Texas College Mathematics Journal
Volume 3, Number 2, Pages 10–26
S applied for(XX)0000-0
Article electronically published on February 3, 2006

A GENETIC ALGORITHM FOR DRAWING ORDERED SETS

BRIAN LOFT AND JOHN SNOW

Abstract. An order relation is a binary relation which is reflexive, antisymmetric, and transitive. Order relations occur naturally in many areas of mathematics (two basic examples are the ordering of the real numbers and the subset relation). The structure of a small set with an order relation can often be completely described by drawing a diagram called a Hasse diagram.

A genetic algorithm is a problem solving technique which treats potential solutions to a problem as biological lifeforms which compete and reproduce in a process similar to natural selection in hopes of generating an acceptable solution.

In this manuscript, we introduce the basics of genetic algorithms and develop a genetic algorithm for drawing Hasse diagrams of ordered sets.

1. Introduction

A genetic algorithm is a problem solving technique which mimics the process of natural selection. Potential solutions to a problem are viewed as biological lifeforms which can reproduce. The process begins with a random population of these lifeforms which are tested to see how close they are to being solutions. The better solutions are deemed more *fit* than other members of the population and are allowed to mate to produce offspring which will compose a new population. This process is repeated until an acceptable solution to the problem is found.

Ordered sets arise naturally and frequently in all areas of general algebra. Ordered sets consisting of subalgebras, normal subgroups, and ideals are some common examples. An indispensable tool when working with ordered sets is the ability to draw diagrams of small ordered sets which expose the structure of the order. In this manuscript we develop a genetic algorithm for drawing ordered sets. The purpose for this algorithm is to generate a clear enough picture of an ordered set so that it may be recognizeable if it is a familiar ordered set. The diagrams which result can often be manipulated to obtain diagrams of higher quality which can be used for publications. Unless otherwise noted, all of the ordered sets in this manuscript were drawn using the genetic algorithm.

Section 2 of this manuscript is a basic introduction to genetic algorithms in general. There is no original material in this section, and everything included here could be found in any survey of genetic algorithms such as [6, 7, 8]. The reader may refer to these for a more thorough treatment of genetic algorithms. Section 3 briefly defines ordered sets and the Hasse diagrams which we will be drawing. Our

Received by the editors August 22, 2005.
1991 *Mathematics Subject Classification.* Primary 68W20; Secondary 06A06.
Key words and phrases. genetic algorithm, ordered set.

specific genetic algorithm for drawing ordered sets and some examples drawn by the algorithm are presented in Section 4.

2. Genetic Algorithms

The idea of employing an "evolutionary strategy" in problem solving was introduced in a 1973 paper of I. Rechenberg [5]. His ideas lead to the development of the concept of a genetic algorithm in 1975 by John Holland and his students [3]. We present a restricted model of a genetic algorithm here as a basic introduction to the idea.

2.1. **Biological Inspiration.** The characteristics of a biological lifeform are determined by genes (blocks of DNA) linked together in strings called *chromosomes*. When two lifeforms reproduce, corresponding chromosomes from each parent are twisted together in a process called *recombination* or *crossover* to form a chromosome in the offspring. This process occurs in the following way. A chromosome from Parent A is paired with a chromosome from Parent B. A random crossover point along the chromosome is chosen, and the two parent chromosomes are cut at this point. The genetic material from Parent A's chromosome before the crossover point is glued together with the material from Parent B's chromosome after the crossover point to form a new chromosome for the offspring (see Figure 1). Recombination

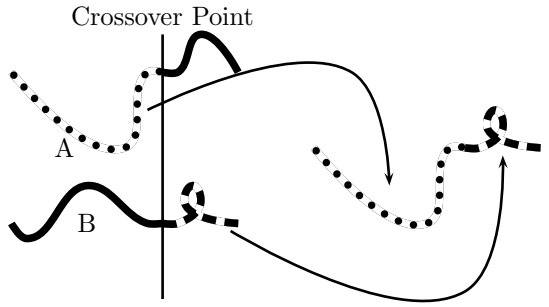

Figure 1. In recombination, the genetic material from the first parent's chromosome before the crossover point is combined with the material from the second parent's chromosome after the crossover point.

has the property that if both parents share a particular pattern in their chromosomes then this pattern will carry over to the offspring. Therefore, good (and bad) qualities which are shared by the parents can be passed on to the offspring.

The process of copying genes from the parents to offspring is imperfect. There are sometimes errors in copying which we call mutation. This mutation introduces a certain randomness into the genetic material of the offspring. This randomness might be negative or positive. It may even have no noticeable effects on the offspring.

If an organism is strong and survives long enough to reproduce, then some of its genetic material is passed on to the next generation. If an organism is too weak to survive long enough to reproduce, then its genetic material is removed from the population.

Genetic algorithms attempt to mimic this situation. Possible solutions to a problem are envisioned as organisms each with a chromosome that contains the genetic material which encodes its individual traits. This chromosome is simply a list of symbols – usually 0's and 1's. At the simplest level, recombination works as described above. A crossover point is chosen at random. The two parents' chromosomes are cut at this crossover point. The first segment of one is glued to the second segment of the other to form the chromosome of the offspring. The offspring's chromosome is then mutated by randomly changing some of the symbols in it.

The genetic algorithm employs a fitness function which determines how good of a solution a particular lifeform is to the problem being addressed. Those organisms which display a greater fitness are given a greater chance to reproduce. First, a random population of organisms is generated, and the entire population is tested for fitness. Then some of the members of the population are selected for reproduction with the most fit organisms being more likely to reproduce. Some of the genetic material in the offspring is mutated. Then the new generation of offspring replaces the previous generation. To insure that the new generation is at least as fit as the previous generation, some of the most fit members of the parent generation may be included with the offspring. This is called *elitism*. The process is repeated until an acceptable solution is found. Here is an outline of a typical genetic algorithm:

(1) **Initial Population**: An initial population of organisms is randomly generated.
(2) **Fitness Testing**: The fitness function is applied to each member of the population.
(3) **Solution**: If an acceptable solution is found in the population during testing then the algorithm terminates.
(4) **Reproduction**:
    4.1 **Selection**: Pairs of organisms are selected from the population for mating with probability based on fitness.
    4.2 **Recombination**: A crossover or recombination operation is applied to the chromosomes of each pair selected for reproduction to produce a new organism.
    4.3 **Mutation**: The chromosome of the new organism is mutated with some small probability.
(5) **Next Generation**: Some or all of the offspring produced in the previous step are chosen to form the new population. This population may also include the most elite members of the previous population.
(6) **Repeat**: Goto step 2.

2.2. **Encoding and Fitness.** The structure of a genetic algorithm is quite general. The means of selection, recombination, and mutation are often not tied to the problem at hand. The specific problem affects most directly how organisms are encoded and how fitness is computed. The most popular means of encoding a chromosome is as a sequence of 0's and 1's as most data can usually be represented in this way. However, genetic algorithms have been applied with chromosomes that are strings of integers, floating point numbers, and arbitrary symbols. John Koza has even applied the ideas behind genetic algorithms to "breed" computer programs in genetic programming [4]. Recombination may be more complicated in

these more general chromosomes, so we isolate our attention to strings of 0's and 1's.

By far the most difficult and most important step in building a genetic algorithm is constructing the fitness function. This is also the most problem-specific step. As we will demonstrate with our genetic algorithm for drawing ordered sets below, the fitness function can be influenced by a variety of parameters. The fitness function must be fast because the fitness of every organism in a population must be tested every generation in a genetic algorithm.

2.3. **Recombination.** The simplest type of recombination is one point crossover. Chromosomes are strings of symbols (here, 0's and 1's). We begin with two chromosomes, one from Parent A and one from Parent B, both the same length. A crossover point is selected randomly. The two chromosomes are cut at this point, and a new chromosome is formed by using the chromosome from Parent A before the crossover point and from Parent B after the crossover point. This is depicted in Figure 2.

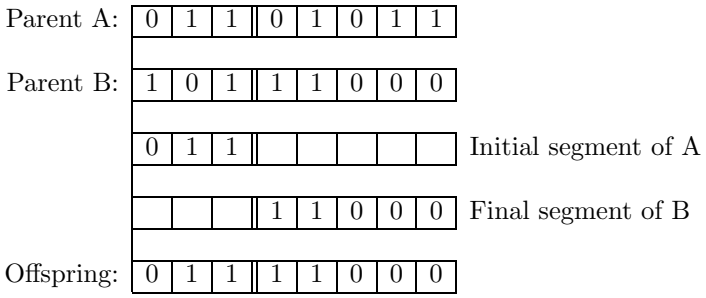

FIGURE 2. One point crossover.

A first generalization of one point crossover is two point crossover. In two point crossover, two crossover points are randomly chosen. Genetic material is copied from A before the first crossover point. Between the crossover points, material is taken from Parent B. After the second crossover point, material is again taken from A. This is depicted in Figure 3.

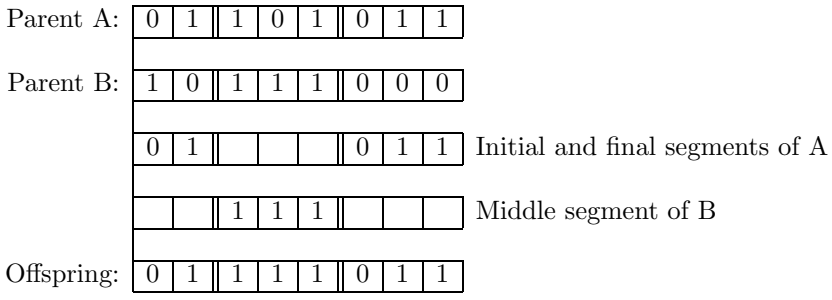

FIGURE 3. Two point crossover.

From two point crossover, one can imagine three point crossover, four point, five point, and so on. The logical conclusion of this is uniform crossover. In uniform

crossover, each symbol in the offspring's chromosome is chosen randomly to be equal to the corresponding symbol of the chromosome of either Parent A or Parent B. This is depicted in Figure 4.

Parent A:	0	1	1	0	1	0	1	1
Parent B:	1	0	1	1	1	0	0	0
Offspring:	0	0	1	0	1	0	0	1

FIGURE 4. Uniform crossover. The underlined symbols are the ones chosen for the offspring.

One observation we should make about crossover in any of these forms is that if a pattern appears in the chromosomes of both parents, then recombination will preserve that pattern. For example, both parents above have 1's as the third and fifth symbol in their chromosomes and 0 as the sixth. You can see that in one point, two point, and uniform crossover the offspring has the same pattern.

2.4. **Mutation.** Mutation is usually applied after the process of recombination. The simplest manner in which this is done is to select a small number of symbols in the offspring's chromosome and replace them with random symbols. If the only symbols are 0 and 1, then this amounts to selecting a few random symbols and toggling them between 0 and 1. In most circumstances, mutation should only be applied in a very limited way.

If a population possesses a few very fit members then it may only take a few generations before the entire population resembles these members – even if they are not acceptable solutions. Mutation helps to avoid this *premature convergence*. In so doing, mutation helps to provide a deeper gene pool so that the genetic algorithm may have more chances to find a good solution.

2.5. **Selection.** Selecting organisms from a population to breed can be done in a number of ways. One popular method is roulette wheel selection. The sum S of the fitness values of the entire population is calculated. For any organism x in the population, denote the fitness of x as $F(x)$. Organisms are selected randomly from the population so that the likelihood of selecting any particular individual x is $F(x)/S$. Of course, for this to work $F(x)$ must not be negative.

The problem with this means of selection is that a single organism's fitness might account for so large a portion of the total sum of fitnesses that it is chosen too frequently for reproduction. This can be avoided by employing a rank function rather than directly using the fitness function. Individuals in the population might be assigned ranks, 1, 2, 3, and so on (with 1 being the worst) based on fitness. These ranks can then be used in place of fitness for roulette wheel selection.

Another alternative is to sort the population by fitness and apply a fixed probability of selection to the most elite organisms. For example, one might insist that the top ten percent of the population is chosen for reproduction eighty percent of the time. An extreme version of this is that the elite of the population are chosen to mate with each other, and the rest of the population is abandoned.

2.6. **The next generation.** Once the population has been tested for fitness, parents have been selected, recombination has occurred, and mutation has been applied, then the next generation of organisms must be chosen. One way to do this is to simply use all of the offspring to make the new generation. A problem with this is that recombination and mutation are not always successful in creating more fit organisms. The children may be less fit than the parents. By throwing away the parent generation, the algorithm would be accepting worse solutions. An alternative is to always include the most fit parents along with the offspring to form a new generation. This is called elitism.

2.7. **Islands.** As generations pass, genetic algorithms tend to converge. After several generations, the population may be biased towards certain genetic patterns, and all of the organisms my resemble each other. Mutation helps to slow this convergence down until the algorithm has had time to explore the gene pool more thoroughly. Another tool for avoiding bias and insuring genetic variety is using multiple populations.

The algorithm can begin with multiple random populations rather than just one. Each of these populations we will call an *island*. Each island is tested for fitness and undergoes selection, recombination, and mutation independent of the others. Genetic material from the islands can then be mixed by migrating organisms from island to island. This migration helps to spread around genetic patterns which develop on different islands. An alternative to migration is letting the islands develop for several generations in isolation until they begin to converge. Then the elite of each island are collected into one *elite population* on which the genetic algorithm can operate, hoping to combine the best traits from the islands to find an organism which is an even better solution than any one island could produce.

2.8. **Why does it work?** Suppose that chromosomes are sequences of 0's and 1's of length k. (For the examples in this section only, we take $k = 8$.) Any sequence of 0's, 1's, and $*$'s we will call a *schema*. A chromosome will be said to fit a schema if it matches the schema in all of the places which are not $*$'s. For example, 00110101 and 00110100 both match the schema $0 * 11 * 10*$, but 11110000 does not. The set of all chromosomes which match a specific schema will be called a hyperplane. There are $3^k - 1$ schemas (ignoring the one that is all $*$'s), and each chromosome lies in $2^k - 1$ hyperplanes. Each chromosome is completely determined by the $2^k - 1$ hyperplanes which contain it. One method for attempting to maximize a fitness function F on the set of all chromosomes is this:

(1) List all possible schema.
(2) Select a random sample from the hyperplane associated to each schema in the list.
(3) Find the average fitness value for each random sample.
(4) Remove from the list those schema for which the random sample gave low average fitnesses.
(5) Goto step 2.

The process is repeated until there are few enough schema in the list so that it is possible to tell where the maximum value(s) of F may occur. The problem with this brute force approach is that if k is very large at all then there are too many schema to list in this manner. The genetic algorithm approximates this search. Each chromosome of each individual in a population is a representative of $2^k - 1$

hyperplanes and can be thought of as a member of a random sample of each these hyperplanes. Thus the chromosomes of one population represent random samples of many hyperplanes. Moreover, if two chromosomes lie in the same hyperplane, then any recombination of those chromosomes will also lie in that hyperplane. Thus selection and recombination of fit lifeforms amounts to performing further sampling in hyperplanes with witnesses of high fitness. By focusing on the lifeforms rather than the hyperplanes, genetic algorithms attempt to sample and test multiple hyperplanes in parallel.

3. Lattices and order

A binary relation $\leq$ on a set A is an *order relation* if for all x, y, z in A the following three conditions hold:

(1) $x \leq x$ ($\leq$ is reflexive).
(2) If $x \leq y$ and $y \leq x$, then $x = y$ ($\leq$ is antisymmetric).
(3) If $x \leq y$ and $y \leq z$, then $x \leq z$ ($\leq$ is transitive).

These are all properties of the usual order relation on the real numbers. The set A along with the order $\leq$ is called an ordered set.

For any two elements x and y of an ordered set, if $x \leq y$ and $x \neq y$, then we can write $x < y$. If $x < y$ and there is no z with $x < z$ and $z < y$, then we will say that y is a *cover* of x. In symbols, we write this as $x \prec y$. If x and y are elements of an ordered set and $x \leq y$ or $y \leq x$, then we say that x and y are *comparable*. Otherwise, x and y are *incomparable*.

A *Hasse diagram* of an ordered set is a picture of the ordered set drawn in the following way. Each element of the ordered set is represented by a small circle. If $x \leq y$ in the ordered set, then the circle for x must be located lower in the diagram than the circle for y. Lines segments are then added to the diagram to depict the order relation. If $x \prec y$, then a line segment is drawn from the circle for x to the circle for y.

For example, suppose that $A = \{0, 1, a, b, c, d\}$ and the order on A is such that 0 is covered by a, b, and c, each of a, b, and d is covered by 1, and $c \prec d$. Then the Hasse diagram for this ordered set is depicted in Figure 5.

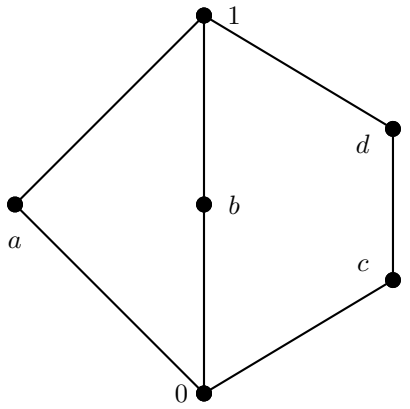

Figure 5. An example of a Hasse Diagram.

Most ordered sets obtained from algebraic structures (such as ordered sets of ideals or subgroups) have two special properties: For any two elements x and y, there is a least element which is greater than both x and y – called the least upper bound of x and y. Also, for any two elements x and y, there is a greatest element which is less than both x and y – called the greatest lower bound. For example, if H and K are subgroups of a group G, then the greatest lower bound of H and K is $H \cap K$, and the least upper bound of H and K is the subgroup generated by $H \cup K$. An ordered set in which every pair of elements has a greatest lower bound and a least upper bound is called a *lattice*. The ordered set in Figure 6 is not a lattice. The elements x and y have no common lower bound (and hence no greatest lower bound). They do have common upper bounds, but no least upper bound. Either of these failure would have prevented the ordered set from being a lattice.

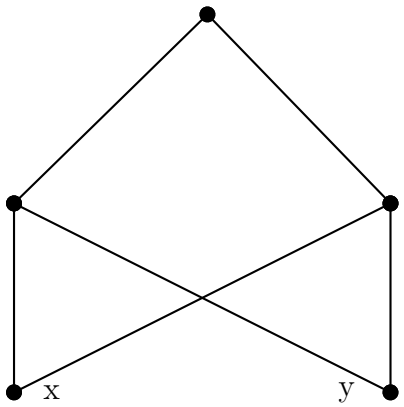

FIGURE 6. An ordered set which is not a lattice.

An algorithm for drawing Hasse diagrams by hand might look something like this:

(1) Scatter circles labelled by the elements of the ordered set on a page.
(2) Slide the circles around to get lesser elements of the ordered set lower on the page.
(3) Draw line segments representing the covering relation of the order.
(4) Slide the circles around on the drawing to *untangle* the line segments.

In step 2, care should be taken to make sure that whenever $x \leq y$ the circle for x is lower than the circle for y. The untangling should clean up the picture enough so as to be aesthetically pleasing while at the same time exposing the structure of the ordered set. For example, in Figure 7, the diagram on the left depicts the same ordered set as the one on the right, but is essentially random. The one on the right, although complicated, is spaced evenly, has many parallel line segments, and exposes the structure of the ordered set as consisting of two identical components (the lower left half looks just like the upper right). There are a variety of opinions about what makes a *good* diagram of an ordered set. One could try to minimize the number of line crossings or minimize the number of different slopes of the line segments. In [1] diagrams are drawn to minimize the total length of all of the line segments. In [2] R. Freese describes an algorithm which tries to balance attractive

forces between comparable elements and repulsive forces between incomparable elements.

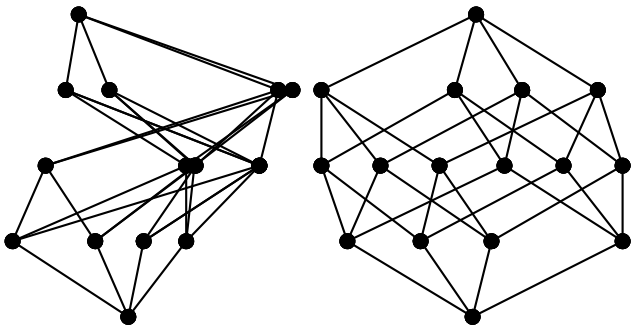

FIGURE 7. These two Hasse diagrams depict the same ordered set. The one on the right does so in a clearer and more aesthetically pleasing way.

4. Breeding Lattices

We will employ a genetic algorithm to draw ordered sets. Each circle in the Hasse diagram of an ordered set A will be determined by a pair of coordinates (x, y) with x and y varying between 0 and 255. The y (vertical) component of each point is determined by the ordering of A independent of the genetic algorithm. A chain in an ordered set is an increasing list of elements

$$a_0 < a_1 < a_2 < a_3 < \ldots < a_l.$$

The length of this particular chain is l. The height of an element a in A will be the length of the longest chain in A with top element a. We will denote this by $h(a)$. The depth $d(a)$ of a is the length of the longest chain in A whose least element is a. The height H of our ordered set is the maximum value of $h(a)$ taken over all $a \in A$. Two possible numbers that could be used to describe the height of a are $h(a)$ and $H - d(a)$, depending on if we care about how far an element is from the bottom or from the top of the ordered set. We will let the y coordinate of the circle corresponding to a (denoted $y(a)$) be the average of these two measures scaled to give a value between 0 and 255. That is, $y(a)$ is the integer part of

$$\frac{h(a) + (H - d(a))}{2} \cdot \frac{255}{H}.$$

Assuming our ordered set consists of n elements and that we need only determine the x coordinates of n points, it is convenient to encode each ordered set as a chromosome with $8n$ symbols (or as n chromosomes, each with 8 symbols) – where each symbol is either a 0 or a 1. Each 8 symbols (called a byte) represents the binary expansion of the x coordinate of one of the circles in our diagram. The numbers expressible by a binary integer of length 8 are 0 through 255 – hence the range for possible x values.

4.1. **Our specific genetic algorithm.** We use an elitist genetic algorithm which will breed through a fixed number of generations on multiple islands followed by a fixed number of generations beginning with a elite population with representatives from each island. The algorithm will use uniform crossover and a high rate of mutation. We mutate an entire byte in the chromosome at one time. This mutation seemed reasonable for this application as changing one byte amounts to changing the position of one circle in the diagram. To balance the high rate of mutation, we only accept those mutations that increase fitness.

Let E be a positive integer. We will have E different islands, each with a population of E^2 organisms. In each population, the E most fit members will all mate with each other (and themselves) to form the E^2 members of the next population. When each organism mates with itself, the offspring is identical to the parent, so the next generation will contain copies of the elite from the previous generation. After a fixed number G of generations pass on each island, the most fit organism on each island is selected to join a elite population. This population will reproduce for G generations.

Here is the outline of our algorithm. We first list a routine for reproduction which will be called by the main algorithm.

Routine: **NextPopulation**
Input: A population of E^2 organisms.
Output: A population of E^2 organisms.

(1) Test the fitness of each organism in the input population.
(2) Sort the input population by fitness.
(3) For $i = 1$ to E and for $j = 1$ to E
  3.1 Let x be a uniform crossover of input organisms i and j.
  3.2 Let y be x with one chromosome byte randomly replaced.
  3.3 Test the fitness of x and y.
  3.4 Place the more fit of x and y in the output population.
(4) Return the output population.

Now we can outline the entire genetic algorithm:

Genetic Algorithm

(1) Fill each of E islands with a random population of E^2 organisms.
(2) For each of the E islands do
  2.1 For $g = 1$ to G replace island population P with **NextPopulation**(P).
  2.2 Test the fitness of each member of the population.
  2.3 Sort the population by fitness.
  2.4 Place the most fit organism from the population into the Elite Population.
(3) For $i = 1$ to G replace the Elite Population with **NextPopulation**(Elite Population)
(4) Test the fitness of each member of the Elite Population.
(5) Sort the Elite Race by fitness.
(6) Return the E most fit members of the Elite Population.

4.2. **Recombination and Mutation Example.** We give here an example of recombination and mutation in action with our ordered set lifeforms. Pictured in Figure 8 are two random arrangements of the same ordered sets. Figure 9 depicts

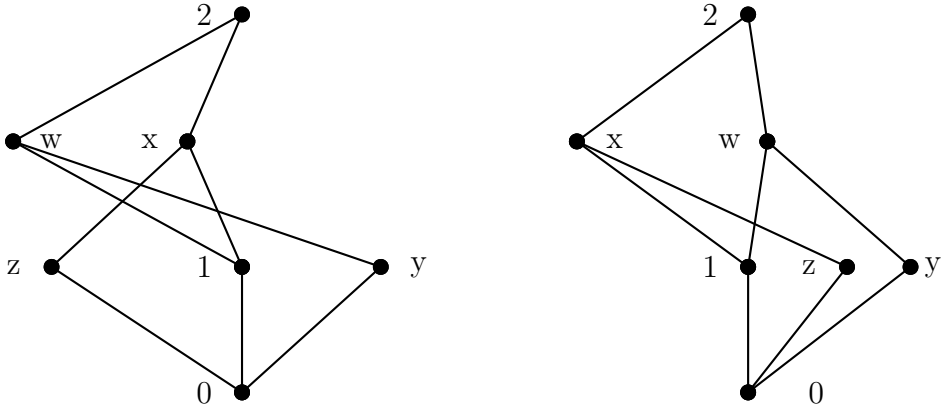

FIGURE 8. Two random arrangements of the same ordered set.

the recombination of the parents from Figure 8 with uniform crossover. Notice how the elements 0, 1, and 2 have the same positions in both parents. This characteristic is preserved by the recombination. Also notice that x is to the left of the center and y to the right in both parents. This is also the case in the offspring. We

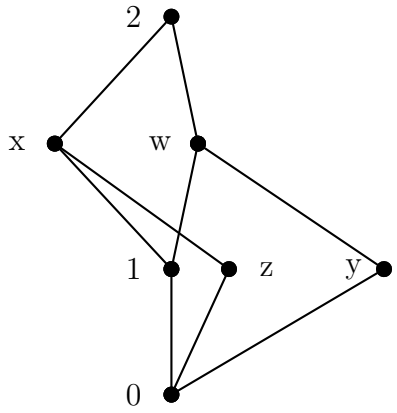

FIGURE 9. A recombination of the parents in Figure 8 using uniform crossover. Notice how the configurations of 0, 1, 2, x, and y which were common to both parents were preserved by recombination.

will now mutate the lifeform in Figure 9 by replacing one byte in the chromosome randomly. This has the affect of randomly moving one circle in the diagram. The mutation is depicted in Figure 10.

4.3. **The fitness function.** In designing the fitness function, we maintained two basic values:

(1) Comparable elements should be close to each other.
(2) Incomparable elements with similar vertical positions should not be too close horizontally.

We chose to measure:

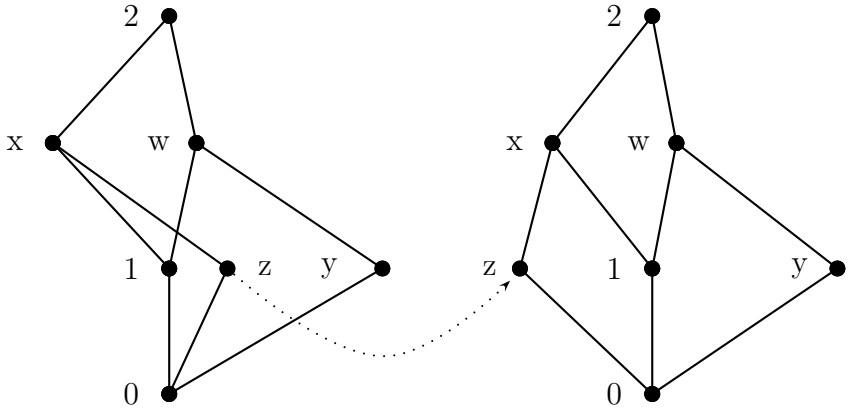

FIGURE 10. In this mutation the byte for element z is randomly replaced.

C: The average squared horizontal distance between elements a and b for which $a \prec b$.

Y: The average squared horizontal distance between elements with the same y value.

H: The minimum squared horizontal distance between elements with the same depth.

D: The minimum squared horizontal distance between elements with the same height.

We chose our fitness function to be a linear combination of these variables so that the coefficient of C is negative and the other coefficients are positive. The effect of this is that large values of C (covers with large distances) lower fitness while small values of C (covers with small distances) contribute to a larger fitness value. Similarly, large values of other variables increase fitness, and small values serve to make fitness lower. This should result in a genetic algorithm which tries to make C small while making the other variables large.

A great deal of trial and error was necessary to find values of the coefficients which would give acceptable diagrams for a variety of ordered sets. If the coefficient of C is too large compared to the other variables, then the algorithm focuses too much on the closeness of compatible elements and quickly converges to a diagram in which all x values are equal. On the other hand, if the coefficients of the other variables are too large, then the diagram which results typically has all x values of 0 or 255. For the examples below, we chose the fitness function to be

$$Y + \frac{1}{2}H + \frac{1}{2}D - 8C.$$

We arrived at these coeficients in the following manner. Since scaling the fitness function would not change the rank of members of a population, we (arbitrarily) selected the coefficient of Y to be 1. To maintain vertical symmetry (when it exists), we decided that the coeficients of H and D should be the same. With this arrangement, we needed only select two values, the common coefficient of H and D and the coefficient of C. We ran the algorithm on a test suite of ordered sets with values for these coefficients varying in magnitude from 0 to 10 by integer values. The values of 0 and 8 or 1 and 8 seemed to give the largest number of acceptable

diagrams, so we selected -8 for the coefficient of C and then tested values for the other coeficients between 0 and 1, finally settling on 0.5.

4.4. **Some examples.** We include here some examples of diagrams drawn by our genetic algorithm. Each diagram is scaled so that the ratio of the height of the picture to the width of the picture is the same as the ratio of the height of the ordered set to the width of the ordered set. Here, width corresponds to the largest number of elements with the same y value. For these ordered sets, we used $E = 20$ and $G = 30$ in the genetic algorithm.

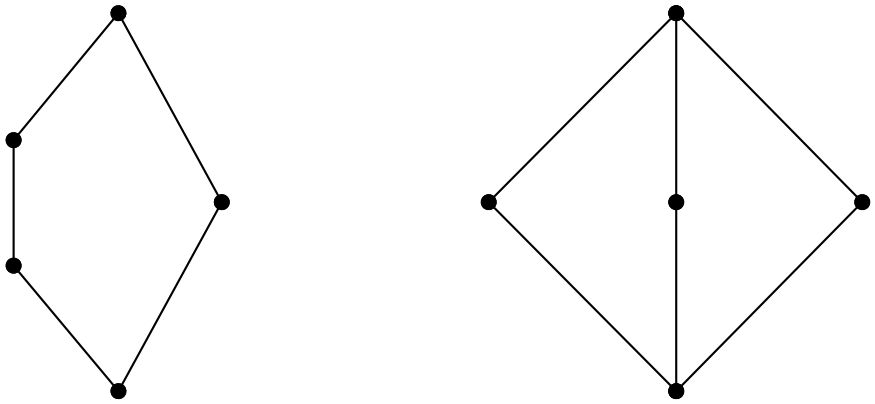

FIGURE 11. The lattices $\mathbf{N}_5$ (left) and $\mathbf{M}_3$ (right) are drawn by the algorithm more or less as expected. (Although $\mathbf{N}_5$ might be considered backwards.)

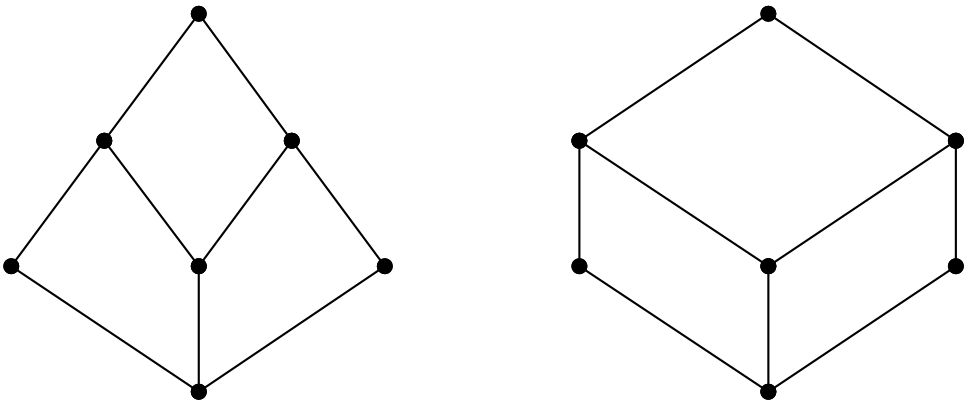

FIGURE 12. The lattice $\mathbf{D}_2$ as drawn with the default choices for coefficients (left) and with a larger Y coeficient (right).

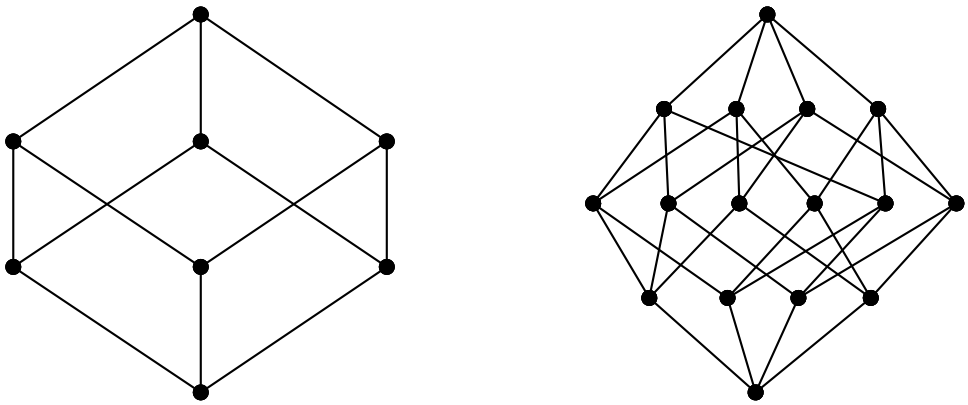

FIGURE 13. The eight and sixteen element boolean lattices are drawn by the algorithm about as one would draw them by hand.

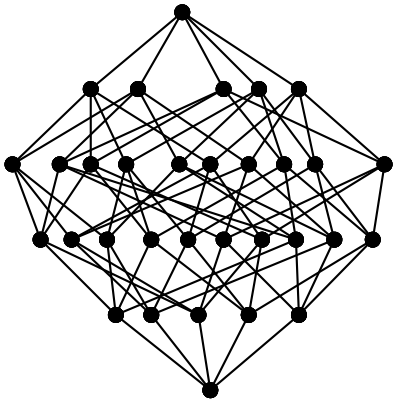

FIGURE 14. The 32 element boolean lattice comes out a bit less symmetric but is recognizable.

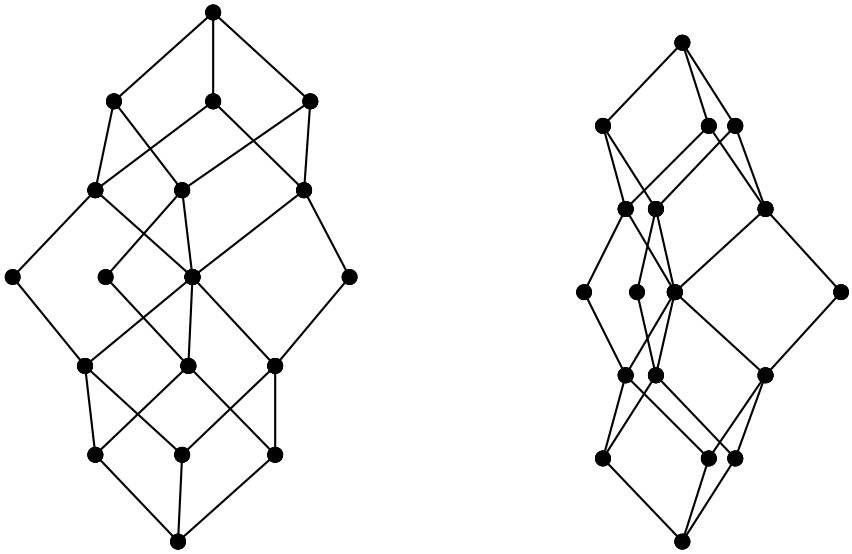

FIGURE 15. The free distributive lattice on three generators as drawn by the genetic algorithm (left) and as drawn by R. Freese's algorithm [2] (right).

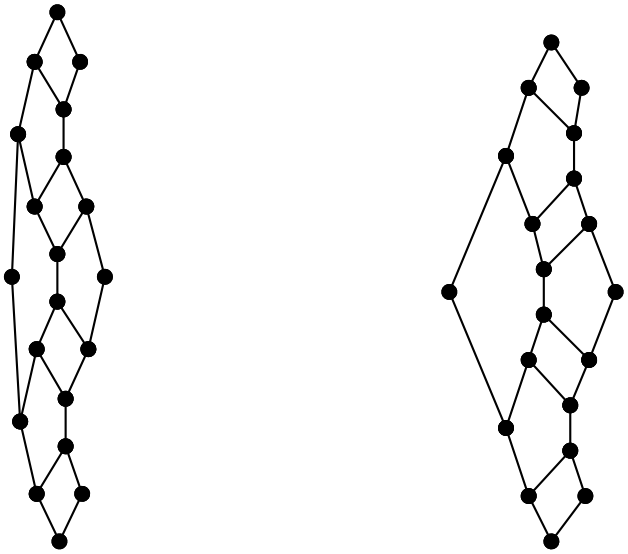

FIGURE 16. The lattice $\mathbf{FL}(3+1)$ drawn by the genetic algorithm (left) and drawn by R. Freese's algorithm [2] (right).

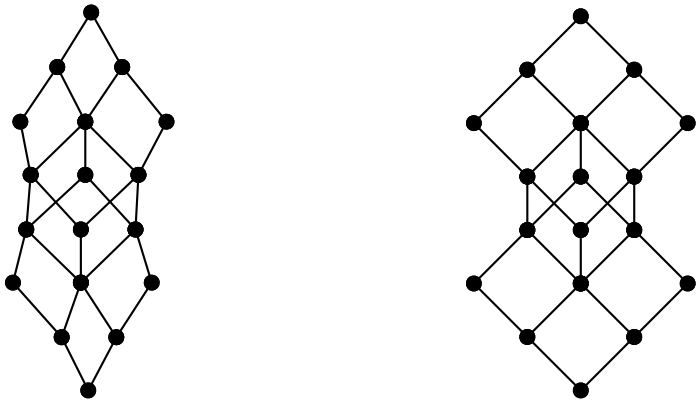

FIGURE 17. The lattice $\mathbf{FM}(2+2)$ drawn by the genetic algorithm (left) and drawn by hand (right).

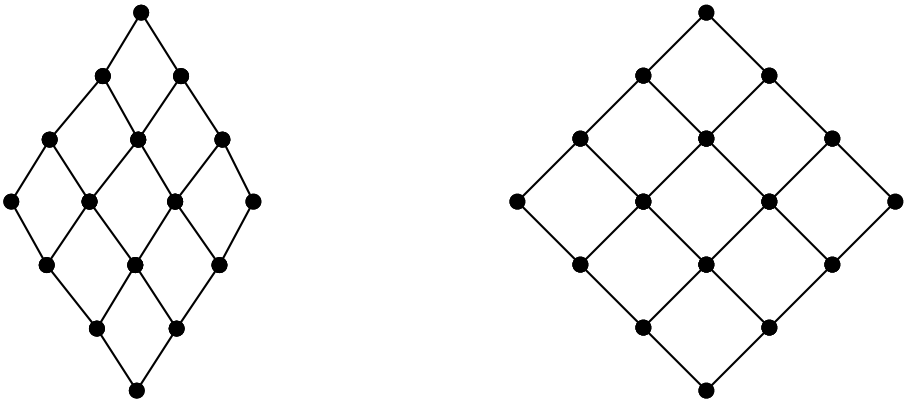

FIGURE 18. A grid-like lattice drawn by the genetic algorithm (left) and by hand (right).

5. Conclusion

In summary, the genetic algorithm here was successful at drawing Hasse diagrams of ordered sets for the purpose of identification. Some of the diagrams generated may not appear "crisp" enough for publication purposes, but the coordinates of the circles in the generated diagrams can usually be adjusted by hand to give more acceptable pictures.

Further work in this area may include a more rigorous manner of determining useful fitness coefficients (for example, using a least-squares approach). Rather than selecting one set of fitness coefficients, it may be more reasonable to select multiple sets of coefficients, each of which perform well on certain types of ordered sets, and then have the program output several options for a Hasse diagram. Perhaps each set of coefficients could correspond to an island in the genetic algorithm process. A more sophisticated approach would examine the ordered set and select the coefficients to be used based on individual characteristics of the poset.

References

[1] A. Aeschlimann. Drawing orders using less ink. *Order*, 9:5–13, 1992.
[2] R. Freese. Automated lattice drawing. *Lecture notes in artificial intelligence*, 2961:112–127, 2004. Springer, Berlin.
[3] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.
[4] J. Koza. *Genetic Progamming: On the programming of computers by means of natural selection*. 1992.
[5] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer System nach Prinzipien der biologischen evolution*. Frommann-Holzboog Verland, Stuttgart, 1973.
[6] L. M. Schmitt. Theory of genetic algorithms. *Theoretical Computer Science*, 259:1–61, 2001.
[7] M. Vose. *The simple genetic algorithm: Foundations and Theory*. MIT Press, Cambridge, MA, 1999.
[8] D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4:65–85, 1994.

Department of Mathematics and Statistics, Sam Houston State University, Huntsville, Texas 77341-2206
E-mail address: loft@shsu.edu

Department of Mathematics and Statistics, Sam Houston State University, Huntsville, Texas 77341-2206
E-mail address: jsnow@shsu.edu