented in Figure 4 , suggest that interactions are relevant predictors for this data. For the chromosomes 1 and 22 taken independently, the predictive accuracy of WHInter is better than that of the simple LASSO for almost every value of λ. By contrast, for the whole genome, the LASSO clearly performs better, which may underline statistical issues due to the huge number of variables in this case (Donoho and Tanner, 2009).

6 Discussion

We presented WHInter, a working set algorithm designed to solve large scale LASSO problems with interaction terms. WHInter implements a new branch pruning bound to efficiently delineate the working set among the many possible interaction variables, and a variant of MIPS solver that provides a further speed up. We showed that WHInter is up to two orders of magnitudes faster than competing approaches. While we presented WHInter for binary data, it could also be used for data rescaled in $[0, 1]$, provided that an appropriate solver is picked for the MIPS problems. As for future work, one could exploit the recent works on approximate MIPS (Shrivastava and Li, 2014; Teflioudi and Gemulla, 2016) to obtain an additional speed up for the computationally intensive updates, and possibly rely on recent post selection-inference (Suzumura et al., 2017) frameworks to characterise the approximate solution obtained.

Acknowledgements

We thank Nino Shervashidze for helpful discussions.

Annexes

A Proof of Lemma 3.1

Lemma 3.1. *For any* $\mathbf{X} \in \{0,1\}^{n\times p}$, $\mathbf{v} \in \mathbb{R}^n_+$, $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^n$, $j \in [\![p]\!]$, $\mathcal{I} \subset [\![p]\!]$ *and* $\alpha \in \mathbb{R}$, *the following holds:*

$$\max_{k\in\mathcal{I}} \left| \boldsymbol{\theta}_2^\top (\mathbf{v} \odot \mathbf{X}_k) \right| \leq |\,\alpha\,| \max_{k\in\mathcal{I}} \left| \boldsymbol{\theta}_1^\top (\mathbf{v} \odot \mathbf{X}_k) \right| + \zeta(\boldsymbol{\theta}_2 - \alpha\boldsymbol{\theta}_1, \mathbf{v})\,,$$

where

$$\forall (\mathbf{u}, \mathbf{v}) \in \mathbb{R}^n \times \mathbb{R}^n_+\,, \quad \zeta(\mathbf{u}, \mathbf{v}) = \max\left(\sum_{i:\mathbf{u}_i>0} \mathbf{u}_i\mathbf{v}_i, -\sum_{i:\mathbf{u}_i<0} \mathbf{u}_i\mathbf{v}_i \right).$$

Proof. With the notations of Lemma 3.1 , we have:

$$\begin{aligned}
\max_{k\in\mathcal{I}} \left| \boldsymbol{\theta}_2^\top (\mathbf{v} \odot \mathbf{X}_k) \right| &\leq \max_{k\in\mathcal{I}} \left| \alpha\boldsymbol{\theta}_1^\top (\mathbf{v} \odot \mathbf{X}_k) + (\boldsymbol{\theta}_2 - \alpha\boldsymbol{\theta}_1)^\top (\mathbf{v} \odot \mathbf{X}_k) \right| \\
&\leq |\,\alpha\,| \max_{k\in\mathcal{I}} \left| \boldsymbol{\theta}_1^\top (\mathbf{v} \odot \mathbf{X}_k) \right| + \max_{k\in\mathcal{I}} \left| (\boldsymbol{\theta}_2 - \alpha\boldsymbol{\theta}_1)^\top (\mathbf{v} \odot \mathbf{X}_k) \right| \\
&\leq |\,\alpha\,| \max_{k\in\mathcal{I}} \left| \boldsymbol{\theta}_1^\top (\mathbf{v} \odot \mathbf{X}_k) \right| + \max_{\mathbf{X}\in\{0,1\}^n} \left| (\boldsymbol{\theta}_2 - \alpha\boldsymbol{\theta}_1)^\top (\mathbf{v} \odot \mathbf{X}) \right| \\
&= |\,\alpha\,| \max_{k\in\mathcal{I}} \left| \boldsymbol{\theta}_1^\top (\mathbf{v} \odot \mathbf{X}_k) \right| + \zeta(\boldsymbol{\theta}_2 - \alpha\boldsymbol{\theta}_1, \mathbf{v})\,.
\end{aligned}$$

□

B Computing η_{min}

In this section we characterise the existence and an algorithm to compute, for any fixed $(\mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\theta}', m) \in \mathbb{R}^n_+ \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}$:

$$\eta_{min}(\mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\theta}', m) := \min_{\alpha\in\mathbb{R}} \eta_\alpha(\mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\theta}', m)\,, \tag{S1}$$

where η_α is defined in Section 3.2. For that purpose, let us introduce for any $\alpha \in \mathbb{R}$ the functions:

$$\begin{cases}
\gamma_p(\alpha) &= \displaystyle\sum_{i:\boldsymbol{\theta}'_i - \alpha\boldsymbol{\theta}_i > 0} \mathbf{v}_i (\boldsymbol{\theta}'_i - \alpha\boldsymbol{\theta}_i)\,, \\
\gamma_m(\alpha) &= \displaystyle\sum_{i:\boldsymbol{\theta}'_i - \alpha\boldsymbol{\theta}_i < 0} \mathbf{v}_i (\boldsymbol{\theta}'_i - \alpha\boldsymbol{\theta}_i)\,,
\end{cases}$$

such that:

$$\eta_\alpha(\mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\theta}', m) = |\alpha|\, m + \max\left(\gamma_p(\alpha), -\gamma_m(\alpha)\right). \tag{S2}$$

Let us first characterise the existence and properties of the solution to the minimisation problem (S1).

Theorem B.1. *For any* $(\mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\theta}', m) \in \mathbb{R}^n_+ \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}$, *the function*

$$\alpha \in \mathbb{R} \to \eta_\alpha(\mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\theta}', m)$$

is continuous, piecewise affine, convex and nonnegative. It reaches at least a minimum at a value $\alpha^* \in \mathcal{B}$ *where*

$$\mathcal{B} = \{0\} \cup \left\{ \frac{\boldsymbol{\theta}'_i}{\boldsymbol{\theta}_i} \,:\, i \in \mathit{supp}(\boldsymbol{\theta}) \cap \mathit{supp}(\mathbf{v}) \right\} \cup \{\alpha \in \mathbb{R} : \gamma_p(\alpha) = \gamma_m(\alpha)\} \,.$$

Proof. For any $i \in [\![n]\!]$, let

$$\forall \alpha \in \mathbb{R}\,, \quad \phi_i(\alpha) = \mathbf{v}_i \max\left(0, \boldsymbol{\theta}'_i - \alpha \boldsymbol{\theta}_i\right) \,.$$

Since $\mathbf{v}_i \geq 0$, $\phi_i(\alpha) = \mathbf{v}_i \max(0, \boldsymbol{\theta}'_i - \alpha\boldsymbol{\theta}_i)$ is continuous, piecewise affine, convex and nonnegative. It has a single breakpoint at $\alpha_i = \boldsymbol{\theta}'_i/\boldsymbol{\theta}_i$ if $\boldsymbol{\theta}_i \neq 0$ and $\mathbf{v}_i > 0$, and is constant otherwise. Since $\gamma_p(\alpha) = \sum_{i=1}^n \phi_i(\alpha)$, γ_p is also continuous, piecewise affine, convex and nonnegative with breakpoints in $\{\boldsymbol{\theta}'_i/\boldsymbol{\theta}_i \,:\, i \in \operatorname{supp}(\boldsymbol{\theta}) \cup \operatorname{supp}(\mathbf{v})\}$. Taking $\psi_i(\alpha) = \mathbf{v}_i \max(0, \alpha\boldsymbol{\theta}_i - \boldsymbol{\theta}'_i)$ shows similarly that $-\gamma_m(\alpha) = \sum_{i=1}^n \psi_i(\alpha)$ has the same properties. Consequently, the function $\alpha \mapsto \max(\gamma_p(\alpha), -\gamma_m(\alpha))$ is also continuous, piecewise affine, convex and nonnegative, with possible breakpoints in

$$\left\{\boldsymbol{\theta}'_i/\boldsymbol{\theta}_i \,:\, i \in \operatorname{supp}(\boldsymbol{\theta}) \cup \operatorname{supp}(\mathbf{v})\right\} \cup \{\alpha \in \mathbb{R} : \gamma_p(\alpha) = \gamma_m(\alpha)\} \,.$$

Since $\alpha \to |\,\alpha\,|$ is also continuous, piecewise affine, convex and nonnegative, and has a breakpoint for $\alpha = 0$, Theorem B.1 follows by observing that a continuous, piecewise affine, convex and nonnegative function necessarily reaches a minimum at one of its breakpoints. □

Let $S = |\operatorname{supp}(\boldsymbol{\theta}) \cap \operatorname{supp}(\mathbf{v})|$. Theorem B.1 shows that it suffices to compute the values of η_α on at most $S+2$ values for α to find the global minimum. However, a naive computation of η_α using (S2) takes $O(|\operatorname{supp}(\mathbf{v})|)$ for each α, hence a total complexity $O(S \times |\operatorname{supp}(\mathbf{v})|)$ to find the minimum of η_α.

This can be improved to $O(|\operatorname{supp}(\mathbf{v})| + S \ln S)$ by first sorting the $S+1$ breakpoints $b_i = \boldsymbol{\theta}'_i/\boldsymbol{\theta}_i$ for $i \in \operatorname{supp}(\boldsymbol{\theta}) \cap \operatorname{supp}(\mathbf{v})$ and $b_{S+1} = 0$ in increasing order:

$$b_{\pi(1)} \leq b_{\pi(2)} \leq \ldots \leq b_{\pi(S+1)},$$

which takes $O(S \ln S)$ time. Adding by convention $b_{\pi(0)} = -\infty$ we observe that on each interval $(b_{k-1}, b_k]$ the functions γ_p and γ_m are affine, of the form:

$$\forall \alpha \in (b_{k-1}, b_k]\,, \quad \begin{cases} \gamma_p(\alpha) & = s_p^k - \alpha t_p^k \,, \\ -\gamma_m(\alpha) & = s_m^k - \alpha t_m^k \,. \end{cases}$$

From the properties of $\gamma_p(\alpha) = \sum_{i=1}^n \phi_i(\alpha)$ and $-\gamma_m(\alpha) = \sum_{i=1}^n \psi_i(\alpha)$, we get the coefficients for $k = 1$, i.e., for the interval $(-\infty, b_{\pi(1)}]$ in $O(|\operatorname{supp}(\mathbf{v})|)$ as follows:

$$\begin{cases} s_p^1 & = \sum_{i \in \operatorname{supp}(\mathbf{v}) : \boldsymbol{\theta}_i > 0} \mathbf{v}_i \boldsymbol{\theta}'_i + \sum_{i \in \operatorname{supp}(\mathbf{v}) : \boldsymbol{\theta}_i = 0} \mathbf{v}_i \max(0, \boldsymbol{\theta}'_i) \,, \\ t_p^1 & = \sum_{i \in \operatorname{supp}(\mathbf{v}) : \boldsymbol{\theta}_i > 0} \mathbf{v}_i \boldsymbol{\theta}_i \,, \\ s_m^1 & = -\sum_{i \in \operatorname{supp}(\mathbf{v}) : \boldsymbol{\theta}_i < 0} \mathbf{v}_i \boldsymbol{\theta}'_i + \sum_{i \in \operatorname{supp}(\mathbf{v}) : \boldsymbol{\theta}_i = 0} \mathbf{v}_i \max(0, -\boldsymbol{\theta}'_i) \,, \\ t_m^1 & = \sum_{i \in \operatorname{supp}(\mathbf{v}) : \boldsymbol{\theta}_i < 0} \mathbf{v}_i \boldsymbol{\theta}_i \,. \end{cases} \tag{S3}$$

This allows in particular to compute $\gamma_p(b_{\pi(1)})$, $\gamma_m(b_{\pi(1)})$, and therefore $\eta_{b_{\pi(1)}}$ from (S2). We can then iteratively compute the coefficients for $k+1$ from the coefficients for k in $O(1)$ only, by observing that between the intervals $(b_{k-1}, b_k]$ and $(b_k, b_{k+1}]$, the only change in slope and intercept of γ_p is due to the function $\phi_{\pi^{-1}(k)}$, when $\pi^{-1}(k) \neq S+1$. Let $i = \pi^{-1}(k)$. When $\boldsymbol{\theta}_i > 0$, the slope of ϕ_i increases by $\mathbf{v}_i\boldsymbol{\theta}_i$ and its intercept decreases by $\mathbf{v}_i\boldsymbol{\theta}'_i$ at b_i. When $\boldsymbol{\theta}_i > 0$, its slope increases by $-\mathbf{v}_i\boldsymbol{\theta}_i$ and its intercept increases by $\mathbf{v}_i\boldsymbol{\theta}'_i$. This translates into the following recursive formula for the coefficients of γ_p:

$$s_p^{k+1} = \begin{cases} s_p^k - \mathbf{v}_i\boldsymbol{\theta}'_i & \text{if } \boldsymbol{\theta}_i > 0\,, \\ s_p^k + \mathbf{v}_i\boldsymbol{\theta}'_i & \text{if } \boldsymbol{\theta}_i < 0\,, \end{cases}$$

and

$$t_p^{k+1} = t_p^k - \mathbf{v}_i|\boldsymbol{\theta}_i|\,.$$

A similar analysis on γ_m leads to the following recursion:

$$s_m^{k+1} = \begin{cases} s_m^k - \mathbf{v}_i\boldsymbol{\theta}'_i & \text{if } \boldsymbol{\theta}_i > 0\,, \\ s_m^k + \mathbf{v}_i\boldsymbol{\theta}'_i & \text{if } \boldsymbol{\theta}_i < 0\,, \end{cases}$$

and

$$t_m^{k+1} = t_m^k - \mathbf{v}_i|\boldsymbol{\theta}_i|\,.$$

We can thus iteratively compute the coefficients on each interval, and thus the values of η_α on each breakpoint, with complexity $O(1)$ per breakpoint. Since $\alpha \mapsto \eta_\alpha$ is convex, we stop at the first k such that $\eta_{b_{\pi(k+1)}} \geq \eta_{b_{\pi(k)}}$. From the equations of γ_p and γ_m on $(b_{\pi(k)}, b_{\pi(k+1)}]$ we can additionally check if there is a crossing point $\bar{\alpha} \in (b_{\pi(k)}, b_{\pi(k+1)}]$ such that $\gamma_p(\bar{\alpha}) = \gamma_m(\bar{\alpha})$, in which case we also compute $\eta_{\bar{\alpha}}$. The global minimum of $\alpha \mapsto \eta_\alpha$ is then $\min(\eta_{b_{\pi(k)}}, \eta_{\bar{\alpha}})$.

The overall algorithm is detailed in Algorithm S1.

C Alternative solver for working set updates

In this section, we present an alternative solver to the inverted list approach (algorithm 3 in section 3.3), which we call $MIPS1$, to compute the working set updates (12). It relies on a pruning technique and does not require storing extra indices for the data. The main idea of this alternative approach is to compute inner products on a progressively growing subset of dimensions, and to maintain an upper-bound on the maximum attainable score on the remaining dimensions. This allows to discard a probe as soon as its maximum attainable score drops below the maximum score achieved so far without computing the inner product in its entirety. Algorithm S2 presents the procedure in details. It takes as input $\mathcal{Q}$ which contains the indices that define the queries of interest and outputs the updated working set $\mathcal{W}$ and $\mathbf{m}^{ref}$. For each query, we start by precomputing the partial inner product bounds $\mathbf{r}^+ \in \mathbb{R}^n$ and $\mathbf{r}^- \in \mathbb{R}^n$, where $\mathbf{r}_i^+$ and $\mathbf{r}_i^-$ are respectively the maximum and minimum attainable inner products between the query and any probe in the database on the dimensions from $i+1$ to n. Formally, $\mathbf{r}^+$ and $\mathbf{r}^-$ are defined for a given query j by:

$$\forall i \in [\![n]\!]\,, r_i^+ = \sum_{m>i;\ \boldsymbol{\theta}_m>0} \mathbf{X}_{mj}\boldsymbol{\theta}_m \tag{S4}$$

$$\forall i \in [\![n]\!]\,, r_i^- = \sum_{m>i;\ \boldsymbol{\theta}_m<0} \mathbf{X}_{mj}\boldsymbol{\theta}_m \tag{S5}$$

Algorithm S1 Minimise η in α

Input: $(\mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\theta}', m) \in \mathbb{R}_+^n \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}$.

Output: $\eta_{min}(\mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\theta}', m)$

$S \leftarrow$ indices in $\text{supp}(\mathbf{v}) \cap \text{supp}(\boldsymbol{\theta})$
$N \leftarrow \text{length}(S)$
$v \leftarrow \left[0, \frac{\boldsymbol{\theta}'_{S[1]}}{\boldsymbol{\theta}_{S[1]}}, \ldots, \frac{\boldsymbol{\theta}'_{S[N]}}{\boldsymbol{\theta}_{S[N]}}\right]$
$\texttt{ind} \leftarrow [\texttt{none}, S[1], \ldots, S[N]]$
$\texttt{rank} \leftarrow \text{sort}(v)$ (in increasing order)
$v \leftarrow v[\texttt{rank}]$; $\texttt{ind} \leftarrow \texttt{ind}[\texttt{rank}]$
Initialise s_p, s_m, t_p, t_m via (S3)
$\texttt{min} \leftarrow +\infty$
for i in $1 \ldots N+1$ **do**
  $\texttt{newmin} \leftarrow |v[i]|\, m + \max\left(s_p - v[i]t_p,\ s_m - v[i]t_m\right)$
  if $\texttt{newmin} < \texttt{min}$ **then**
    $\texttt{min} \leftarrow \texttt{newmin}$
    if $\texttt{ind}[i] \neq \texttt{none}$ **then**
      $t_p \leftarrow t_p - \mathbf{v}_{\texttt{ind}[i]} \left| \boldsymbol{\theta}_{\texttt{ind}[i]} \right|$
      $t_m \leftarrow t_m - \mathbf{v}_{\texttt{ind}[i]} \left| \boldsymbol{\theta}_{\texttt{ind}[i]} \right|$
      if $\boldsymbol{\theta}_{\texttt{ind}[i]} > 0$ **then**
        $s_p \leftarrow s_p - \mathbf{v}_{\texttt{ind}[i]} \boldsymbol{\theta}'_{\texttt{ind}[i]}$
        $s_m \leftarrow s_m - \mathbf{v}_{\texttt{ind}[i]} \boldsymbol{\theta}'_{\texttt{ind}[i]}$
      else
        $s_p \leftarrow s_p + \mathbf{v}_{\texttt{ind}[i]} \boldsymbol{\theta}'_{\texttt{ind}[i]}$
        $s_m \leftarrow s_m + \mathbf{v}_{\texttt{ind}[i]} \boldsymbol{\theta}'_{\texttt{ind}[i]}$
      end if
    end if
  else
    Check if there exists $\bar{\alpha} \in [v[i-1], v[i]]$ s.t. $\gamma_p(\bar{\alpha}) = \gamma_m(\bar{\alpha})$
    Return min($\texttt{newmin}$, $\eta(\alpha^{intersection})$)
  end if
end for

and provide an upper bound on inner products with the query $\mathbf{X}_j \odot \boldsymbol{\theta}$ as follows:

$$\begin{aligned}\forall k \in [\![p]\!],\quad (\mathbf{X}_j \odot \boldsymbol{\theta})^\top \mathbf{X}_k &= \sum_{m \leq i} \mathbf{X}_{mj}\boldsymbol{\theta}_m \mathbf{X}_{mk} + \sum_{m > i} \mathbf{X}_{mj}\boldsymbol{\theta}_m \mathbf{X}_{mk} \\ &\leq \sum_{m \leq i} \mathbf{X}_{mj}\boldsymbol{\theta}_m \mathbf{X}_{mk} + \sum_{m > i;\, \boldsymbol{\theta}_m > 0} \mathbf{X}_{mj}\boldsymbol{\theta}_m \\ &= \sum_{m \leq i} \mathbf{X}_{mj}\boldsymbol{\theta}_m \mathbf{X}_{mk} + r_i^+\end{aligned}$$

The bound involving $\mathbf{r}^-$ can be obtained analogously. These bounds simply assume there is a probe vector which has ones in front of every positive entry of the query and none in front of its negative entries, or the reverse. Once these bounds have been precomputed, the inner product between the query and a probe is computed up to a certain dimension, and every $n_c \in \mathbb{N}$ dimensions we check whether there is a possibility that the inner product being computed becomes larger than the current maximum, or larger than λ. If it is impossible, then the probe can be safely discarded and the algorithm proceeds with the next probe. If not, the inner product is computed on n_c more dimensions and a new check is performed. For all our simulations and real data experiments, we set n_c to a default of 20. If a probe cannot be discarded then the algorithm updates when appropriate the active set $\mathcal{W}$ and/or the current maximum absolute inner product obtained $\mathbf{m}_j^{ref}$. For the pruning to be effective, we reorder the dimensions $1 \dots n$ so that queries are sorted in decreasing order in absolute value. As a consequence, the partial inner product bounds $\mathbf{r}_i^+$ and $\mathbf{r}_i^-$ are computed with the $n - i$ smallest entries in absolute value of the queries which makes them tighter than with any other ordering of the dimensions.

We now compare $MIPS1$ to its naive counterpart (which we will call $Naive$ from now on) on several benchmark datasets in order to assess the speed-up obtained with the pruning. To be more specific, $Naive$ is implemented similarly to $MIPS1$ except the lines specific to pruning, i.e., lines 5, 12 and 13 in Algorithm S2, are removed. The benchmark datasets we use are designed in such a way that the pruning rate achievable varies. To do this, we simulate a matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, with $n = p = 1000$, where the features are drawn from a Bernoulli distribution, whose parameter is itself drown from a uniform distribution $\mathcal{U}_{[0.1,0.5]}$. Then $\boldsymbol{\theta} \in \mathbb{R}^n$ is built in such a way that the cumulative sum of the vectors obtained by sorting $\boldsymbol{\theta}_{|\boldsymbol{\theta} \geq 0}$ and $|\boldsymbol{\theta}_{|\boldsymbol{\theta} < 0}|$ follows the function $f(x) = \frac{1}{1-e^{-\mu}}\left(1 - e^{-\mu x}\right), x \in \{0, 1\}$ for a given parameter $\mu \in \mathbb{R}^+$. The area under this cumulative sum, which is $\kappa(\mu) = \frac{1}{1-e^{-\mu}} - \frac{1}{\mu} \in [0.5, 1]$, characterises the different vectors $\boldsymbol{\theta}_\kappa$ obtained with different values of μ. Figure S1a shows how the cumulative sums are modified with μ. The interest of simulating different $\boldsymbol{\theta}_\kappa$ is that the rate of pruning achievable increases with κ: the closer κ is to 1, the higher the pruning rate. In the experiments presented hereafter, all p features were taken as queries, i.e., $\mathcal{Q} = [\![p]\!]$, and we took $\lambda = +\infty$ and $\mathcal{W} = \emptyset$. The results are presented in Figure S1b. The pruning rate, which we define as the average number of non-zero coordinates of the queries which were pruned out of their total number of non-zero coordinates, widely varies from 8% for $\kappa = 0.55$ to 84% for $\kappa = 0.95$. Moreover, the speed-up obtained with $MIPS1$ compared to $Naive$ is almost equal to 1 minus the pruning rate. That means $MIPS1$ is twice as fast as $Naive$ when it can prune half of the total number of coordinates.

We now compare the performance of $Naive$, $MIPS1$ and IL on the benchmark datasets (Figure S2). $MIPS1$ is the only method whose speed depends on κ since it is the only method to implement pruning. It has the same performance in terms of speed as $Naive$ for the lowest pruning rate, while it is as fast as IL for the highest pruning rates. For vectors $\boldsymbol{\theta}$ following classical distributions such

Algorithm S2 MIPS1

Input: $\mathbf{X} \in [0,1]^{n \times p}$, $\boldsymbol{\theta} \in \mathbb{R}^n$, $\mathcal{Q} \subset [\![p]\!]$, $\lambda \in \mathbb{R}$, $\mathcal{W} \subset [\![D]\!]$
Param: $n_c \in \mathbb{N}$
Output: $\mathcal{W}$, $\mathbf{m}^{ref}$.

Reorder the dimensions $1 \dots n$ such that $\boldsymbol{\theta}$ is sorted in descending order in absolute value and reorder the dimensions of $\mathbf{X}$ accordingly.
Reorder the columns of $\mathbf{X}$ in descending order of vector size.
for $j \in \mathcal{Q}$ **do** $\mathbf{m}_j^{ref} \leftarrow 0$
for $j \in \mathcal{Q}$ **do**
  Compute $\mathbf{r}^+ \in \mathbb{R}^n$ and $\mathbf{r}^- \in \mathbb{R}^n$ via (S4) and (S5).
  for $k \in [\![p]\!]$ **do**
    if $k \in \mathcal{Q}$ and $k > j$ then `continue`
    $d \leftarrow 0$ (inner product initialization); c $= 0$ (counter initialization);
    for $i \in \text{supp}(\boldsymbol{X}_j)$ **do**
      $d \leftarrow d + \mathbf{X}_{ij}\mathbf{X}_{ik}\boldsymbol{\theta}_i$
      $c \leftarrow c + 1$.
      if c `mod` $n_c = 0$ **then**
        if $(d + \mathbf{r}_i^+) < \min(\mathbf{m}_j^{ref}, \lambda)$ `and` $\left|(d + \mathbf{r}_i^-)\right| < \min(\mathbf{m}_j^{ref}, \lambda)$ **then** go to next probe.
      end if
    end for
    if $\mathbf{m}_j^{ref} < d < \lambda$ **then** set $\mathbf{m}_j^{ref} = d$
    if $d \geq \lambda$ and $\tau(k,j) \notin \mathcal{W}$ **then** add $\tau(k,j)$ to $\mathcal{W}$
  end for
end forreturn $\mathcal{W}, \boldsymbol{m}^{ref}$

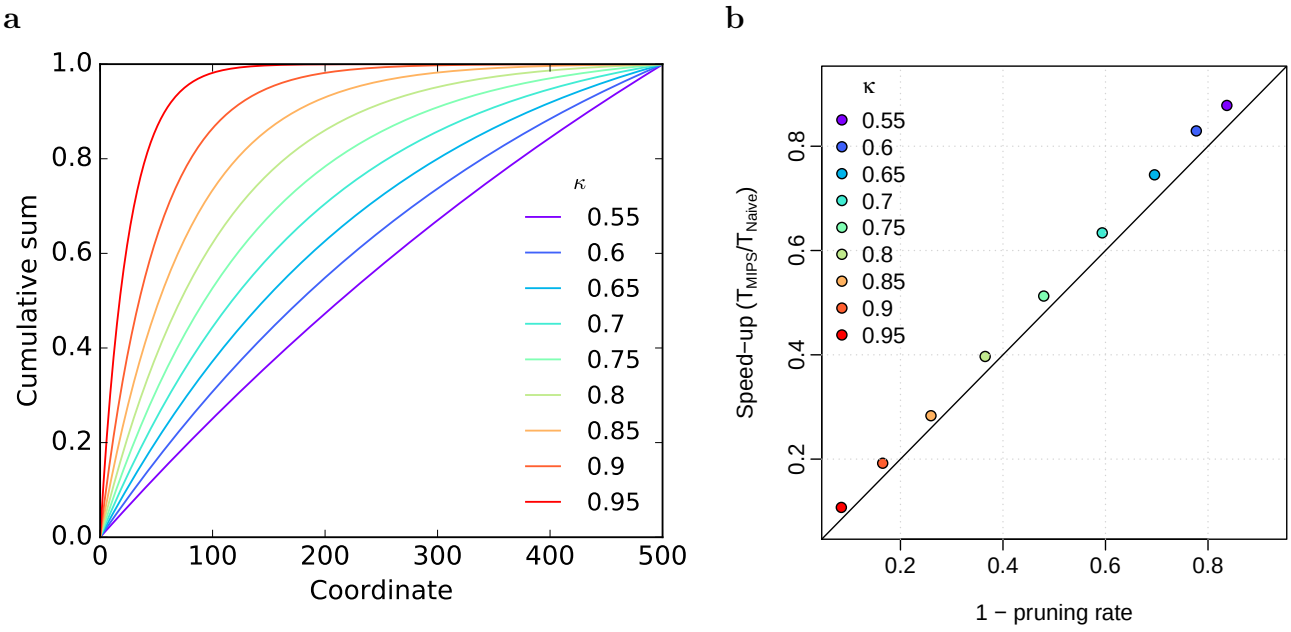

Figure S1 – **(a)** Cumulative sum of the vector obtained by sorting the positive entries of $\boldsymbol{\theta}_\kappa$ in decreasing order. **(b)** Speed-up obtained with $MIPS1$ compared to $Naive$ for different vectors $\boldsymbol{\theta}_\kappa$ as a function of the pruning rate. The pruning rate is defined as the average proportion of coordinates in the queries which are pruned.

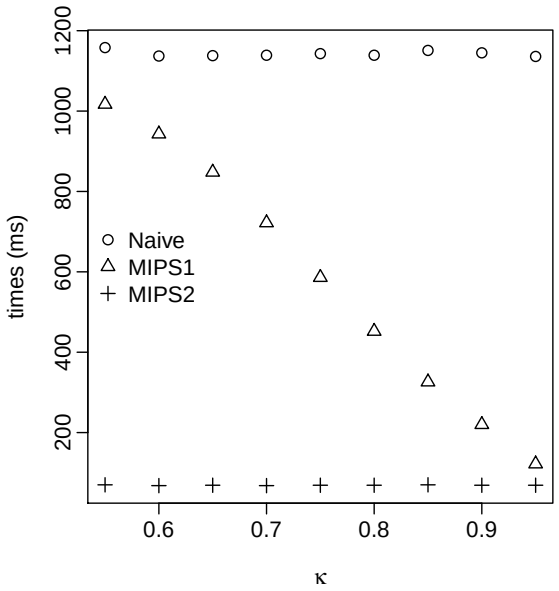

Figure S2 – Time (in ms) taken by $Naive$, $MIPS1$ and $MIPS2$ to solve Maximum Inner Product Search problems with responses characterised by different κ.

as the gaussian distribution, $\kappa \approx 0.7$ and $MIPS1$ is therefore expected to be $\times 1.6$ times faster than $Naive$ but $\times 11$ times slower than IL. An analysis of the complexity of $MIPS1$ and IL can help to understand these results. For a given query, $MIPS1$ requires to compute inner products (although partially) with all p vectors in the database. In our implementation, the vectors are encoded as sparse vectors, i.e., the vector $\mathbf{X}_j$ is represented by the list of its non-zero indices. If we assume that the number of non-zero elements in the query is $|q|$ and that the total number of non-zero elements of the vectors in $\mathbf{X}$ in nnz, then $MIPS1$ has a $O(p|q| + nnz)$ complexity to compute the p inner products with the query. By contrast, the inverted index approach has a $O(|q|\frac{nnz}{n})$ complexity, where $\frac{nnz}{n}$ is the average length of an inverted index. As the number of non-zero elements $|q|$ in the query will typically be a fraction of the total number of samples n, the inverted index approach is expected to be faster than $MIPS1$ even though the pruning in $MIPS1$ can make it faster. This however may not be the case with dense data instead of sparse data.

D SPP: depth-first vs breadth-first

The Safe Pattern Pruning algorithm presented in Nakagawa et al. (2016) deals with pairwise interactions but also higher-order interactions, and relies on a depth-first search scheme to explore the tree of patterns. However in our setting where we only consider pairwise interactions, we find that it is more efficient to implement a breadth-first search scheme for SPP. Indeed, the breadth-first search first identifies all the branches which can be screened. Then with this knowledge, we can restrict the number of interactions which are visited to those which only involve main effects whose corresponding branch was not screened. Basically, if we consider a case where p_s branches were screened among p branches, then the total number of nodes visited will be $p + \frac{(p-p_s)(p-p_s-1)}{2}$. Figure (S3) illustrates the difference in performance obtained with the original SPP and the breadth-first search version in the case of pairwise interactions. The speed up obtained with the breadth-first search version ranges from $\times 1.2$ for $n = p = 1000$ to $\times 1.6$ for $n = 1000, p = 10000$. We therefore use the breadth-first search version of SPP as a comparison baseline in all our experiments.

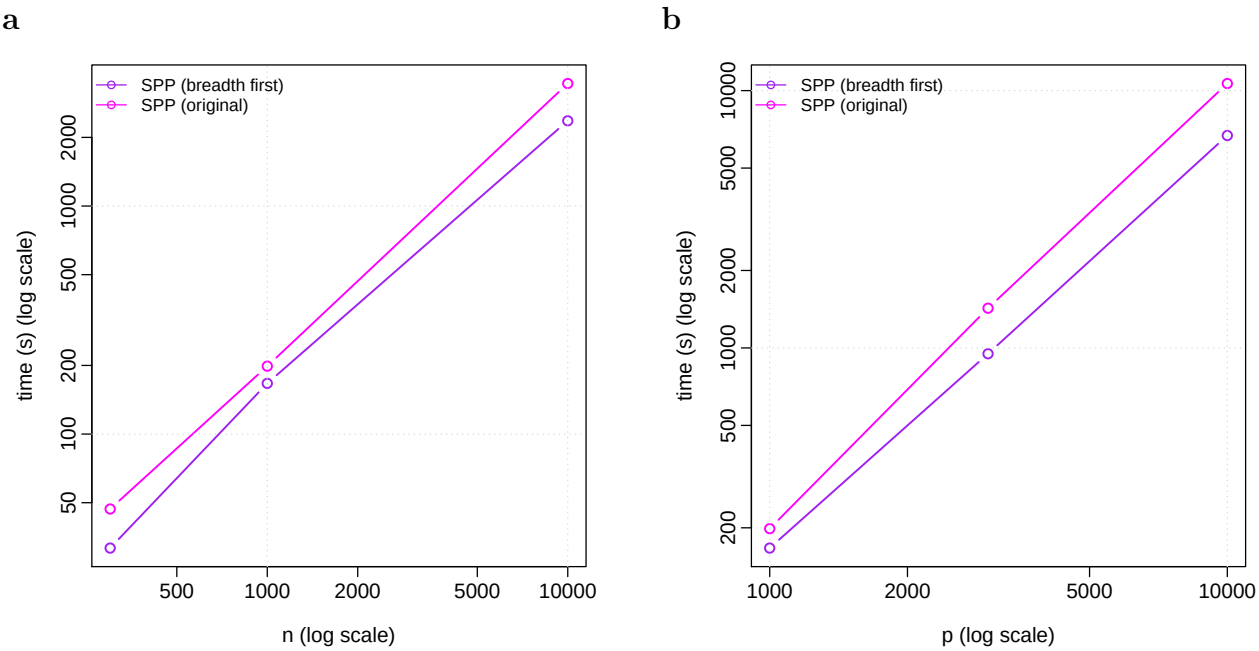

Figure S3 – **Safe Pattern Pruning performance on simulated data for an entire regularisation path**. The breadth-first search SPP (which is adapted to order-2 interactions only) is in purple and the original depth-first search SPP (which is adapted to order-2 interactions and more) is in magenta. (**a**) Time in seconds for $p = 1000$ fixed and n varied. (**b**) Time in seconds for $n = 1000$ fixed and n varied.

References

P. J Bickel, Y. Ritov, and A. B. Tsybakov. Hierarchical selection of variables in sparse high-dimensional regression. In *Borrowing Strength: Theory Powering Applications – A Festschrift for Lawrence D. Brown*, pages 56–69. Institute of Mathematical Statistics, 2010.

J. Bien, J. Taylor, and R. Tibshirani. A lasso for hierarchical interactions. *Ann. Stat.*, 41:1111–1141, 2013.

A. Bonnefoy, V. Emiya, L. Ralaivola, and R. Gribonval. Dynamic screening: Accelerating first-order algorithms for the lasso and group-lasso. *IEEE Trans. Signal Process.*, 63(19):5121–5132, 2015.

D. Donoho and J. Tanner. Observed universality of phase transitions in high-dimensional geometry, with implications for modern data analysis and signal processing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 367(1906): 4273–4293, 2009.

F. Eduati, L. M. Mangravite, T. Wang, H. Tang, J. C. Bare, T. Huang, T. Norman, M. Kellen, M. P. Menden, J. Yang, X. Zhan, R. Zhong, G. Xiao, M. Xia, N. Abdo, O. Kosyk, NIEHS-NCATS-UNC DREAM Toxicogenetics Collaboration, S. Friend, A. Dearry, A. Simeonov, R. R. Tice, I. Rusyn, F. A Wright, G. Stolovitzky, Y. Xie, and J. Saez-Rodriguez. Prediction of human population responses to toxic compounds by a collaborative competition. *Nat. Biotechnol.*, 33: 933–940, September 2015.

L. El Ghaoui, V. Viallon, and T. Rabbani. Safe feature elimination in sparse supervised learning. *Pacific J. Optim.*, 8(4):667–698, 2012.

J. Fan and J. Lv. Sure independence screening for ultrahigh dimensional feature space. *J. R. Stat. Soc. Ser. B*, 70(5):849–911, 2008. doi: 10.1111/j.1467-9868.2008.00674.x. URL http://dx.doi.org/10.1111/j.1467-9868.2008.00674.x.

O. Fercoq, A. Gramfort, and J. Salmon. Mind the duality gap: Safer rules for the lasso. In *Proc. 32nd Int. Conf. Mach. Learn.*, pages 333–342, Lille, France, 2015.

M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Y Zien. Evaluation Strategies for Top-k Queries over Memory-Resident Inverted Indexes. *Proc. VLDB Endow.*, 4(12):1213–1224, 2011.

Y. Fujiwara, Y. Ida, H. Shiokawa, and S. Iwamura. Fast Lasso Algorithm via Selective Coordinate Descent. In *Proc. 30th Conf. Artif. Intell. (AAAI 2016)*, pages 1561–1567, 2016.

N. Hao and H. H. Zhang. Interaction Screening for Ultra-High Dimensional Data. *J. Am. Stat. Assoc.*, 109(507):1285–1301, 2014.

A. Haris, D. Witten, and N. Simon. Convex Modeling of Interactions With Strong Heredity. *J. Comput. Graph. Stat.*, 25(4):981–1004, 2016.

T. Johnson and C. Guestrin. BLITZ: A principled meta-algorithm for scaling sparse optimization. In *Proc. 32nd Int. Conf. Mach. Learn. - ICML '15*, pages 1171–1179, 2015.

T. B. Johnson and C. Guestrin. StingyCD: Safely Avoiding Wasteful Updates in Coordinate Descent. *Proc. 34th Int. Conf. Mach. Learn. - ICML '17*, 70:1752–1760, 2017.

M. Kowalski, P. Weiss, A. Gramfort, and S. Anthoine. Accelerating ISTA with an active set strategy. In *OPT 2011: 4th International Workshop on Optimization for Machine Learning*, page 7, 2011.

M. Lim and T. Hastie. Learning Interactions via Hierarchical Group-Lasso Regularization. *J. Comput. Graph. Stat.*, 24(3):627–654, 2015.

A. Malti and C. Herzet. Safe screening tests for LASSO based on firmly non-expansiveness. In *IEEE Int. Conf. Acoust. Speech Signal Process.*, pages 4732–4736, 2016. ISBN 9781479999880. doi: 10.1109/ICASSP.2016.7472575.

M. Massias, A. Gramfort, and J. Salmon. From safe screening rules to working sets for faster Lasso-type solvers. *arXiv Prepr. arXiv1703.07285*, 2017.

K. Nakagawa, S. Suzumura, M. Karasuyama, K. Tsuda, and I. Takeuchi. Safe Pattern Pruning: An Efficient Approach for Predictive Pattern Mining. In *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. - KDD '16*, pages 1785–1794, 2016.

E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. GAP safe screening rules for sparse multi-task and multi-class models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Adv. Neural Inform. Process. Syst.*, pages 811–819. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5699-gap-safe-screening-rules-for-sparse-multi-task-and-multi-class-models.pdf.

E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. Gap safe screening rules for sparsity enforcing penalties. *J. Mach. Learn. Res.*, 18(128):1–33, 2017.

A. L Price, N. J. Patterson, R. M. Plenge, M. E. Weinblatt, N. A. Shadick, and D. Reich. Principal components analysis corrects for stratification in genome-wide association studies. *Nat. Genet.*, 38:904–909, August 2006.

P. Radchenko and G. James. Variable selection using Adaptive Nonlinear Interaction Structures in High dimensions. *J. Am. Stat. Assoc.*, 105(492):1541–1553, 2010.

A. Raj, J. Olbrich, B. Gärtner, B. Schölkopf, and M. Jaggi. Screening Rules for Convex Problems. *arXiv Prepr. arXiv1609.07478*, 2016.

R. D Shah. Modelling interactions in high-dimensional data with backtracking. *J. Mach. Learn. Res.*, 17(207):1–31, 2016.

Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Adv. Neural Inf. Process. Syst. - NIPS '14*, pages 2321–2329, 2014.

Shinya Suzumura, Kazuya Nakagawa, Yuta Umezu, Koji Tsuda, and Ichiro Takeuchi. Selective Inference for Sparse High-Order Interaction Models. In *Proc. 34th Int. Conf. Mach. Learn. - ICML '17*, volume 70, pages 3338–3347, 2017.

C. Teflioudi and R. Gemulla. Exact and Approximate Maximum Inner Product Search with LEMP. *ACM Trans. Database Syst.*, 42(1), 2016.

R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B*, 58(1): 267–288, 1996.

R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *J. R. Stat. Soc. Ser. B*, 74(2):245–266, 2012.

M. J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programming (lasso). *IEEE Trans. Inf. Theory*, 55(5):2183–2202, 2009.

J. Wang, J. Zhou, P. Wonka, and J. Ye. Lasso screening rules via dual polytope projection. In *Adv. Neural Inf. Process. Syst. - NIPS '13*, pages 1070–1078, 2013.

T. T. Wu, Y. Fang Chen, T. Hastie, E. Sobel, and K. Lange. Genome-wide association analysis by lasso penalized logistic regression. *Bioinformatics*, 25(6):714–721, 2009.

Z. Xiang, H. Xu, and P. J. Ramadge. Learning sparse representations of high dimensional data on large scale dictionaries. In *Adv. Neural Inf. Process. Syst. - NIPS '11*, pages 1–9, 2011.

Z. J. Xiang and P. J. Ramadge. Fast lasso screening tests based on correlations. In *IEEE Int. Conf. Acoust. Speech Signal Process.*, pages 2137–2140, 2012.