Some preliminary terms:

BF brute force monte carlo simulation (without feature space recombination)
CNS Correlation Networked Sampling
G.A. Polish Genetic Algorithm performs recombination and mutation on a pre-existing population; in this example, the top 5,000 subsets were selected from the brute force monte-carlo simulation of runLength 10,000.
Relatively Independent Subset: a subset whose maximum pair-wise, cross correlation is less than a hard preset correlation threshold
error the mean of squared or absolute residuals generated by the fitness function, ppNN, a localized ridge regression (see footnotes).
bench error bench mark error in a population generated by a monte-carlo simulation (bf or ga). It is the minumum of the regression errors in the population of associated time-series subsets.
the **genotype** is an integer, a column number of a matrix, array, data.frame, relational table
the **phenotype** is a time-series or named vector corresponding to that column number
Feature Space each feature is a possible subset of expl. vars. The BF simulation and the g.a. both sample from this space of subsets.

Abstract:
We will be using monte-carlo simulations and genetic algorithms to suggest non-parametric correlations between the most frequently occurring explanatory variables and the response; suggest panels of relatively independent subsets and their regressive models that may indicate the future direction of the response. It may also assist in developing more complex structural models.

This technique concerns the use of brute-force monte-carlo simulation whose resulting population of subsets will be polished by a genetic algorithm. This hybrid approach appears to work better than either one alone. This paper also demonstrates the possible efficacy of subtly biased sampling techniques in these simulations. It also demonstrates the efficacy of Correlation Networked Sampling to generate populations of relatively independent subsets. Within the context of financial and econometric time series, it appears that brute-force monte-carlo simulation followed by mutation worked better than crossover.

Motivation:
Collinearity caused:
- due to numerical instability introduced by redundacy, long simulations crashed or produced slightly inaccurate results from singular matrix inversion methods
- as a mathematical statistician, i believe that out-of-sample forecasts would be more inaccurate
- inconsistent parameter estimates made models that were hard to defend and compare in particular, inaccurate significance tests;
- redundant variables can also lead to overfitting, a serious problem in forecasting with local regression such as loess and particularly vulneable, neural nets.

My efforts at neuro-evolutionary programming stumbled on this boulder of collinearity.

As a consequence, I set out to generate subsets from the column space of a matrix or data.frame with two properties:

1. the explanatory variables (corresponding to the integers in the subset) are relatively uncorrelated;

2. a fitness functional exists that reduces each subset of time-series to a positive scalar. (in this way, the subsets can be ordered and filtered);

Performance metrics:

error mean std. error from the residuals of a regression on the time series
this constitutes the fitness functional;
elapsed time dedicated system elapsed time (the cost);

Pseudo-code for Brute Force Simulation:

```
    set n to some positive integer
    set runLength to some postive integer
    set sample size to n, some pos. int. where n <= max( column space )
    set correlation threshold has high as you can tolerate ( 0<coreThresh <= 1 )
# the loop to generate a population of relatively uncorrelated subsets
    while( number of subsets < runLength )
        sample n integers from the column space
        map those integers to the n corresponding time series in the data.frame
        compute maximal time-series subset cross-correlation
        regress this time series matrix against the response variable
        compute the mean std error of the regression
    loop back;
Filter by maximal subset cross-correlation
Sort and filter by fitness
```

Data Context:

We will perform these simulations on data freely available on the internet.
I downloaded time-series data from the U.S. Federal Reserve of St. Louis (https://fred.stlouisfed.org/tags/series?ob=pv&od=desc) and yahoo.finance (https://finance.yahoo.com/quote/YHOO/history?p=YHOO). Including lags of 1 to 3 months, this is a data.frame of 206 rows corresponding to monthly measures going back to 1999-01-01 and 1476 columns (all potential explanatory variables) for the response, Con_Agra, a food packaging corporation.

Row.Names	Column 1	Column 2	Column 3	Column 4	Column 5
Date	djia_L1	Harmonized_Index_of_Consumer_Prices_Overall_Index_Excluding_L1	Index_of_Aggregate_Weekly_Hours_Production_and_Nonsupervisory_L1	Trade_Balance_Goods_and_Services_Balance_of_Payments_Basis_L1	Indexes_of_Aggregate_Weekly_Hours_of_Production_and_Nonsupervisory_L1
1999-04-01	9786.160	72.36	100.5	-17,900	108.3
1999-05-01	10789.040	72.65	101.1	-18,254	110.8
1999-06-01	10559.740	72.74	101	-20,229	112.7
1999-07-01	10970.800	72.73	101.5	-23,115	114.9
1999-08-01	10655.150	72.73	101.7	-23,421	114.2
1999-09-01	10829.280	72.85	101.8	-22,977	115.8
1999-10-01	10336.950	73	101.7	-22,899	114
1999-11-01	10729.860	73.08	102.4	-23,531	115.6
1999-12-01	10877.810	73.2	102.6	-25,224	115.2

2000-01-01	11497.120	73.47	102.9	-26,337	113.8
2000-02-01	10940.530	73.4	103	-27,088	109.5

The column space: integer domain { 1, 2, 3, ..., 1476 }.
The sample: 4 integers selected almost at random from the column space;
Since these integers correspond to a subset of time-series columns in the data.frame, we compute that subset's maximal pair-wise cross-correlation and run a regression on that subset against the response.
We subsequently filter by cross-correlation and sort and filter by fitness error.

Sample output from brute force monte carlo simulation:

chromosome number	Gene ind.var.1	Gene ind.var.2	Gene ind.var.3	Gene ind.var.4	bench error	max. subset cross correlation
1	383	987	1001	423	2.7272	0.5568
2	114	89	328	1076	2.8743	0.6935
3	92	898	330	283	2.9224	0.683
4	311	818	766	1349	2.9993	0.3193
5	1284	418	1040	916	3.0099	0.5728
6	741	123	1129	89	3.0554	0.4652
7	114	688	362	185	3.1289	0.6609
8	184	640	435	628	3.1812	0.5819
9	215	180	418	452	3.1932	0.5331
10	640	1130	190	341	3.2051	0.6162

The first row, chromosome 1 contains the column numbers: 383, 987, 1001, 423 and these integers (genotypes) correspond to these time series (phenotypes):

Polo_Ralph_Lauren_Corp_L1
Index_of_Aggregate_Weekly_Hours_Production_and_Nonsupervisory_L3
Japan_US_Foreign_Exchange_Rate_L3
SP_Global._Inc_L1

where L1 stands for a lag of one month, L3 stands for a lag of 3 months;

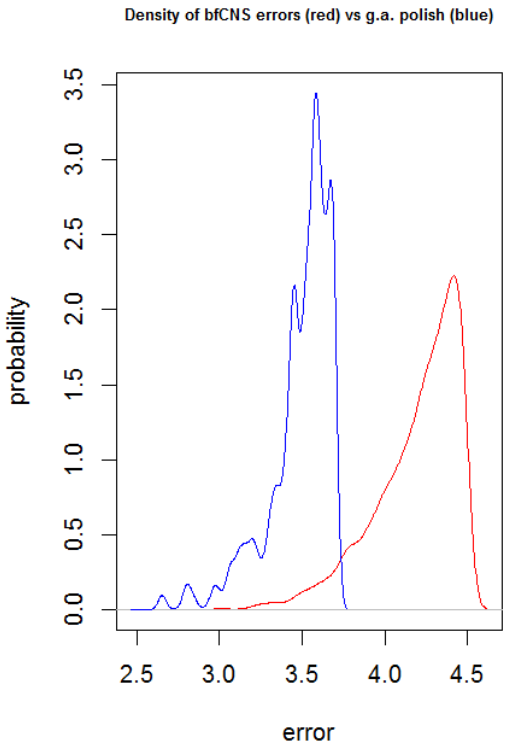
The Results:

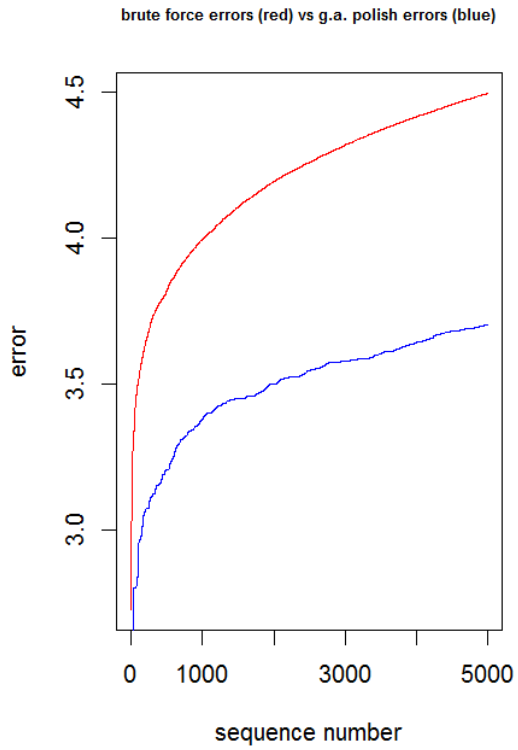
Description	**runLength**	**acceptance%**	**bench error**	**elapsed sys time**
Brute force simulation	200,000	50.00%	2.8894	345.260
BF with CNS	**200,000**	**100.00%**	**2.7272**	**390.830**
BF with CNS	10,000	100.00%	3.0677	41.990
G.A. Polish on best 5000	*5,000	100.00%	2.5340	41.310
Total	**15,000**		**2.5340**	**83.300**

* note: cross over percentage of the initial 5,000 is 10% or 500 CNS samples

mutation percentage is 90% or 4500 CNS samples
for a total of: 5000 CNS samples

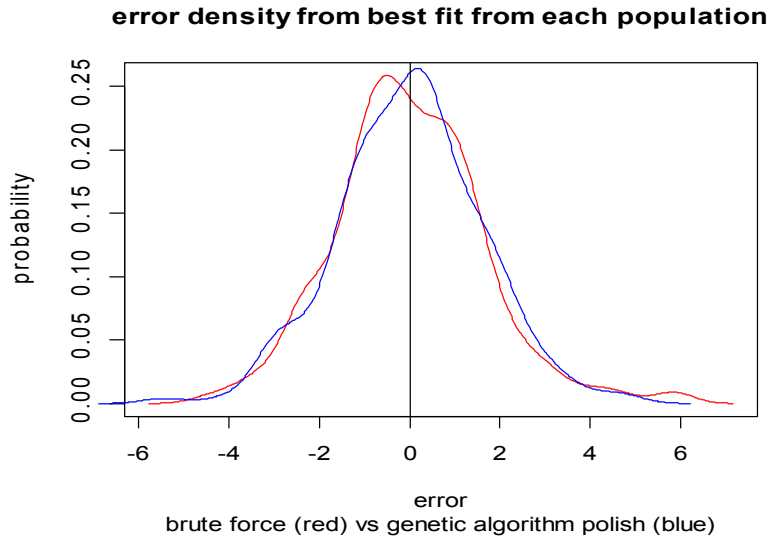
Below we see a comparison of population wide errors for brute force cns and g.a. polish:



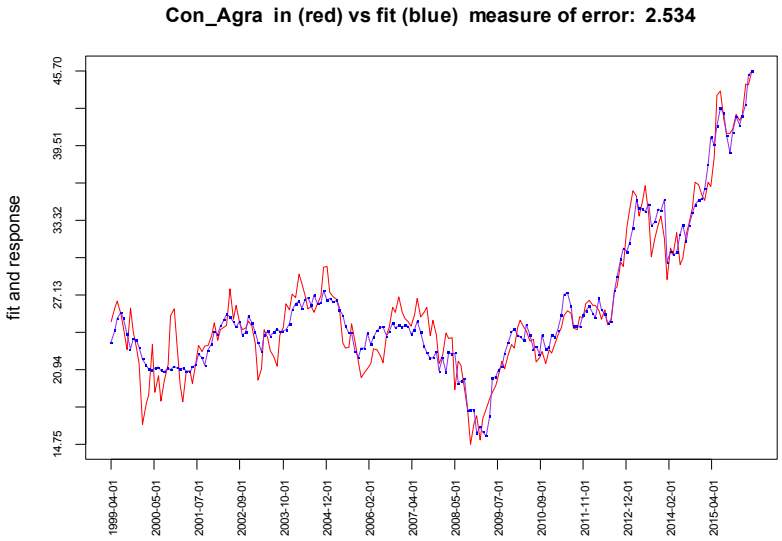


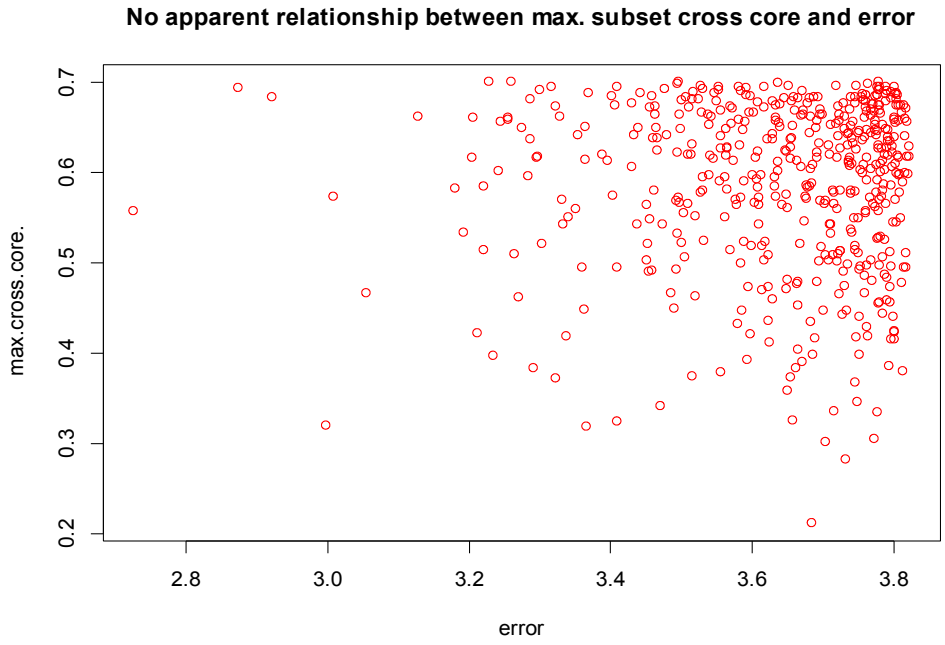
bfCNS Brute Force Monte Carlo Simulation with Correlation Networked Sampling
fitness function ppNN a localized ridge regression used in all simulations
error squared residual between the response (Con_Agra) and the fit from the regression based on the selected subset of 4 relatively independent variables

Comparing the brute force bench mark residuals against the g.a. bench mark regression's residuals reveals only a slight improvement.



The 'best' fit from the genetic algorithm polishing the brute force population.





Conclusion:

This technique demonstrates the possible efficacy of subtly biased sampling techniques (ex. Correlation Networked Sampling).

The Genetic Algorithm with Correlation Networked Sampling improved performance:

an 80% reduction in elapsed time;
a 7% reduction in benchmark error;
a great improvement concerning population-wide errors (see graphs above).
100% of the subsets in the population were relatively independent instead of only 50% with purely random sampling.

Surprizingly in these elitist simulations, *crossover* played a minor role:

Brute force 10,000 subsets
mutation 4,500 subsets
total number: 14,500 subsets

compared to only 500 subsets from crossover (500/15000 = 3.33%).

In this data context, this result may imply that evolution relies more heavily on massive parallelism and mutation rather than crossover.

Here is the same g.a. simulation with recombination percentages reversed:

Description	runLength	Mutation Pct	Cross Over Pct	bench error	elapsed sys time
G.A. Polish Mutation	5000	0.9	0.1	2.5340	41.310
G.A. Polish Crossover	5000	0.1	0.9	2.7272	40.990

As a caveat: in the face of explosive combinatorics for the cardinality of the feature space, hundreds (or thousands) of populations will produce similar or nearly identical bench errors in the same

elapsed time. In fact, any population will contain hundreds or thousands of nearly equivalent models. This is rather humbling. In this case study, these simulations tried to explore a feature space with approximately 4.669e+12 subset combinations! Hence, there may be millions of models that perform nearly well as your own carefully considered jewel of thought.

Foot Notes:
The fitness function used in this case study is a locally fitted ridge regression, called ppNN or projection-pursuit neural-net. Jerome Friedman at Stanford wrote ppr (ppNN) in Fortran IV. His opus is an unrecognized masterpiece.

I wrote the brute force simulator (mcSim) and the integer-valued genetic algorithm with integrated CNS and the time-series module entirely in R. Although I have tested these programs and database on AWS (vertically scaled up to 8 cpu's 2 gpu's, a cuda), I ran these simulations on my own desktop computer.