A Genetic Algorithm for k-distinct lattice paths

March 2, 2025

Abstract

This paper uses a small combinatorial problem to illustrate the power of genetic algorithms. Finding the number of distinct paths on an $m \times n$ lattice is canonical problem, which can be extended by asking that the set of paths be k-distinct, with no two paths sharing k or more edges. In previous work,Gillman et. al. conjectured that these sets could be found with a greedy algorithm. However, Engstrom & Yager showed that this was false using an inefficient brute force algorithm which becomes computationally expensive for large lattices. We use a genetic algorithm to efficiently find maximum sets of k-distinct lattice paths and compare its performance to the other two algorithms.

1 Introduction

On an $m \times n$ lattice, there are a total of $m + n$ edges in each path from the southwest corner to the northeast corner (only moving East and North), and two paths are k-distinct if they share fewer than k edges. Otherwise, they are k-equivalent. A set of k-distinct paths, $\mathbf{P}$, contains paths $p_1, p_2, \ldots, p_P$ for a total of P paths. These paths may be ordered alphabetically with E moves before N moves. We want to determine the maximum number, $P(m, n, k)$, of k-distinct paths on the $m \times n$ lattice. We know that the set of all possible paths, $\mathbf{C}$, has order $C = \binom{m+n}{n}$.

A simple example is shown in Figure 1. The green path is 2-equivalent to the blue path and the red path. However, the blue and red paths are not equivalent to each other. This lack of transitivity is what makes the problem challenging and interesting.

A greedy algorithm which finds an approximation to the maximal set has been known for some time [5], [6]. Engstrom & Yager constructed a brute-force algorithm to investigate the accuracy of the greedy algorithm. In the process of finding solution errors in the greedy algorithm, they also discovered that their algorithm was exceedingly slow [10].

Our approach is to use a genetic algorithms to solve the problem. A genetic algorithm is a type of optimization algorithm that uses techniques inspired by evolution to search for optimal solutions. Specifically, our algorithm generates a

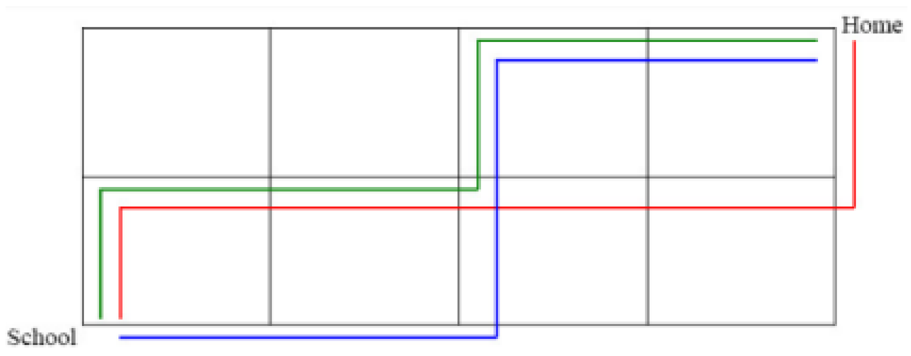

Figure 1: A Simple Illustration of the Problem

population of sets of paths and evaluates them based on the number of distinct paths that can be found. The genetic algorithm then uses selection, crossover, and mutation to evolve the population towards better solutions[8].

2 Prior Algorithms

Before discussing the greedy algorithm, we take a few paragraphs to sketch both the greedy algorithm and the brute force algorithm.

The greedy algorithm begins with an alphabetized list of the paths in **C**. The first path in **C** is added to our candidate set **P**, then each subsequent path is tested (in alphabetical order) to see if it is k-distinct from the paths already in **P**. A possible improvement, not explored, called *backtracking*, involves replacing the last path added to a solution to see if this leads to an improved solution.

The brute force algorithm begins with the same set of paths, **C**, not necessarily in alphabetical order. Then the algorithm constructs all combinations of order c of these paths. If a set of k-distinct paths is found, c is incremented by 1 and the process is iterated.

Both of these methods yield results consistent with proven results on extreme values of m, n, and k. (As does the genetic algorithm.)

At the end of the paper, we share the time complexity of these two algorithms (the first is fast and the second exceedingly slow), as well as some evidence about their accuracy in mid-range values (the first is increasingly wrong and the second is always correct.)

3 Notation and Processes

3.1 Basic Definitions

Here are the basic definitions required to develop our genetic algorithm.

- We will call a lattice path, p, a *gene*.
- Two genes are said to be k-distinct if they share at most $k-1$ edges. Otherwise, they are considered k-equivalent. A *individual* is a set of genes, and is denoted ϕ.
- A *population* is a set of individuals and has an order denoted by s.
- A *solution* is a individual who has genes that are mutually k-distinct. An *optimal solution* is a solution with a maximal number of genes.

3.2 Fitness

The *fitness* of an individual, ϕ, is a fraction, $f(\phi)$ that represents how k-distinct its constituent paths are. The fitness of ϕ, is given by

$$f(\phi) = \frac{1}{x+1},$$

where x is the number of pairs of genes in ϕ that are k-equivalent. In an individual with t genes, the maximum fitness is 1 and the minimum fitness is $\frac{1}{t(t-1)+1}$. Any individual with a maximum fitness of 1 has mutually distinct genes and is therefore a solution.

3.3 Divergence

The *divergence* of an individual, $d(\phi)$ is a measure of how different an individual is from the rest of the population. In each generation, the distribution of $\mathbf{C}$, the set of all possible genes, over the current population is calculated. Any gene p in $\mathbf{C}$, can be, at maximum, in all of the individuals in the population and, at a minimum, in none. So a measure of how p is distributed in the population is the proportion of individuals that contain the gene.

$$d(p) = \frac{\sum_{\phi | p \in \phi} 1}{s}$$

This calculation is done for every gene and then the divergence of an individual

$$d(\phi) = 1 - \frac{\sum_{d(p) | p \in \phi} 1}{t},$$

where s is the population size, and t the number of genes in an individual.

4 Creating the next Generation: Evolution

The next generation of individuals is created, and evolution occurs, via an optimization process by mating individuals in the population in each generation to produce new, fitter, individuals. The process is briefly summarized in the following paragraphs.

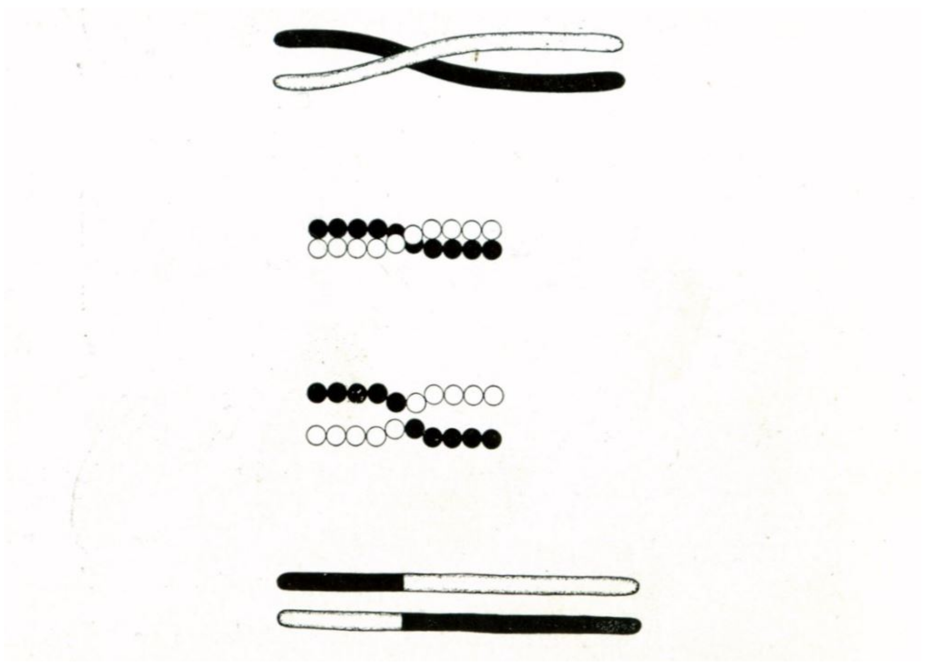

Figure 2: Crossover of genes between two parents to produce two individuals

In a population of size s, we let r denote the mating rate of the population. That is, there will be $r \times s$ matings, hence involving $r \times s \times 2$ individuals. Setting r to zero implies that the next generation is simply a copy of the current generation. (A specific value for r, and other parameters, is given in Section 7.)

When two individuals are selected to mate, the new individuals they produce is simply a splicing of the genes of the two parents. We list the genes of each parent in alphabetical order, and a random point is selected along the sequence and the genes before that point will be taken from one parent, and the genes after it will be taken from the other parent. Using both options, this process results in two new individuals.

This process is illustrated in Figure 2. The individuals that are selected to reproduce are chosen randomly from a Poisson distribution favoring those with higher fitness. The mating probability of an individual, $MP(\phi)$, is a probability that determines how likely an individual is going to be selected for mating. It is a function of the fitness $f(\phi)$ and divergence $d(\phi)$ of the individual and is calculated as follows:

$$MP(\phi) = (u \times f(\phi) + (1 - u) \times d(\phi)) \times \max(f(\phi)),$$

where u is a real number in the range $[0, 1]$ indicating the relative importance of fitness and divergence.

Given that there are $r \times s$ matings, we can create a mating pool with $n = r \times s \times 2$ slots. To determine the number of copies of itself each individual, ϕ , has in the pool, we multiply its mating probability, $MP(\phi)$, with n, and give the whole number part as the number of slots. Then we perform a Bernoulli trial on the decimal part to determine whether or not we should give an additional slot to the individual.

We always mutate one of the two offspring by having one of its genes randomly swapped for another from the set of all possible genes. Since we can know which genes are k-equivalent to others, we can exploit that to increase the pace of the search. The second offspring is only mutated once in every 100 mating, when one of its genes is replaced at random.

4.1 Life and Death

At the end of the mating and mutating events, there will be more individuals available than need to move forward to the next generation. At this point, we introduce a survival competition to reduce the population to the appropriate size. Briefly, we randomly select two individuals from the original population. The one with the greater fitness "lives" while the other "dies." This is repeated until the population is reduced to its established size, s.

5 Refinements

Two problems that a genetic algorithm might encounter are early convergence and low fitness variation.

Early convergence occurs at the start of the process. Suppose an individual has a significantly higher fitness than the other individuals, although probably not the optimal solution. This will lead to an unusually high mating probability for that individual and will lead the population to converge towards this individual, since it mates more often and its offspring are likely to be like it, with high but not optimal fitness.

Low fitness variation happens after many generations, as the average population fitness increases and stagnates. The differences in individual fitness will be very small, even if there is a high divergence, thereby slowing down the improvement in the evolution towards the maximum fitness since no individual is particularly fitter than the others.

To address these problems, we introduce a scaled fitness, Sf, to calculate the mating probabilities. Specifically, rather than using fitness to calculate the mating probabilities, we scale the fitness in every generation such that for the first generation the maximum scaled fitness is slightly above the average fitness and after many generations, the maximum scaled fitness is much higher than the average fitness. To accomplish this, we introduce a new parameter, v, which is the desired scaling factor for fitness in the final generation and let

$$v_i = d + (v - d) \times (i/g),$$

be the scaling factor in generation i with $1.5 < d < 2$, and thus in a given generation,

$$\max(Sf(\phi)) = (\sum f(\phi)/s) * v_i.$$

We want to ensure that the average scaled fitness of each generation is the same as its average fitness, as given by the equation

$$\sum (Sf(\phi)/s = \sum f(\phi)/s$$

Finally, we also want the minimum scaled fitness to be close to 0.

$$0 < \min(Sf(\phi) << 1$$

Using the conditions described above, we solve the system of equations at every generation to obtain values a and b in each generation i to obtain that generation's scaled fitness,

$$Sf(\phi) = f(\phi) \times a + b,$$

In summary, we will use the following equation for computing mating probabilities.

$$MP(\phi) = (u \times Sf(\phi) + (1-u)d(\phi)) \times \max(Sf(\phi)),$$

6 The Algorithm

We present an overview of the algorithm in this section, while the detailed coding is available at [7].

Due to the random nature of this search, we add one more level of complexity by implementing the algorithm in multiple ecosystems. Using smaller values for v makes the process explore more of the search space, but might take too much time to find the solution. This implies that ecosystems with higher values of v can find the solution quickly if it is an easy one and those with smaller values of v will explore more if it is a difficult solution. Hence, we can use ecosystems with different values of v to expand and accelerate the search process.

1. Initialize the algorithm.
   (a) Input the problem parameters, m, n, k.
   (b) Input the global computational parameters, g and u.
   (c) Construct the set $\mathbf{C}$.
   (d) Initialize the number of genes in a individual at $t = 2$.
2. For each ecosystem, j,
   (a) Input the ecosystem parameters v_j and the initial value of s_j.
   (b) Construct G_0, with individual size t and population size s_j.

(c) While $i < g$,

   i. Determine the fitness of each individual in G_i.
   ii. Determine the divergence of individuals in G_i.
   iii. Determine the scaled fitness of each individual.
   iv. Determine the mating probability of each individual.
   v. Test for optimal fitness. If an individual is optimal, update the individual size to $t + 1$.
   vi. Check whether any other ecosystem has found an optimal solution. If yes, increment the value of t by 1, and return to step 2. Else if all cores have completed g generations, output value of $P(m, n, k) = t - 1$.

(d) Create generation G_{i+1} with an updated individual size.

   i. Randomly, but biased towards people with high fitness scores, select members of G_i to move to generation G_{i+1}
   ii. Randomly generate children of G_i to add to G_{i+1}
   iii. Randomly add mutations of members of G_i to G_{i+1}
   iv. Randomly "kill" individuals to reduce the population to size s.

7 Comparisons

We experimentally found that using four ecosystems with parameters

$$g = m \times n \times k \times 1000, v = 4, u = 0.5, d = 1.5, r = 0.7$$

and

$$s_j = 1000, 2000, 5000, 10000, \text{ and } v_j = 5, 10, 7, 4$$

for $j = 1, \ldots, 4$ worked well.

Table 1 displays a comparison of the findings of the greedy algorithm, the brute-force algorithm, and the genetic algorithm for $n = 3$ and small values of m and k. The values found by the greedy algorithm are in black. The two values found by the brute-force algorithm that differed from the greedy algorithm are in red and the algorithm took most of a semester to do these computations. The genetic algorithm found the corrected values given in blue in much less time.

All three algorithms identify the individual with the optimal solution. However, the brute force algorithm and the genetic algorithm identify all such individuals. Thus, back to the original lattice problem, we can also display at least one set of paths justifying each entry in Table 1.

The greedy algorithm executes in $O(n^2)$ time[3].In contrast the brute force algorithm is very computationally complex. Generating all combinations of a set of a given size requires $O(n^k)$ time according to Canonne [4].

The time complexity of the genetic algorithm is $O(s \times m \times n \times k)$ [8], or approximately $O(n^4)$, and Figure 3 illustrates the speed and accuracy with which the genetic algorithm converges on the optimal solutions.

$_m\backslash^k$	1	2	3	4	5	6	7	8	9	10
3	2	4	8	10	20					
4	2	4	67	13	19	35				
5	2	4	6	910	20	28	56			
6	2	4	5 6	8 9	13 15	30	44	84		
7	2	4	5	7 8	11 12	18 21	42	60	120	
8	2	4	5	6 7	10	16	25 27	57	85	165
9	2	4	4	5 6	8	12 13	20 21	33 36	76	110
10	2	4	4	4 5	6 7	10 11	16	24 28	41 46	98
11	2	4	4	4 5	6	8 9	13	18 21	31 34	50 56

Table 1: $P(m, n, k)$ for $n = 3$ and small values of m and k Solutions generated by the greedy algorithm are in black. The brute force corrections are in red and the genetic algorithm corrections are in blue.

In Figure 3, we see that an optimal solution is found after approximately 40 generations, and that the entire population also achieves a high average fitness rating. Further, it does this with consistency - there is correspondingly low divergence among the individuals in the population.

Figure 4 and Figure 5 are sample figures showing the relation between runtime, m, and k. We can see that the runtime increases linearly with respect to m and k, and that the runtime is a matter of minutes.

Hence, in comparison to the brute force algorithm, the genetic algorithm, though not deterministic, is much quicker and just as accurate. When compared to the greedy algorithm, the genetic algorithm is nearly as fast, but far more accurate.

8 Extensions

We have not proven anything in this paper, but rather have demonstrated that using a genetic algorithm can offer a pathway to efficiently solving complex problems.

For example, early work on this lattice problem determined that it was equivalent to finding the independence number of graphs defined on the paths of lattice. Several papers by Gillman and his students [1], [6] explored the properties of these graphs. While it is clear that not every graph corresponds to a lattice graph, the insights gained from the approach used in this paper, that is, using genetic algorithms to find solutions, may contribute to finding efficient solutions to the independence number problem which is known to be NP-Complete. [11]

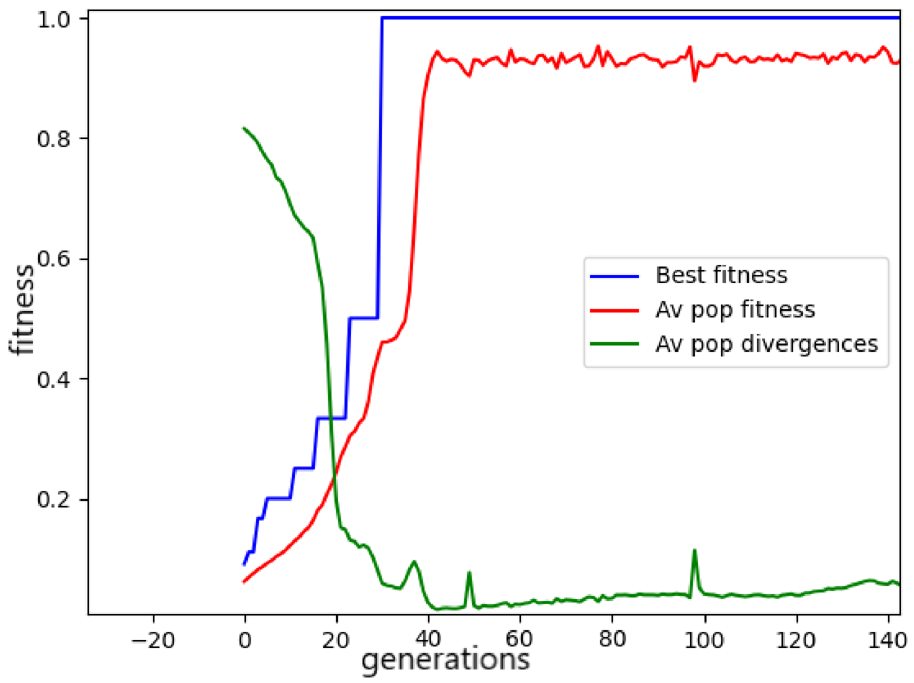

Figure 3: fitness vs generation

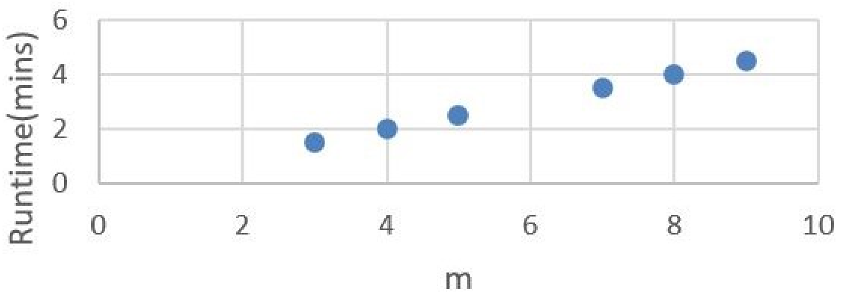

Figure 4: Runtime(mins) vs m [n=3, k=4]

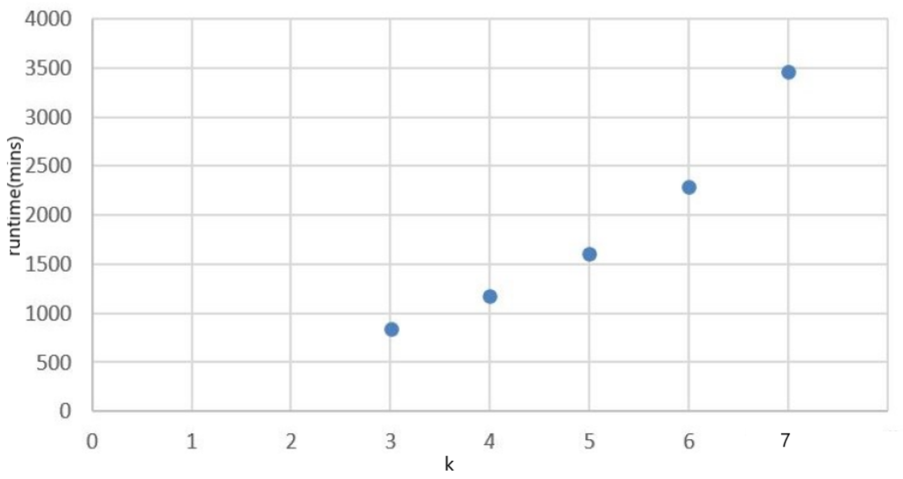

Figure 5: Runtime(mins) vs k [m=11, n=3]

References

[1] Brewer, M., et al. "Graphs of Essentially Equivalent Lattice Paths." *Geombinatorics*, vol. 13, no. 1, 2003, pp. 5-9.

[2] Bona, M. (editor) Handbook of Enumerative Combinatorics CRC Press, Boca Raton, 2015.

[3] Brown, R. Advanced Mathematics: Precalculus with Discrete Mathematics and Data Analysis. Houghton-Mifflin Co., Boston, 1994.

[4] Canonne, C. "Approximation of combination $\binom{n}{k} = \Theta\left(n^k\right)$?" *Stack Exchange*, May 3, 2015, math.stackexchange.com/questions/1265519/approximation-of-combination-n-choose-k-theta-left-nk-right/4134185

[5] Gillman, R. "Enumerating and Constructing Essentially Equivalent Lattice Paths." *Geombinatorics*, vol. 11, no. 2, 2001, pp. 37-42.

[6] Gillman, R., et al. "On the Edge Set of Graphs and Lattice Paths." *International Journal of Mathematics and Mathematical Sciences*, vol. 61, no. 1, 2004, pp. 3291-3299.

[7] Ateufack Zeudom, F. Lattice Path Research Code [Computer software]. github.com/arielfayol37/Lattice_paths.

[8] Goldeberg, D. Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[9] Gillman, R.,et al “On the Edge Set of Graphs and Lattice Paths” International Journal of Mathematics and Mathematical Sciences

[10] Yager, E. and Engstrom, M. (2023) ”k-Distinct Lattice Paths,” Rose-Hulman Undergraduate Mathematics Journal: Vol. 24: Iss. 2, Article 6. Available at: https://scholar.rose-hulman.edu/rhumj/vol24/iss2/6

[11] Wilf, H. Algorithms and Complexity, 2nd Edition, AK Peters, Massachusetts, 2002.