Scaling up Greedy Equivalence Search for Continuous Variables

Joseph D. Ramsey
Philosophy Department
Carnegie Mellon University
Pittsburgh, PA, USA
jdramsey@andrew.cmu.edu

Technical Report
Center for Causal Discovery
Pittsburgh, PA, USA

7/25/2015

Abstract

As standardly implemented in R or in the Tetrad program, causal search algorithms that have been most widely or effectively used in scientific problems have severe dimensionality constraints. However, implementation improvements are possible that extend the feasible dimensionality of search problems by several orders of magnitude. We describe optimizations for the Greedy Equivalence Search (GES) that allow search on 50,000 variable problems in 13 minutes for sparse models with 1000 samples, on a 4 processor 8G laptop computer, and in 18 hours for sparse models with 1000 samples on 1,000,000 variables on a supercomputer node at the Pittsburgh Supercomputing Center with 40 processors and 384 G RAM, on data generated i.i.d. from a linear, Gaussian model.

1. Introduction.

Under the auspices of the Center for Causal Discovery, the Greedy Equivalence Search (GES) algorithm has recently been scaled so that it will handle variable sets in the tens of thousands on a laptop computer, and in the hundreds of thousands on a supercomputer, with top sizes reached of 100,000 and 1,000,000 variables, respectively, for sparse models (with the number of edges equal to the number of nodes) and 1000 samples, for continuous variables. This is an improvement over previous work, which for our group maxed out at 5000 variables, requiring days of computation. In this report, we document the strategies used to achieve the scale-up and give some reference results.

The revised implementation of GES we call FastGES. In what follows we discuss it in stages. First, we discuss the basic assumptions and structure of the GES algorithm as given in Chickering

---

1 Research supported by the National Institutes of Health under Award Number U54HG008540. The content is solely the responsibility of the author and does not necessarily represent the official views of the National Institutes of Health. Special thanks to Clark Glymour, Peter Spirtes, and Erich Kummerfeld for algorithmic help, to Nick Nystrom and John Urbanic for help in running the algorithm on Pittsburgh Computing Center machines, and to Ricardo Silva for the original code that was optimized. Note that this report may be revised in light of code review.
(2002), with some adjustments due to Ricardo Silva. Next, we go over the main structural optimizations used for the FastGES implementation. It will be taken for granted that basic code profiling and common sense programming techniques will have been used; these will not be discussed. Next, we will define a simulation framework and give results for both a laptop computer and a supercomputer node. We will end with a brief discussion.

2. Assumptions.

GES assumes the data are generated from parameterizations of a true causal model characterized by a directed acyclic graph (DAG), in which directed edges $X \rightarrow Y$ indicate that $X$ is a cause of $Y$. A directed graph over a set of variables $V$ consisting entirely of directed edges. A path is a concatenation of edges, e.g. $X \rightarrow Y \rightarrow Z$; a directed path is a path consisting entirely of directed edges pointing in the same direction. A directed graph is acyclic just in case it contains no directed path from a variable to itself. Some variables may be measured, some unmeasured; the unmeasured ones are call latent variables. GES assumes additionally that the graph over the measured variables does not contain variables with latent common causes--that is, unmeasured variables $L$ such that $X \leftarrow L \rightarrow Y$ for $X$ and $Y$ measured. Under these assumptions, GES (like several other algorithms) produces a pattern or CPDAG, which is a mixed graph consisting of both directed and undirected edges representing an equivalence class of DAGs.

Let an unshielded collider be a path of the form $X \rightarrow Y \leftarrow Z$ where $X$ and $Z$ are not adjacent. The pattern will have oriented each unshielded triple in the graph, together with as many other orientations as can be inferred without introducing new unshielded colliders. A set of rules for accomplishing this is given in Meek (1995). All other edges will remain unoriented. A process for obtaining DAGs in the pattern suggests itself. One simply takes an edge that is unoriented in the pattern and gives it a direction, then orients as many other edges as possible by the Meek rules. This process may be repeated until a DAG has been created. The equivalence class consists of all DAGs that can be generated in this way.

For Gaussian and multinomial joint distributions, under the specified assumptions and i.i.d. sampling, GES converges to a pattern containing the true graph and all DAGs in the pattern are statistically indistinguishable by conditional independence relations among the measured variables.

3. Performance Measures.

Algorithm performance is reported in terms of elapsed time and accuracy. There will be four dimensions to the measurement of accuracy. First, the number of graph adjacencies in the output graph of an algorithm that do not exist in the true model are counted; these will be called the false positive adjacencies. Likewise, the number of graph adjacencies in the true model generating the data that do not appear in the algorithm result graph are calculated; these are called the false negative adjacencies. Third, the number of arrowheads in the output graph of the algorithm that do not appear in the corresponding place in the pattern of the graph of the true model generating the data are calculated; there are called the false positive arrowheads. And fourthly, the number of arrowheads in pattern of the true model generating the data that do not occur in the corresponding place in output graph of the algorithm are calculated; these are the false negative arrowheads. Algorithms may be sorted as more or less accurate depending on how small these numbers are. An ideal algorithm would have very few false positive and false negative adjacencies, and very few false positive and false negative arrowheads.
To be clear about false positive and negative arrowheads, consider some simple examples. Let the true model be $X \rightarrow Y$; the pattern for this is $X \rightarrow Y$. If the output model is $X \rightarrow Y$ or $X \rightarrow Y$, in each case, this will count as an arrowhead error. Likewise, if there is in fact no edge between $X$ and $Y$, if the output model is $X \rightarrow Y$ or $X \rightarrow Y$, this will count as an adjacency error and a direction error. And so on. Note that this method of counting compares the estimated pattern to the true pattern, not to the true graph, though adjacencies of the true graph are the same as the adjacencies of the true pattern. For instance if the true graph is $X \rightarrow Y$, the true pattern will be $X \rightarrow Y$, so if the estimated pattern is $X \rightarrow Y$, this will be counted as correct, even if the true direction is not known.

To make comparisons between models of various sizes easier, we further calculate false positive and negative adjacency and arrowhead error rates. Letting $CA$ be the number of correct adjacencies (that is, adjacencies that are in both the true and estimated patterns), $FPA$ the number of false positive adjacencies, and $FNA$ the number of false negative adjacencies, we calculate the adjacency false positive rate (FPR) as $FPA / (CA + FNA)$ and the adjacency false negative rate (FNR) as $FNA / (CA + FNA)$. Similarly for arrowhead false positive and negative rates.

It is worth noting up front that in the tested configuration increasing the amount of RAM does not necessarily improve running time or the ability of the process to finish without throwing exceptions. The reason is that covariances are calculated on the fly, so that only the data set need be stored in memory, together with requisite fields in the heap for processing the algorithm. This is a deliberate choice, since pre-calculating the covariance matrix severely limits the number of variables that can be analyzed. For problems of even moderate size, covariance matrices cannot be stored in memory for the machines we tested against; they are therefore not stored in matrix form but are rather computed on the fly as needed. (Note that if the number of samples exceeded the number of variables, or were even very much larger than 1000, this advice would be reversed.)

4. Overview of GES Algorithm.

GES uses a scoring to produce an output pattern, adding or removing one edge at a time in the space of patterns until the final pattern is arrived at. The algorithm breaks down into two pieces. In the first piece, the maximum scoring edge among all possible edges satisfying certain conditions is added to the pattern, and the orientation of the pattern adjusted for the addition. In the second piece, the maximum scoring edge among all possible edges satisfying certain conditions is removed, and the orientation of the pattern adjusted for the removal. In either case, the score of the model is monotonically increased.

The score used in the algorithm must be decomposable (Chickering 2002)--that is, the score of the model must be the sum of scores for each variable given its parents in the graph. The log of the Bayesian decomposition of the model is a decomposition of this nature; if $P(M) = P(X_1 \mid Pa(X_1)) P(X_2 \mid Pa(X_2)) \ldots P(X_n \mid Pa(X_n))$, where $X_1, \ldots, X_n$ are variables in graph $G$, $Pa(X)$ are the parents of $X$ in graph $G$, and $P(M)$ is the probability of the model, then taking logs, we have $\ln P(M) = \ln P(X_1 \mid Pa(X_1)) + \ldots + \ln P(X_n \mid Pa(X_n))$. I assume that the probability of the model is decomposable in this way. This formulation translates directly into a formulation for log likelihoods: $L = \ln P(M \mid D) = \ln P(X_1 \mid Pa(X_1), D) + \ldots + \ln P(X_n \mid Pa(X_n), D)$, where $D$ is a data set over variables $X_1, \ldots, X_n$. Since the likelihood of a variable given its parents for the linear, Gaussian case is equal to $-n \ln s + C$ for sample size $n$, residual variance $s$ after regressing $X$ onto its parents, and constant $C$, we have that the BIC score is equal to:

$$BIC = 2L - k \ln n = -n \ln s - k \ln n + C$$
(Schwarz, 1978) where $k$ is the number of degrees of freedom of the model. Notice that BIC is decomposable, since $L$ is decomposable and the penalty $k \ln n$ is the sum of penalties for individual BIC scores of variables given their parents. This score is adjusted in the direction of stringency for large models as follows:

\[
\text{BIC2} = 2L - c\, k \ln n
\]

where $k = 2p + 1$, for a variable given its parents $Y_1, \ldots, Y_p$, and $c$ is a penalty discount chosen empirically to be by default 2.

As indicated earlier, the forward phase of the algorithm consists of a loop for adding edges to the model, and the backwards phase consists of a loop for removing edges from the model. The forward phase begins with the problem of determining which single edge to add to the model to get things started. This is a very expensive step, especially for large models, and is quadratic in the number of variables. For each 2-set of variables $\{X, Y\}$ GES must score the model consisting of $Y$ with no parents, yielding score $s_1$, and then the model consisting of $Y$ with $X$ as a parent, yielding score $s_2$. The difference between these scores, $s_1 - s_2$, may be positive or not. If it is not positive, the edge $X \rightarrow Y$ is not added to the model. If it is positive, it is considered as a candidate to be added to the model. Out of all such candidates to be added to the model, the one with the highest difference in score is chosen, and this edge is added to the model. Note that the difference in score for $X \rightarrow Y$ is the same as the difference in score for $Y \rightarrow X$; only one of these need be tested. After this step, the graph is reverted to its pattern, in this case, by rendering the directed edge $X \rightarrow Y$ as undirected $X \leftrightarrow Y$.

For the next phase, for each edge $X \rightarrow Y$ considered for addition. Two sets are required for this, for each edge, NaYX and T (Chickering 2002). NaYX is the set of all $Z$ that can be reached from $Y$ by an undirected edge and are adjacent to $X$; $T$ is the set of all $Z$ that can be reached from $Y$ by an undirected edge and are not adjacent to $X$. Together, NaYX and T comprise all of the variables that can be reached from $Y$ by an undirected edge. Iterating over subsets $S$ of $T$, $X \rightarrow Y$ may be added to the graph if $\{Y\} \cup S \cup \text{NaYX}$ is not a clique and adding $X \rightarrow Y$ to the graph does not result in a cycle (that is, there must be no semidirected path from $Y$ to $X$, where a semidirected path from $Y$ to $X$ is a path from $Y$ to $X$ that consists entirely of undirected edges and edges pointing toward $X$). If these checks pass, GES then scores $Y$ with all of $S$ as parents, yielding score $s_1$, and $Y$ with all of $S$ as parents and $X$ in addition as a parent, yielding score $s_2$; the score difference is $s_1 - s_2$. If this difference in score is positive and is the highest out of all such calculated scores across all edges and all subsets $S$ of $T$ for each edge, the edge $X \rightarrow Y$ (with subset $S$) is added to the graph and $S' \rightarrow Y$ is reoriented for each $S'$ in $S$. (This reorientation of $S'$ into $Y$ is how colliders get oriented in the graph.) The graph is then reverted to its pattern by rendering as undirected all edges not involved in unshielded colliders and then applying the Meek rules. Note that this step may orient edges in the graph that were previously unoriented and may unorient some edges that were previously oriented. The forward step stops when there are no more such edges to add to the graph.\footnote{Note that this is somewhat different from the formulation in Chickering (2002), though compatible; we use this formulation to illustrate how FastGES is developed.}

Note that the size of the subsets $S$ of $T$ may be bounded if one expects most edges for large hubs to point away from a node; we have implemented this bound.
The backward step is similar to the forward step, except that subsets H of NaYX are calculated instead of subsets of T, the clique check is for NaYX \ H, the cycle check is not necessary, and when edges X→Y are removed, edges Y→H' are rendered as Y→H for each H' in H.

5. Optimizations for FastGES.

Profiling a naive implementation of the version of the GES algorithm reveals two basic bottlenecks. First, there is considerable redundancy in scoring. The initial step of adding a single edge to the graph is quadratic in the number of variables, and adding each additional edge to the graph is quadratic in the number of variables. Most of the scoring in each of these steps is repeated many times over. Much of the redundancy in this scoring can be reduced. Second, even if scoring is reduced to a minimum, the process of calculating BIC scores itself is extraordinarily expensive. While this step can be optimized by calculating regressions directly from the covariance matrix, it is still expensive.

There are many ways to address the redundant scoring problem; our method is as follows. We keep a list L of score differences, along with associated information, consisting of a structures of "arrows" A = <d, X→Y, S, NaYX> for the forward step. In these structures, d is the score difference, X→Y is the edge whose addition to the graph is contemplated, S is a subset of the set of variables (T) adjacent to Y by undirected edges but not adjacent X, and NaYX is the set of variables adjacent to Y by undirected edges and also adjacent to X. L need only contain arrows with positive score differences; arrows with non-positive score differences are excluded from L. The list L is kept in sorted order high to low by d; a collection class is available in Java for this (ConcurrentSkipListSet). Note that the edge X→Y may appear in more than one arrow, since such arrows are calculated for each subset of the T set described above, though when an arrow is removed from the list, all other arrows with different subsets of T must be removed as well, or at least ignored, in order to maintain the consistent property that the list of edges consists of a subset of edges that would have been calculated at a particular stage of the algorithm. The edge with the highest score is always added to the graph, so long as it is a legal edge (i.e. passes the clique and cyclicity tests). When an edge, say X→Y, is added to the graph, possibly reverted to X←Y, scores for edges into X or Y, say, W→Y, need to be rescored, since Y possibly has a new parent, say X→Y←W, now with Y having two parents where before it just had one. The full rescore step takes every node W in the graph other than X or Y and rescores the edge W→X, and if X is adjacent to Y, X→W, and similarly for W→Y and Y→W. This is an expensive step, but it is linear in the number of variables rather than quadratic, so there is significant time saving.

The revised algorithm, therefore, constructs a list of arrows sorted by score differences. After an extensive search, the first edge X→Y is added to the graph and removed from L, and the graph reverted to its pattern, X←Y. Then in the manner described above, new edges are added to L for variables adjacent to X or to Y whose score differences are positive. Then the first arrow in L is removed, tested for legality, and if the tests pass, the edge A of that arrow is added to the graph, selected adjacent edges are reoriented, the graph reverted to a pattern, and then arrows for directed edges adjacent to the endpoints of A are added to L, if their score differences are positive. And so on, until the forward stage is finished, which occurs when the list L is empty. The backwards phase then commences, populating the list L and proceeding until L is empty, and the final graph is reported. The implementation with this change alone scales easily to 1000 variables, and with great patience to 5000 variables.

3 That is, by removing all arrowpoints not part of unshielded triples and then applying the Meek rules, in this case a trivial operation.
There is still quite a lot of redundant scoring. If Y with no parents is scored, and then X→Y is scored, the score for Y with no parents is scored twice. However, eliminating this redundancy for large models requires expensive maps, which have not been pursued successfully here.

Instead, the scoring of the algorithm has been speeded up through parallelization. In the above algorithm, there are two steps than can be parallelized easily. First, there is the initial step of adding a single edge to the graph. For this each single edge needs to be evaluated; these operations can be done in any order, independently, with the same result. It's a very expensive step, and a good candidate for parallelization. The other step that can be parallelized to good effect is the scoring of additional edges after the addition of each edge to the model. Once X→Y is added to the graph, for each W in the graph other than X or Y, W→X, X→W, W→Y, and Y→W need to be reevaluated. These are independent checks and can easily be parallelized. With these steps parallelized, the algorithm can be scaled up to 35,000 variables on 8 G, 4 processor Macbook Air with patience.

Scaling up further is possible; one possibility is as to strengthen a particular assumption of GES. GES as defined does not assume faithfulness per se but something weaker, more akin to triangle faithfulness (Spirtes and Zhang, 2014). GES can actually discover the model A→B→C→D, A→D, even if the A→B→C→D path exactly cancels the A→D path.\(^4\) This assumption is strengthened to a weak version of faithfulness when doing scoring regressions—namely, that if it is known that A has zero total effect on D, then A can be removed as a parent when regressing D on its parents A and C. This allows us to do a preprocessing step in which all pairs of variables <X, Y> for which X and Y have zero total effect on one another are identified, and X→Y and Y→X can simply be removed from all scoring regression models. This allows us to speed up the algorithm considerably. The step of identifying the first edge X→Y to be added to the graph is unchanged, but after X→Y is added, we do not need to consider all W adjacent to X or to Y, just those such that <W, X> or <W, Y> are effective connections—i.e. have non-zero total effect. Doing this on the above MacBook Pro reduces running time by a factor of 5 and saves a good deal of memory. Notice that this is primarily a strengthening of assumptions for purposes of scaling up; for smaller models, where more recall is desired, and where one has more time to wait for a result to come back, a version of GES that does not make this strengthening is preferable.

Also, the step of rebuilding the pattern after each edge addition may be optimized. The method outlined above is a global method. One eliminates all orientations from the graph that are not directly involved in unshielded colliders and then applies the Meek orientation rules to the entire graph. This is onerous when the graph becomes large. Moreover, it is unnecessary; one may limit revisions of orientation to just the variables directly involved in orientation changes. When X→Y is added to the graph, one needs to reorient edges about X and edges about Y and adjust any other orientations in the graph that result. So about X and Y, one can eliminate orientations not directly involved in unshielded colliders, and then apply Meek rules to X and Y, and the surrounding variables. This has two kinds of implications beyond X and Y and their adjacent. First, new orientations may propagate by the Meek rules applied to adjacent variables; these new orientations need to be allowed to propagate and quite substantial. Second, if for instance triangles are formed in the graph, some orientations adjacent to X and Y may be eliminated, and this may result in further orientations being eliminated. So elimination of orientations needs to be allowed to propagate as well. This may be accomplished by identifying edges W→X or W←X that have been unoriented as W→X by applying the above procedure (reverting to unshielded

\(^{4}\) Personal communication from Peter Spirtes.
collider orientations and propagating Meek rules) and then applying the same procedure to W (and similarly for Y). This way, all new orientations and unorientations will be propagated, and the effect on the graph will be the same as if all directed edges not part of unshielded colliders had been replaced by undirected edges and the Meek rules run on the entire graph. Doing this considerably speeds up the algorithm, by a factor of 5.

A notable point is that the ConcurrentSkipListSet collection uses the score difference to order arrows from largest score difference to smallest, but this alone does not account for the possibility that two arrows may have the same score difference. In this case, both arrows need to be added to the list. This issue is finessed by adjusting the ordering of arrows in the ConcurrentSkipListSet; arrows with the same score difference are arbitrarily ordered by subtracting the hash codes of the two arrows, where these hash codes are carefully selected to distinguish among arrows.

6. What is parallelized?

It helps to pause a moment and review what exactly has been parallelized in this implementation of the algorithm. It only makes sense to run an algorithm on a supercomputer if it has been sufficiently parallelized, and GES is not an algorithm that lends itself to easy parallelization. Nevertheless, it can be parallelized fruitfully in a few places, one of which is of great help.

First, the preparation of the covariance matrix can be parallelized by variables. If the covariance matrix is being precalculated (which it is not here), it may be parallelized by arranging all of the covariances as a two-dimensional array, where only the lower triangle need be computed, and then dividing this array into column-wise groups, sending each group to a different thread—so-called “striping.” We are calculating covariances on the fly, but variances are heavily used and deserve precalculation. These can be parallelized by simply dividing the variances into groups.

The most effective parallelization done in the algorithm is to precalculate the possibilities for the initial edge to add to the graph in the first step. For sparse graphs, this is the most expensive step in the algorithm, usually taking far and away the most time to complete. The edges that must be considered by the procedure may be arranged as a lower triangle of a two-dimensional array; these may again be striped by dividing the columns of this array into groups and sending each column to a different thread. This procedure uses almost all processors for even very large machines. If it doesn't, the algorithm can be adjusted so that it uses more, by perhaps dividing up tasks more evenly among threads, or by adjusting the number of columns sent to each thread individually.

Beyond that, one has choices as to what to parallelize, and none of them is particularly good. The overall structure of the second part of the GES algorithm (which includes most of the algorithmic code) is not terribly amenable to parallelization, for one simple reason. The overarching idea of GES in the forward phase is to find the best edge to add to the graph, then given that edge, to find the next best edge, and given those two edges, the third best edge, and do on. These are conditional decisions and must be done in serial, not in parallel, and there are a lot of them. However, within each decision some reasoning may be parallelized to a degree. For instance, in the forward phase, after an edge has been added, one must recalculate quite a number of score differences that may have been affected by the new edge—that is, cases where there may be a new parent of a node given the edge, or its reverse, in cases where the direction of the edge is not fixed by Meek orientation. These rescorings can be listed, and the list tackled by the rescorer in

---

5 This was pointed out by P. Spirtes and E. Kummerfeld, in conversation.
parallel, by simply breaking the list down into smaller lists. This can be done both in the forward phase and in the backward phase. This generally increases the number of processors dedicated to the forward (or backward) phase from one to upwards of six, but generally no more. On our Pittsburgh Supercomputer node with 40 processors, calculating the initial edge easily uses all 40 processors, but the second step of the algorithm, calculating the rest of the edges, uses four to six processors. The saving grace is that the second part of the algorithm takes a lot less time to execute than the first, even with both parallelized. Note that with denser graphs, more processors will be used in the second step, since the size of the T sets will be greater.

Beyond that, no parallelization has been done; speedups have been due to overarching strategic choices and detailed profiler work.

7. Simulation Framework.

We aim at doing well on simulation tests of the following form. For each simulation, choose a sparse graph with the number of edges equal to the number of nodes, obtained by adding random forward edges to nodes arranged in a list. Parameterize each such graph as a structural equation model (SEM), for instance this one with the graph $X\rightarrow Y$:

\[
X := eX \\
Y := a X + eY
\]

In this model, $X$ and $Y$ are continuous variables, $eX$ and $eY$ are disturbances, and $a$ is a linear coefficient. We assume that $eX$ and $eY$ are distributed according to some zero-centered Gaussian distribution. The procedure of testing is to generate a random graph, to parameterize this graph as a structural equation model, as above, with coefficients randomly drawn uniformly from $(-1.5, -0.5)$ $U (0.5, 1.5)$ and coefficients from $U(1, 3)$, then to draw 1000 samples from the specified distribution recursively and i.i.d., and then to run the optimized GES algorithm on this data, comparing edges in the result back to the equivalence class (i.e. the pattern, or CPDAG) of the original graph, reporting back discrepancies in directions and arrowhead orientations. Thus, all models tested are linear and Gaussian, all variables continuous, and all data i.i.d.; we are testing scale-up under these conditions and leaving for future work the problem of testing scale-up when these conditions do not obtain. Because the sizes of the models are large, only one repetition is tested per run.

It is one of the goals of the Center to allow algorithms to be run on laptop computers as well as on a supercomputer, when the capacities of laptop computers are insufficient. For testing, we choose as an example laptop computer a MacBook Air with 2 cores (each with 2 hardware threads, for a total of 4 processors) and 8G RAM. For a supercomputer machine, we choose a node at the Pittsburgh Supercomputer Center (PSC) with 40 processors and 384 G of RAM. In both cases, using a machine with more processors would significantly reduce running times and allow larger models to comfortably be explored. A MacBook Air seems like a fairly good choice for a laptop machine on which to test the performance of GES; it is not especially large, even in its maximum configuration—in this case with 2 processors, 2 hardware threads each, and 8 G of RAM; at the present time, this is a middle of the road machine, certainly with no more capacity than one would expect for anyone who might want to use this algorithm. We are careful, as indicated earlier, to calculate covariances on the fly, which is feasible for a thousand samples; representing covariance matrices directly, even using 4-byte numerical representation, would quickly use up all of the available 8 G of RAM and would severely limit the size of the problem that can be considered.
8. Results.

On the MacBook Air, we test sparse models of 10,000 to 100,000 variables. Table 1 shows the results. The false positive rates are bounded above by 1% throughout this range for adjacencies and 3% for arrowhead orientations, and the false negative rates for adjacencies do not exceed 3% for adjacencies or 4% for arrowhead orientations. These are very usable results. Because of the size of the models, the number of repetitions per result is limited to 1.

Table 1. Results for FastGES on the 16 MB, 8 Aircessor MacBook Air. "AH" abbreviates "Arrowhead."  

| # Vars | Time (min) | Adj FPR | Adj FNR | AH FPR | AH FNR |
|--------|------------|---------|---------|--------|--------|
| 10000  | 0.5        | 0.47%   | 2.84%   | 1.17%  | 4.47%  |
| 20000  | 2.1        | 0.31%   | 2.55%   | 1.05%  | 3.92%  |
| 30000  | 4.5        | 0.37%   | 2.49%   | 1.02%  | 4.07%  |
| 40000  | 8.8        | 0.47%   | 2.50%   | 1.22%  | 3.82%  |
| 50000  | 13.0       | 0.66%   | 2.55%   | 1.78%  | 3.95%  |
| 60000  | 17.4       | 0.71%   | 2.48%   | 1.73%  | 3.89%  |
| 70000  | 25.3       | 0.82%   | 2.46%   | 1.86%  | 3.88%  |
| 80000  | 33.6       | 0.98%   | 2.47%   | 2.38%  | 3.73%  |
| 90000  | 43.5       | 0.99%   | 2.59%   | 2.32%  | 3.93%  |
| 100000 | 50.5       | 1.13%   | 2.49%   | 2.73%  | 3.80%  |

Timing results unsurprisingly increase quadratically with the number of variables. On the MacBook Air, one gets rather quickly to the point where virtual memory is used, though the algorithm is able to keep going to higher numbers of variables. Unsurprisingly, the larger the computer, the easier it is to ascend to larger numbers of variables; an 8 processor 16 G MacBook Pro already handles these larger numbers much more easily.

For the Pittsburgh Supercomputer node, though, with 40 processors and 384 G of RAM, it is considerably easier to ascend to larger numbers of variables; a table extending the above to 1,000,000 variables is given in Table 6. The memory limit for representing covariances directly on this machine was under 200,000 variables; by calculating covariances on the fly one obtains results for larger numbers of variables. (With larger machines the covariances can be stored directly; we did not test on such machines.) For all results except for 1,000,000 variables a penalty discount of 2 is used; for the 1,000,000 variable result, a penalty discount of 4 is used. The reason why we increased the penalty discount for the 1,000,000 variable run was the trend in increasing false positive rates for the models with 10,000 to 500,000 variables; increasing the penalty discount led to lower false positive rates and higher false negative rates. Different choices could have been made.

---

6 Note as explained earlier that adjacency and arrowhead orientation percentages are with respect to the true pattern.
Table 2 Results for FastGES on a supercomputer node with 40 processors and 384 MB RAM on a 1,000,000 variable problem, calculating covariances on the fly. "AH" abbreviates "Arrowhead." See text.¹

| # Vars | Time (min) | Adj FPR | Adj FNR | AH FPR | AH FNR |
|--------|------------|---------|---------|--------|--------|
| 10000  | 0.2        | 0.20%   | 2.60%   | 0.69%  | 3.68%  |
| 20000  | 0.4        | 0.32%   | 2.59%   | 0.90%  | 3.74%  |
| 30000  | 0.9        | 0.39%   | 2.81%   | 1.04%  | 4.35%  |
| 40000  | 1.5        | 0.54%   | 2.49%   | 1.29%  | 3.83%  |
| 50000  | 2.4        | 0.61%   | 2.46%   | 1.53%  | 3.88%  |
| 60000  | 3.1        | 0.74%   | 2.56%   | 1.75%  | 3.96%  |
| 70000  | 4.2        | 0.78%   | 2.53%   | 2.04%  | 3.90%  |
| 80000  | 5.8        | 0.89%   | 2.58%   | 2.03%  | 3.89%  |
| 90000  | 7.0        | 1.08%   | 2.54%   | 2.53%  | 3.92%  |
| 100000 | 8.5        | 1.15%   | 2.51%   | 2.72%  | 3.83%  |
| 200000 | 31.6       | 2.24%   | 2.52%   | 4.97%  | 3.84%  |
| 500000 | 202.6      | 5.43%   | 2.51%   | 11.62% | 3.81%  |
| 1000000| 1070.5     | 0.02%   | 6.53%   | 0.13%  | 9.60%  |

In general, false positive rates increase with increasing numbers of variables with the current implementation and noticeably so when the number of variables becomes large. False negative rates remain firm, bounded above by about 3 for adjacencies and 4 for orientations—the 1 million variable case excepted, but even there false negative orientations are under 10%. As indicated, the false positive rates may be controlled by increasing the penalty discount of the search. Time increases quadratically, with the largest search (1,000,000 variables) taking 18 hours. Again, these are very usable results.

8. Discussion.

Software, including Java code, will be made available once code review is complete, hopefully very soon. This technical report will be updated at that time with instructions on how to download the code, which will be made available through the Center for Causal Discovery. Feedback is very welcome. There was no sense that we had reached the limit of optimization for this problem,

¹Note as explained earlier that adjacency and arrowhead orientation percentages are with respect to the true pattern.
just that we had gone much further than previous attempts. We would welcome ideas for further optimizing the code.

It is clear that Java is very fast for this problem, despite early dire warnings to the contrary. Perhaps a C++ implementation would be even faster. However, the Java implementation would clearly be faster as well on a machine with more processors; that hasn’t been tried yet.

Although we haven't discussed it here, the algorithm can be used to analyze discrete variables using a discrete decomposable score such as BIC or BDeu; we use BDeu, as recommended by Chickering (2002). We have scaled this up to 10,000 variables, with excellent precision and fair recall. (The accuracy very much depends on the choice of values for hyperparameters of the score, a cautionary note.) Work is underway to try to scale this discrete score up further; for large sample sizes, AD trees will be incorporated; for increasing the number of variables, we are looking into techniques to accomplish this.

FastGES handles somewhat denser models relatively easily, though this does slow the algorithm down. Nevertheless, anecdotaly, accuracy is good even when the number of edges is two or even three times the number of nodes.

It turns out that a fast and accurate pattern search such as FastGES can serve as a component in modifications of other algorithms, such as FCI (Spirtes et al., 2000), with improvements in speed and accuracy. We have successfully performed such combinations and hope to bring out modified algorithms in the near future with these changes.

In short, FastGES is a very fast and accurate implementation of GES for the continuous variable case, suitable to application to large data sets. One side comment (as suggested briefly earlier) is that in applying FastGES to large data sets, one may discover the existence of massive hubs. It is reasonable to suspect that such hubs exist in graphs the structure of which is suspected to be scale free, such as biological networks. So long as edges point away from these hubs, FastGES works fairly well, in our experience. The existence of such hubs slows the algorithm down to an extent, so the number of variables that can comfortably be analyzed might be more on the order of 300,000 or 400,000, still within the range of many big data problems.

References

D. Chickering (2003). Optimal structure identification with greedy search. The Journal of Machine Learning Research 3: 507-554.

C. Meek (1995). Causal inference and causal explanation with background knowledge. Proceedings of the Eleventh conference on Uncertainty in artificial intelligence: 403-410.

G. Schwarz (1978). Estimating the dimension of a model. Annals of Statistics 6 (2):461-464.

P. Spirtes, C. Glymour, and R. Scheines (2000). Causation, Prediction, and Search. MIT Press.

P. Spirtes and J. Zhang (2014). A uniformly consistent estimator of causal effects under the k-triangle-faithfulness assumption. Statistical Science:29, 662-678.