Computations On Distributed Data Without Aggregation

Balasubramanian Narasimhan

Department of Health Research and Policy
Department of Statistics
Stanford University

June 30, 2015

Setting

In many situations, data of interest is distributed in several places.

- Medical databases containing clinical data
- Genomic data generated at institutions
- Clinical trials data

Analyzing aggregated data promises many advantages.

- Reliable and stable modeling of outcomes
  - Larger N
  - Precision of estimates
  - Power for detecting differences
- Richer feature sets for use in models
- More chance for finding "patients like me"

The Problem

There are high (and growing) barriers to aggregation of medical data, particularly between centers/researchers.

- Lack of standardization of ontologies
- Privacy concerns
- Reluctance to cede control (once flown...)
- Proprietary attitude towards institution's data

There have been several large efforts in these areas often very costly and yielding mixed results.
The aggregation step may sometimes even be undesirable, due to the sheer scale of the data.

Distributed data will do

It has been long known that data aggregation is unnecessary; aggregating computational summaries suffices. Thus,

- Data can stay at site;
- A master process can aggregate the summaries from sites and perform statistical calculations, e.g., an MLE calculation

Many model fitting computations can be so implemented.

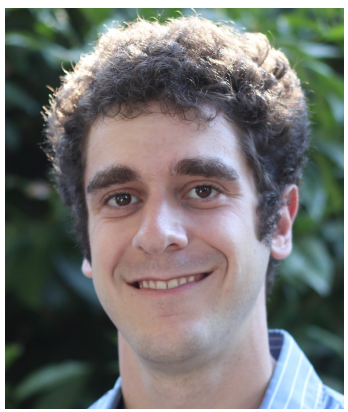
GLM, MLE Likelihood breaks up into sums of likelihood computed on local data at sites.

SVD Iterative algorithms are available for computing singular values using quantities computed on local data at sites.

Indeed, sometimes distribution of the calculation among sites is necessary to share a heavy computational burden.
We present an R-centered approach, using **opencpu** and **shiny**.

Collaborators and Acknowledgements

Marina Bendersky



Samuel Gross

Philip Lavori

Daniel Rubin

Research supported by Stanford Cancer Institute, NIH, NSF (Nos. P30 CA124435-01, UI1 RR025744, LM-07033)

Previous Work

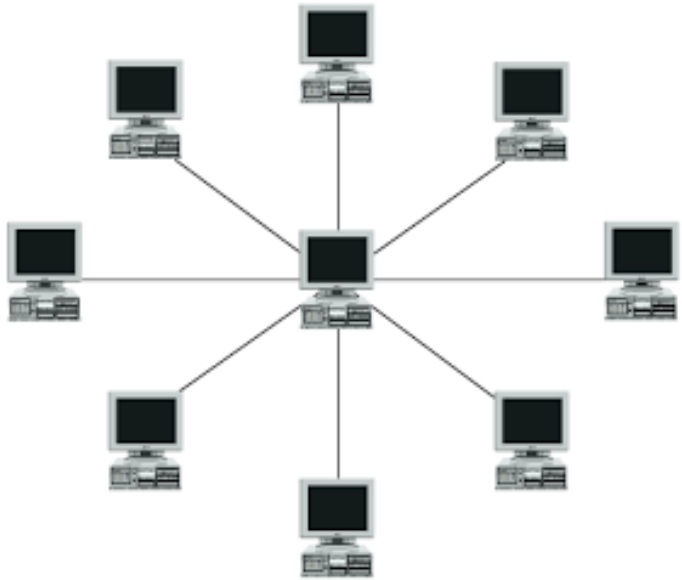
The idea that computations can be distributed is not new.

- Jiang et. al.(*Bioinformatics* 2013) describe the **WebGLORE**: Web-based Grid LOgistic REgression service
- Wolfson, et. al.(*IJE, 2010*) describe fitting generalized linear models (GLMs) by aggregating anonymous summary-statistics from harmonized individual-level databases (DataSHIELD).

> *OBiBa suite includes advanced software components enabling data harmonization and federation for study networks that aim to harmonize and share securely data among their members...*

Source: `http://www.obiba.org`
This is project is not unlike ours, using R, and implements the iteratively weighted least squares (IRLS) algorithm for fitting a full GLMs using only summary statistics computed from the distributed data.

Assumptions



- Star topology, master in center;
- Transmitting summaries is ok; in fact, master makes an unlimited number of calls on worker sites;
- Some degree of trust between sites via agreements between CIOs etc.

Singular Value Decomposition

The SVD of $\boldsymbol{X}_{n\times p}$, is $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{D}$ such that

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top,\ \boldsymbol{U}^\top\boldsymbol{U} = I,\ \boldsymbol{V}^\top\boldsymbol{V} = I,\ \text{and } \boldsymbol{D} \text{ is diagonal.}$$

- Decomposes the variance of $\boldsymbol{X}$ into what are called principal components.
- v_1, the first column of $\boldsymbol{V}$, the first principal component of X maximizes of $\mathrm{var}(\boldsymbol{X}v)$.
- u_1 indicates how much of factor v_1 is present in each observation.
- $d_1^2/\sum_j d_j^2$ is the proportion of the variance of $\boldsymbol{X}$ that can be explained by v_1.
- The first k vectors can used to get a k approximation to $\boldsymbol{X}$.

Efficient Implementations in LAPACK, which much software builds upon.

There is a well-known power method for computing a singular vector corresponding to the largest eigenvalue.

Data: $\boldsymbol{X} \in \mathcal{R}^{n \times p}$

Result: $u \in \mathcal{R}^n$, $v \in \mathcal{R}^p$, and $d > 0$

$u \leftarrow (\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$;

repeat

$v \leftarrow \boldsymbol{X}^\top u$;

$v \leftarrow v / \|v\|$;

$u \leftarrow \boldsymbol{X} v$;

$d \leftarrow \|u\|$;

$u \leftarrow u / \|u\|$;

until *convergence*;

Note that the operations involve inner products and sums and therefore distribute over sites.

Singular vectors can be found successively by removing the effect of the top singular vector and then finding the rank-1 approximation again.

Privacy-preserving rank-k SVD

Data: each site has private data $\boldsymbol{X}_j \in \mathcal{R}^{n_j \times p}$

Result: $V \in \mathcal{R}^{p \times k}$, and $d_1 \geq \ldots d_k \geq 0$

$V \leftarrow 0$, $d \leftarrow 0$ **foreach** *site* j **do**
  $U^{[j]} = 0$;
  transmit n_j to master;
end
for $i \leftarrow 1$ **to** k **do**
  foreach *site* j **do** $u^{[j]} \leftarrow (1, 1, \ldots, 1)$ of length n_j;
  $\|u\| \leftarrow \sqrt{\sum_j n_j}$;
  transmit $\|u\|$, V, and D to sites;
  repeat
    foreach *site* j **do**
      $u^{[j]} \leftarrow u^{[j]} / \|u\|$;
      calculate $v^{[j]} \leftarrow (\boldsymbol{X}^{[j]} - U^{[j]} D V^\top)^\top u^{[j]}$;
      transmit $v^{[j]}$ to master;
    end
    $v \leftarrow \sum_j v^{[j]}$; $v \leftarrow v / \|v\|$;
    transmit v to sites;
    foreach *site* j **do**
      calculate $u^{[j]} \leftarrow \boldsymbol{X}^{[j]} v$;
      transmit $\|u^{[j]}\|$ to master;
    end
    $\|u\| \leftarrow \sum_j \|u^{[j]}\|$;
    transmit $\|u\|$ to sites;
    $d_i \leftarrow \|u\|$;
  until *convergence*;
  $V \leftarrow \texttt{cbind}(V, v)$;
  foreach *site* j **do** $U^{[j]} \leftarrow \texttt{cbind}(U^{[j]}, u^{[j]})$;
end

Site Stratified Cox Model

The Cox PH model assumes a hazard function of the form

$$\boldsymbol{\lambda}_{n\times 1}(t) = \lambda_0(t)\exp(\boldsymbol{X}_{n\times p}\boldsymbol{\beta}_{p\times 1}),$$

Model fitting and inference is accomplished by maximizing a partial likelihood function (see Therneau and Grambsch) of the form:

$$l(\boldsymbol{\beta}|\boldsymbol{X}) = \sum_{i=1}^{n}\int_0^{\infty}\left[Y_i(t)\boldsymbol{X}_i(t)\boldsymbol{\beta} - \log\left(\sum_j Y_j(t)r_j(\boldsymbol{\beta},t)\right)\right]dN_i(t).$$

In multicenter studies, the *stratified* Cox model is often used where each site is a stratum. This allows for different a baseline hazard for each site, yet a single $\boldsymbol{\beta}$ is fit.
It turns out again that the overall log-likelihood is a sum over the strata. Same for score, information matrix etc. So once again the computation can be distributed.

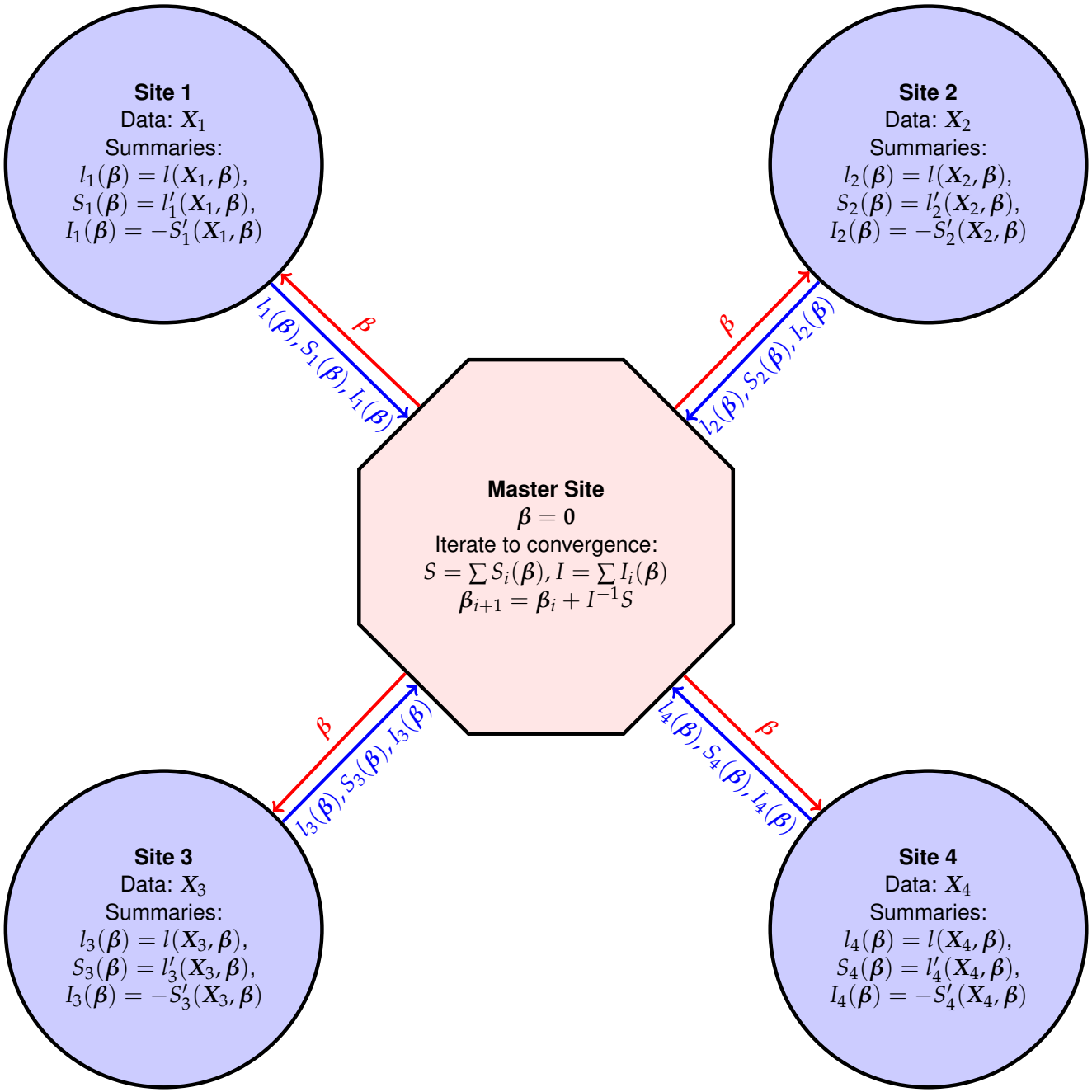
Maximization via Newton-Raphson

K sites, $l_k(\boldsymbol{\beta})$, $S_k(\boldsymbol{\beta})$, $I_k(\boldsymbol{\beta})$ are site-specific likelihood, score and information matrix.

0. Set $i = 0$, $\boldsymbol{\beta}_0 = 0$, a tolerance ϵ and a maximum number of iterations B.
1. Transmit $\boldsymbol{\beta}_i$ to each site
2. Each site k sends back $l_k(\boldsymbol{\beta}_i)$, $S_k(\boldsymbol{\beta}_i)$ and $I(\boldsymbol{\beta}_i)$
3. Compute $l(\boldsymbol{\beta}_i) = \sum_{k=1}^{K} l(\boldsymbol{\beta}_)$, $S(\boldsymbol{\beta}_i) = \sum_{k=1}^{K} S_k(\boldsymbol{\beta}_i)$, $I(\boldsymbol{\beta}_i) = \sum_{k=1}^{K} I_k(\boldsymbol{\beta}_i)$,
4. Set
$$\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i + I^{-1}(\boldsymbol{\beta}_i) S(\boldsymbol{\beta}_i)$$
5. Stop if converged or iteration count exceeded. Else increment i and repeat step 1.

For the Cox Model, the convergence is very fast with a slight tweak such as step-halving.

Schematic


Site 1
Data: X_1
Summaries:
$l_1(\beta) = l(X_1, \beta)$,
$S_1(\beta) = l_1'(X_1, \beta)$,
$I_1(\beta) = -S_1'(X_1, \beta)$
Site 2
Data: X_2
Summaries:
$l_2(\beta) = l(X_2, \beta)$,
$S_2(\beta) = l_2'(X_2, \beta)$,
$I_2(\beta) = -S_2'(X_2, \beta)$
Master Site
$\beta = 0$
Iterate to convergence:
$S = \sum S_i(\beta), I = \sum I_i(\beta)$
$\beta_{i+1} = \beta_i + I^{-1}S$
Site 3
Data: X_3
Summaries:
$l_3(\beta) = l(X_3, \beta)$,
$S_3(\beta) = l_3'(X_3, \beta)$,
$I_3(\beta) = -S_3'(X_3, \beta)$
Site 4
Data: X_4
Summaries:
$l_4(\beta) = l(X_4, \beta)$,
$S_4(\beta) = l_4'(X_4, \beta)$,
$I_4(\beta) = -S_4'(X_4, \beta)$
β
$l_1(\beta), S_1(\beta), I_1(\beta)$
β
$l_2(\beta), S_2(\beta), I_2(\beta)$
β
$l_3(\beta), S_3(\beta), I_3(\beta)$
β
$l_4(\beta), S_4(\beta), I_4(\beta)$

Components of a Distributed Approach

- Readily available computing power (a unix server box) on local data.
- Tools to propose, define and refine computation tasks (R package shiny, R package opencpu)
- An extensible, open source environment to implement such tools and algorithms in order than Joe and Pam can be peers (R for us, our package distcomp)
- Secure means of exposing computation to site (SSL)

There are several social aspects of the collaboration need to be engineered.
We describe our progress so far with our R package distcomp.

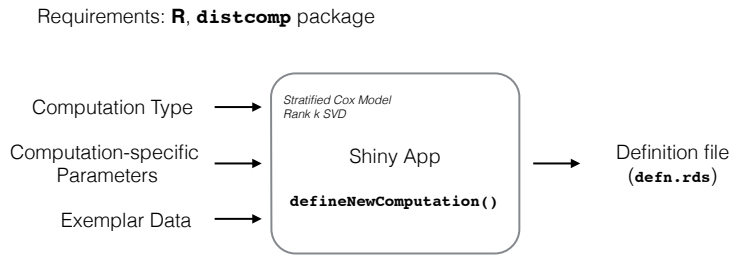
Implementation Details

- Definiation of a computation are stored in an object, tagged by unique names and each instance is given a unique identifier
- For each computation there is a master object (`CoxMaster`, `SVDMaster`, etc.) and a corresponding worker object (`CoxSlave`, `SVDSlave`, etc.) implemented using `R6` classes
- Worker objects can be *stateful*. So for example, they may change states during iterations (serialized to workspace)
- Function calls are via `opencpu` to `distcomp` package (`httr::POST` calls) invoking methods on instantiated objects
- Data is JSON serialized in current implementation

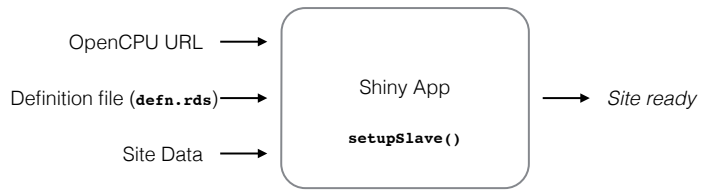
Executing a Distributed Computation

The main steps are the following.

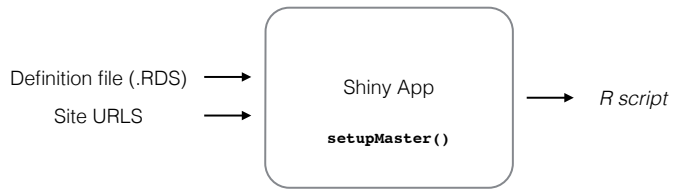
1. Define the computation:



2. Set up worker processes:



3. Create master script:



The resulting R script can then be executed.

Example: SVD Computation

Simulated dataset on three sites, 20×5 matrix at each site.
Aggregated SVD is:

```
> set.seed(12345)
> x <- matrix(rnorm(100), nrow = 20))
> svd(x)$d
[1] 9.707537 8.199827 7.982888 7.257286 6.235182
> svd(x)$v
            [,1]        [,2]        [,3]        [,4]        [,5]
[1,] -0.17946375  0.08268613 -0.01644895 -0.98010572 -0.00883063
[2,] -0.78963831  0.34694371  0.34328503  0.16509457  0.33316749
[3,]  0.21305901  0.91839439 -0.25083926  0.04461477 -0.21505068
[4,]  0.54504905  0.16843629  0.53318714 -0.10009622  0.61663844
[5,] -0.04232602 -0.03120945 -0.73121540  0.01126215  0.68002329
```

The Distributed SVD Fit

The code produced in the master setup is:

```
library(distcomp)
defn <-
structure(list(id = "c83c9bd59551df3a", compType = "RankKSVD",
    projectName = "SVDTest", projectDesc = "SVD Test Example",
    rank = 2L, ncol = 5L), .Names = c("id", "compType", "projectName",
"projectDesc", "rank", "ncol"), row.names = c(NA, -1L), class = "data.frame")
sites <-
list(structure(list(name = "Site1", url = "http://127.0.0.1:5999/ocpu"), .Names = c("name",
"url")), structure(list(name = "Site2", url = "http://127.0.0.1:5999/ocpu"), .Names = c("name",
"url")), structure(list(name = "Site3", url = "http://127.0.0.1:5999/ocpu"), .Names = c("name",
"url")))
master <- makeMaster(defn)
for (site in sites) {
   master$addSite(name = site$name, url = site$url)
}
result <- master$run()
print(master$summary())
```

The SVD Output

If you run that program, after a while it spits out the following:

```
$v
            [,1]        [,2]
[1,]  0.17947030  0.08275684
[2,]  0.78969198  0.34634459
[3,] -0.21294972  0.91875219
[4,] -0.54501407  0.16784298
[5,]  0.04229739 -0.03032954

$d
[1] 9.707451 8.200043
```

If you actually ask for $k = 5$, it gives:

```
> result$d
[1] 9.707451 8.200043 7.982650 7.257355 6.235351

> result$v
            [,1]        [,2]       [,3]        [,4]         [,5]
[1,]  0.17947030  0.08275684  0.0165604  0.98008722 -0.008933396
[2,]  0.78969198  0.34634459 -0.3437723 -0.16504730  0.333181988
[3,] -0.21294972  0.91875219  0.2496210 -0.04479619 -0.214978886
[4,] -0.54501407  0.16784298 -0.5334277  0.10025749  0.616612820
[5,]  0.04229739 -0.03032954  0.7312254 -0.01140918  0.680060781
```

Example: Cox Model

Hosmer and Lemeshow data on time until return to drug use for patients enrolled in two different residential treatment programs. Aggregated fit is:

```
> uis <- readRDS("uis.RDS")
> coxOrig <- coxph(formula = Surv(time, censor) ~ age + becktota +
+                      ndrugfp1 + ndrugfp2 + ivhx3 +
+                      race + treat + strata(site), data = uis)
> summary(coxOrig)
Call:
coxph(formula = Surv(time, censor) ~ age + becktota + ndrugfp1 +
ndrugfp2 + ivhx3 + race + treat + strata(site), data = uis)

n= 575, number of events= 464
(53 observations deleted due to missingness)

               coef exp(coef)  se(coef)      z Pr(>|z|)
age       -0.028076  0.972315  0.008131 -3.453 0.000554 ***
becktota   0.009146  1.009187  0.004991  1.832 0.066914 .
ndrugfp1  -0.521973  0.593349  0.124424 -4.195 2.73e-05 ***
ndrugfp2  -0.194178  0.823512  0.048252 -4.024 5.72e-05 ***
ivhx3TRUE  0.263634  1.301652  0.108243  2.436 0.014868 *
race      -0.240021  0.786611  0.115632 -2.076 0.037920 *
treat     -0.212616  0.808466  0.093747 -2.268 0.023331 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
...
```

The Distributed Cox Fit

Let's do it!

Data in Databases

The examples shown here use CSV files for demonstration. It is straightforward to use data in databases, Redcap for example.

- We use the redcap-api package to get data in R
- The shiny apps ask for Redcap secret token instead

This is currently being tested. Note that the Redcap server only need to talk to the opencpu server and not the outside world. This approach can be replicated for any database, where instead of uploading a CSV file, one would specify database parameters.

Ongoing Work

- User-friendly tools for collaboration
- Improving error messages, better fault-tolerance
- Designing dashboards for audits
- Efficient serialization: JSON versus Protocol Buffers
- Implementing more models

References

- N. et.al. *Software for Distributed Computation on Medical Databases: A Demonstration Project*. Arxiv paper: `http://arxiv.org/abs/1412.6890`
- Software on on CRAN and on Github: `http://github.com/hrpcisd`