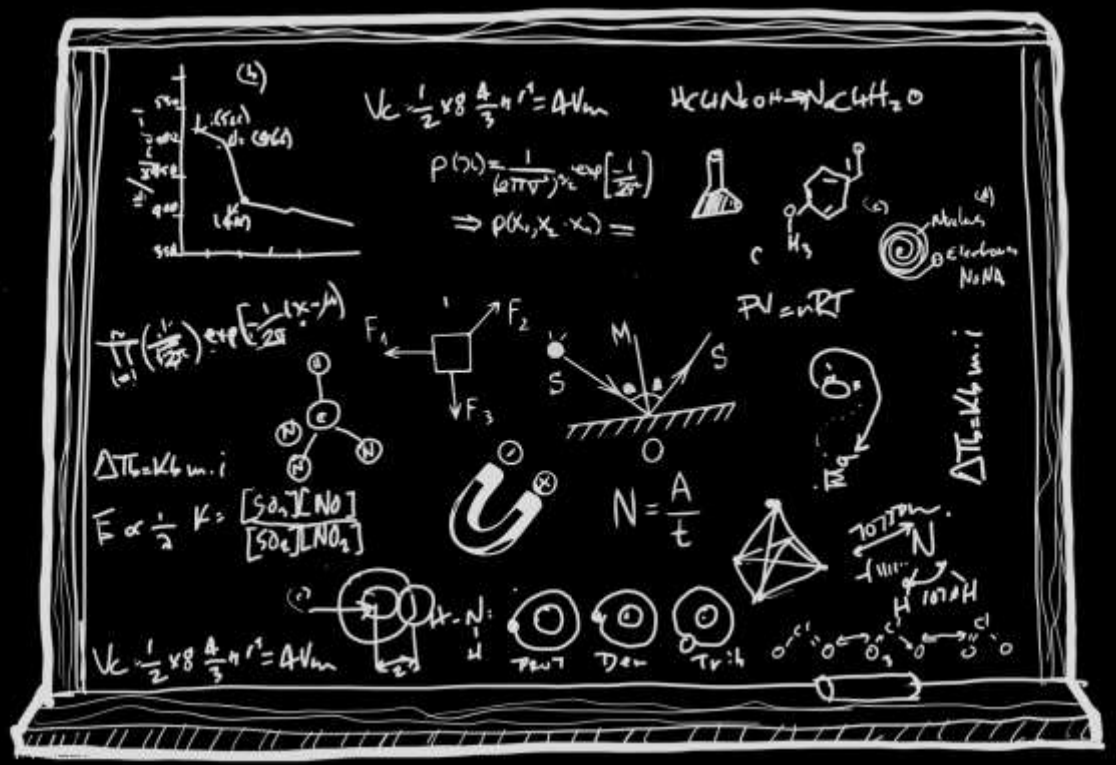
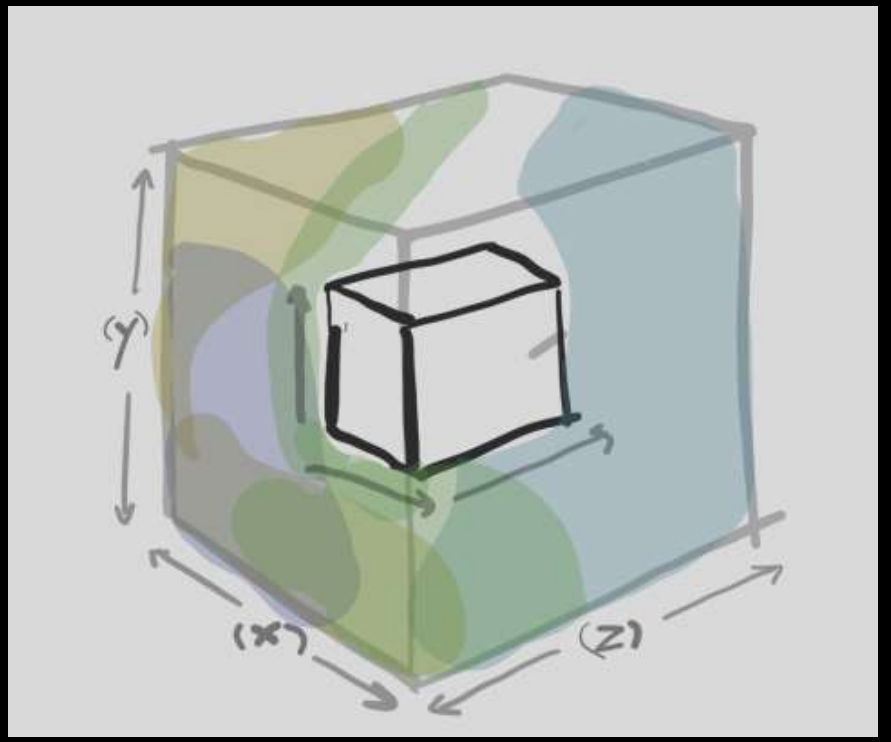
力任せだけど
繊細なアルゴリズムについて

コンピュータの性能を活用しよう！
でも、より賢く、より要領よく！
【JMPの将来バージョンにおいて実装しようとしている、開発段階の機能を紹介します。】

ジョン・ソール（John Sall）

克服したい数学的難問：入力変数の仕様限界を設定したい





入力変数の非線形関数で表される出力変数が
特定の範囲内に収まるように、入力変数の矩形領域、
つまり、長方形の領域を求める

克服したい数学的難問：ゲノム領域での階層型クラスター分析

アトラス

アルキメデス

仕様限界（規格限界）

ジュリア・オネイル様のインタビュー

クオリティ・バイ・デザイン（QbD）

「仕様限界（規格限界）」に関係するQbDの用語

重要工程パラメータ（CPP: Critical Process Parameters）

重要品質特性（CQA: Critical Quality Attributes）

デザインスペース

立証された許容範囲（PAR: Proven Acceptable Ranges）

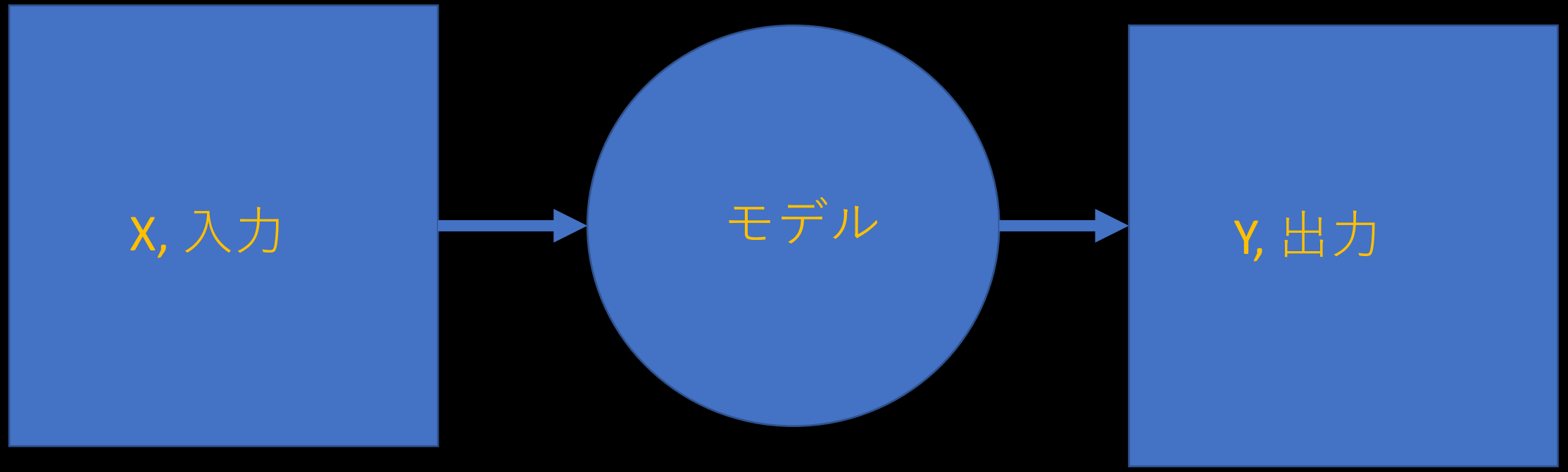
通常の稼働範囲（NOR: Normal Operating Ranges）

ICH-Q8(R2)におけるデザインスペース（設計領域）の定義

「**デザインスペース**」

- 「品質を確保することが立証されている入力変数（原料の性質など）と工程パラメータの多元的な組み合わせと相互作用。」
- 「このデザインスペース内で運用することは変更とはみなされない。」
- 「デ ザインスペース外への移動は変更とみなされ、通常は承認事項一部変更のための規制手続きが開 始されることになる。」
- 「デザインスペースは申請者が提案し、規制当局がその評価を行って承認する。」

入出力のモデル



入力に対する仕様限界を推測する

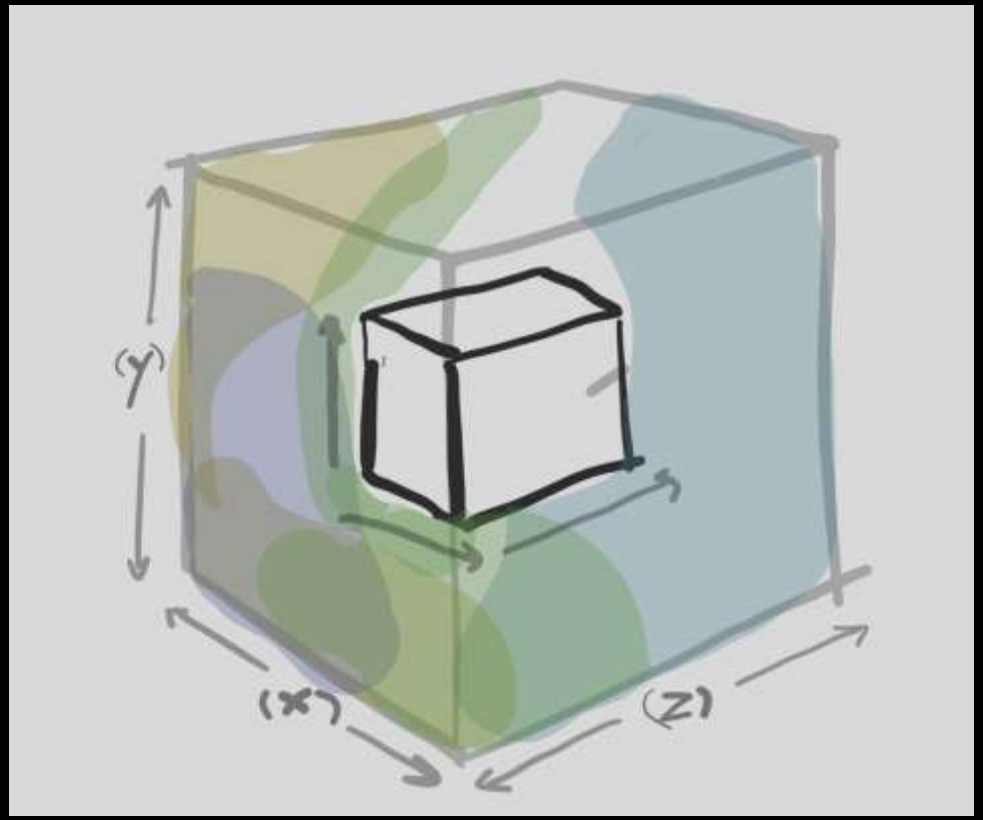
出力に対する仕様限界から始めて...

デザインスペースは矩形領域（長方形の領域）に限らないが
矩形領域にすると入力を独立して管理できる

デザインスペース（設計領域）

- 許容できる出力となる入力の領域を求める
- 許容できる出力となる入力の仕様限界（＝矩形領域）を求める。矩形である必要はないが、矩形にすれば１因子ずつ独立に管理できる。
- 許容できる出力となる入力の矩形領域のうち、体積が最大となる領域を求める（＝これは難問）

この問題を解くために、入力に対して、出力が仕様を満たしているかどうかを、グラフに描いてみる。そして、そこから矩形領域を求めてみる。



出力に対する次の条件を満たすもののうち、その体積が最大となる入力の矩形領域（仕様限界）を求めよ。

「摩擦」 ≧ 120 &
「引張応力」 ≧ 1200 &
「伸び」 ≧ 350 & 「伸び」 ≦ 500
「硬度」 ≧ 65 & 「硬度」 ≦ 75

ただし、各出力は、入力（シリカ・シラン・硫黄）の2次式で表されており、次の誤差標準偏差であるとする。

「摩擦」s.d.＝3
「引張応力」s.d.＝100
「伸び」s.d.＝10
「硬度」 s.d. = 0.6

デザインスペースの定式化:
タイヤのトレッドに関する実験の例

デモ

ゲノム領域での階層型クラスター分析

計算時間を短くする

ジョン・ソール（John Sall）

問題克服までの道のり
- コンガライン法を用いる
- ケビン・ベーコン問題を解決する
- VT木を用いる
- 乱数を掛け合わせる

テーマ：力任せだけど慎重な方法（矛盾した用語）

克服したい問題

- ゲノム領域では、データの分布を理解するのに階層型クラスターがよく使われている。
- ゲノム領域では、行方向だけでなく、列方向でもクラスタリングが行われる。そして、ヒートマップが描かれる。
- しかし、ゲノム領域のデータは大規模。何千〜何十万の列、何百〜何千もの行。
- 一方、大規模データに対する階層型クラスターの計算時間は長い。
- この計算時間を速めたい。

ゲノム領域の横長な（ワイドな）データ

データセット名	行数	列数	セル数
gtex_gene_tpm_log2	17,382	54,592	948,918,144
chr10_emaize_wide_num2	6,210	135,608	842,125,680
pmra_390_gt_wid3	390	902,560	351,998,400
pmra_390_gt_wid4	390	667,505	260,326,950
Baron_countPhenoCombined	7,266	20,124	146,220,984
Morocco_numgeno_194_maf02_pcs	194	552,145	107,116,130
comb_geno_w_num_49041NoMiss	249	49,041	12,211,209
giant_island_mice_wide	100	25,097	2,509,700
breastcancer_MAQC	230	10,787	2,481,010
mmap_wid_log2	79	31,271	2,470,409

計算時間が長すぎる……

- Morocco_numgeno_194_maf02_pcsデータ: 194行, 552,145列
- 列に対する高速Ward法: 171,893秒 （約48時間）
- これだけ計算時間がかかると、対話的な分析とは言えない。

前提知識

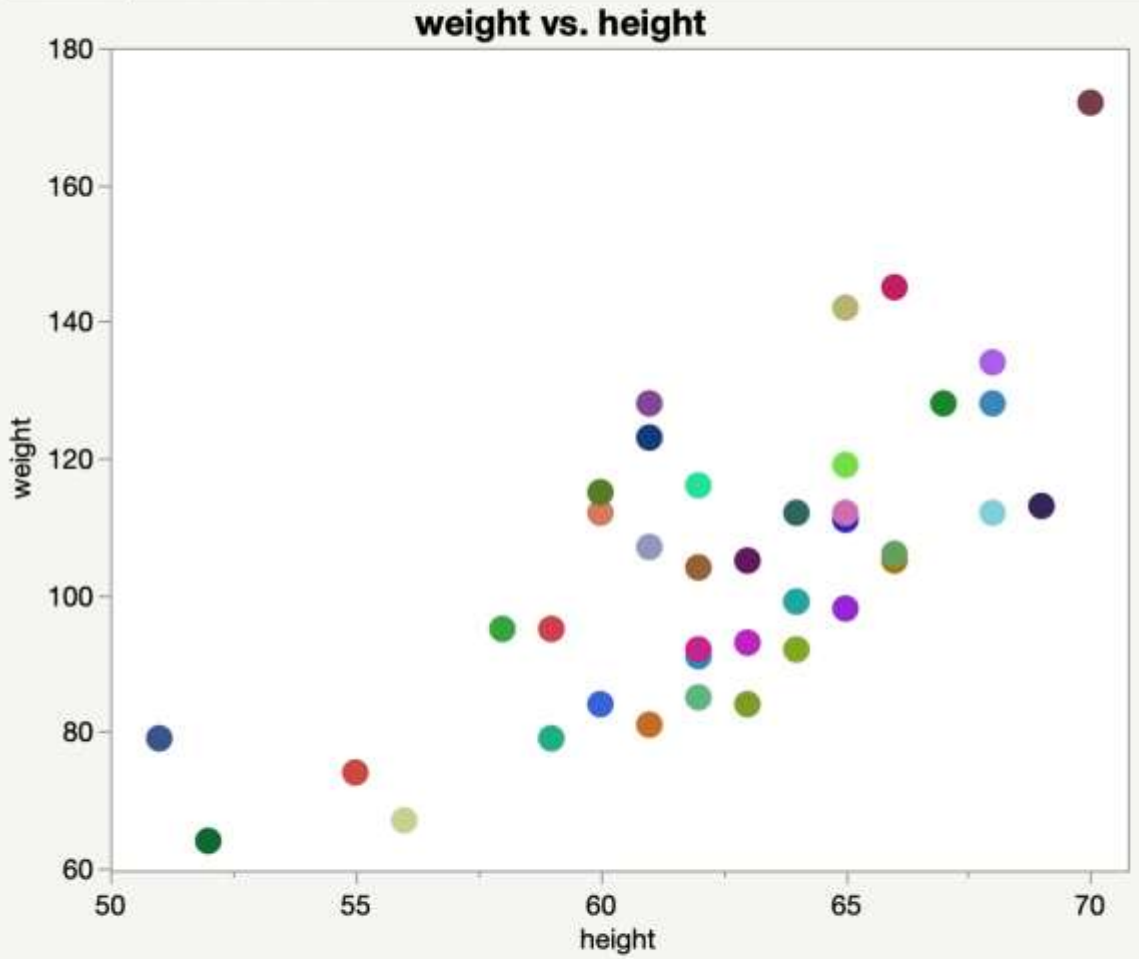
階層型クラスター分析は、凝集型アルゴリズムである。

- n個のクラスターから始める。1クラスターがデータの1行。
- 最短距離のペアを見つけ、そのペアのクラスターを併合する。
- 全体で1つのクラスターになるまで、上記の処理を続ける。
- 結合していく様子は樹形図で描かれる：「デンドログラム」
- 「距離」の測り方には、いくつかの方法が提案されている。
- そのなかでWard距離は、最小2乗推定と相性がよい。 Ward距離は重み付きのEuclid距離。

Hierarchical Clustering

Window Tools Graph Tools Show Data Table Local Data Filter Show Properties

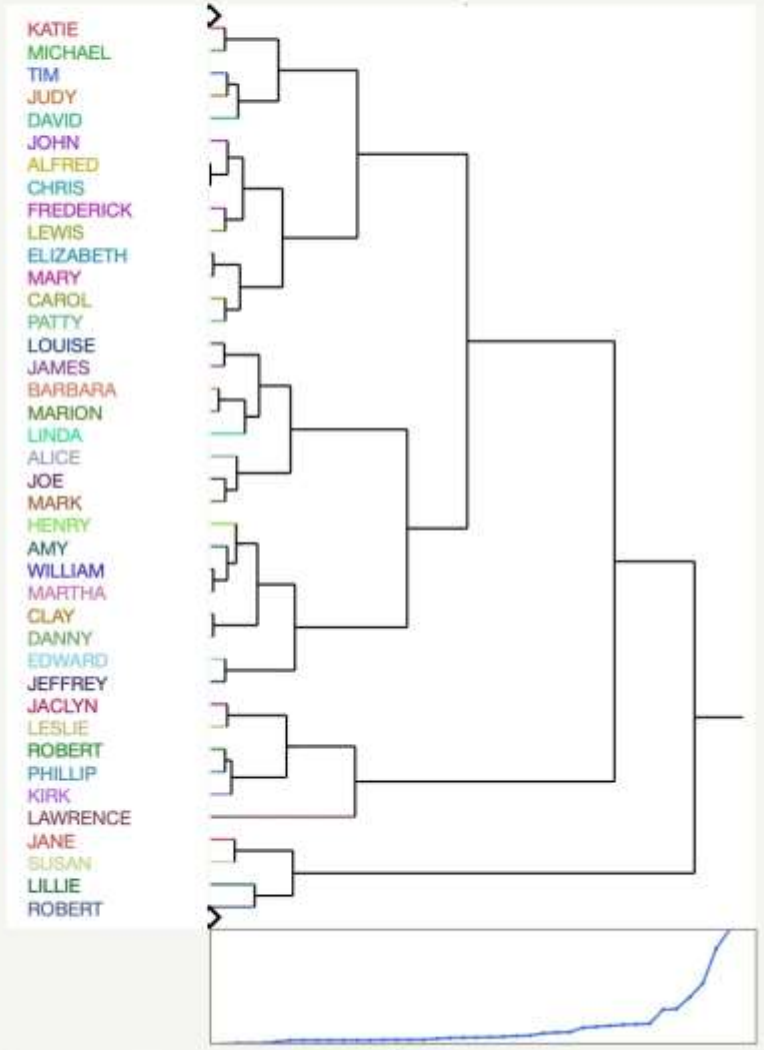
Graph Builder



Hierarchical Clustering

Method = Ward

Dendrogram



Clustering History

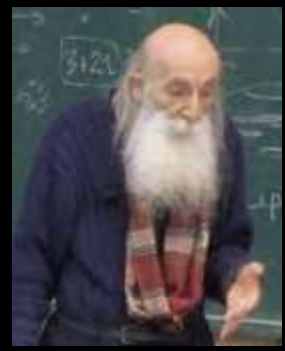
古典的なアルゴリズム

距離行列に基づくアルゴリズム

- n行のデータから、n×nの距離行列を計算する。
- 距離のなかから、最短のものを探す。
- それらを併合する。結合先の行を削除し、結合元の行における距離を更新する。この処理を反復する。
- このアルゴリズムは大規模データでは計算負荷が高い。
  - 計算のオーダーは $n^2m + n^3$（n^2のなかから最短距離をさがすのを n回繰り返す）。必要メモリはn^2のオーダー。
  - ゲノムでは、mやnは10^5。1,000行 100,000列ならば、$(10^5)^2 \times 10^3 + (10^5)^3 \simeq 10^{16}$ぐらいの計算量となる。とてつもない計算量。

より高速なアルゴリズム:
最近隣連鎖法
（別名：コンガライン法）

Fionn Murtagh
Jean-Paul Benzécri
J. Juan David Eppstein



- https://en.wikipedia.org/wiki/Nearest-neighbor_chain_algorithm
- n個のクラスター（データ1行が1個のクラスター）から開始。
- Sをスタックとする。最初、Sは空集合とする。
- クラスターが1個になるまで、次の処理を行う。
  - Sが空集合の場合、現在のクラスターから1個を選び、Sに追加する。
  - Cを、Sの頂上にあるクラスターとする。Cから最近隣にあるクラスターを選ぶ。そのクラスターをDとする。
  - Dが、すでにSになるならば、それはCのすぐ一つ下にあるクラスターのはず。CとDの両方をポップする。
  - DがSにないならば、DをSにプッシュする。

最近隣連鎖法

- すべての距離（$n^2/2$個の距離）を計算する必要なし。
- 最近隣を探す速い方法があれば、各ステップでn個の距離を再計算する必要なし。
- 最近隣を探す上手い方法の１つ: kd木
- ノードを追加・削除するときに、木を更新する必要あり
- kd木は、最近隣探索および追加・削除の効率的なデータ構造
- JMPでは、Craige Halesが開発担当（KDTree.cpp）。
- 「高速Ward法」は、小規模なデータを除き、デフォルトの方法。

最近隣連鎖法の問題: まだ遅すぎる...

- ゲノム領域のデータを扱うには、最近隣連鎖法はまだ遅すぎる。
- 最近隣連鎖法はマルチスレッド化が難しい。
- kd木は良いデータ構造だが、高次元データに対しては、より効率的なアルゴリズムがある。しかし、それらのアルゴリズムは、ノードの追加・削除が非効率的。
- そのため、新しい方法・アルゴリズムが必要。
- 大規模データに対するクラスター法はいくつかの論文があるが、私は単純な方法を実装したい。

発想

- 階層型クラスター分析において、厳密な階層構造に興味があるのは、樹形図の上部における数百の結合だけ。
- 樹形図の下部では、厳密な階層構造に興味はない。
- 最初のほうのクラスタリングでは、厳密に行わないでもよい（それでも目的に適っている “fit for purpose”）
- 最初のほうのクラスタリングでは、最初の距離のみだけを考慮し、結合したものの距離を再計算しなければ、計算は速くなる。

いくつかの２段階法はすでに提案されている

Bioinform Biol Insights. 2016; 10: 237–253.
Published online 2016 Nov 30. doi: 10.4137/BBI.S38316
PMCID: PMC5135122
PMID: 27932867

Clustering Algorithms: Their Application to Gene Expression Data

Jelili Oyelade,[1,2,*] Itunuoluwa Isewon,[1,2,*] Funke Oladipupo,[1] Olufemi Aromolaran,[1] Efosa Uwoghiren,[1] Faridah Ameh,[1] Moses Achas,[3] and Ezekiel Adebiyi[1,2]

CHAMELEON：最初のほうで計算が簡単なクラスタリングを行い、その後、2段階目で通常の階層型クラスタリングを行う。

私が提案する折衷型Ward法

- VP木（バンテージポイント木）の利用：VP木は、JMPのいくつかのプラットフォームですでに利用されている。
- 各データ点に対して、k近傍点（k = 10）を挙げる。
- 全部でk ×n 個の距離を、短い順に並べる。
- 半分（k ×n / 2）の距離について次の計算を行う。
  - 各ペアについて、まだどことも結合していないもの同士に限り、結合させて新しいクラスターとする。
- そうして得られたクラスターの個数が十分に小さければ、それらのクラスターからWard法を行う。
- もしクラスターの個数が多いならば（約400個以上あれば）、上記の計算サイクルを、現在の結合したクラスターに対して実行する。

折衷型Ward法： 良さげな感じがするが…

- VT木はマルチスレッドにはできないが、計算は速い。
- kd木と異なり、VP木は一度にすべての変数の情報を用いる。そのため、何千もの変数がある場合、kd木よりも効率的。
- 各点のk近傍を求める処理は、マルチスレッドにできる。
- 樹形図の上部にはWard法を適用するので、厳密に階層構造が担保される。樹形図の下部では、結合されたデータ点は近傍にあるので、まあ十分だろう。
- 良いアイデアだ！ 何か悪い点があるだろうか？
- マーフィーの法則：「失敗する可能性があるものは失敗する」

問題点「人気がありすぎるデータ点」

- Baron_countPhenoCombined.jmp: 20,124列、7,266行；列でクラスタリングする。
- 20,124列に対して10個の近傍点（つまり、全部で201,240組の距離）。同じ組は削除し、距離が短い順に並び替え、そのうちの半分だけを抜き出す。93,265組のペア。
- 20,124列の多くのものが結合することを期待したが、2,563点しか結合しなかった。
- 多くの点は、「人気がある」。他の多くの点の10近傍点に含まれている点がいくつもある。
- 9460番目の列は「人気があり」、6,556列において10近傍点に含まれている。 18124番目の列も「人気がある」。
- このようなKevin Bacon点がいくつもある。
- １つのサイクルにおいて、１つの近傍点は1回の結合しか許されていない。

10近傍点に含まれている回数

列番号	度数
9460	6556
18124	6549
18123	6394
9279	6140
4523	5770
2675	5538
18325	5195
11387	4870
19218	4416
18285	3762
16811	2717
9364	104
7860	101
598	100
4155	98
766	96
3180	92
7189	91
13232	86
4152	85

「人気がありすぎるデータ点」

- あるデータ点に関して、なぜ、近くにあるデータ点はそれほど人気がないのに、その点自身はなぜ人気があるのか？
- たとえば、9460番目の列は 18399番目の列に近い。しかし、18399番目の列は他の列とは近くない。
- 高次元空間は、摩訶不思議。
- 9460番目の列は、他の列よりもゼロを多く含んでいる。ほとんどの列は、ゼロを多く含んでいる。そのため、 9460番目の列は、他の列よりも、多くの列との距離が短くなっている。
- 「人気がありすぎるデータ点」があるため、VP木による近隣クラスタリングでは、クラスター数がそれほど少なくならない。

列番号	度数
9460	6556
18124	6549
18123	6394
9279	6140
4523	5770
2675	5538
18325	5195
11387	4870
19218	4416
18285	3762
16811	2717
9364	104
7860	101
598	100
4155	98
766	96
3180	92
7189	91
13232	86
4152	85

「人気がありすぎるデータ点」

この問題はデータ依存である。

事前にこの問題が生じるかどうかは分からない。

折衷型Ward法の例

Baron_countPhenoCombined.jmp: 行のクラスタリング

折衷型Ward:

サイクル	n	VP作成	VP探索	結合数
1	7266	8.599	10.618	1645
2	5621	6.284	6.163	1527
3	4094	4.362	3.746	1180
4	2914	2.862	1.992	868
5	2046	1.865	1.209	643
6	1403	1.201	0.751	479
7	924	0.721	0.369	339
8	585	0.428	0.172	226
9	359	0.23	0.072	137

20,124遺伝子, 7,266サンプル

高速Ward法:
KD作成時間 =2.116; クラスタリング時間=287.068; 消化時間=0.342
総計算時間=289.604秒 (4.8分)　ほぼ5倍

折衷型計算時間 = 52.163; 結合数=7,044; 残ったクラスター数=222
KD作成時間=0.041; クラスタリング時間=2.18; 消化時間=0.01
総計算時間 = 66.267秒 (1.1分)

この例は成功

Baron_countPhenoCombined.jmp 列のクラスタリング

折衷型Ward:

サイクル	n	VP作成	VP探索	結合数
1	20124	9.15	129.138	1696
2	18428	8.228	109.908	1254
3	17174	7.642	97.04	953
4	16221	7.334	87.131	712
5	15509	7.068	80.138	587
6	14922	6.68	73.187	488
7	14434	6.288	66.886	436
8	13998	6.153	62.178	400
9	13598	5.885	58.241	370
10	13228	5.806	54.467	364
11	12864	5.621	50.918	337
12	12527	5.615	48.05	324

折衷型計算時間=999.87; 結合数=7,921; 残ったクラスター数=12,203

KD作成時間=1.514; クラスタリング計算時間=1,039.388 =17.3分　消化時間=0.259

注意: 折衷型クラスタリングではクラスター数を減らせられなかった。高速Ward法に切り替え。

KD作成時間=2.479
クラスタリング計算時間=2,273.327;
消化時間=0.399
総計算時間=2,276.281秒 =37.9分

chr10_emaize_wide_num2.jmp

135,608列、 6,210行
折衷型による２方向クラスタリング：324分 (5.4時間)

サイクル	n	VP作成	VP探索	結合数
1	6210	50.743	534.356	1349
2	4861	36.482	351.139	1257
3	3604	26.072	201.931	1048
4	2556	17.637	102.235	787
5	1769	11.256	48.962	556
6	1213	7.1	22.938	522
7	691	3.705	7.267	274
8	417	2.124	2.538	199

折衷型計算時間=1,428.044; 結合数=5,992;
残ったクラスター数=218
KD作成時間=0.295; クラスタリング時間=23.388; 消化時間=0.068

この例は成功

サイクル	n	VP作成	VP探索	結合数
1	135608	68.708	8533.715	30131
2	105477	53.394	5214.711	34713
3	70764	35.243	2388.443	26955
4	43809	19.628	900.824	16331
5	27476	12.507	352.271	10718
6	16757	6.894	131.908	6344
7	10413	4.144	50.558	3783
8	6630	2.576	20.567	2207
9	4423	1.534	8.8	1373
10	3050	0.94	4.127	960
11	2090	0.612	1.899	754
12	1336	0.365	0.764	550

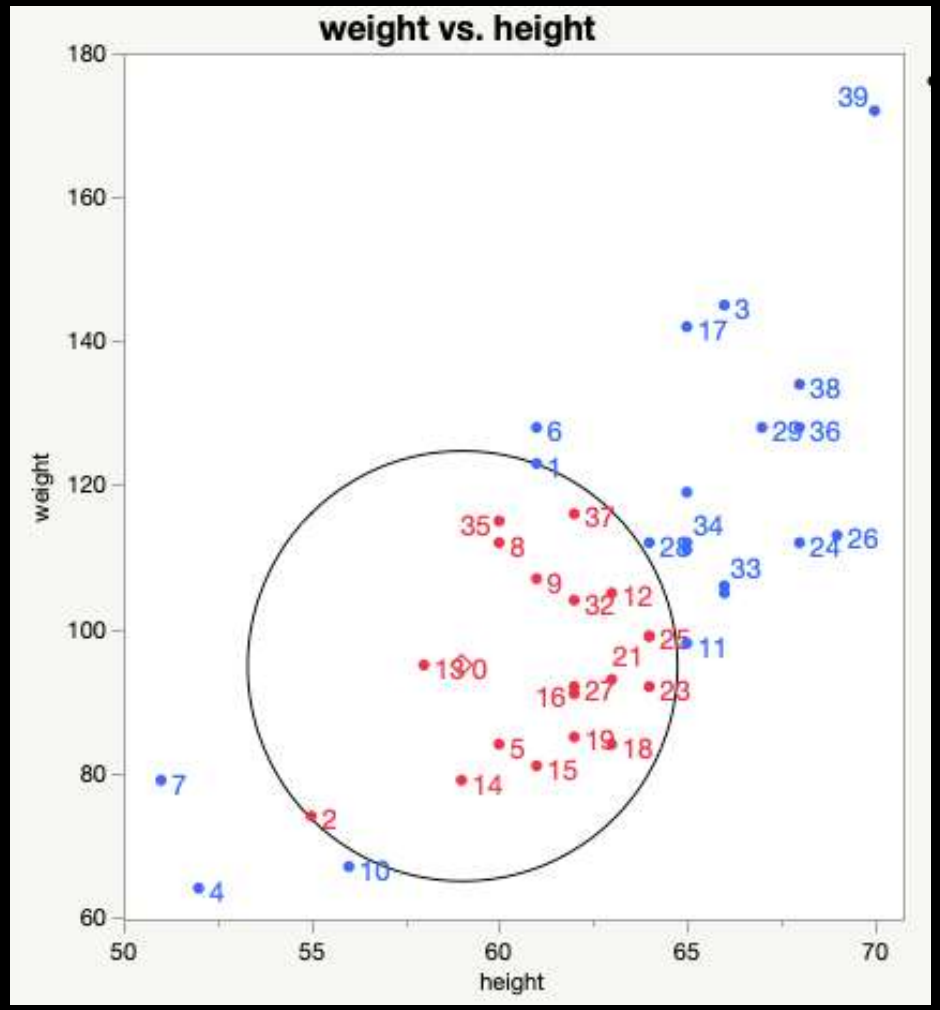
折衷型計算時間=17,817.22; 結合数=134,819; 残ったクラスター数=789
KD作成時間=0.036; クラスタリング時間=12.558; 消化時間=0.011
総計算時間 = 19,486.217

折衷型Ward法： 失敗することが多すぎる…

データ名	行クラスタリング	列クラスタリング
Baron	成功	失敗
EMaize	成功	成功
mmap	成功	成功
combG	成功	成功
bCancer	成功	失敗
mice	成功	失敗
gtex	失敗	失敗
Morocco	失敗	失敗

VP木: バンテージポイント木

- 左ノード： 円内
- 右ノード： 円外



2次元だとVP木のありがたみはないが、高次元だと効率的。

rootNode 0 1.34639
nearNode 20 0.548669
nearNode 23 0.471438
nearNode 21 0
farNode 25 0.715559
farNode 18 0
farNode 27 0.315289
nearNode 16 0
farNode 19 0.931155
farNode 12 0
farNode 32 1.01673
nearNode 9 0.430582
nearNode 8 0
farNode 35 0.473585
farNode 37 0
farNode 5 0.326008
nearNode 15 0
farNode 14 0.758231
farNode 13 1.180989
farNode 2 0
farNode 1 1.46866
nearNode 30 0.358604
nearNode 34 0.235719
farNode 28 0.543404
farNode 33 0
farNode 31 1.062428
nearNode 22 0
farNode 29 0.787326
farNode 17 1.134303
farNode 6 0
farNode 11 2.130022
nearNode 24 0.72066
nearNode 26 0
farNode 36 0.270247
farNode 38 0
farNode 3 4.230726
nearNode 39 0
farNode 10 0.952509
farNode 4 0.715559
farNode 7 0

VP木:バンテージポイント木

- 多くの計算科学者たちが、VT木に関心をもってきた。「探索」にはビジネスチャンスがある？

Near Neighbor Search in Large Metric Spaces

Sergey Brin*

Department of Computer Science

Stanford University

November 20, 1995

Abstract

Given user data, one often wants to find approximate matches in a large database. A good example
images similar to a given image in a large collection of images. We focus on the important and tech
where each data element is high dimensional, or more generally, is represented by a point in a large
distance calculations are computationally expensive.

In this paper we introduce a data structure to solve this problem called a GNAT -- Geometric Near-
based on the philosophy that the data structure should act as a hierarchical geometrical model of the
simple decomposition of the data which doesn't use its intrinsic geometry. In experiments, we find t
previous data structures in a number of applications.

Keywords -- near neighbor, metric space, approximate queries, data mining, Dirichlet domains, Vor

5.2 Data Structures Supported

The final implementation supports a number of data structures including GNAT's[7].

VP-Trees -- See Section 2 for a brief description. In the tests presented here, we did not use any sampling technique to chose vantage points since we could not be sure that we would do it identically to [Yia93]. However, some limited tests with sampling indicated that savings were in the 10% range for images and were negligible for text and random vectors.

VP$_k$-Trees A generalization of VP-Trees which differs in that at each node, instead of the remaining data points being split into two halves based on their distance from the vantage point, they are split into k sections of equal size (also based on distance from the vantage point). These were found to perform very similarly to vp-trees (sometimes a little better even) but there was not a sufficiently large difference to warrant further investigation.

GH-Trees -- See Section 2 for a brief description. They are essentially GNAT's of constant degree 2 without the sampling for split points and degree variation throughout the tree. Since they perform worse than GNAT's of degree 2, not many experiments were performed.

OPT-Trees -- These use a much smaller number of distance computations for queries than any other structure but they lose out by having far more costly other computations (even superlinear in the number of data points). The idea here is to pick a number of vantage points. Measure the distances from each to all the other points and store these in a table. When a query comes along, measure its distance to the first vantage point and based on that weed out all of the impossible data points. Then take the next vantage point and do the same. Continue until no more data points are pruned. Then, check each of the remaining ones individually. Up to the choice of vantage points this gives more or less the optimal performance, in terms of distance calculations. This structure can serve as a lower bound for distance calculations but is not a realistic goal to shoot for if one wants a scalable structure.

GNAT's -- This is the main data structure of this paper, described in Section 4 .

For each of these, we check how many distance calculations are used to both build the structure and perform queries.

VP木：
最近隣連鎖法で kd木の代わりにVP木が使えないか？

- 高次元の場合、VP木はkd木よりも効率的。 KD木は、1回につき1変数しか見ない。分散が大きなものを1つずつ見ていく。また、VP木は探索においてマルチスレッドが使える。
- 結合の際に、木から２つのノードを削除し、木に１つのノードを追加する必要がある。
- VP木は、ノードを削除するのが非効率。
- よって削除するのではなく、それらを「引退」させる。つまり、探索において、候補に挙げなくする。この「引退」を木が疎になりすぎて、探索に時間がかかるよういなるまで行う。
- しかし、結合したクラスターを追加する処理も必要。

折衷型クラスター法: 力ずくの解決法

「人気がありすぎるデータ点が多いこと」への対策:

- 対策１：K近傍を求める際に、より大きなK にする（ただし、Kは150以下）。クラスターの減りが弱いなら、Kを適応的に増加させる。
- 対策２：現在のカテゴリー数の半分にならなかったならば、近傍点の半分ではなく、それ以上（たとえば2/3）を候補に挙げる。
- 対策３：サイクルの回数を増やす。

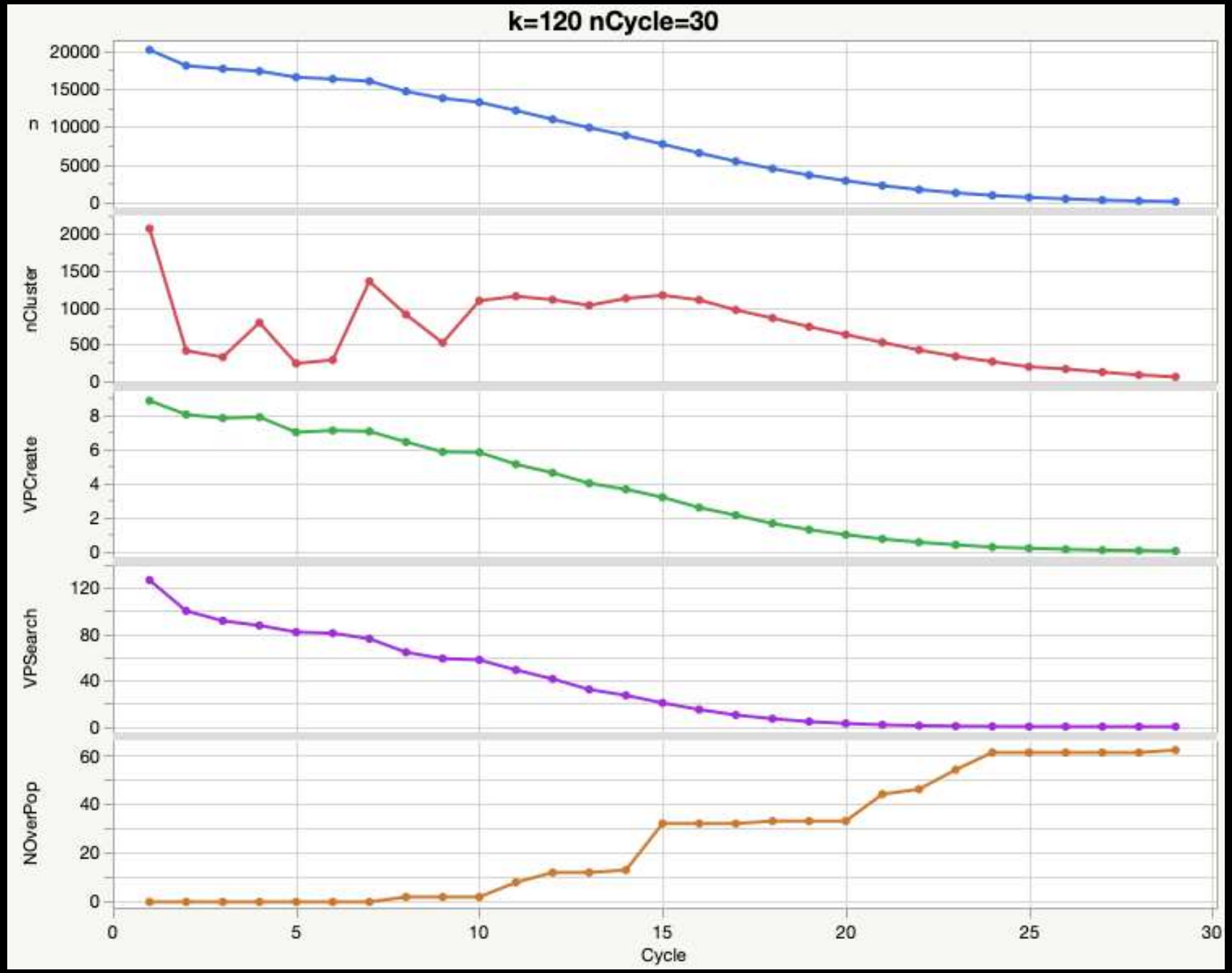
折衷型Ward: より力ずく

Baronの転置データ

20,124行, 7266列

k=120, サイクル数=30

総計算時間 = 1,156.25秒
19.27分



折衷型Ward: より力ずく

Baronの転置データ

20,124行, 7,266列

k=120, サイクル数=30

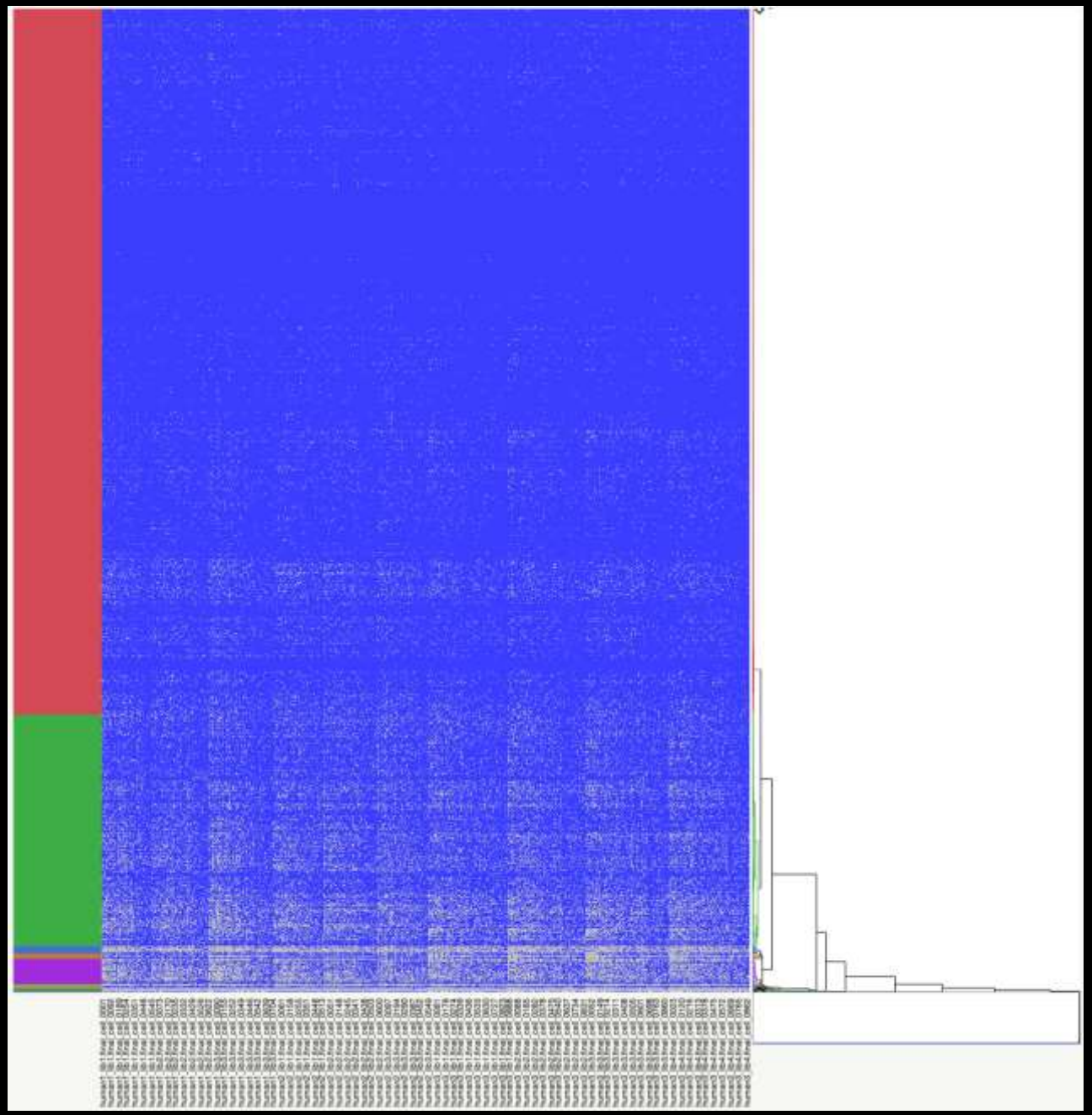
データの分布に注目！

１つの大きなクラスター（赤色＆緑色）と、多くの小さなクラスター。

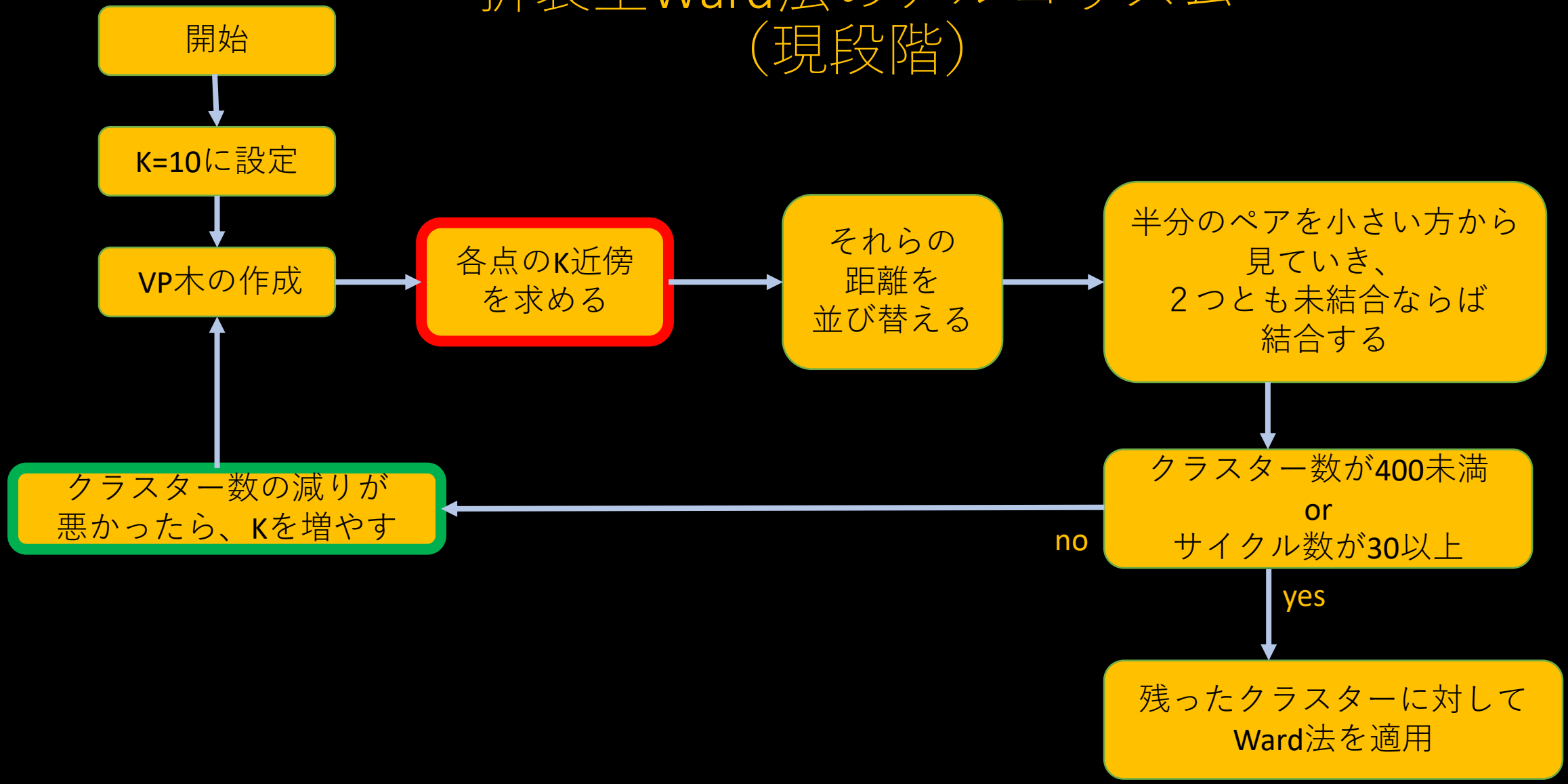
右図は、クラスター数を 50にした結果。

１つの大きなクラスターが２つ（赤色と緑色）に分かれるには、クラスター数を44個まで下げなければいけない。

Level	Count	Prob
1	14431	0.71710
2	4743	0.23569
3	153	0.00760
4	106	0.00527
5	10	0.00050
6	509	0.02529
7	1	0.00005
8	10	0.00050
9	3	0.00015
10	46	0.00229
11	1	0.00005
12	2	0.00010
13	14	0.00070
14	2	0.00010
15	1	0.00005
16	1	0.00005
17	1	0.00005
18	1	0.00005
19	6	0.00030
20	2	0.00010
21	29	0.00144
22	2	0.00010
23	14	0.00070
24	2	0.00010
25	7	0.00035
26	1	0.00005
27	1	0.00005
28	1	0.00005
29	1	0.00005
30	1	0.00005
31	1	0.00005
32	1	0.00005
33	1	0.00005
34	1	0.00005
35	2	0.00010
36	1	0.00005
37	1	0.00005
38	1	0.00005
39	2	0.00010
40	1	0.00005
41	1	0.00005
42	1	0.00005
43	1	0.00005
44	1	0.00005
45	1	0.00005
46	1	0.00005
47	1	0.00005
48	1	0.00005
49	1	0.00005
50	1	0.00005



折衷型Ward法のアルゴリズム（現段階）

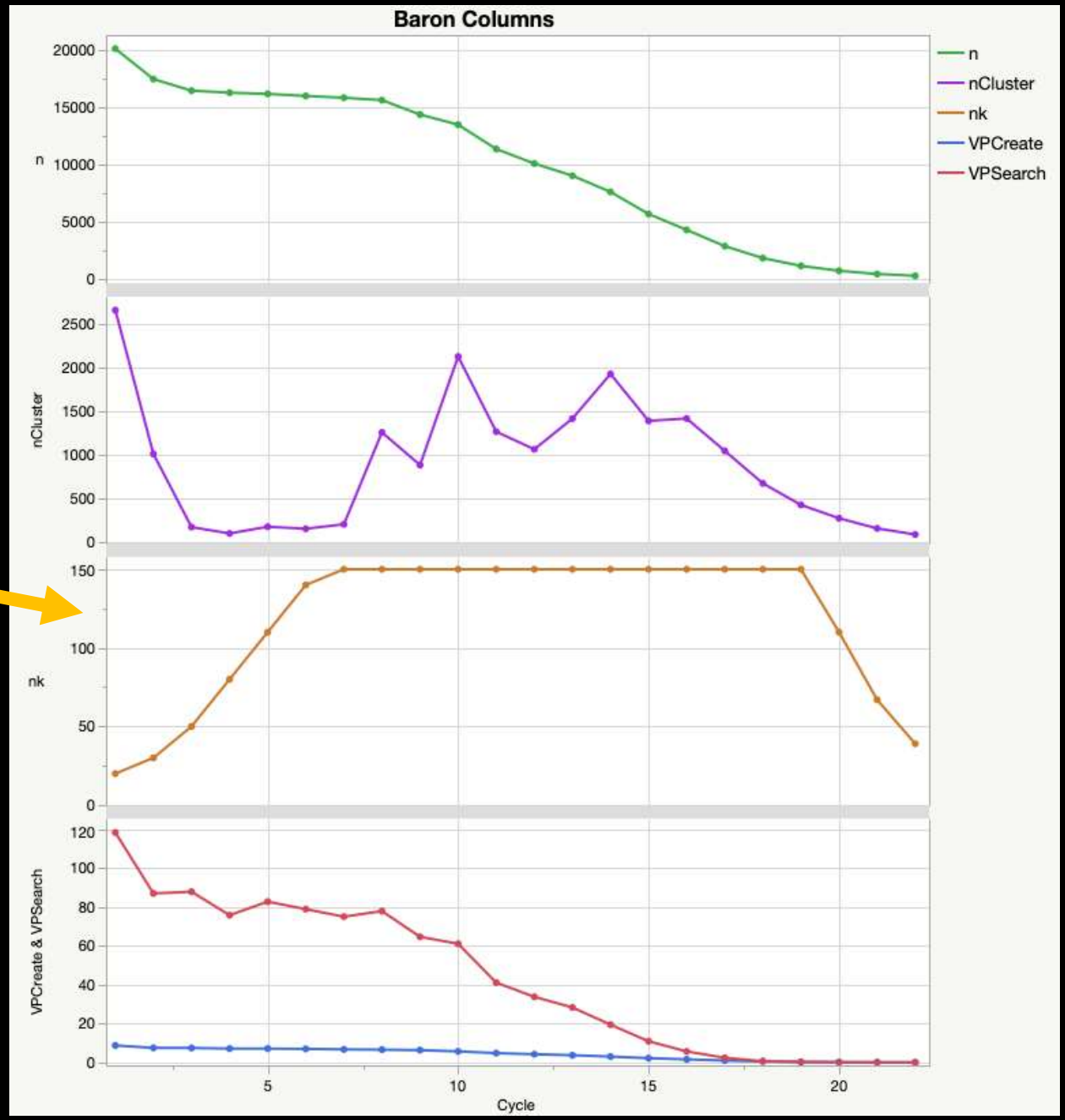


Baronデータ：
列クラスタリング

- 2方向の総計算時間:

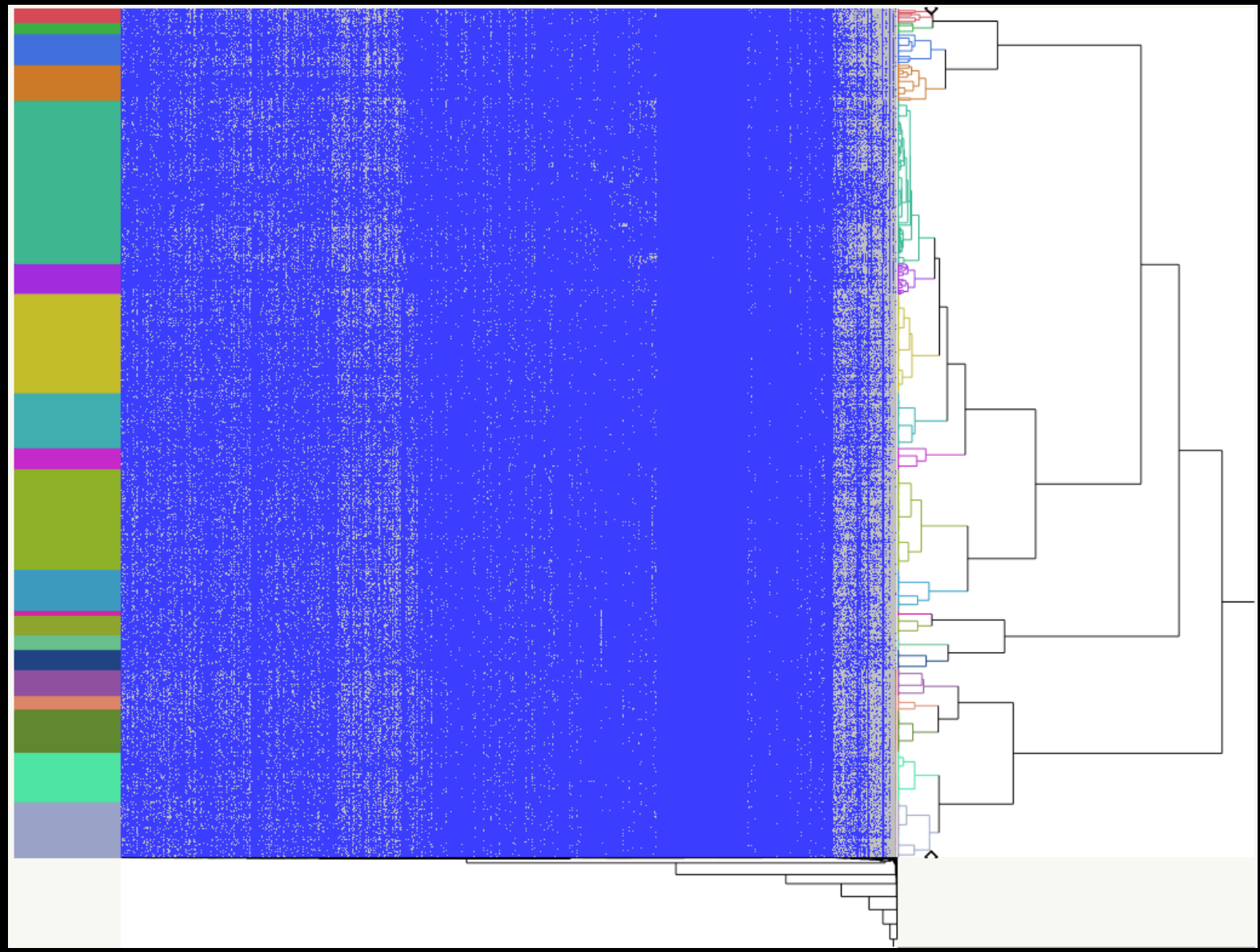
  1,099.35秒 = 18分

- Kが増加されている。150まで増加し、150に張り付いた後、徐々に減少。



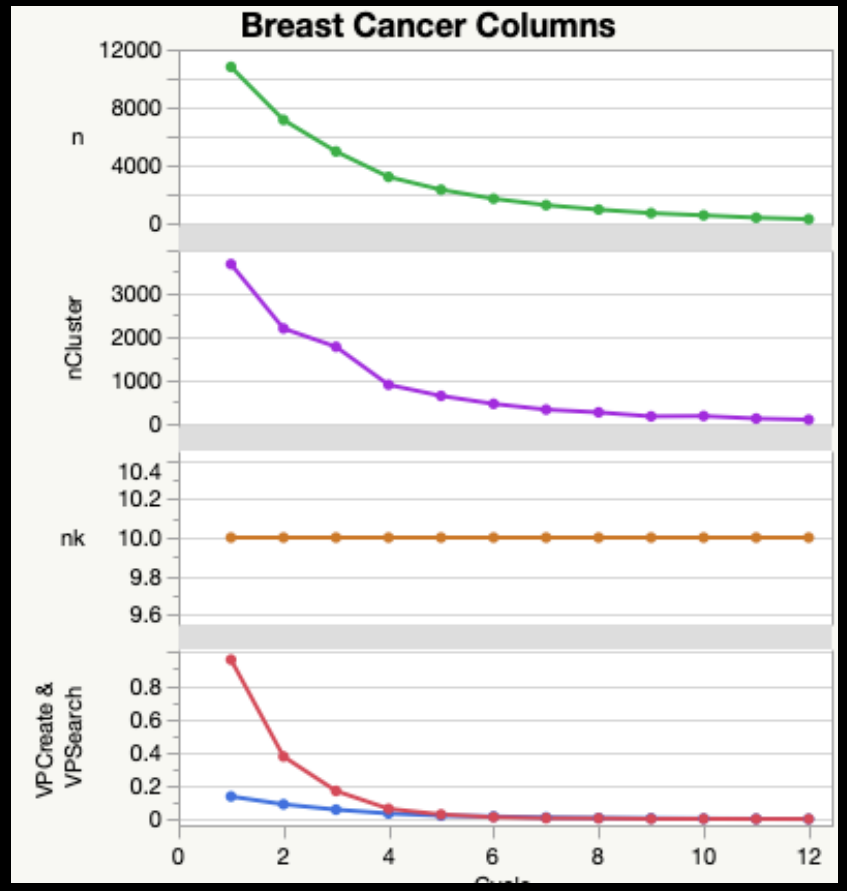
Baronデータ: 2方向

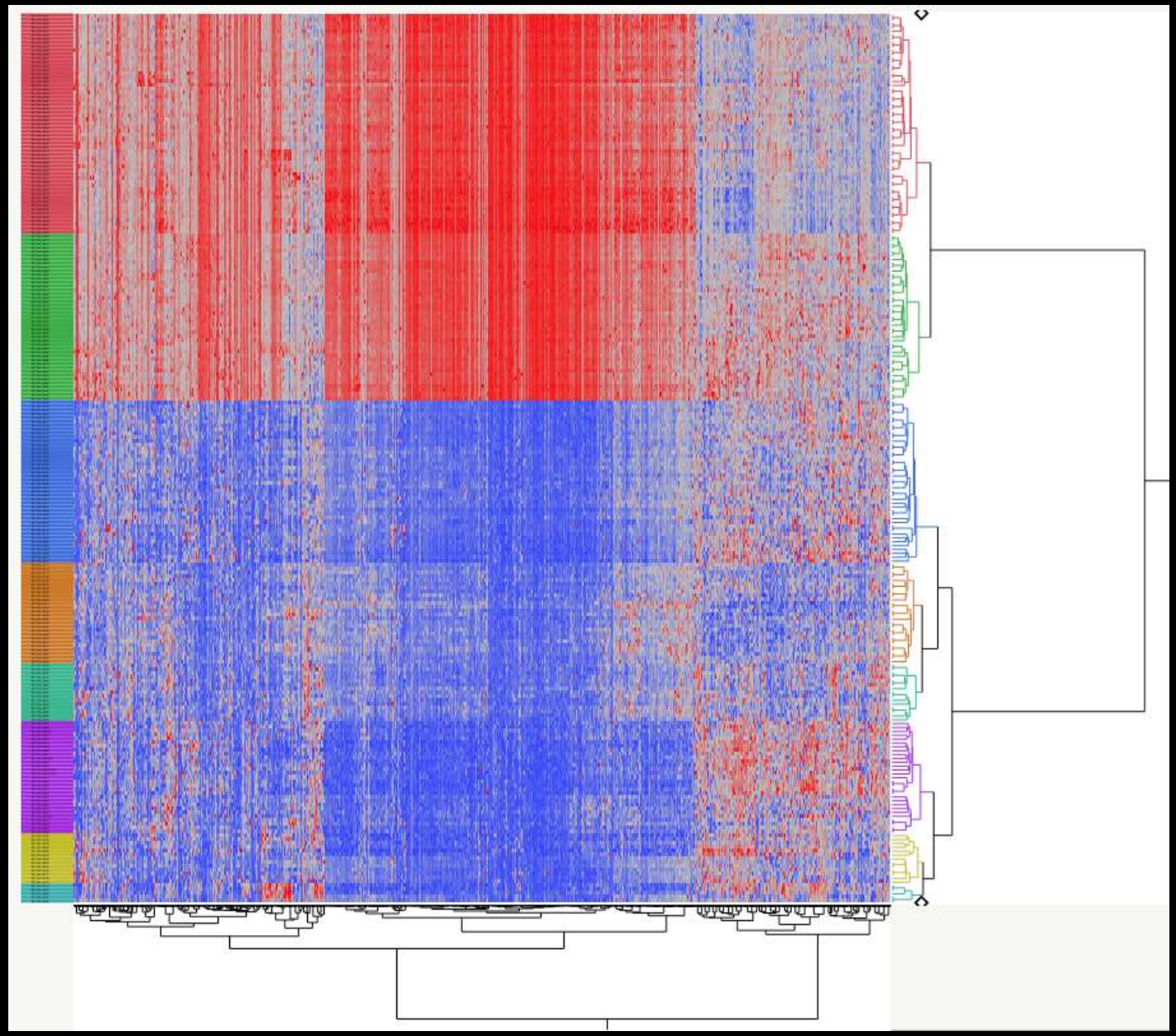
- 行の結果はバランスがとれている
- 列の結果はとてもアンバランス
- 多くの列が１つのクラスターに属しており、情報が乏しい。



breastcancer_MAQC.jmp

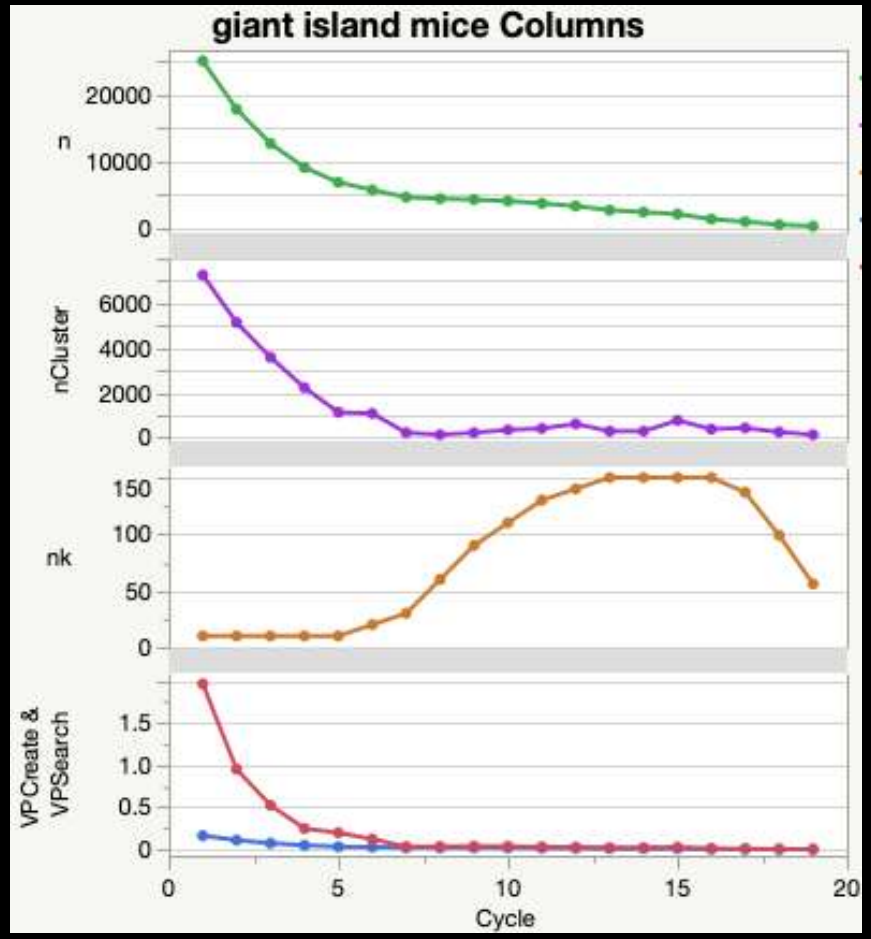
10,787 列、 230行

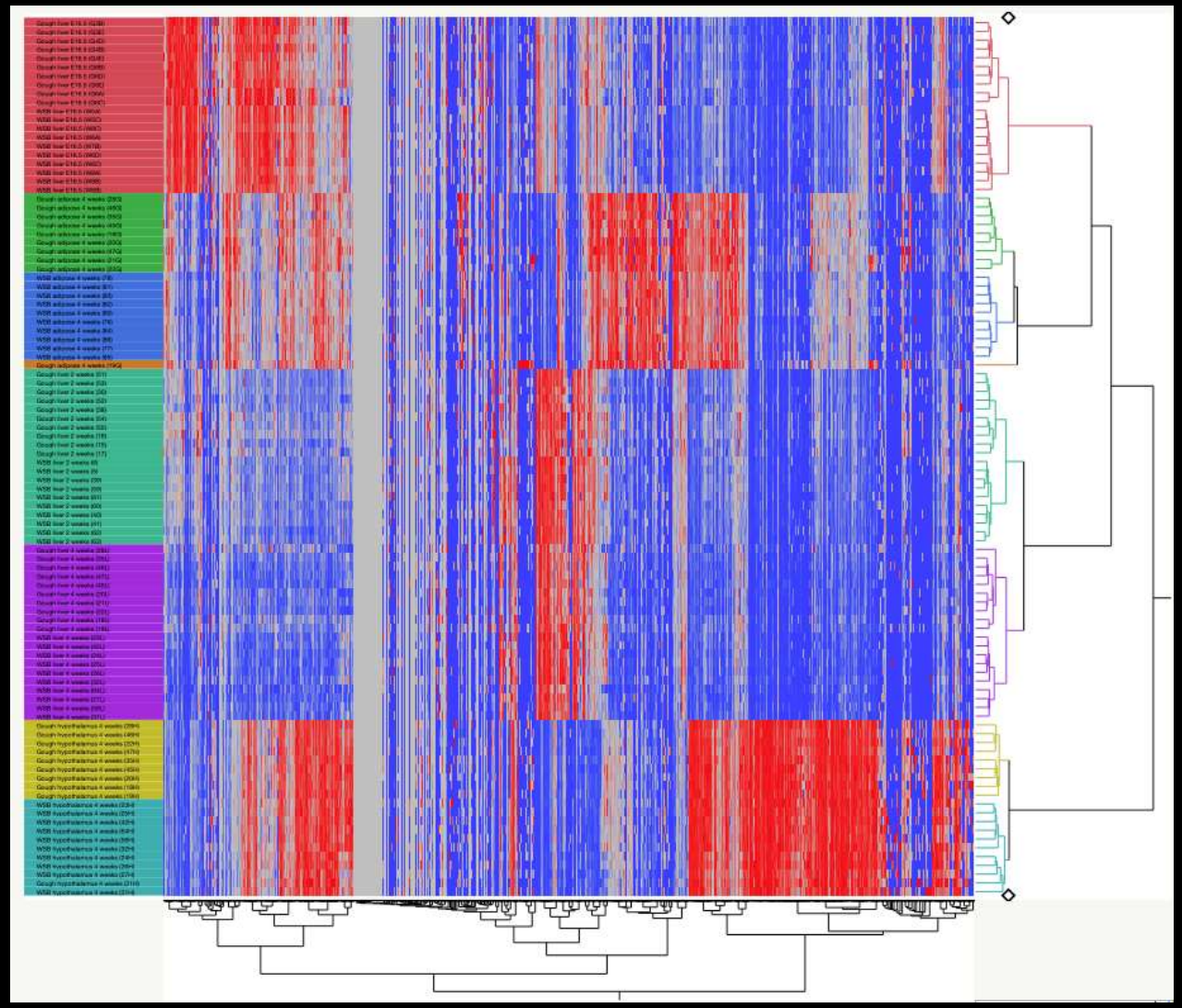




giant_island_mice_wide

25,097列、100行

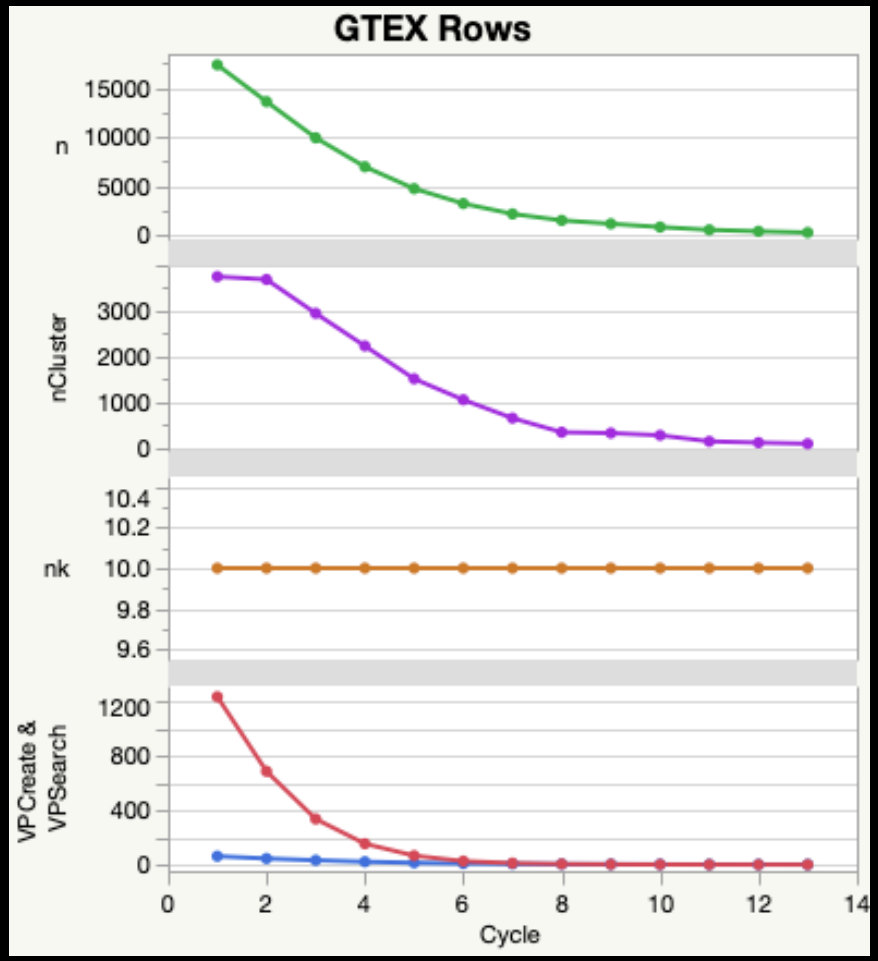


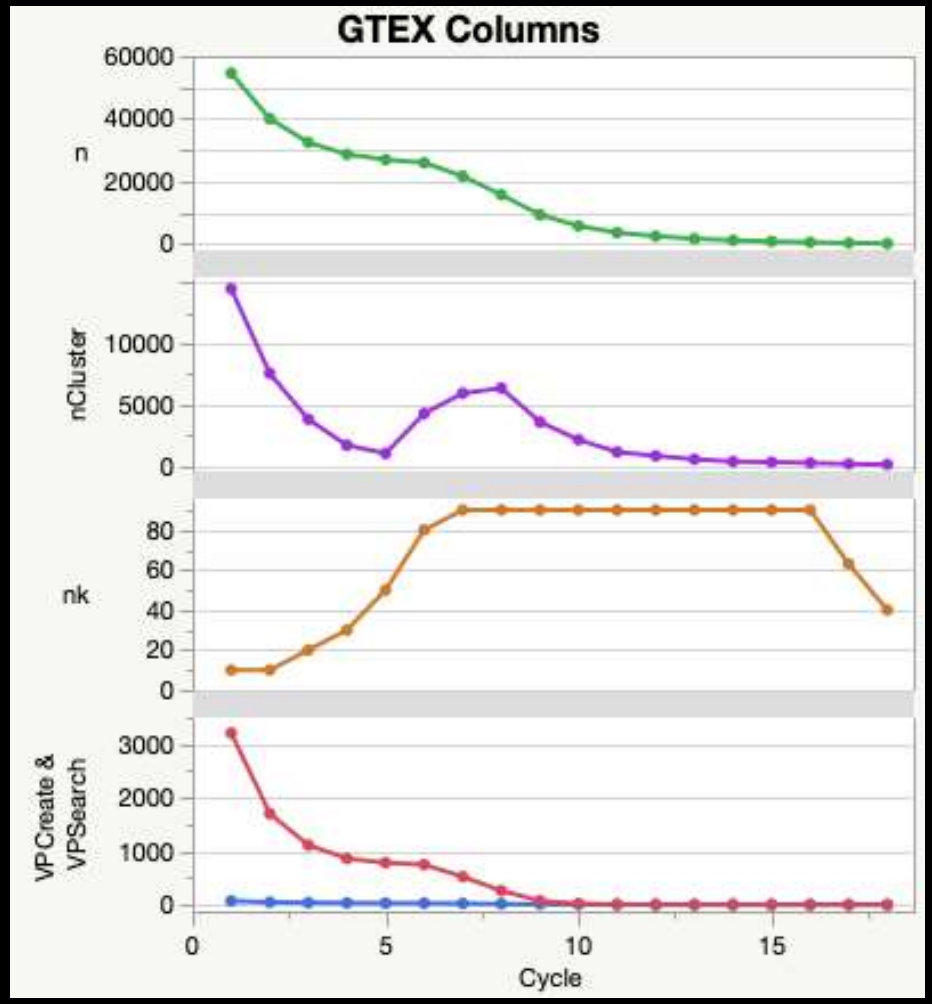


gtex_gene_tpm_log2_std2

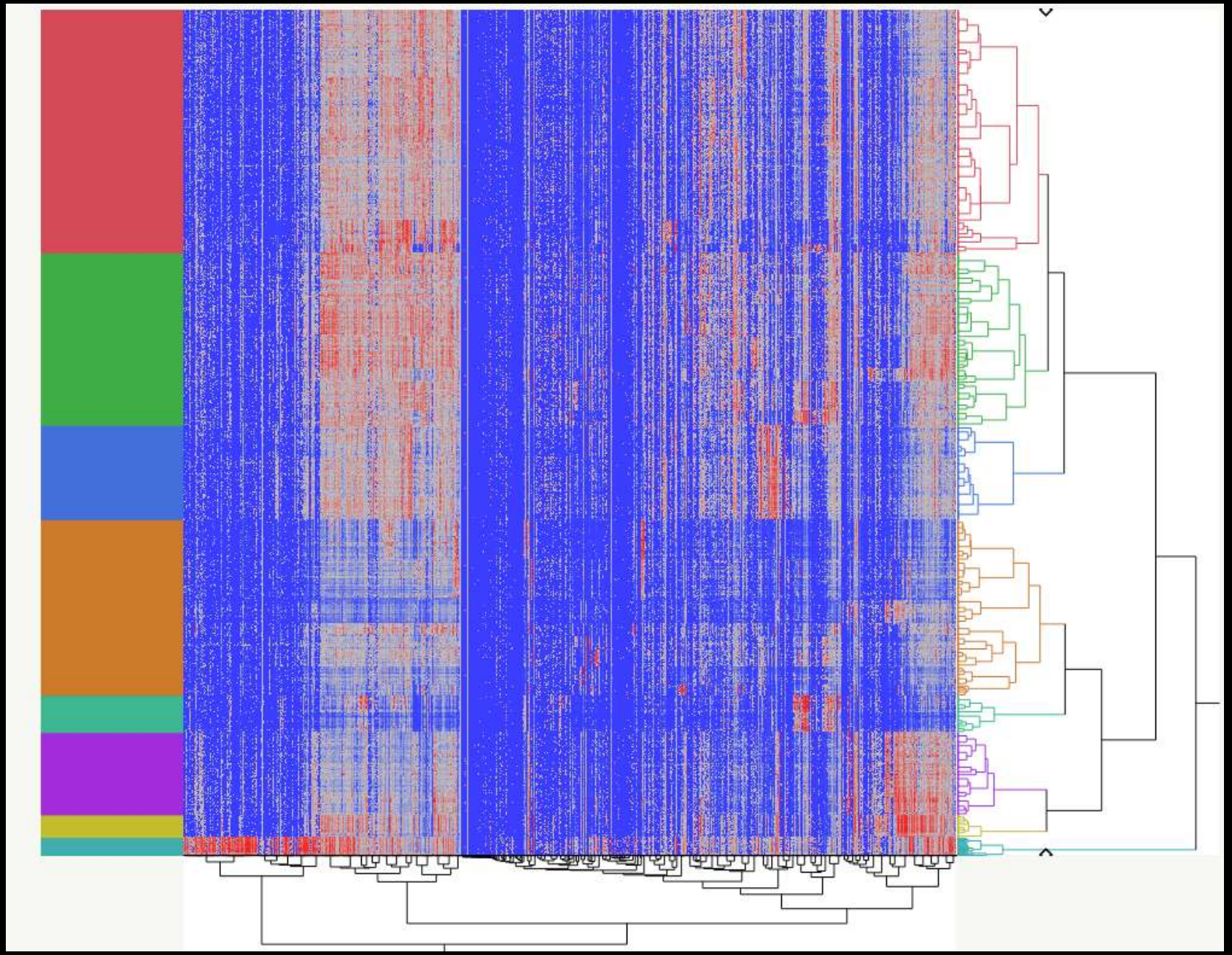
54,592 列、17,382行

総計算時間 = 12,580.4秒 = 209分 = 3.5時間



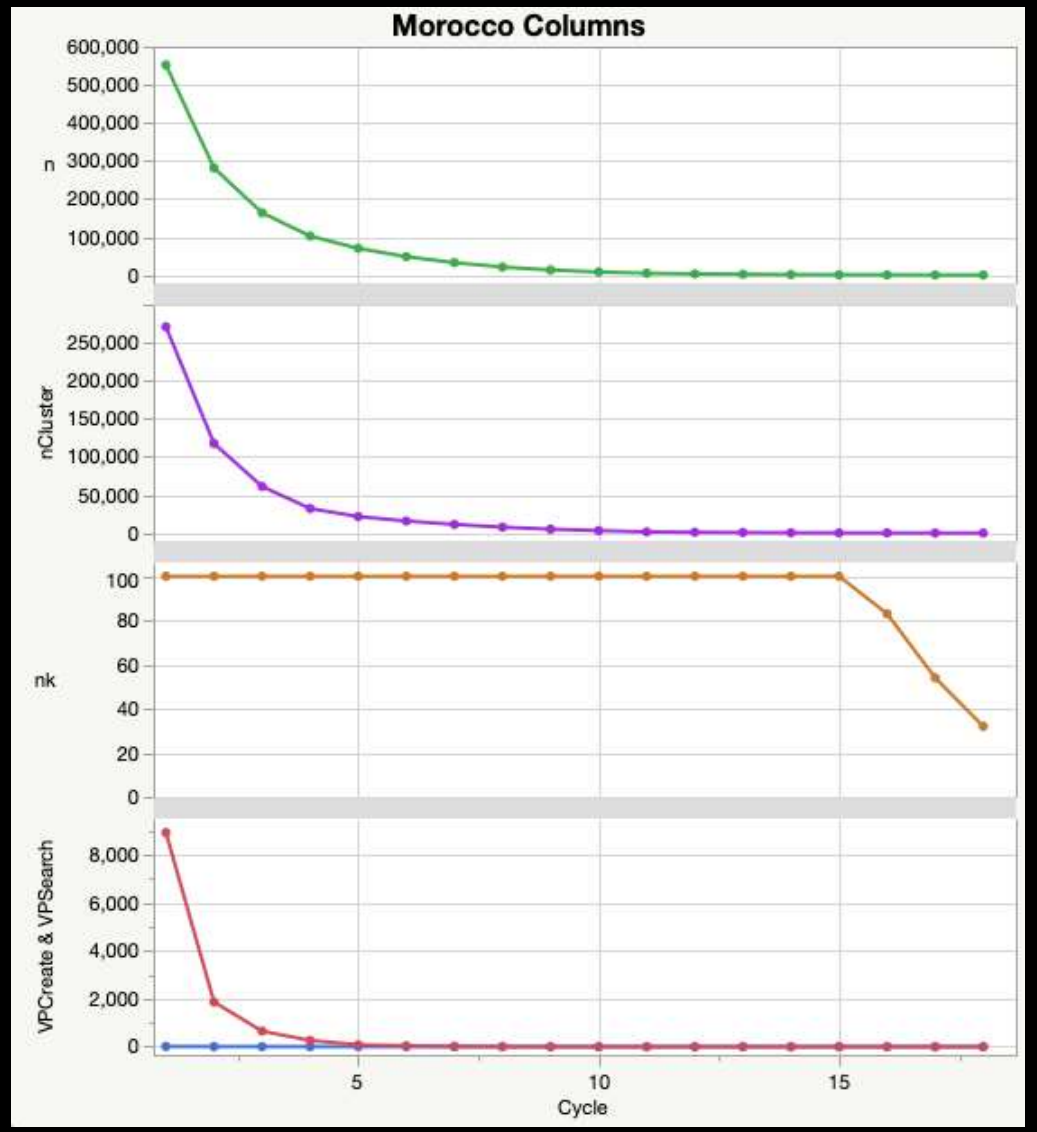


gtex_gene_tpm_log2_std2



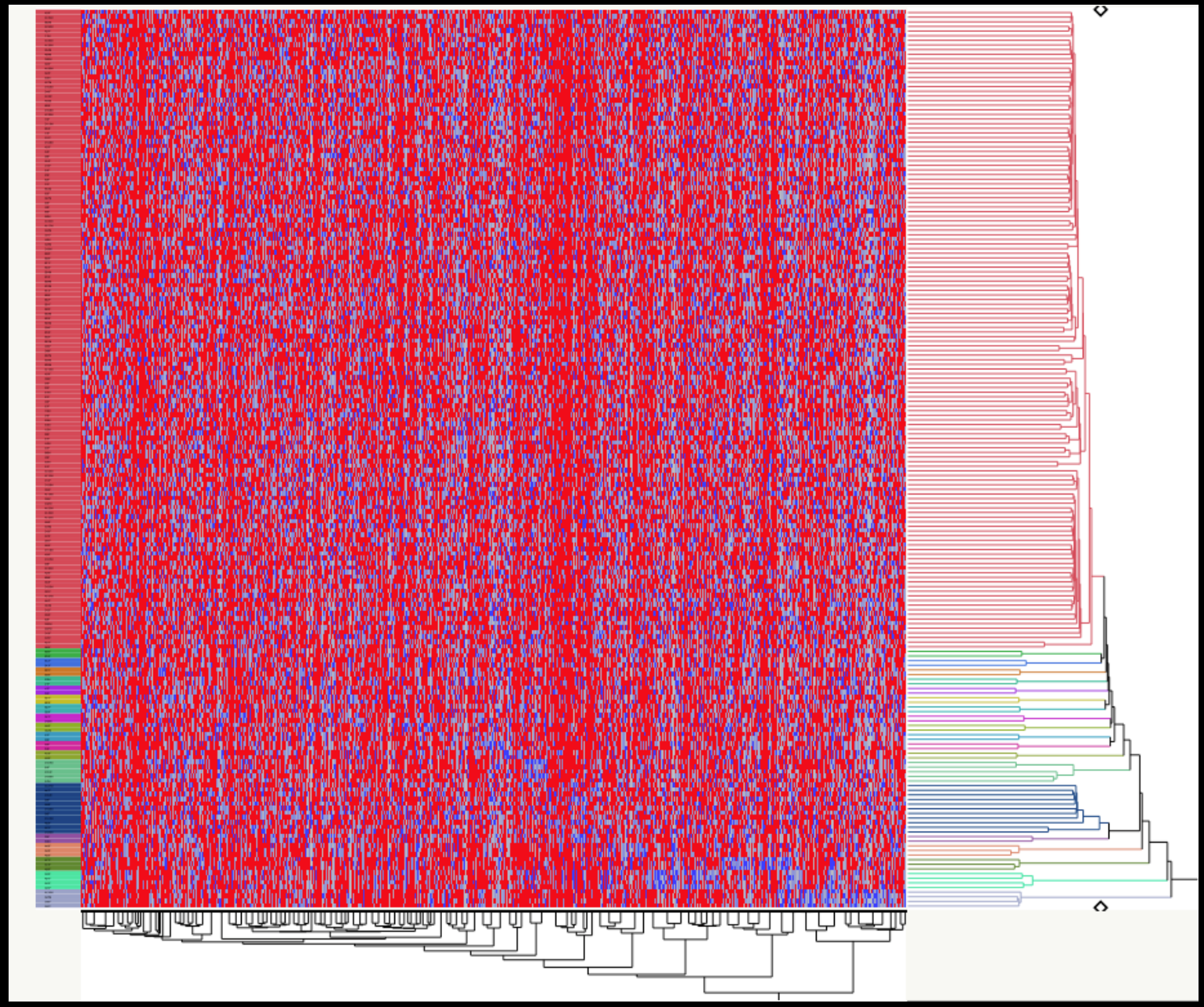
Morocco_numgeno_194_maf02_pcs imputed.jmp

- 552,145列、 194行
- 総計算時間 = 12,006秒 = 200分 = 3.3時間
- 行方向：n = 130; time = 0.001
- 列方向：n = 194; time = 11.94



Morocco_numgeno_194_maf02_pcs imputed.jmp

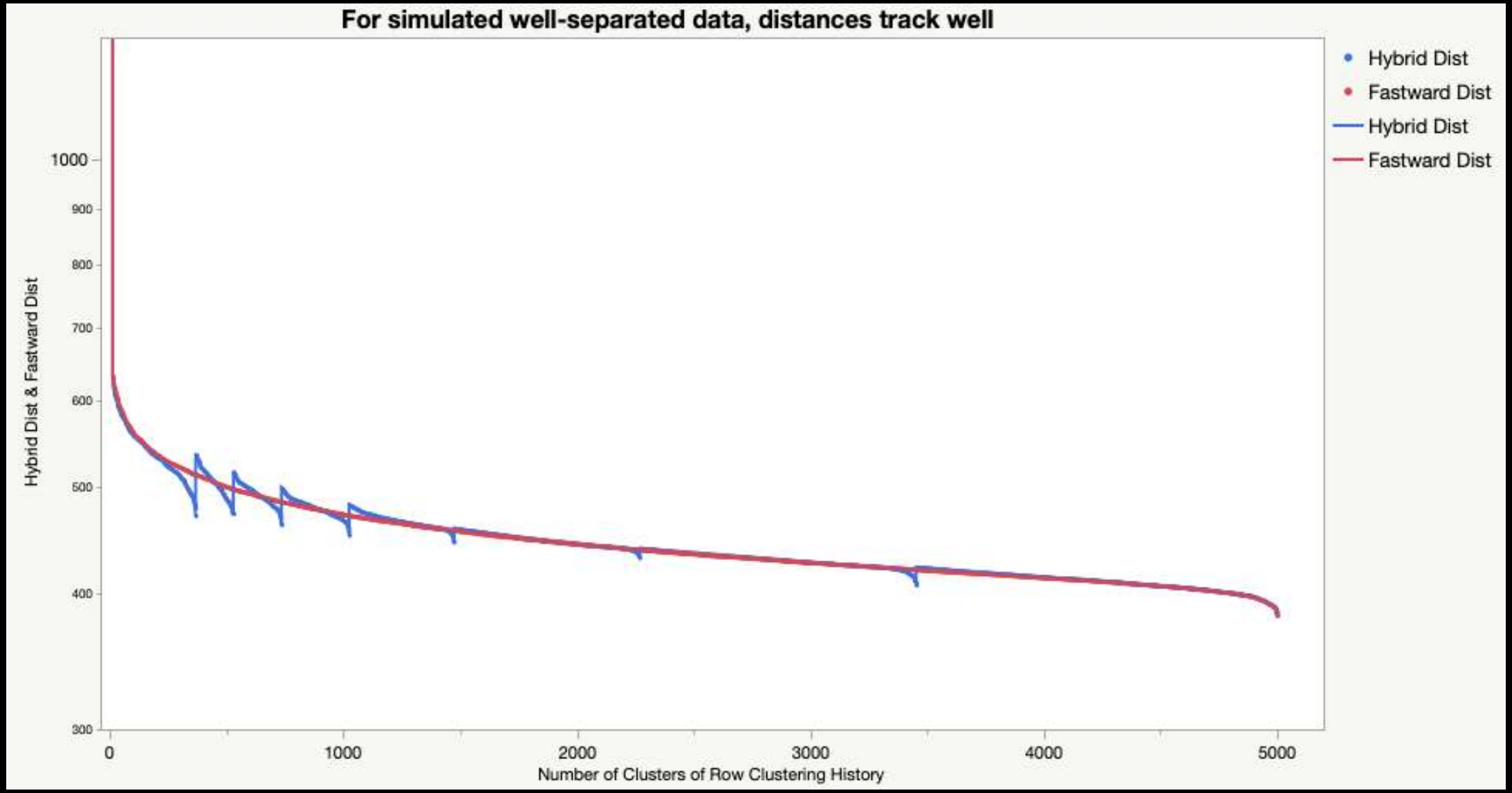
- 552,145列、 194行
- 総計算時間 = 12,006秒
  = 200分 = 3.3時間
- 列方向：n = 130; time = 0.001
- 行方向：n =194; time = 11.94



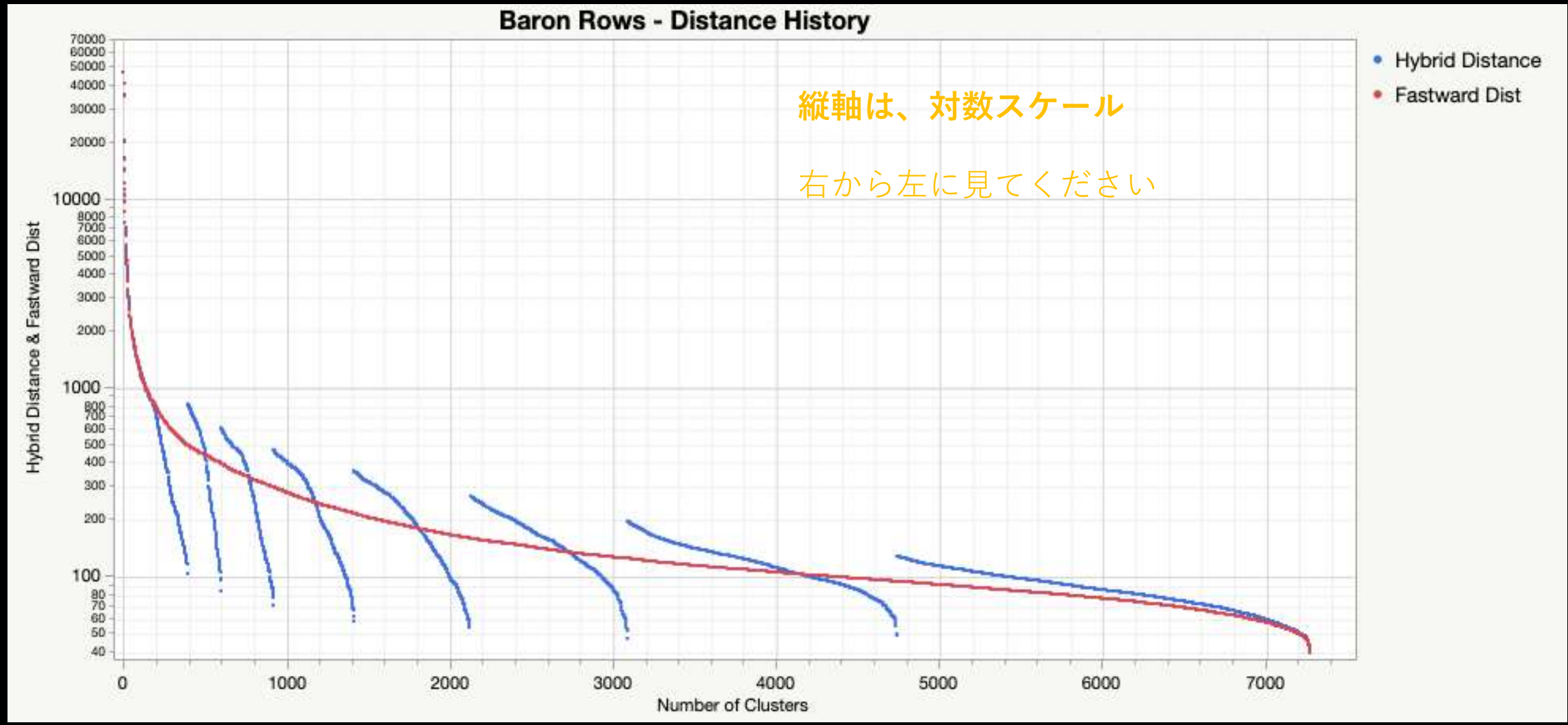
注：右図は、552,145列すべてを
示していない。

乱数で生成した
綺麗に分かれた疑似データ

折衷型Ward法（青色）と高速 Ward法（赤色）
の距離



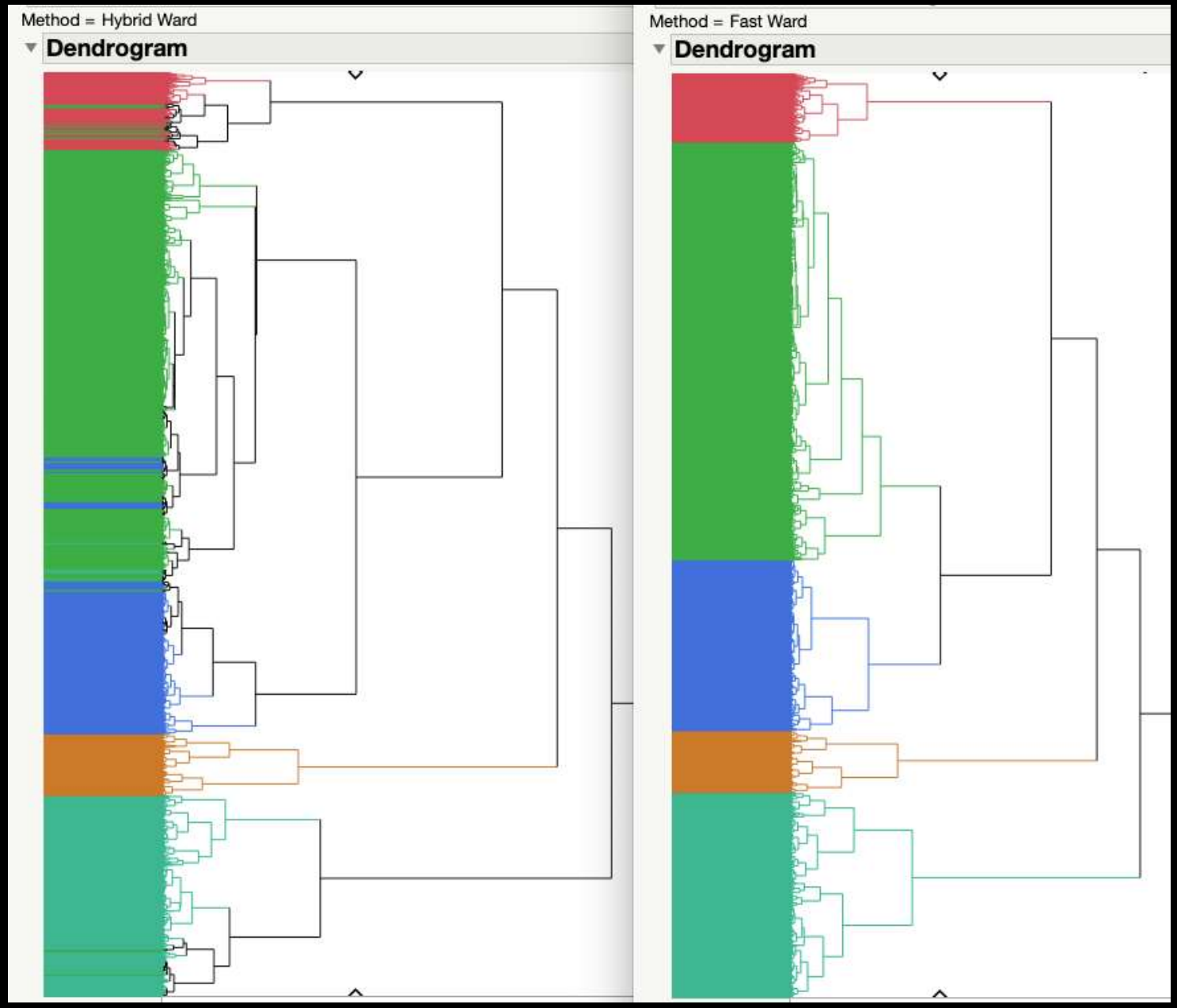
Baron_countPhenoCombined.jmp:　20,124遺伝子, 7,266サンプル



サイクルの期間を短くすれば近似が良くなるだろうけれども、
計算時間は長くなるだろう。
あとのサイクルになるほど近似は悪くなっている（計算時間は短い）。

折衷型の結合処理	46610.08
高速Ward	46653.28

Baron_countPhenoCombined.jmp: rows - 20,124遺伝子, 7,266サンプル



	距離
折衷型Ward結合	46610.08
高速Ward	46653.28

疑問： 折衷型Ward法は、Ward法とどれぐらい同じ結果になる？

5クラスターの結果は、ピボットで動かすといくらか似た結果になっている

高速Ward法の結果（図の右側）で色付けしたもの

- 折衷型Ward法は、まだ遅い。
- データの次元を減らすことができないか？
- 主成分分析（PCA）はどうか?
- 主成分分析は、データ点間の距離をなるべく近似しながらも、より低次元の空間にデータを回転する手法。
- しかし、大規模データの特異値分解（SVD）は、計算時間がかかる。
- 特異値分解を速くする方法はないか?

主成分分析のベンチマーク
GTEX 転置　54,592列、 17,382行; 行のクラスタリング
通常の折衷型Ward = 461
Lanczos法SVD（300次元主成分）: 775 + 折衷型時間: 3;　合計=778
300次元の主成分は、 元データの63%のばらつきしか説明しない

X = U*S*V`　　（特異値分解）
U*S　　行の主成分スコア
V*S　　列の主成分スコア

2011年の技術革新:
乱択特異値分解（Randomized SVD）

- Halko, N., Martinsson, P.-G., & Tropp, J. A. (2011). *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*. *SIAM Review*, *53*(2), 217–288.

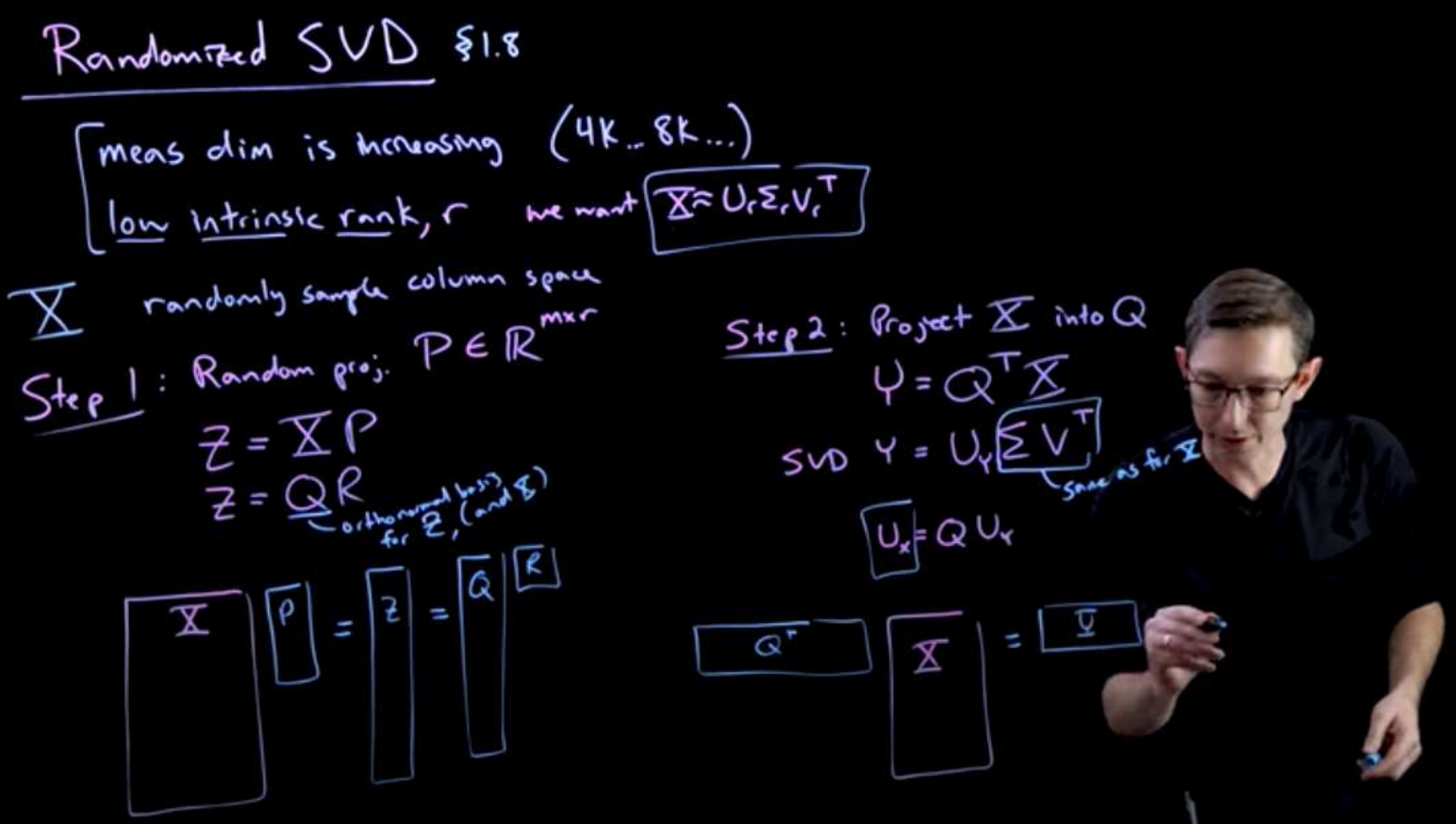
```
//n×m行列のXを、q列に縮約
P = J(m,q,Random  Normal());
Z = X*P;                    // n×q
Q = Ortho(Z);               // 列空間の直交化
Y = Q`*X;                   // q×m
{Uy,S,V} = SVD(Y);          //小さくなった行列のSVD
U = Q*Uy;                   // n×q, X ≈ U S V`
X2 = U*Diag(S);             // 行の主成分スコア
```

論文タイトル「ランダム性で構造を見つける」
矛盾した言葉

X2は、Xの主成分スコアを回転したものの近似になる。
つまり、より少ない次元で、なるべくXの行間における
距離を再現するようなスコアになっている。

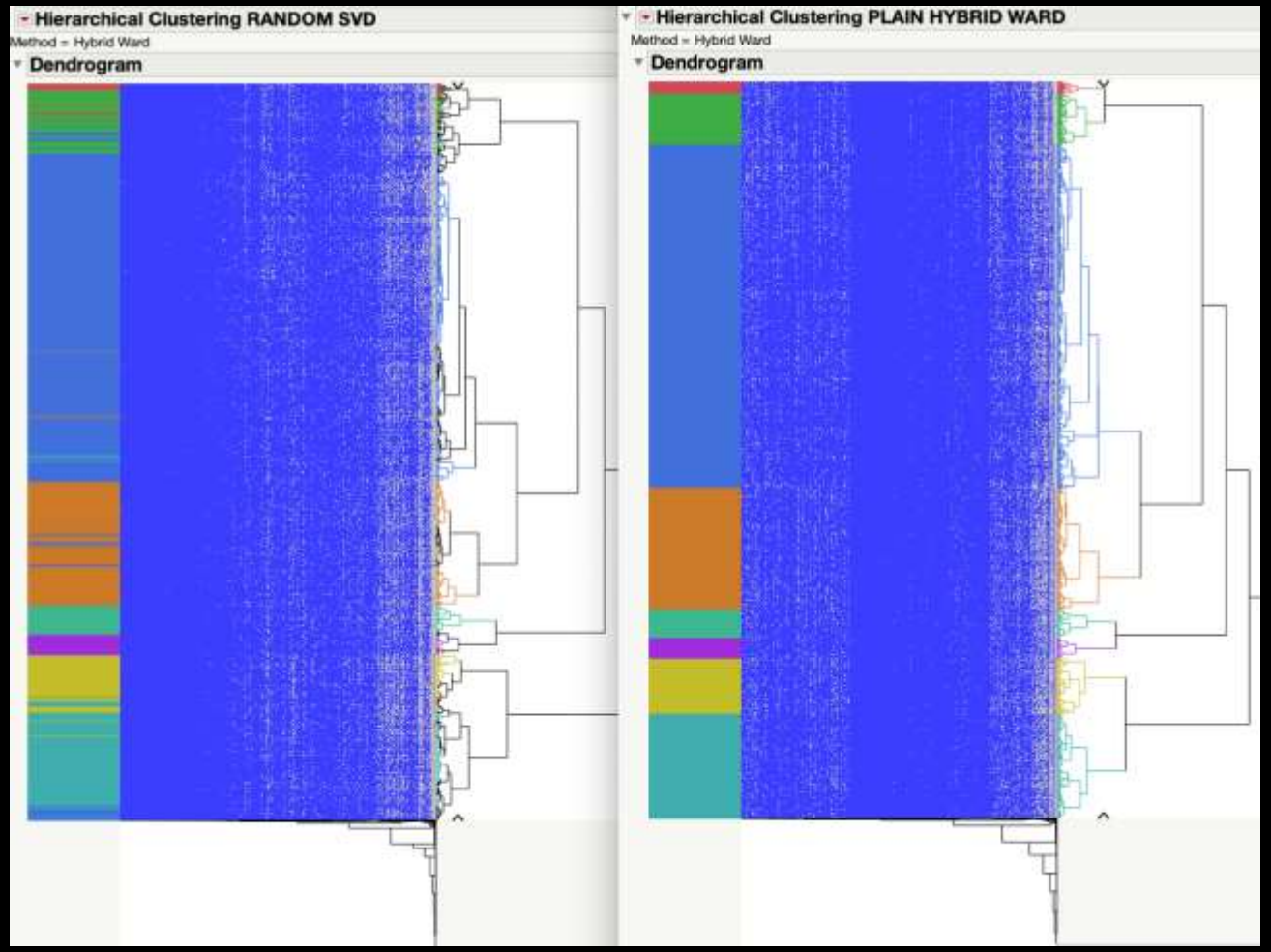
乱択特異値分解

すばらしいチュートリアル動画: Steven Brunton
「Randomized SVD Brunton」で検索してください
https://www.youtube.com/watch?v=fJ2EyvR85ro
余談：Steven Bruntonは右利き？



乱択特異値分解: 距離を近似する主成分スコア

Baronデータで（行および列ともに） 50次元だけを用いた場合



行をクラスタリングするのに、すべての列を用いる必要はない。
列をクラスタリングするのに、すべての行を用いる必要はない。
低次元にすると、計算時間が短くなる。
（直交変換は元の距離を保持。）

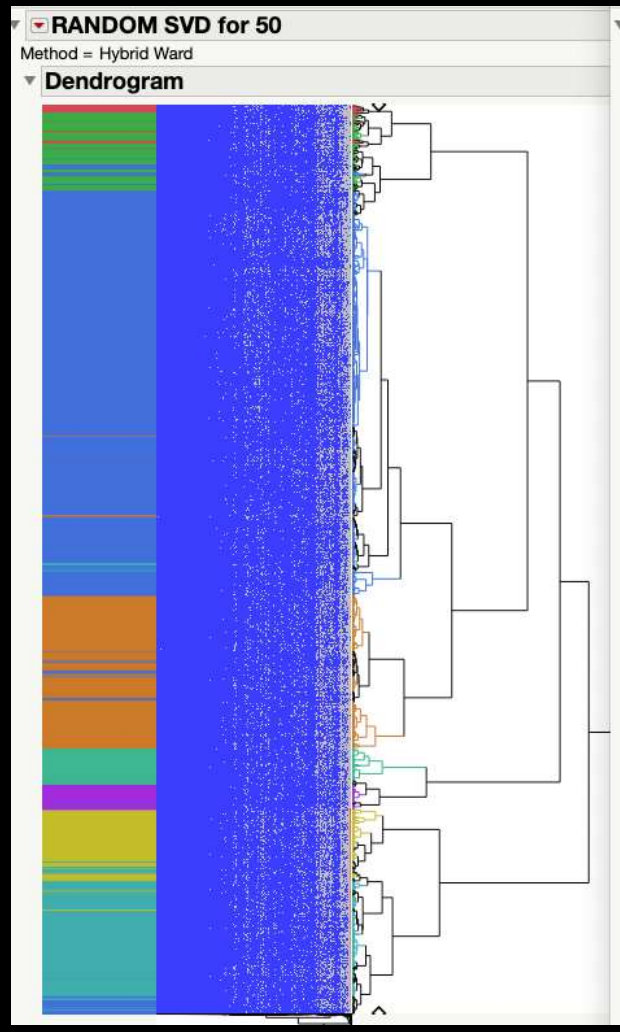
X*P　　時間=8.21
直交化　時間=0.036
Q`*X　　時間=76.599
SVD　　時間=0.114
Q*U*sqrt(S) 時間=0.048

SVD版: 179.68
通常版: 1207.35

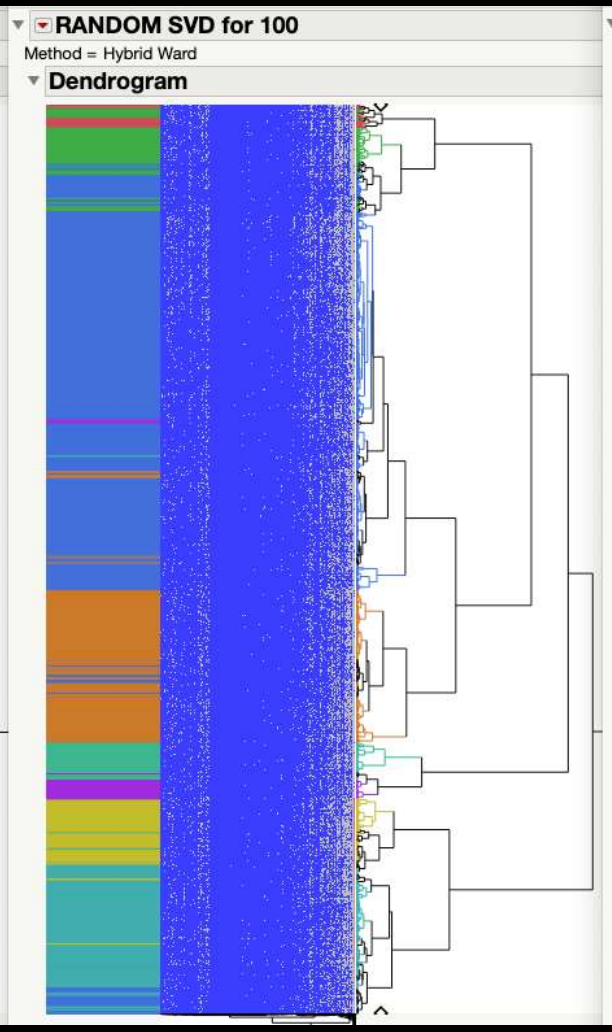
6.7倍、速くなった。

乱択特異値分解: より高次な次元(50, 100, 200)

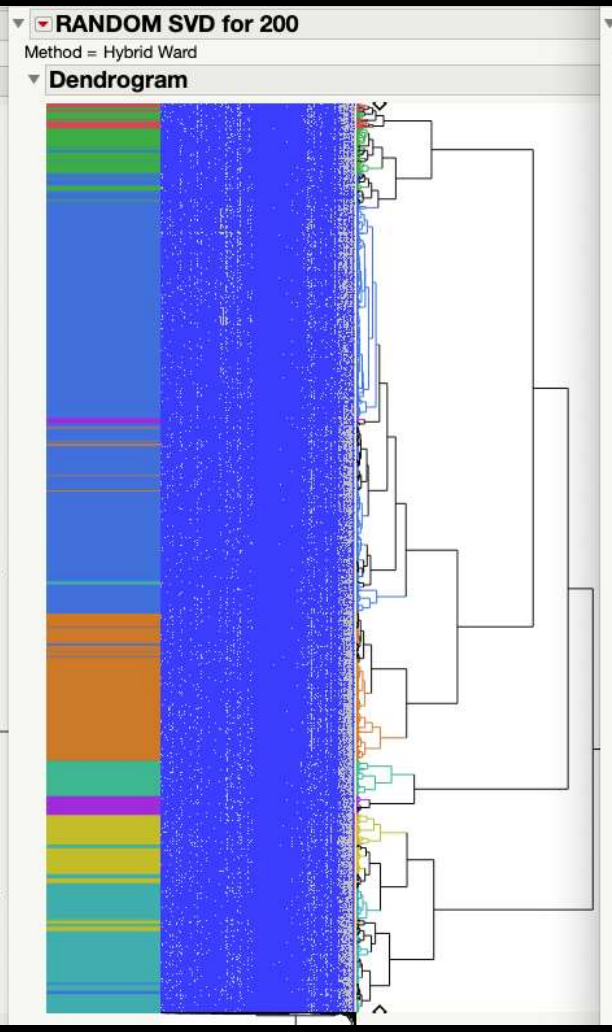
50次元, 179.68秒
100次元, 315.85秒
200次元, 625.63秒



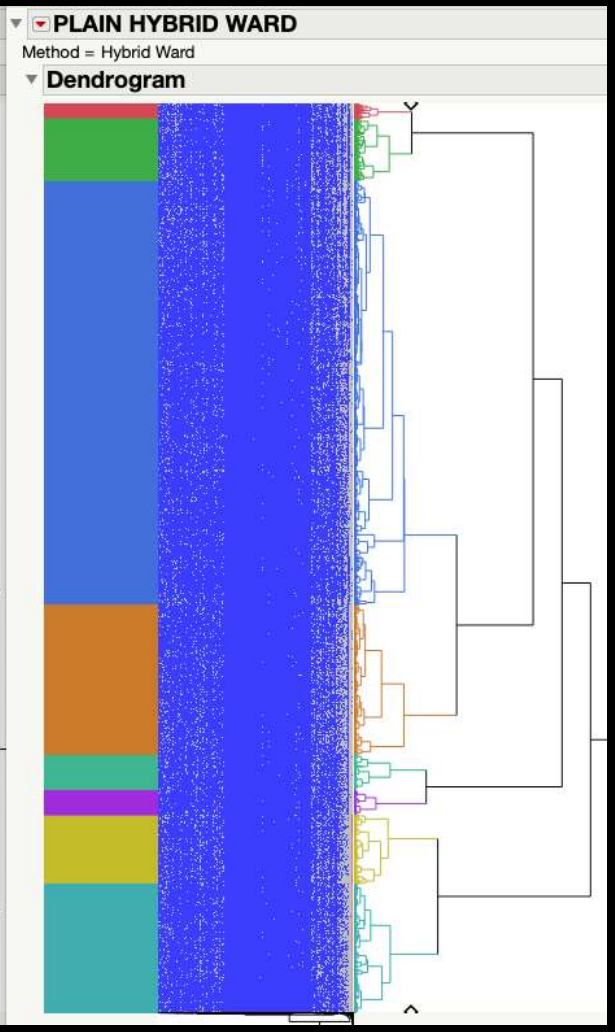
50次元



100次元



200次元



全次元での折衷型

乱択特異値分解

- 乱択特異値分解それ自体は、まだ遅い。
  そのボトルネックは、行列の乗算。

- どうやったら速くなる？
- →行列の乗算をマルチスレッド化。

乱択特異値分解: 次の改良点＝ 行列乗算のマルチスレッド化

行列乗算を非マルチスレッド	
X*P	30.682
直交化	0.205
Q`*X	212.34
SVD	11.526
Q*U*sqrt(S)	0.254
総計算時間 = 255.01	

行列乗算をマルチスレッド	
X*P	1.257
直交化	0.206
Q`*X	19.095
SVD	11.43
Q*U*sqrt(S)	0.018
総計算時間 = 32.01	

Baronデータ: 7,266行, 20,124列, 行のクラスタリング

乱択SVD（200次元）非マルチスレッド：総計算時間＝315.68

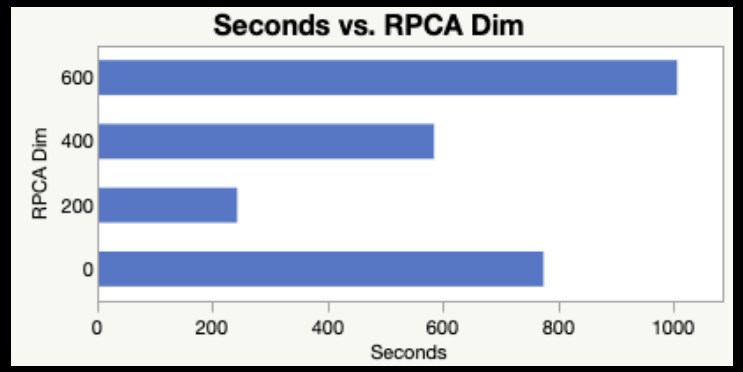
乱択SVD（200次元）マルチスレッド　：総計算時間 = 43.76

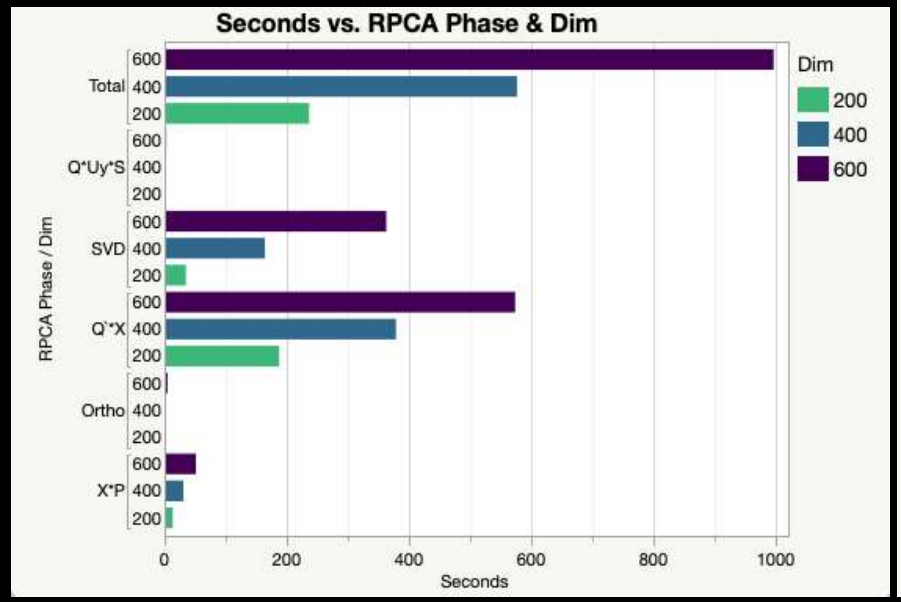
乱択SVDなしの総計算時間は、 53.03秒。

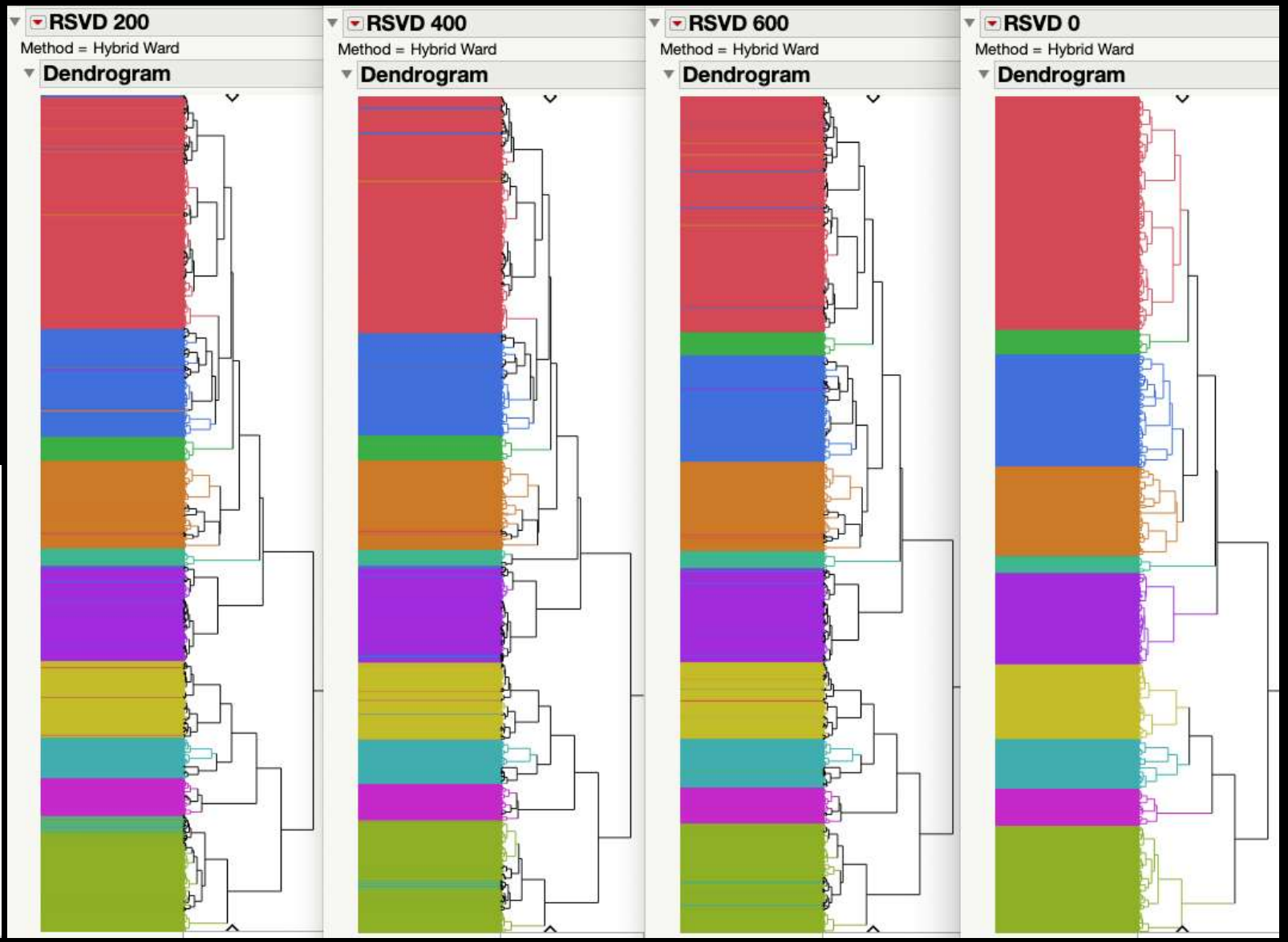
なぜ、乱択SVDにしても計算時間は短くならないのか？　次のようなトレードオフ。

方法	SVD時間	折衷時間	
通常	0.0	37.96	
乱択SVD200	32.01	0.438	
乱択SVD100	12.77	0.269	SVD 100次元にすると計算時間は短くなるが、近似が悪くなる。

乱択特異値分解: GTEX







この例では200次元で近似は十分で、かつ、計算時間も短い。
600次元にすると、乱択特異値分解なしの処理よりも遅くなる。

乱択特異値分解: perspectives

- 乱択特異値分解の結果は、行のクラスタリング、および、列のクラスタリングの両方に使える。ただし、その利点を使っていない。
- 乱択特異値分解は、折衷型Ward法だけでなく、高速Ward法でも利用できる。次の理由で計算時間が短くなる。(1) kd木は、データをうまく分割する変数を1つずつピックアップしていく。最初のほうの主成分が、データをうまく分割する。 (2) もし、その変数のピックアップにおいて距離が大きいことが分かったならば、最近隣かどうかの判断をショートカットできる。
- 乱択特異値分解は、主成分分析・ロバスト特異値分解などでも利用できるアルゴリズムである。
- 乱択特異値分解の結果はまちまち。いくつかの低次元なデータでは、かなり良い結果が得られる。
- 特異値分解において外れ値は頭痛のたね。それは他の多くの行にも影響するから。将来にはロバスト特異値分解を実装したほうがいい？

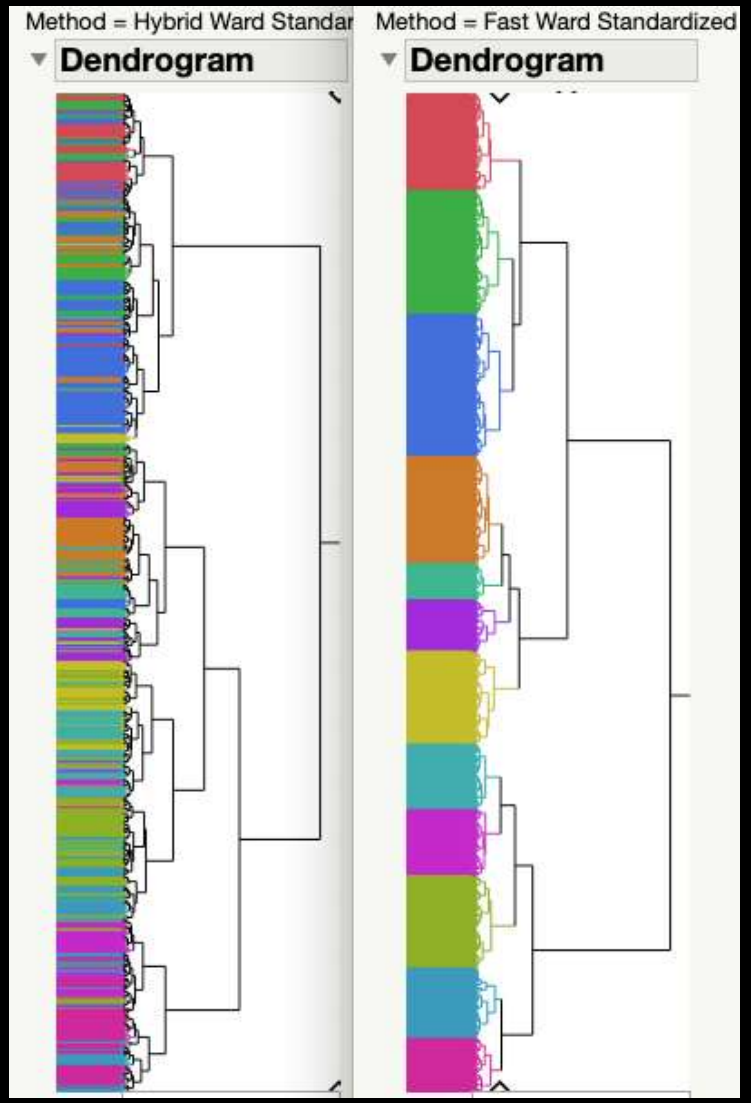
次にどうやって改良する?

- 作業完了: ~~折衷型ウォード法~~
- 作業完了: ~~「人気がありすぎる点」の対処（~~K~~を適応）~~
- 作業完了: ~~乱択特異値分解による次元削減~~
- 作業完了: ~~行列乗算のマルチスレッド化~~
- これから：折衷型クラスター法は常に良い結果？

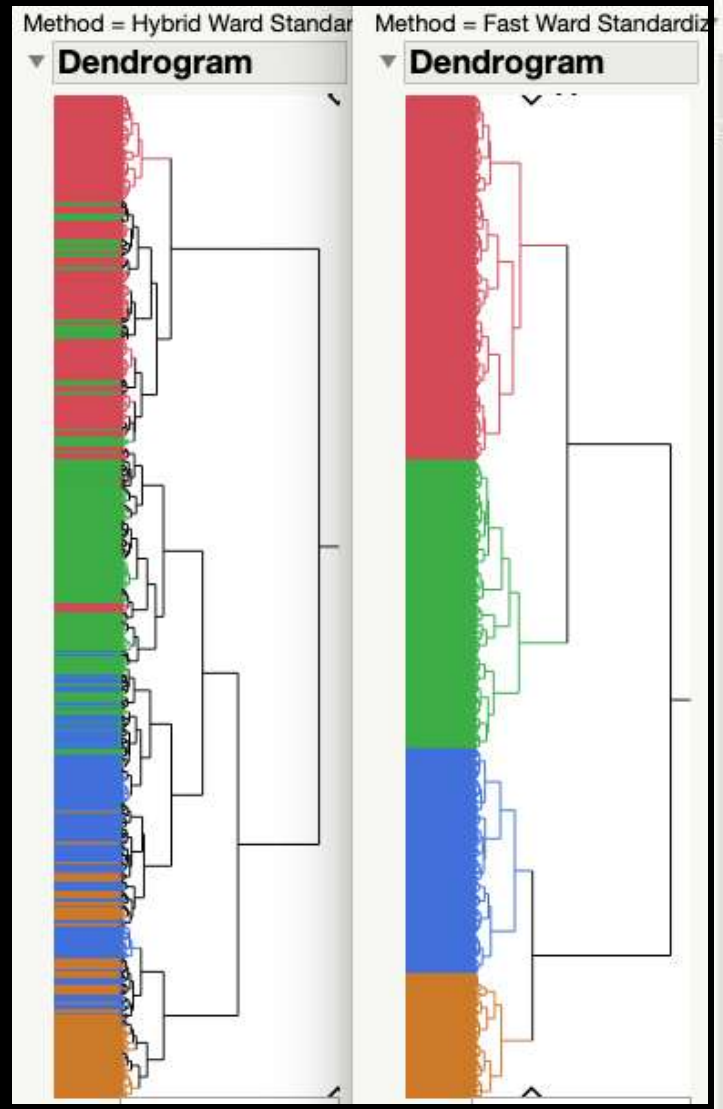
折衷型クラスターの結果は常に良い？

- ansurAnthropometry.jmp
- 3,982行, 131列
- 最良なクラスター数は?

12クラスター

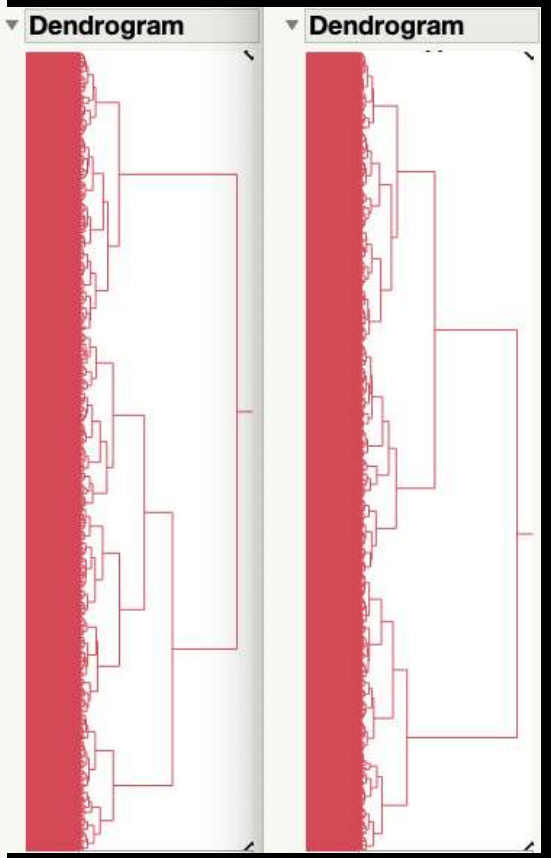


4クラスター



Cubic Clustering Criterion

	Number of Clusters	CCC
->	1	0.00
	2	-11.53
	3	-14.93
	4	-16.31
	5	-20.74
	6	-22.75
	7	-21.43
	8	-19.75
	9	-18.31
	10	-18.10
	11	-17.63
	12	-17.02
	13	-16.33
	14	-15.70
	15	-15.47
	16	-15.12
	17	-14.79
	18	-14.76
	19	-14.67
	20	-14.55
	21	-14.52
	22	-14.47
	23	-14.37
	24	-14.26
	25	-14.10
	26	-13.96
	27	-15.44
	28	-15.25
	29	-15.04
	30	-14.86



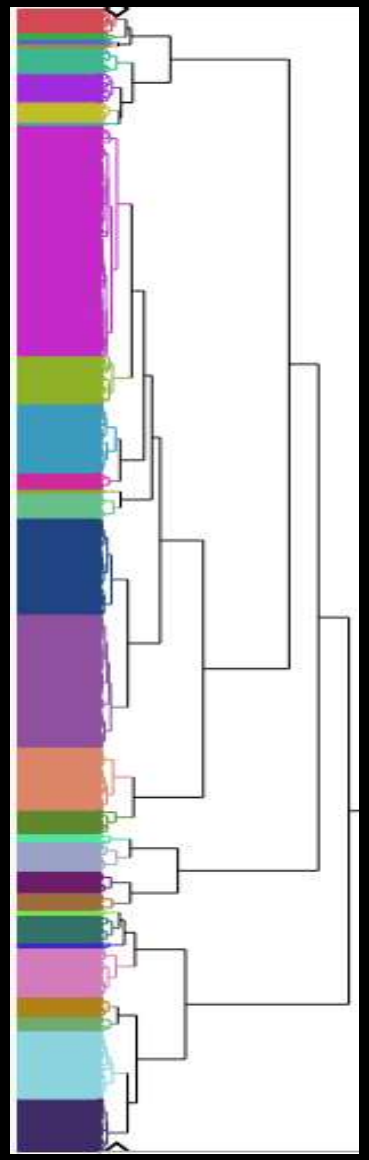
クラスター数を決める規準：
立方体クラスター規準（CCC; Cubic Clustering Criterion）

- 共分散行列を計算する（共分散行列の計算は以前のJMPですでにマルチスレッド化）。
- 共分散行列の固有値を求める。

- 共分散行列の計算量は、n×m^2/2。また、大きな行列の固有値計算もかなり計算量が多い。
- Baronデータ（ m=20,124, n=7266）の共分散行列は、計算量 = 1,471,275,541,008, 要素数=202,487,688。

- CCC計算における特異値計算アルゴリズムを乱択特異値分解にすると、特異値の計算には13.2秒で、CCCの計算にはわずか44.3秒だった。

立方クラスター規準（CCC）: Baronデータ
（左図が行クラスタリング、右図が列クラスタリング）

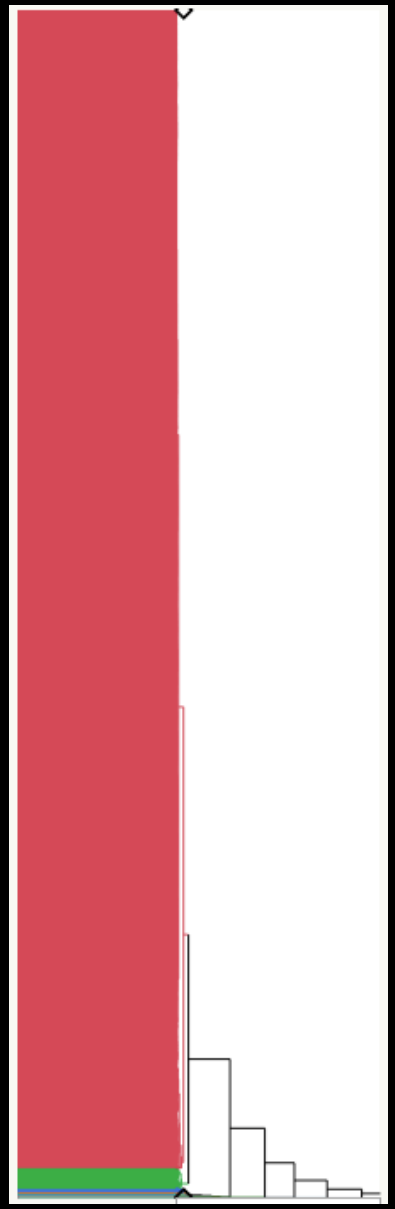


Cubic Clustering Criterion

	Number of Clusters	CCC
	1	0.00
	2	47.39
	3	82.35
	4	134.78
	5	158.40
	6	161.09
	7	166.74
	8	173.24
	9	177.47
	10	179.76
	11	180.04
	12	178.44
	13	177.45
	14	176.85
	15	176.88
	16	177.21
	17	177.88
	18	178.99
	19	180.69
	20	181.82
	21	183.28
	22	184.63
	23	185.54
	24	185.57
	25	185.72
	26	185.90
	27	186.04
	28	186.45
	29	186.66
->	30	186.81

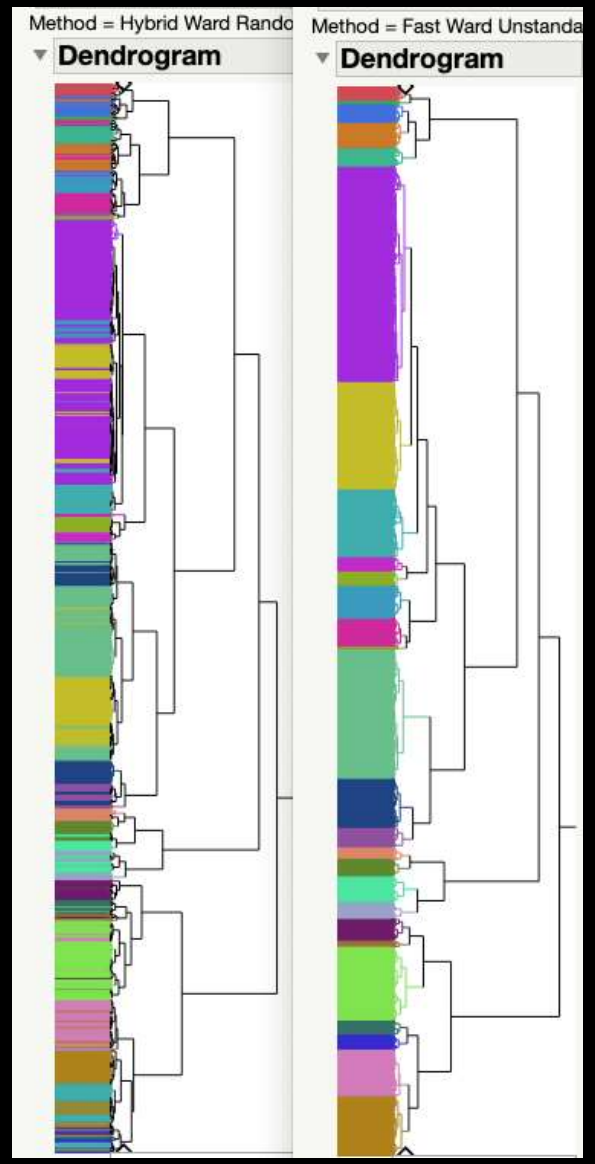
Cubic Clustering Criterion

	Number of Clusters	CCC
	1	0.00
	2	49.15
	3	144.96
	4	242.38
	5	367.19
	6	384.13
	7	402.69
	8	431.22
	9	472.05
	10	504.33
	11	545.65
	12	563.59
	13	566.69
	14	571.74
	15	576.02
	16	582.72
	17	588.80
	18	595.89
	19	599.57
	20	603.17
	21	608.44
	22	612.58
	23	617.25
	24	620.87
	25	625.13
	26	629.82
	27	633.24
	28	637.11
	29	641.45
->	30	645.48

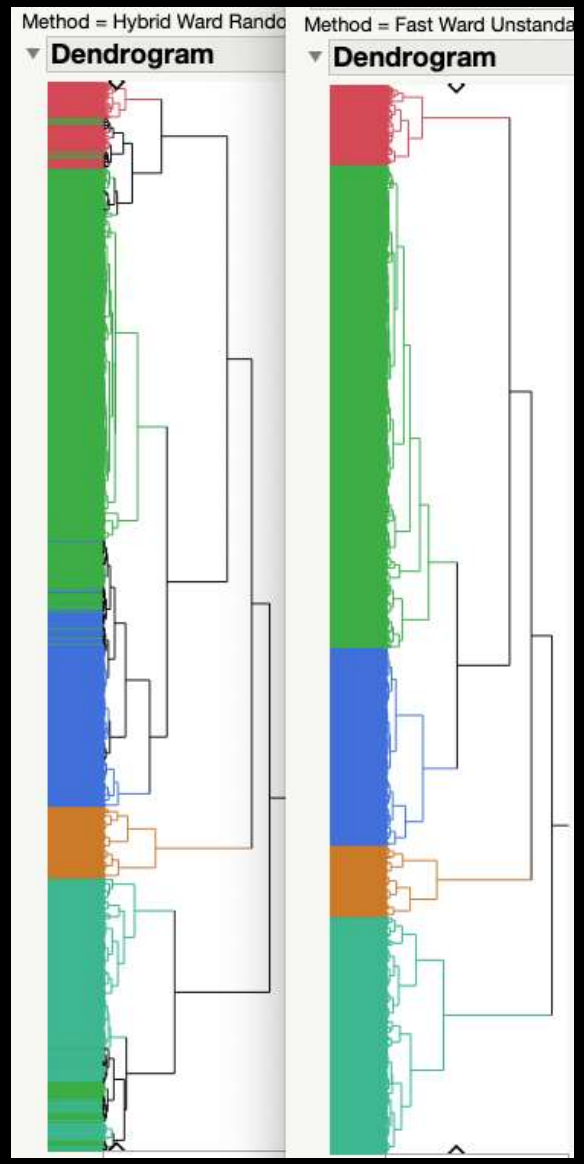


立方クラスター規準（CCC）：Baronデータ

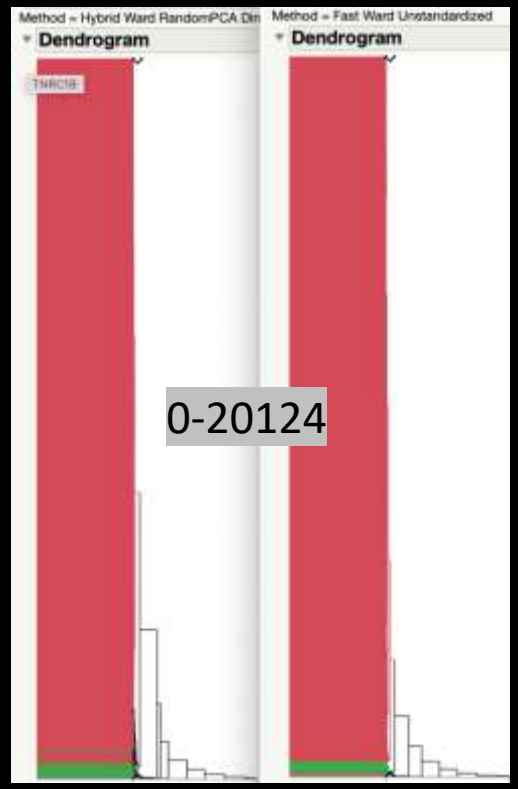
Baron 列, 27クラスター



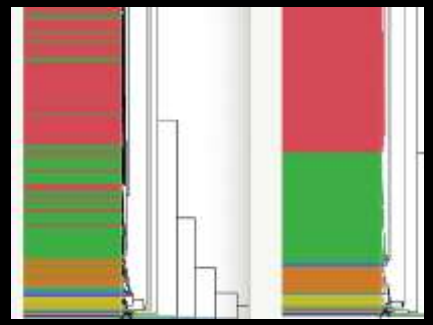
Baron 行, 5クラスター



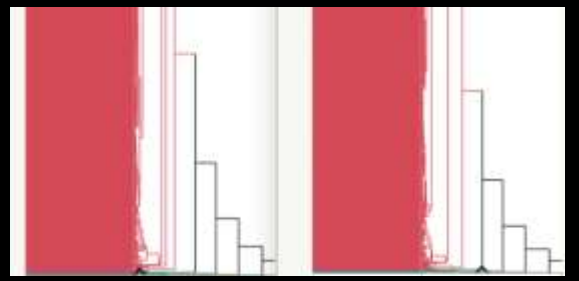
Baron 列, 30クラスター



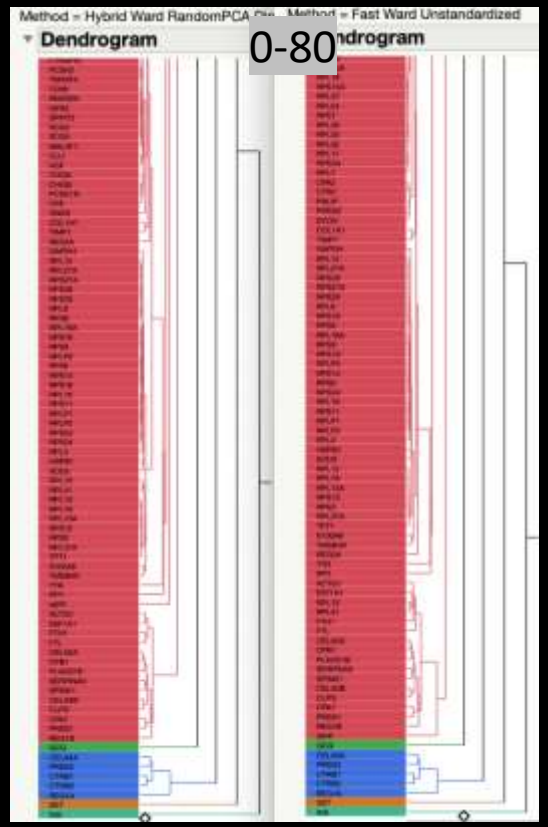
0-20124



0-800



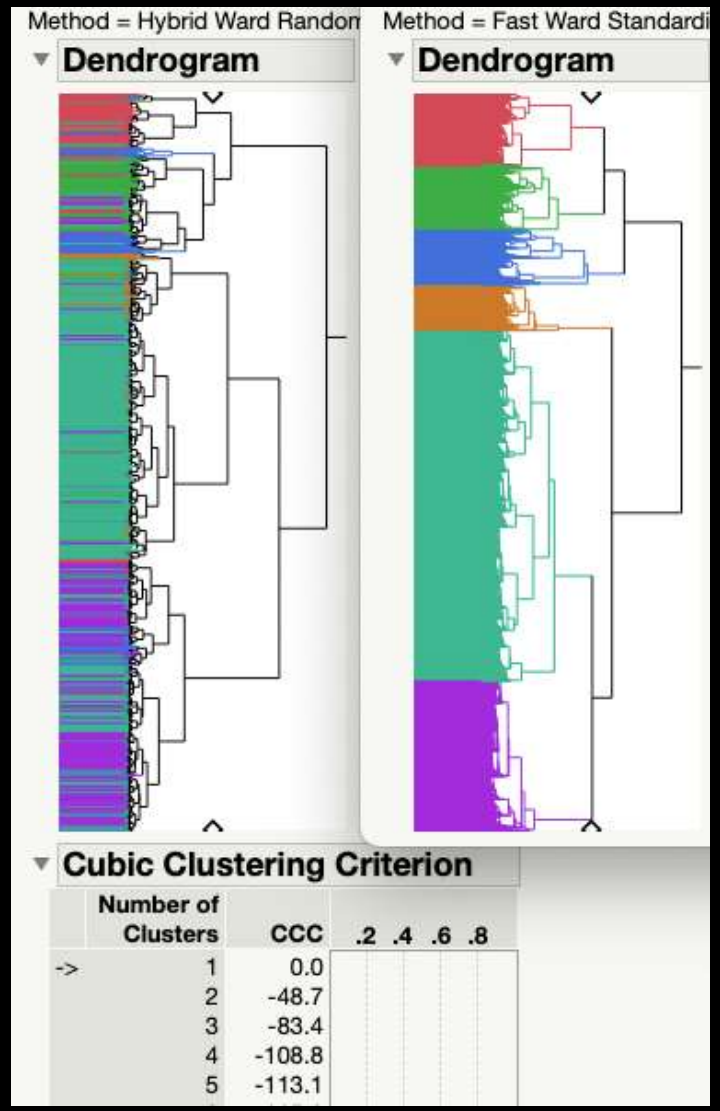
Baron 列, 5クラスター



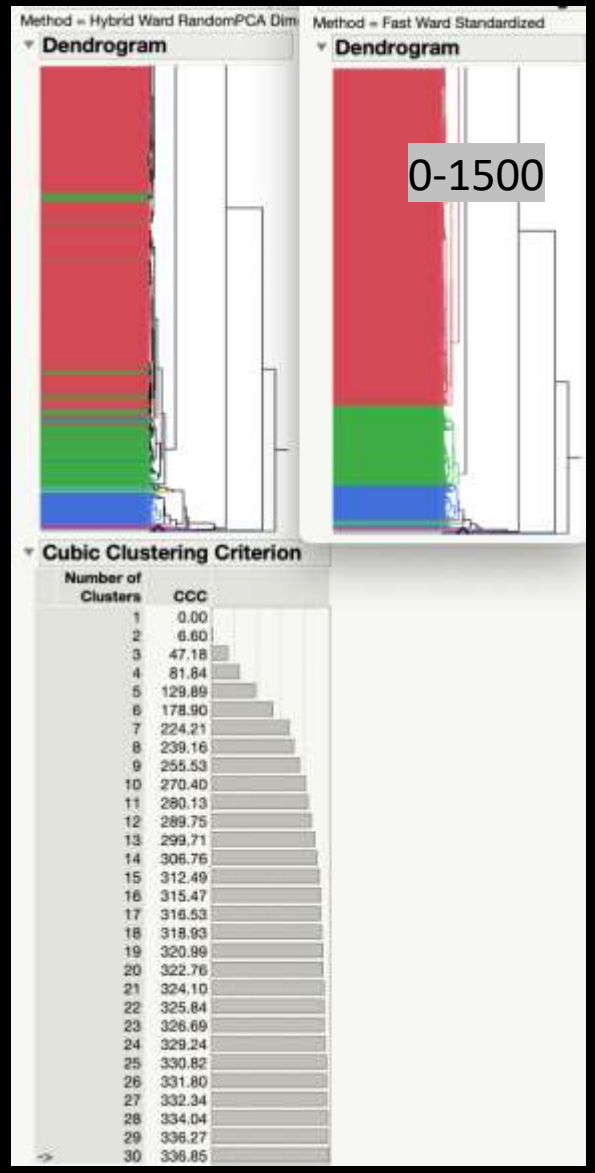
0-80

立方クラスター規準（CCC）：Baronデータ

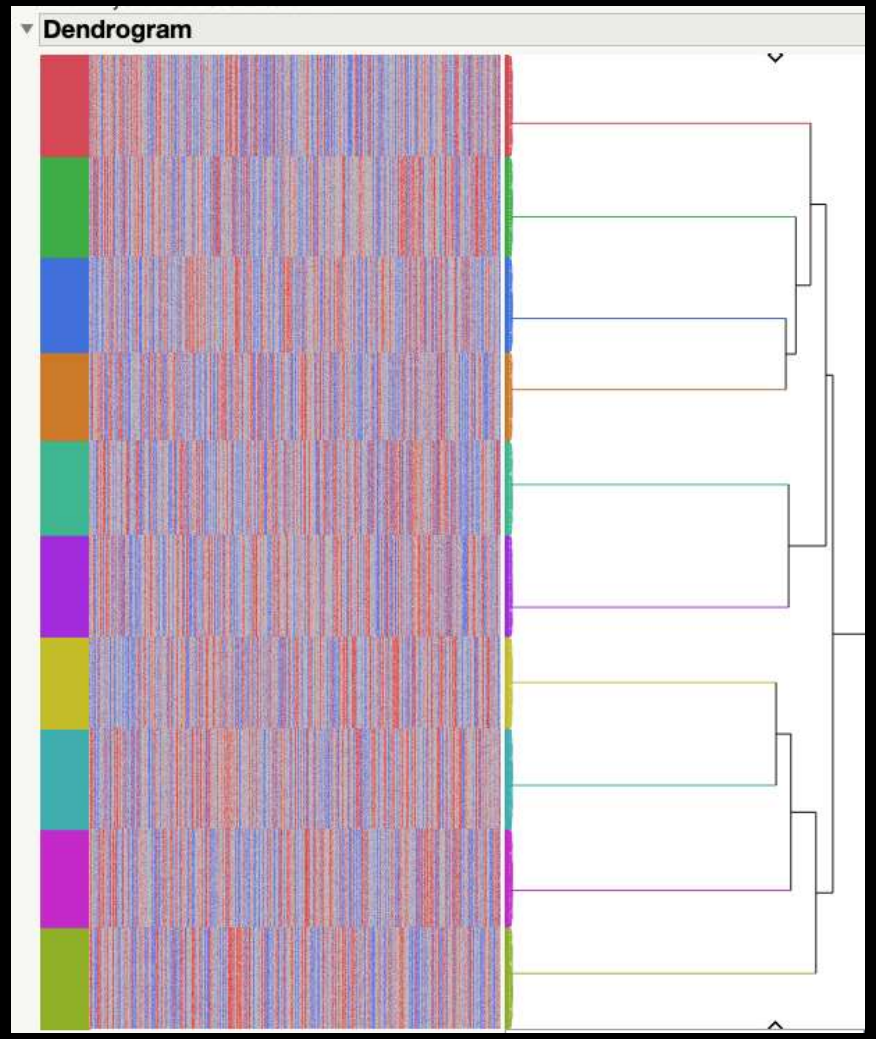
Baron 行, 6クラスター



Baron 列, 15クラスター



立方クラスター規準（CCC）： 人工データ



Cubic Clustering Criterion

	Number of Clusters	CCC
	1	0.00
	2	-64.60
	3	-69.61
	4	-66.93
	5	-60.03
	6	-50.87
	7	-77.94
	8	-52.48
	9	-4.69
->	10	72.08
	11	68.78
	12	65.83
	13	68.23
	14	65.60
	15	67.56
	16	65.17
	17	62.95
	18	60.87
	19	58.92
	20	57.09
	21	58.72
	22	56.97
	23	55.31
	24	53.73
	25	52.22
	26	53.53
	27	52.07
	28	50.67
	29	49.32
	30	48.02

この疑似データのように、データの分布がきれいにクラスターに分かれていたならば、折衷型Ward法の結果は、従来の方法と一致する。

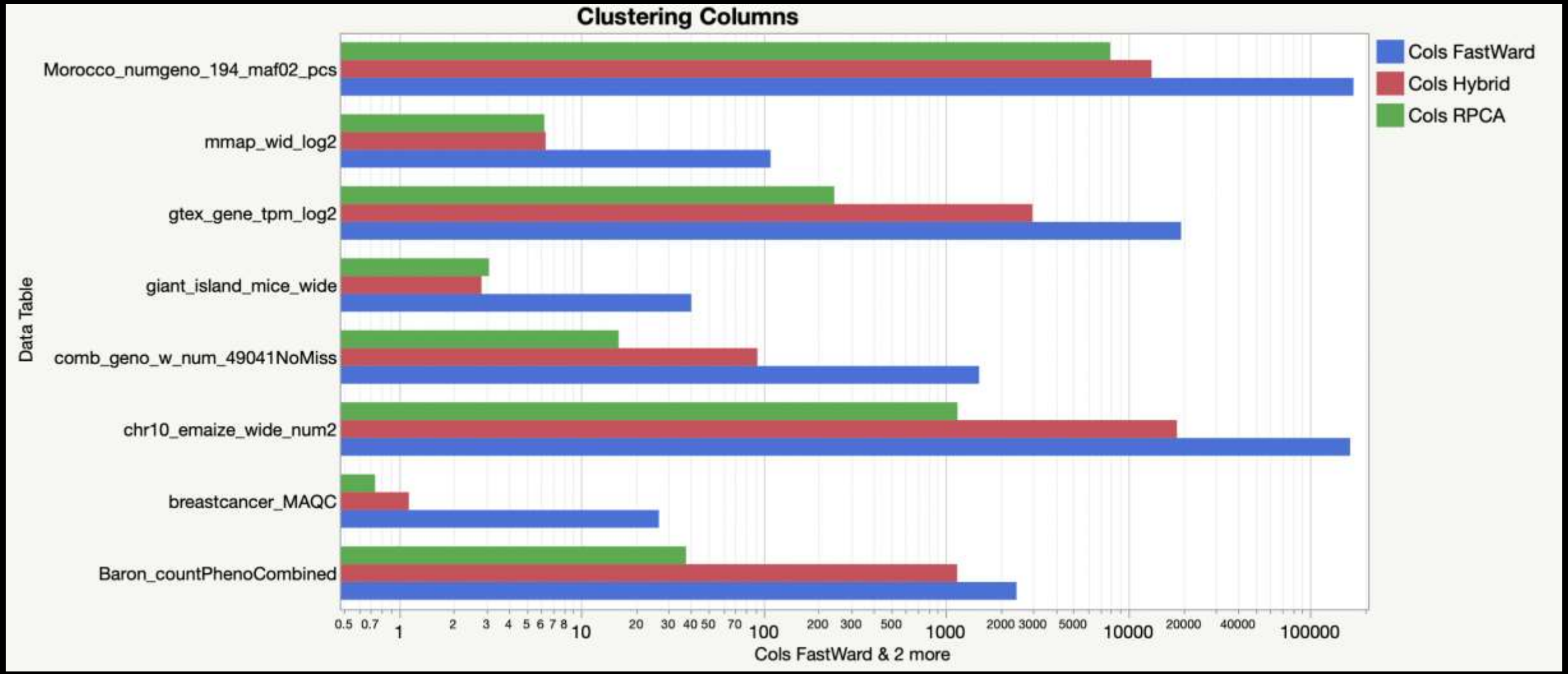
また、この疑似データでは、クラスー多数が何個かもはっきりわかる。

現実のデータは、このようなきれいな結果にはならない。

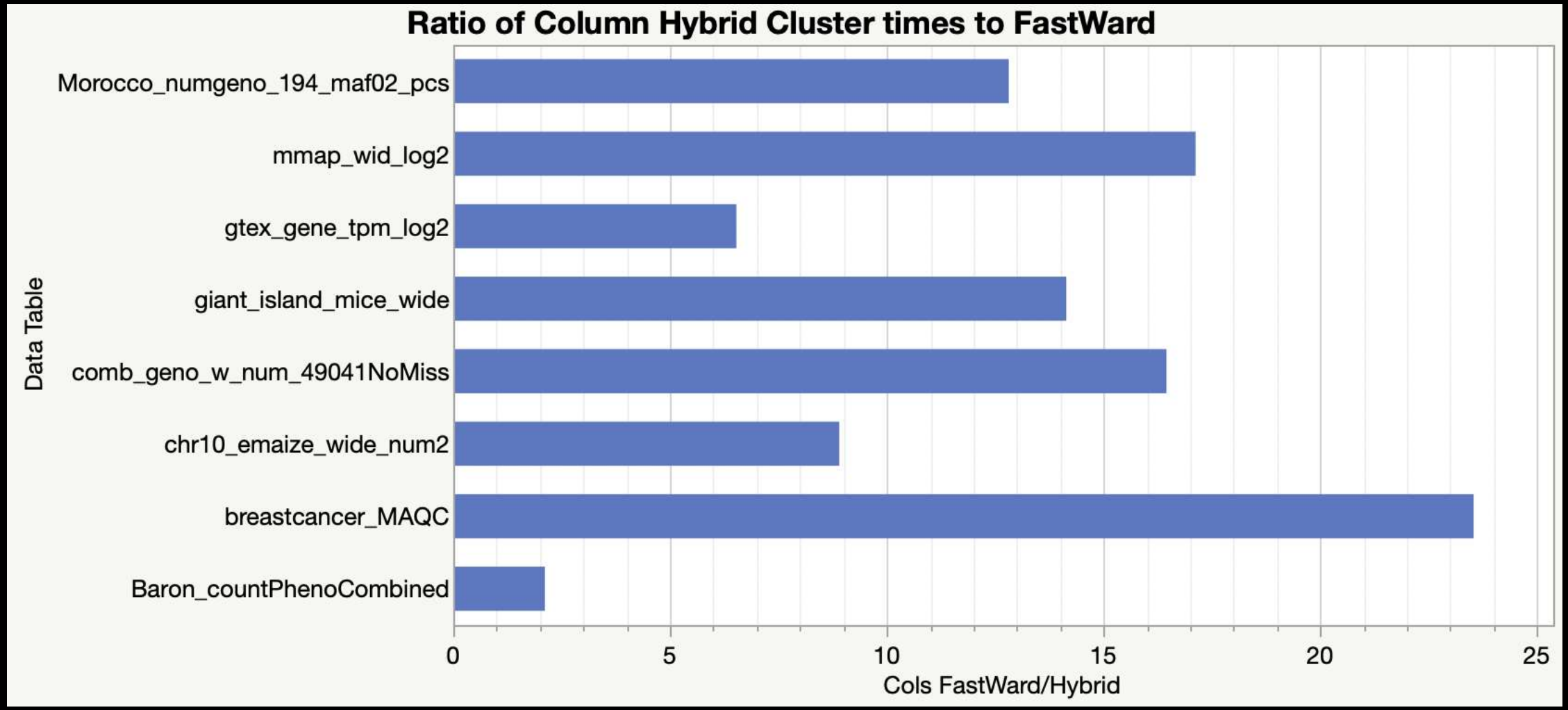
要約

古典的な階層型クラスタリング	遅すぎる
改良した方法：高速Ward	まだ遅すぎる
折衷型	「人気がありすぎる点」問題
Kを適応した方法	まだ遅すぎる
乱択SVD	まだ遅すぎる
行列乗算のマルチスレッド化	結果の妥当性が疑問
CCC（立方クラスター規準）	遅すぎる
CCC計算に乱択SVD	良くなったが、まだまだ改良余地。

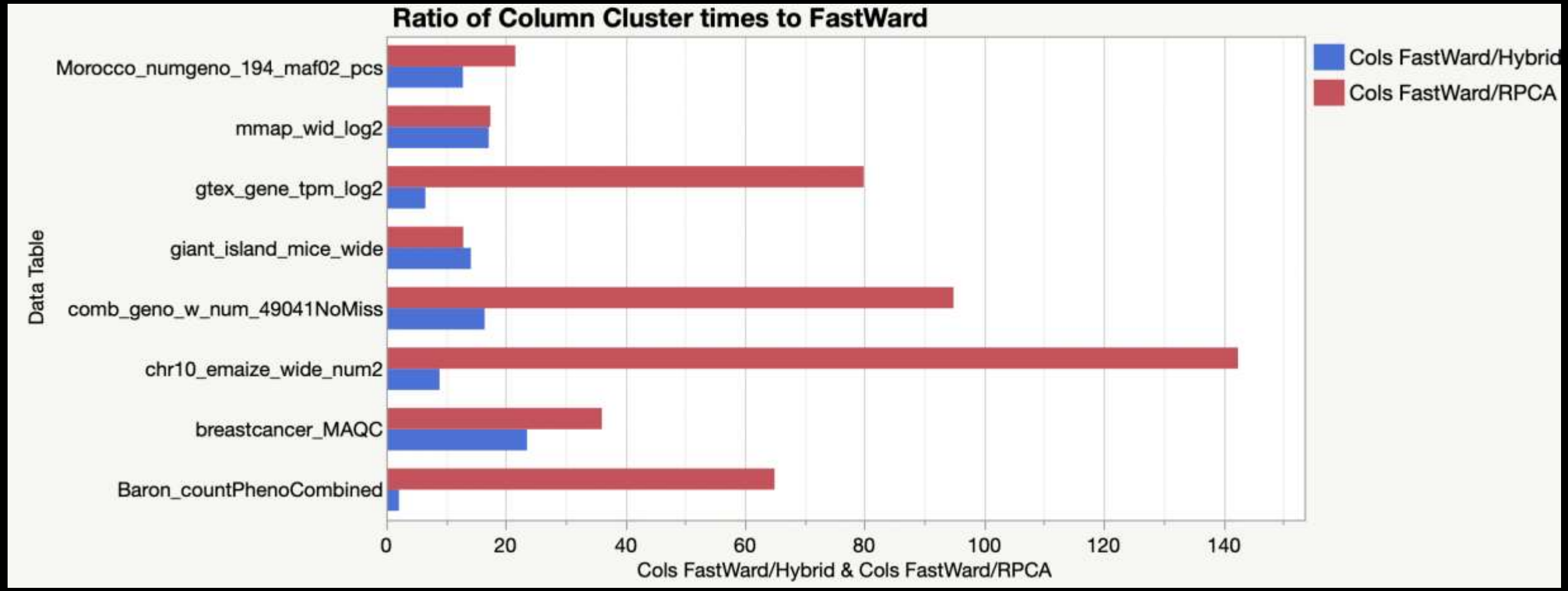
ゲノムの横長データ（ワイドなデータ）でのベンチマーク



高速Ward法に対する折衷型の比



高速Ward法での
通常方法に対する乱択SVDの比



結論

弊社開発者のJeremy Ashによるベンチマーク

50,000行、 210列

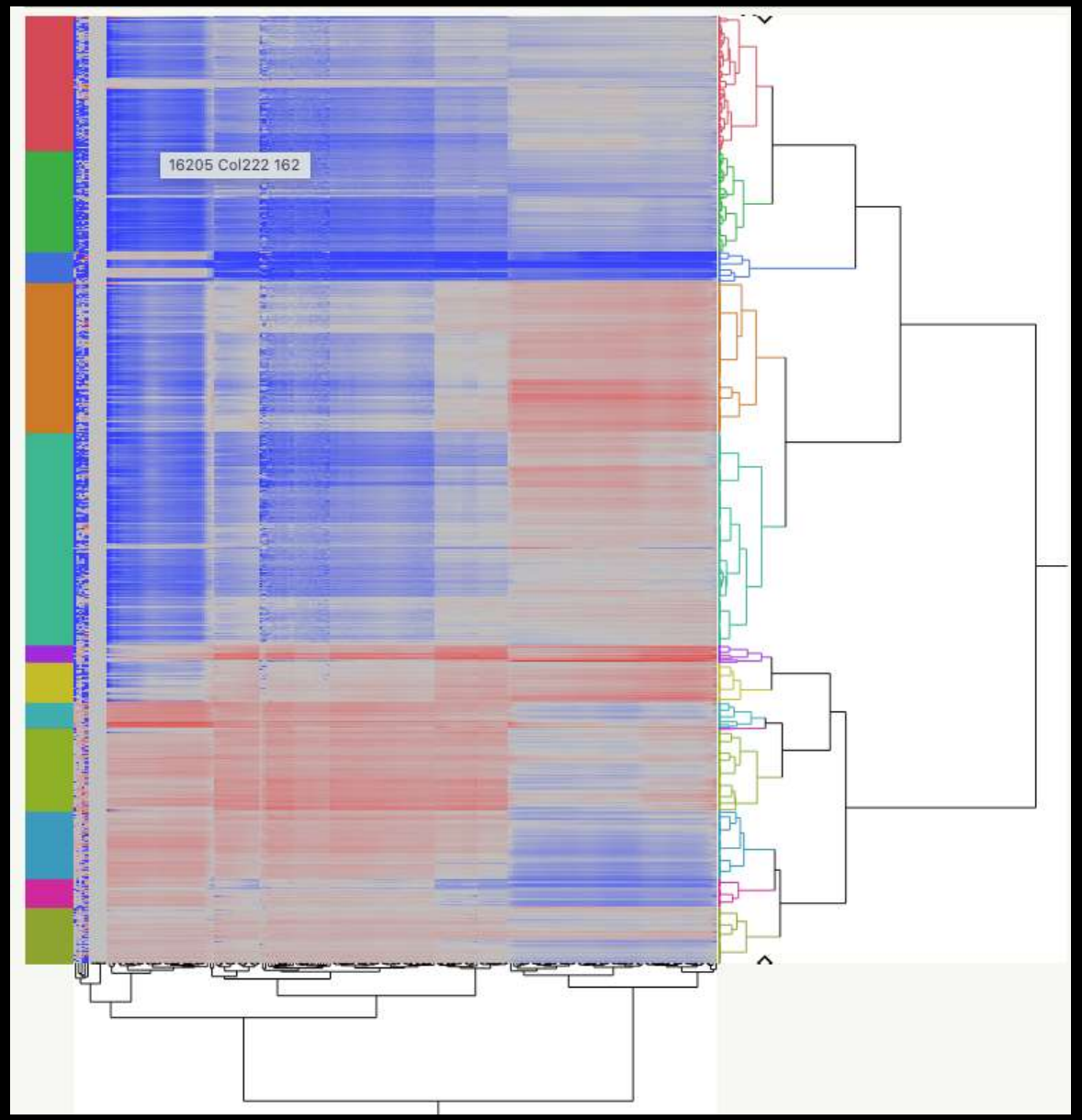
- fastcluster (R): 約58分 （3,465秒）
- 高速Ward (JMP): 約8 minutes (479秒)
- 折衷型Ward: 22秒

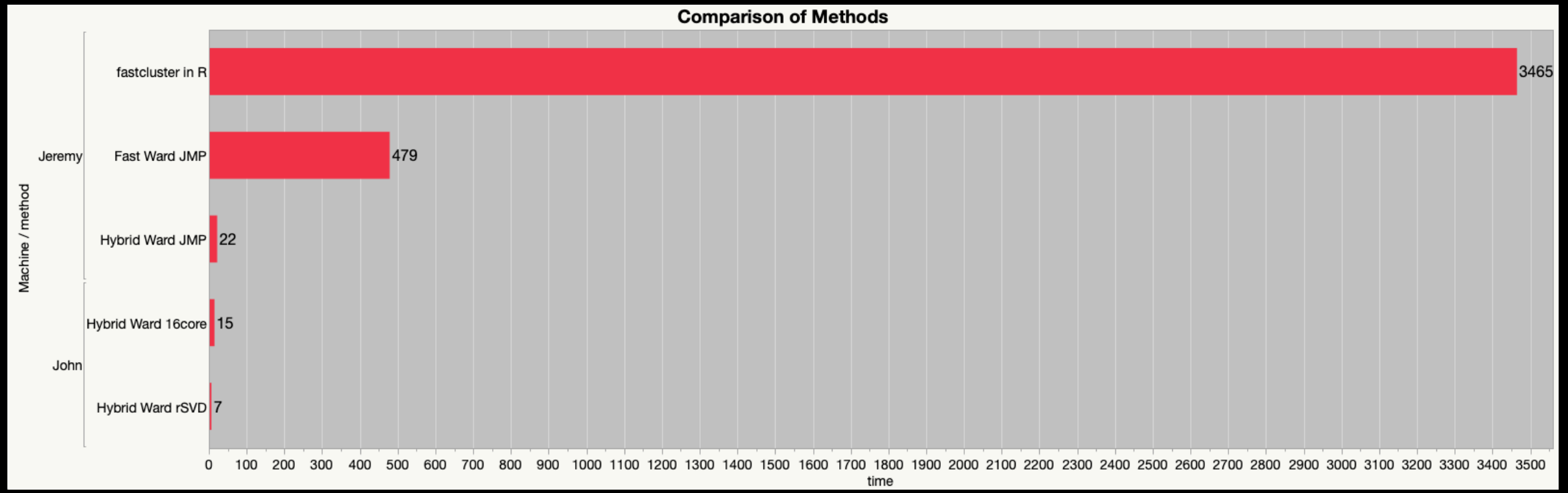
折衷型Ward法は、以前のJMPの方法（高速Ward法）よりも21倍速い。また、Rのfastclusterよりも157倍速い。

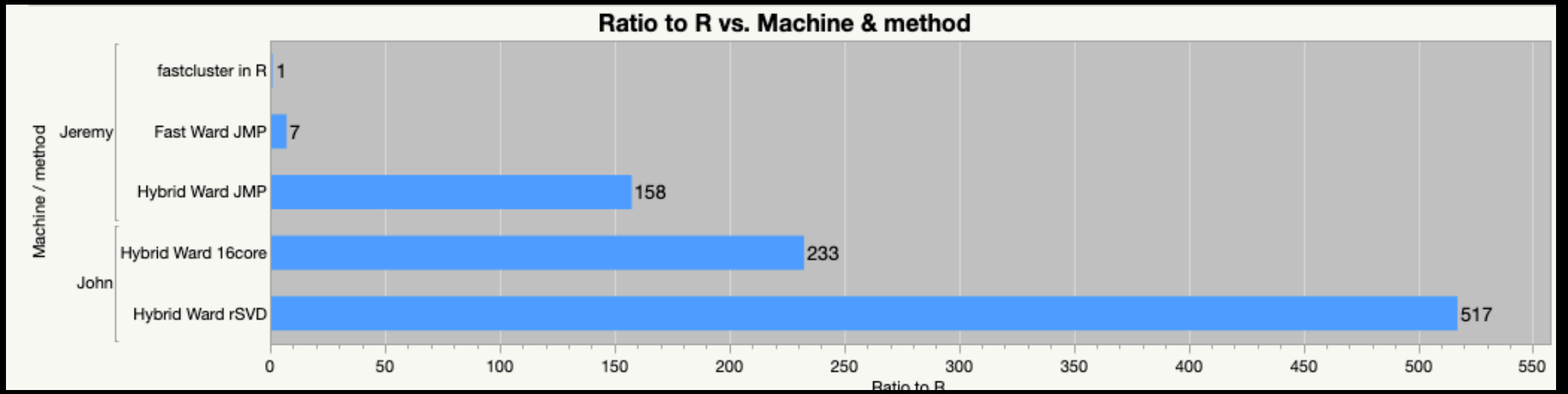
私のデスクチップパソコンでは、以前のJMPの方法（高速Ward法）よりも、折衷型Ward法は46倍速かった。私のマシンは16CPUコアを搭載しており、計算時間は14.9秒だった。Ashのマシンよりも、CPUコアが搭載されているので、計算が短くなったのだろう。

折衷型Wardをいくつかのアイデアでさらに改良すると、6.7秒となった。さきほどよりも2倍の向上。

この改良は革新的になりえて、従来の分析を繰り返し対話的に行えるようにするだろう。




Comparison of Methods
Machine / method
Jeremy
John
fastcluster in R
Fast Ward JMP
Hybrid Ward JMP
Hybrid Ward 16core
Hybrid Ward rSVD
3465
479
22
15
7
0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600 1700 1800 1900 2000 2100 2200 2300 2400 2500 2600 2700 2800 2900 3000 3100 3200 3300 3400 3500
time


Ratio to R vs. Machine & method
Machine / method
Jeremy
John
fastcluster in R
Fast Ward JMP
Hybrid Ward JMP
Hybrid Ward 16core
Hybrid Ward rSVD
1
7
158
233
517
0 50 100 150 200 250 300 350 400 450 500 550
Ratio to R

ほかにも、いろいろな改良点が考えられる

- 距離の単調増加性が大きく崩れることを避けるために、サイクルの後期において目的関数を変化させる。
- 列クラスタリング・行クラスタリング・CCCのすべてを、１回だけの乱択特異値分解でまとめて計算する。
- 近似的な近傍点探索に関して、より高速なアルゴリズムを用いる。たとえば、Hierarchical Navigable Small World (HNSW) グラフ？
- 外れ値の処理（予備処理で、1個だけのクラスターが生まれないようにする）
- 保存したデータ（オフライン）からのクラスタリングを表示する機能。
- どのような標準化が妥当か？（特に、2方向のクラスタリングにて）
- 「ロバスト主成分」などによる欠測値補完も、計算時間を短くすべき。乱択特異値分解を用いて、その結果をクラスタリングに用いるべき？
- 先行研究の精査―　何百もの良い方法が提案されている

お願い

- 計算時間が短くなるのは有益だ。流れるように分析できるようになり、多くの分析を行えるようになり、データがあなたに何を語っているかを理解できるようになる。
- 高性能のコンピュータによる荒業を、いくつかの繊細なエンジニアリングと組み合わせることで、大規模な問題が短時間で解ける場合がある。

- 私たちは、お客様からの声に基づいて開発を進めています。「計算時間を速くしたい」などの声もお聞かせください。

力ずくだけど繊細なアルゴリズム

コンピュータの性能を利用しよう！
しかし、より賢く、より要領よく！

ご清聴ありがとうございました