[[1]](#footnote-1)

Genetic algorithm and wisdom of artificial crowds to solve the longest path problem(November 2023)

Corey A. St Jean and Roman V. Yampolskiy

*Abstract*—The paper describes use of a genetic algorithm along with a novel wisdom of crowds’ algorithm for solving the decision version of the longest path problem on undirected cyclic graphs. Seven different graphs were tested with four, six, eight, ten, twelve, fourteen, and sixteen vertices. For these seven graphs, four different amounts of experts were taken. These tests resulted in a data set explaining the effects of vertex amount and expert amount on time taken and results obtained. The approach can solve the problem accurately and the time taken is dependent on the number of vertices in the graph as well as the number of experts (runs) calculated for use in the WOC algorithm. From the data tested there was a 100% success rate in finding a path of set length in moderate time meaning that the genetic algorithm along with wisdom of artificial crowds can accurately return a viable solution to the longest path problem.

*Index Terms*— artificial intelligence, AI, genetic algorithms, GA, longest path problem, LPP, graphs, NP-complete, optimization, wisdom of artificial crowds, WoAC.

# INTRODUCTION

## Genetic Algorithms

The Genetic Algorithm (GA) is a term used to describe the family of algorithms based on evolutionary concepts[13]. The algorithm starts with a random set of individuals, picks individuals to parent a new generation, and performs crossovers of the parents to create a new generation. A mutation effect is also present in which a new individual within the new generation undergoes some kind of random event on their genome to increase variation. The parents are selected based on a fitness score calculated from a fitness equation and are typically the best individuals of a generation[9]. This pattern of picking parents, crossing over parents, mutating new individuals will create a new generation and is repeated until some stopping factor is reached. After the stopping point is reached the algorithm should return a fairly viable solution depending on the problem it is applied to.

## Wisdom of Artificial Crowds

Wisdom of crowds (WoC) is an idea stemming from psychology and statistics. The idea is that a crowd made up of random individuals could produce a correct solution via combination of expert solutions. This process typically performs better than picking a random individual and using their solution[18].

Wisdom of artificial crowds (WoAC) is a way of implementing the WoC idea into a computer program. The crowds being utilized are made up of individuals found via a different process. WoAC is described by Yampolskiy as “a post-processing algorithm in which independently-deciding artificial agents aggregate their individual solutions to arrive at an answer which is superior to all solutions present in the population” [20]. This requires the creation of the artificial agents from some other system or process, in the case of this study, the GA will be utilized to create artificial agents that will be used in WoAC to create a better solution on average than the individuals making up the population.

## Nondeterministic Polynomial

Nondeterministic polynomial (NP) is a class of problems in which a solution can be solved in polynomial time as long as the machine utilized is nondeterministic[12]. This class of problems is directly related to the class of NP-Hard and NP-complete problems covered later.

## NP-Hard

NP-Hard is a different class of problems that the NP class. NP-Hard problems are ones that were reduced from a problem in the NP class in polynomial time[12]. Another way to explain this is if problem A can be reduced from problem B in the NP class in polynomial time, problem A is an NP-Hard problem.

## NP-Complete

The NP-Complete class of problem are problems that are both NP and NP-Hard in nature. These problems are decision problems meaning that a yes or no is output based on results gained and input[12]. Since NP-Complete problems are both NP and NP-Hard, meaning that they can be solved in polynomial time if the problem was not deterministic as well as reduced from a problem in the NP-Class. NP-Complete problems are typically deterministic meaning they are derived from an NP problem in polynomial time.

## Genetic Algorithm for NP-Complete

Genetic algorithms are a great tool for estimating results for NP-complete problems. The GA does not typically find the best solution to a problem, instead it finds a correct solution that the best solution can be inferred from[1]. The reason the algorithm is effective on NP-Complete problems is that they take advantage of assumed domains of the problem as well as forcing a problem to be implemented in its own terms[7]. This makes the GA very effective for solving NP-complete problems.

## Wisdom of Crowds for NP-Complete

WoAC is a great method for solving NP-complete problems when combined with the GA[2]. This is because it takes the solutions created by the GA, picks the best results created artificially, and aggregates them to form a better solution. WoAC does not always need to be combined with the GA, but it needs some sort of agent creating a “crowd” of results to pick experts from and combine/average[21].

## Longest Path Problem

The longest path problem (LPP) is a problem that can take on many forms with differing levels of difficulty. The main goal of the LPP is to find the longest simple path through a graph. This problem in itself is typically described as NP-Hard[6]. But the decision version of this problem is NP-complete. This version of the problem asks the question “Does a path of length X exist in the graph?” in which a yes or no (true or false) answer is required. The decision version of the LPP is NP-complete in most cases. The reason this version is not always NP-complete is because it depends on the type of graph being tested[17]. An acyclic digraph can be solved via a small manipulation of Dijkstra’s algorithm. Acyclic implies the graph does not have any cycles or loops, and digraph implies that edges are directional. This version of the problem would not be NP-complete because it can be solved in less than polynomial time. The specific version of the problem being tested in this case is the decision version of the LPP on cyclic undirected graphs. These graphs have edges that can be traversed in either direction as well as cycles or loops present. This version of the problem is NP-complete because no known algorithms to solve in polynomial time as well as it being the decision version[3][11]. This problem has less practical applications than the Travelling Salesman Problem (TSP) because in most cases the longest path is the worst possible outcome. TSP is used for routing directions, planning highways, planning factories, and much more. The LPP has practical applications mostly in networks because the longest path is the critical path or in just finding worse case scenarios[16].

## Longest Path Problem is NP Complete

The proof of the NP-completeness is reduced from the Hamiltonian path problem. This is because both problems are very similar in nature. The Hamiltonian path problem states that if a path of length N-1 (N = # Vertices) exists in a graph, then a Hamiltonian path exists in the graph. The LPP states that if a path of length N exists, and K <= N, then a path of length K exists[4][14]. Like mentioned previously, this problem by itself is just NP-hard because it is derived from an NP problem. This problem becomes NP-complete when it becomes the decision version because at that point it is both NP and NP-hard meaning that it is derived from an NP problem while also being an NP problem itself.

## Hamiltonian Path is NP-Complete

To prove the LPP is NP-complete, the Hamiltonian path problem must first be proven NP-complete and then the LPP is reduced from that proof. To prove that the Hamilonian path is NP-complete, it must be proven to be NP as well as NP-hard. To be NP, a solution to the problem must be able to be proven in polynomial time. This is possible in a variety of ways and has been proven to be truevia creation of an algorithm that first guesses a path, then checks that all edges of said path are viable edges, then checks that vertices are only visited once at most. If all is true the results are accepted, which can be accomplished in polynomial time[8]. Then for the problem to be NP-hard, it must be reducible from a problem that is NP itself. The problem that can be reduced to Hamiltonian path is the 3SAT making the Hamiltonian path problem NP-hard[19]. With both NP and NP-Hard being proven, this in turn proves that the Hamiltonian path problem is NP-complete which then goes to prove than the LPP is itself NP-complete because it is reduced from another NP-complete problem.

# Review of Literature

In the paper “Long Path Problems,” Horn et al. studies how well the GA performs in the longest path problem compared to various hillclimber algorithms. The point of the study is to see if the GA performs better than a hillclimber algorithm as the number of vertices in a graph increases. The study examines GA with crossover only as well as the GA with crossover as well as mutations. The specific hillclimber algorithms studied are the steepest ascent hillclimber, the next ascent hillclimber, fixed rate mutation algorithm, then mutation with steepest ascent and next ascent. The study found that the GA with crossover only can find better solutions to the problem in less time on the larger graphs. This shows that the GA is effective and efficient in solving the longest path problem, as well as the difficulty in solving the longest path problem. On simpler problems though, the hillclimber is more effective, suggesting that the best general solution to the problem could be one that performs hill climbing as well as evolution.[10]

In the paper “A Study of Genetic Algorithms for Approximating the Longest Path in Generic Graphs,” Portugal explores four different types of GA implementations to see how well each performs on the LPP. The study specifically examines how the GA can be used to efficiently find high quality solutions to difficult problems. The four types of GAs implemented include GA using non-intersecting paths, GA using intersecting paths, GA using both pairs of paths, and GA with a mutation mechanism. GA using non-intersecting paths only crosses over simple paths that do not share any vertices, allowing paths to be crossed over without having to deal with duplicate vertices. This version of the GA is effective in finding long paths and involves much simpler computations because paths used in crossover are much simpler and are distinct. GA using intersecting paths only uses parents in which common vertices are shared. This implementation starts off very efficient because not many crossovers occur due to not many paths intersecting. But as longer paths are found, more intersections occur, requiring more crossovers. Next, GA using both pairs of paths combines both previously discussed approaches. This means that this approach can crossover paths regardless of vertices used, but the crossover method differs based on the path. The final approach is GA with mutation. This involves performing mutations on individuals in the current population to create the next generation without use of crossover. The results obtained from the study show that approach best at finding solutions was the GA using non-intersecting paths. It had the highest success rate, but also took a lot of time to compute. The worst performance was the GA using intersecting paths. It took very long and often did not compute the correct results. But accounting for time required and quality of solutions, GA with mutation was most effective. GA with mutation had successful results, while also taking less time than most of the other algorithms. GA with mutation was less successful than GA using non-intersecting paths but took much less time. This means that estimations of the longest path can be effectively made with GA with mutation in comparatively short time. But, if finding the best solution is required, then GA using non-intersecting paths is the best solution but will require more time[15].

# Proposed Approach

An implementation of Wisdom of Crowds using experts created by the GA was created in Python 3.8. The algorithm also included dynamic visual representation of the graphs and paths calculated via the tkinter and matplotlib libraries. Other libraries utilized are the time, threading, math, and random libraries used for fitness calculations, randomization of initial populations, timing of runs for the algorithm, and for multiple threads to be used for GUI and the algorithm.

All testing was completed on the same system. The system has an Intel Core i5 9600K as the central processing unit and 32 gigabytes DDR4 ram as memory. The system is running Windows 11 Home 64-bit as the operating system.

The GA initially creates a completely random population of individuals, all making up simple paths through the graph. The fitness values of all individuals of the initial population are then calculated via the distance formula:

for q and p in V.

Where q and p represent coordinates for adjacent vertices in the list of vertices V.

Higher fitness values indicate better individuals in the population. Next, four parents are selected from the population to be used in crossover to create the next generation. Two parents are chosen at random from the list of four parents created. These parents are crossed over at a shared vertex to create a new individual. By the end of crossover, 496 individuals are created. Next a mutation function mutates individuals randomly at a rate input by the user. All the runs tested had a mutation rate of 75% meaning there is a 75% chance an individual will mutate. Mutation could occur in two different ways via two different functions. First an insert mutation can occur in which a vertex that is able to be inserted into the path along a viable edge is inserted to the path. This insertion ensures that the path remains simple and only travels along viable edges in the graph. The other mutation is a swap and reverse mutation. This mutation finds two vertices in the path that form a viable edge. Then the path is broken in two at the point of one of the vertices. The latter sub path is reversed, and the other sub path is added to the end. This changes the order of vertices visited in a path potentially increasing fitness.

After crossover and mutation, the four parents selected are readded to the list of new individuals so that the best individuals of each generation will never get worse. This makes it so the best fitness levels are either the same as the previous generation or increase. This process continues for all 99 remaining generations, after which four experts are chosen, being the four best individuals of the final population, for use in the WoAC function.

The WoAC function attempts to build a path of a set length indicated by the user. This function takes the experts obtained from the GA, and finds common edges present in more than one expert. Then, from these common edges, a new path is created of the length indicated. If a path can successfully be created out of the common edges, then true is output as well as the path created. On the other hand, if no path can be created, then false is output.

A diagram of a process

Description automatically generatedAll results are output in the GUI in either a pop-up window or the graph window. Graph edges are graphed in blue while the path calculated is graphed in red.

Figure 3.1: Flow chart of GA and WoAC approach

# Experimental Results

## Data

The variables being testing in this study were how the number of experts and the size of a graph (# vertices) affects the results. There are four different amounts of experts being tested, this includes five runs (20 experts), ten runs (40 experts), fifteen runs (60 experts), and twenty runs (80 experts). There are seven different graphs being tested of lengths four, six, eight, ten, twelve, fourteen, and sixteen. The purpose of this data is to see if false results eventually begin being obtained as the number of potential paths in a graph increases. Also, to see if a small number of experts can sometimes lead to a false result on a graph where a true result should be obtained. Time is another factor being studied, as the size of the graph or the number of runs increases, the time taken should also increase, and how much of an increase is caused.

The datasets tested were all in the form of text files created from Concorde TSP Solver[5]. These text files contain the coordinates for vertices as well as the vertices that make up edges. The location of vertices and vertices used in edges are random, but that graphs also went through minor changes to clean up crossing edges and make the graphs more legible.

Table 4.1: Genetic Algorithm Results (No WoAC)

## Results

The GA along with WoAC was able to successfully create/find paths of set length with 100% accuracy. This means that for all graphs, all amounts of experts, and all length of paths searched in the testing returned true as well as an example path for proof. Results for GA and GA along with WoAC were both saved from comparison. The time taken for GA along with WoAC was saved as well to see how expert amount, graph size, and length of path being searched for affects time taken. Table 4.1 shows the results obtained from running GA by itself with no WoAC. The table shows the average length of experts obtained from the algorithm. Also, the minimum length of the experts and the maximum length of the experts are also noted. The GA simply just attempts to find the longest path, regardless of length input by the user.

Table 4.2: Genetic Algorithm along with Wisdom of Artificial Crowds results

The results from running the GA along with WoAC are within table 4.2. The “Path Length” column represents the length of the path being searched for. The “# Vertices” column represents the number of vertices present in a graph. So, each graph has three different lengths of path being searched for. Also, for each graph, four different numbers of runs/experts are tested. So, in total, each graph is tested twelve times, with four differing amounts of experts being tested for each length being searched for. Then, the results obtained are time in seconds, fitness of the final path (length of the final path), and whether a path was found. The time taken for each of the run categories was averaged for easier comparison.

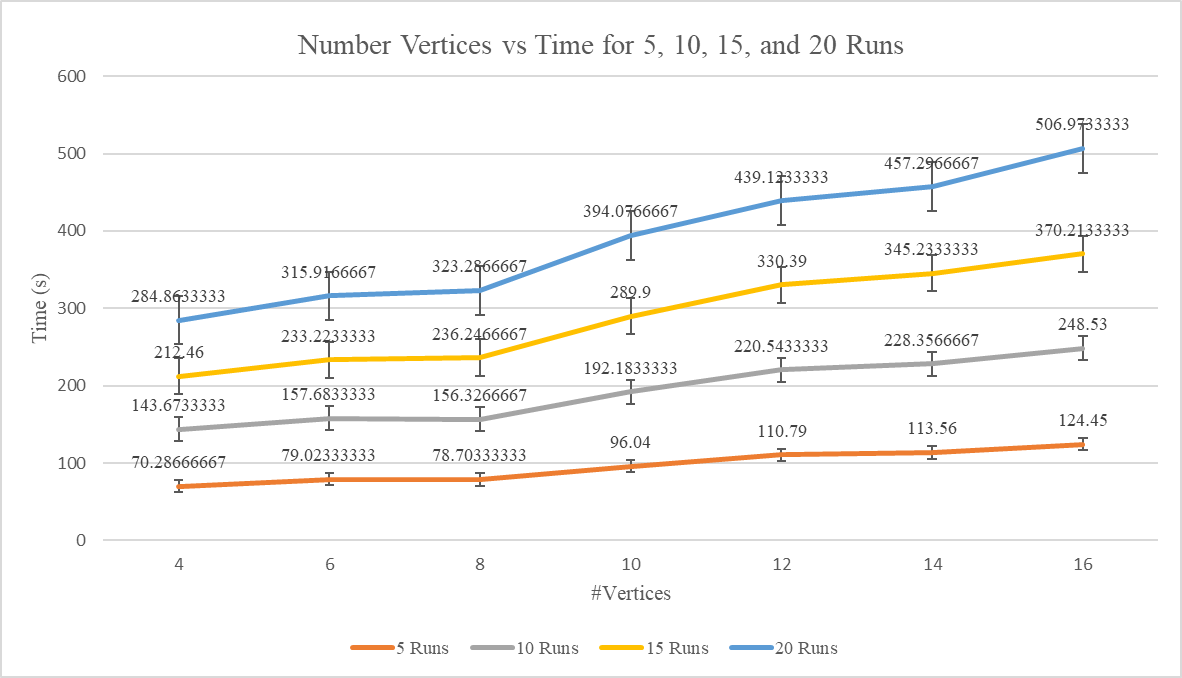


Figure 4.3: Line Chart showing how time is affected by the number of vertices and number of runs. Data from Table 4.3

Figure 4.2: Chart showing Genetic Algorithm results from Table 4.1

## Then, when it comes to charts, Figure 4.2 is a bar chart showing comparisons between the maximum, minimum, and average lengths of GA results obtained without any WoAC. The lengths are on the Y-axis and the number of vertices in the graph are in the X-axis.

Figure 4.3 expresses the relationship between time, number of runs tested, and number of vertices on the graph being tested. The time is graphed in seconds in the Y-axis, and the number of vertices is graphed in the X-axis. Then there are four different lines depicting the four different amounts of runs tested. The bottom orange line depicts the trend for five runs, grey depicts the trend for ten runs, yellow for fifteen runs, and blue for twenty runs.

# Conclusions

## Discussion

From the results obtained, it can be confirmed that the GA with WoAC can be successfully implemented to solve the decision version of the longest path problem. There was never a failure or false result returned during the testing process. This also shows that at most twenty experts are required to gain a meaningful result because regardless of expert amount the implementation was successful. Also, it should be noted that as expert amount increases, the time taken increases as seen in Table 4.3 and Figure 4.3. This means that a successful result could be obtained in less time if the testing was expanded to include lower expert amounts. But the number of

vertices also affect the ease of obtaining results. This means that eventually, given a large enough graph, twenty experts will not be enough to gain accurate results.

From the results obtained from testing GA alone (Table 4.1, Figure 4.2), it can be observed that as the number of vertices in a graph increases, so does the variation in the results obtained from the GA. This once again shows that as the number of vertices increases, results from only twenty experts becomes harder to obtain. But it can also be noted that this is not always the case because there is another factor affecting the results obtained. From Figure 4.2 and Table 4.1, it can be observed that the graph with six vertices had more variation than the graph with eight vertices. This seems to counter the idea of only vertices affecting variation of GA results. This is because the number of edges in the graph and the vertices edges are attached to also has an effect. There could be more edges in a graph with four vertices than in a graph with eight vertices depending on the graphs tested. Also, there could be a vertex connected to most edges in a graph, making finding a long path more difficult.

Time seemed to be affected by both the number of vertices in a graph as well as the number of experts obtained. The number of experts obtained increasing the length of time required makes sense, because more runs of the GA are required for more experts taking more time. In Table 4.2, the time taken essentially doubles for the same graph as the number of experts doubles. Then also, like mentioned before, in most cases the number of vertices in a graph affects the time taken with more vertices taking more time to compute. But once again the time taken between the graph with six vertices and the graph with eight vertices is essentially the same, reflecting the idea that edges and graph complexity affect the time taken.

All in all, the implementation had a 100% success rate or a 0% failure rate. In all tests with all graphs and all amounts of experts a path of a certain length was found and true was output.

## Further Testing

To further research how different factors affect the results and accuracy of the GA and WoAC on the decision version of the longest path problems, more factors need to be analyzed. This includes some ideas mentioned already like edges and complexity of the graph. Testing could also be expanded to see what the true lower limit for the number of experts required is. This could mean testing with sixteen, twelve, eight, and four experts. This would also show where the cutoff is for expert amount for certain graph sizes. For example, it can by hypothesized that as the size of the graph increases, more experts are required and vice versa. Keeping track of the edges present in the graph could also be useful, or maybe even using a constant number of vertices and an increasing number of edges as the data sets to be tested. This would isolate the edges and see if edges are truly to blame for inconsistent variation in the results obtained in this study. Another experiment to further test implementation could be to have a constant number of vertices and edges for a graph but changing the location of the edges. This would isolate how graph complexity (apart from edge and vertex amounts) affects the accuracy of the results.

## Conclusion

This project aims to analyze an approach to solving the decision version of the LPP via the GA and WoAC. The GA alone was successful at finding the longest paths within the graphs in most cases via an evolutionary approach. The approach included a crossover to create generations from the best individuals, and a mutation to increase genetic diversity. WoAC when used on experts obtained from the GA, was successful in creating paths from common edges in experts. Both the GA and WoAC, when used together, are a possible way of gaining good results for NP-complete problems and sometimes even find the best result, but this depends on several factors including but not limited to number of vertices, number of experts used in WoAC, and the length of the path to be found. The approach can solve the problem accurately and the time taken is dependent on the number of vertices in the graph as well as the number of experts (runs) calculated for use in the WOC algorithm. From the data tested there was a 100% success rate in finding a path of set length in moderate time meaning that the genetic algorithm along with wisdom of artificial crowds can accurately return a viable solution to the longest path problem.

References

1. Arabi, B. H. (2016). Solving NP-Complete Problems using Genetic Algorithms. 2016 UKSim-AMSS 18th International Conference on Computer Modelling and Simulation (UKSim).
2. Beck, M. B., Desoky, A. H., Rouchka, E., McClure, P., & Yampolskiy, R. V. (2016). Using Genetic Algorithm and Wisdom of Artificial Crowds to Find Hidden Data in DNA. Embodying Intelligence in Multimedia Data Hiding.
3. Björklund, A. (2003). Approximating Longest Path [Licentiate Thesis].
4. Chauhan, Y. (2023, 2 November). Optimized Longest Path is NP Complete. GeeksForGeeks. <https://www.geeksforgeeks.org/optimized-longest-path-is-np-complete/>
5. Cook, W. (2003). Concorde-03-12.19. University of Waterloo. Retrieved November 10 from https://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm
6. Dahan, G., Tabib, I., Shimony, E. S., & Felner, A. (2022). Generalized Longest Path Problems. Proceedings of the International Symposium on Combinatorial Search.
7. De Jong, K. A., & Spears, W. M. (1989). Using Genetic Algorithms to Solve NP-Complete Problems. ICGA.
8. Gans, G. (2021, 19 May). Hamiltonian Path is NP-Complete. Medium. <https://ggans.medium.com/hamiltonian-path-is-np-complete-e6c5941eb4de>
9. Haldurai, L., Madhubala, T., & Rajalakshmi, R. (2016). A Study on Genetic Algorithm and its Applications. Int. J. Comput. Sci. Eng, 4(10), 139-143.
10. Horn, J., Goldberg, D. E., & Deb, K. (1994). Long Path Problems. Parallel Problem Solving from Nature—PPSN III: International Conference on Evolutionary Computation The Third Conference on Parallel Problem Solving from Nature Jerusalem, Israel, October 9–14, 1994 Proceedings 3.
11. Karger, D., Motwani, R., & Ramkumar, G. (1993). On approximating the longest path in a graph: Preliminary version. Algorithms and Data Structures: Third Workshop, WADS'93 Montréal, Canada, August 11–13, 1993 Proceedings 3,
12. Maji, N. (2015). An Interactive Tutorial for NP-Completeness [Virginia Tech].
13. Mathew, T. V. (2012). Genetic Algorithm. Report submitted at IIT Bombay, 53.
14. Nguyen, T. D. (2018). A Special Case of Longest Path Problem.
15. Portugal, D. (2010). A Study of Genetic Algorithms for Approximating the Longest Path in Generic Graphs. 2010 IEEE International Conference on Systems, Man and Cybernetics,
16. Schmidt, K., & Schmidt, E. G. (2010). A Longest-Path Problem for Evaluating the Worst-Case Packet Delay of Switched Ethernet. International Symposium on Industrial Embedded System (SIES),
17. Scholvin, J. K. (1999). Approximating the Longest Path Problem with Heuristics: A Survey [University of Illinois at Chicago].
18. Surowiecki, J. (2005). The Wisdom of Crowds. Anchor Books.
19. Williamson, M. (2009). NP-Completeness Proofs [PowerPoint Slides]. West Virginia University. <https://community.wvu.edu/~krsubramani/courses/sp09/cc/lecnotes/npcproofs.pdf>
20. Yampolskiy, R. V., Ashby, L., & Hassan, L. (2012). Wisdom of artificial crowds—a metaheuristic algorithm for optimization. Journal of Intelligent Learning Systems and Applications, 4(02), 98-107.
21. Yampolskiy, R. V., & El-Barkouky, A. (2011). Wisdom of Artificial Crowds Algorithm for Solving NP-Hard Problems. International Journal of Bio-inspired computation, 3(6), 358-369.

1. Manuscript received November 9, 2023.

   C.A. St Jean, R.V. Yampolskiy are with the University of Louisville, Louisville, KY 40292 USA (e-mail: c0stje01@louisville.edu). [↑](#footnote-ref-1)