CSE 568 Biocomputing Portfolio Highlights

Rachel Lanier Arozarena
Arizona State University
Tempe, Arizona, USA
rlaniera@asu.edu

ABSTRACT

This portfolio highlights three individual projects completed as part of the CSE 568 Biocomputing course: Introduction to Graph Coloring, Extensions to Balanced Colorings and Neutral Landscapes, and Modeling Collective Behavior in Ants through Agent-Based Modeling. These projects demonstrate the application of biologically inspired computing techniques—such as Genetic Algorithms and agent-based models—to solve complex optimization and simulation problems. Each project was implemented independently and focuses on a different aspect of biocomputing: combinatorial search, evolutionary neutrality, and collective behavior.

CCS CONCEPTS

• **Computing methodologies** → **Agent / discrete models**; • **Theory of computation** → **Evolutionary algorithms**; • **Applied computing** → **Biological networks**.

KEYWORDS

Biocomputing, Graph Coloring, Genetic Algorithms, Agent-Based Modeling

1 INTRODUCTION

This report presents a summary of three individual projects completed during the CSE 568 course on Biocomputing. The course included a total of six individual projects: Introduction to Graph Coloring, Extensions to Balanced Colorings and Neutral Landscapes, Introduction to Negative Selection, Anomaly Detection with Negative Selection [1], Modeling Collective Behavior in Ants, and Ant Colony Optimization for Network Route Repair. Together, these assignments made up 60% of the final course grade.

All projects were implemented independently, following the coursework and materials provided through CSE 568 and its associated Zybook. For this portfolio, I selected Projects 1, 2, and 5—Introduction to Graph Coloring, Extensions to Balanced Colorings and Neutral Landscapes, and Modeling Collective Behavior in Ants—which together account for 30% of the total coursework. These projects resonated with me the most and provide a strong representation of my individual contributions and understanding throughout the course.

2 PROJECT 1: INTRODUCTION TO GRAPH COLORING

Project overview: The graph coloring problem was a foundational challenge in computer science and combinatorial optimization. It involved assigning a color to each vertex in a graph such that no two adjacent vertices shared the same color. The k-coloring problem, where each vertex must receive one of k colors, was NP-hard when $k > 2$, making exact solutions infeasible for large graphs [2]. This motivated the use of heuristic and metaheuristic approaches—such as Genetic Algorithms (GAs) [3]—to search for approximate solutions efficiently.

In this project, I investigated how GAs could be used to solve the graph coloring problem. I utilized the DEAP (Distributed Evolutionary Algorithms in Python) package [3] to implement and run the evolutionary process. The goal was to minimize the number of conflicts—edges connecting vertices of the same color—in a coloring, eventually converging to a valid assignment where no such conflicts existed.

Goal: The project was organized around three main tasks:

- Implemented a parser to convert input graph files into internal data structures (Q1).
- Designed a fitness function that penalized color conflicts and rewarded legal colorings (Q2).
- Tuned genetic algorithm parameters to improve performance on large graphs (Q3).
- Evaluated the effectiveness of the GA on different graph types and sizes.

Solution: I independently implemented a pipeline: parsed input graphs to extract nodes and edges; designed a fitness function that rewarded legal colorings; implemented a genetic algorithm using DEAP with initialization, selection, crossover, mutation, and evaluation; and ran experiments on four graphs. Input graphs were parsed using `init_graph`, which constructed a `Graph` object with adjacency lists. Individuals were encoded as binary strings of length $n \times \lceil \log_2 k \rceil$, decoded into color assignments, and evaluated with `eval_graph`, which computed fitness as the proportion of valid edges. Invalid encodings returned zero fitness. Parameters were tuned to improve convergence: mutation probability and population size were increased, crossover and tournament selection adjusted, and generations extended to 100. Individuals were represented as integer lists, selection used tournament size 5, crossover used two-point recombination, and mutation flipped bits with probability 0.15. The updated implementation converged to fitness 1.0 on `graph_4.txt`.

Results: The graph parsing function correctly processed all tested files, producing the expected adjacency lists. For example, parsing `graph_1.txt` yielded the correct output `[[2], [2, 3], [1, 0], [1]]`, confirming accurate formatting and parsing [2].

The fitness function returned accurate scores between 0 and 1, correctly penalizing conflicts and confirming its reliability. For example, it returned 0.75 for `g3.txt` with input `[1, 0, 1, 0, 1, 1, 1, 0]`. All graph-coloring pairs tested gave consistent results, validating the function's correctness [3].

The genetic algorithm was applied to `graph_4.txt` and evaluated over 100 generations. Average fitness rose from 0.75 to over 0.92 by generation 20, with the best individual reaching a fitness of 1.0, demonstrating convergence to an optimal solution. Table 1 and Figure 1 summarize these results. Minimum fitness improved steadily, and evaluation counts remained stable, confirming that the algorithm evolved toward better solutions as generations progressed [3].

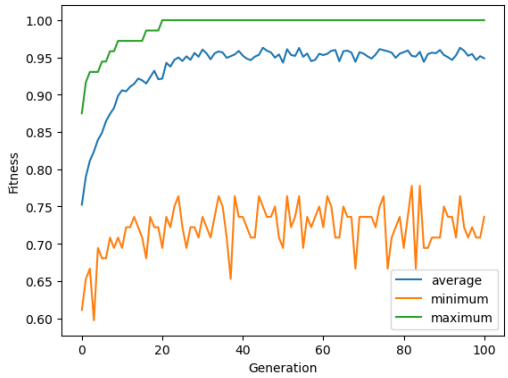

Figure 1: Graph of the fitness values across generations for the Genetic Algorithm applied to graph 4.

3 PROJECT 2: EXTENSIONS TO BALANCED COLORINGS AND NEUTRAL LANDSCAPES

Project overview: This project, Extensions to Balanced Colorings and Neutral Landscapes, built upon the foundation laid in a previous assignment on basic graph coloring using Genetic Algorithms. In the earlier project, the primary objective was to assign colors to vertices such that no two adjacent vertices shared the same color. This time, the challenge extended further by introducing the concept of balanced coloring, where not only must adjacent vertices be colored differently, but the usage of each color must also be evenly distributed across the graph. In addition, the concept of neutral landscapes was explored, where mutations that did not change the fitness of a solution were studied to better understand the structure of the search space.

Goal:

- Implemented a genetic algorithm (GA) to find balanced colorings for a given graph.
- Explored and analyzed the neutrality of graph colorings by calculating 1-step neutrality.
- Extended the functionality of the graph coloring problem by adding a balance constraint and studying the effect of this constraint on fitness.

Solution: I independently modified the fitness function to incorporate a balance factor, ensuring colorings met balanced coloring constraints. Using the DEAP Python package, individuals were encoded as binary strings, with each $\lceil \log_2 k \rceil$-bit segment representing a vertex's color. Invalid color codes ($\geq k$) were immediately assigned a fitness of 0.

The fitness function computed:

- *valid_fraction*: the ratio of valid (differently colored) edges to total edges.
- *balance_factor*: the product of normalized color frequencies, penalizing uneven distributions.

The final fitness was computed as:

$$\text{fitness} = \text{valid\_fraction} \times \text{balance\_factor}$$

and rounded to five decimal places.

To explore neutrality, I implemented a 1-step neutrality function that evaluated how many single-gene (color) mutations resulted in unchanged fitness. Each mutated individual was decoded and re-evaluated using the original `eval_graph` function. The neutrality score was the fraction of neutral mutations among all possible one-color mutations. All methods were tested on various graphs to validate correctness and behavior of the modified fitness and neutrality metrics.

Results: The `eval_balance` function was tested on `graph_2.txt` using a DEAP-based evolutionary algorithm. With a population size of 50, tournament selection, and mutation probability of 0.2, the maximum fitness achieved was **0.032**, with convergence observed around generation 9. Average, minimum, and maximum fitness values increased across generations and stabilized.

The `one_neutral` function passed all test cases on `graph_1.txt` through `graph_4.txt`. For `graph_1.txt`, the original fitness was 1.0 with neutrality 0.25. Test case fitness values were 0.3333, 1.0, 0.6667, 1.0, and 1.0. For other graphs, original fitness ranged from 0.714 to 1.0 and results remained within expected bounds. The function successfully computed neutrality across all graph structures.

4 PROJECT 5: MODELING COLLECTIVE BEHAVIOR IN ANTS

Project overview: This project focused on converting an Ordinary Differential Equation (ODE) model into an agent-based model (ABM) to simulate ant foraging behavior [4]. Using the Mesa package in Python [5], I implemented the Ant agent within an existing model [6], capturing behaviors such as exploration, commitment to food sources, recruitment, and tandem running. These behaviors followed probabilistic rules translated from the ODE model to offer a granular view of individual-level and collective dynamics.

Ant agents moved randomly when uncommitted, chose food sources A or B based on probabilities α_A and α_B, recruited via tandem running, and could uncommit with probabilities λ_A and λ_B. State transitions were based on interactions with food and other ants.

The model operated on a 50x50 grid and included Ant, Nest, and Food agents. Ants had states (uncommitted, committed to A, committed to B), and a step method governed movement and recruitment. Food agents represented strong and weak food sources. The Nest agent served as the starting point. The ABM extended the original ODE model by simulating detailed, interactive, and realistic ant behavior at the individual level.

Goal: The objectives of this project were:

- Learn how to convert an ODE model into an agent-based model.

Generation	Evaluations (nevals)	Average Fitness (avg)	Minimum Fitness (min)	Maximum Fitness (max)
0	200	0.752431	0.611111	0.875000
1	161	0.790139	0.652778	0.916667
2	163	0.811458	0.666667	0.930556
3	157	0.823819	0.597222	0.930556
4	159	0.839375	0.694444	0.930556
5	165	0.848889	0.680556	0.944444
6	141	0.864375	0.680556	0.944444
7	157	0.874167	0.708333	0.958333
8	166	0.882153	0.694444	0.958333
9	156	0.898611	0.708333	0.972222
10	153	0.905903	0.694444	0.972222
⋮	⋮	⋮	⋮	⋮
20	152	0.921528	0.694444	1.000000

Table 1: Fitness values across 100 generations for the Genetic Algorithm applied to graph 4.

- Simulate key ant behaviors and interactions.
- Analyze the dynamics of foraging, recruitment, and commitment.

Solution: I implemented ant agents using Mesa with behaviors based on ODE-derived probabilities. Each ant started at the nest in the `UNCOMMITTED` state with `tandem_running` set to 0. I controlled discovery, recruitment, and uncommitment with parameters α_A, α_B, β_A, β_B, λ_A, and λ_B, adjusting values by a factor of 60 to convert from minutes to seconds. I used $N = 100$ ants and the `SimultaneousActivation` scheduler.

The parameters used in the model are as follows:

- $\alpha_A = 0.0125 \times 60 = 0.75$: Probability per second that an uncommitted ant encountering food source A becomes committed.
- $\alpha_B = 0.0125 \times 60 = 0.75$: Probability per second that an uncommitted ant encountering food source B becomes committed.
- $\beta_A = 0.015 \times 60 = 0.9$: Probability per second that a committed ant successfully recruits another to food source A.
- $\beta_B = 0.006 \times 60 = 0.36$: Probability per second that a committed ant successfully recruits another to food source B.
- $\lambda_A = 0.009$: Probability per second that an ant uncommits from food source A.
- $\lambda_B = 0.038$: Probability per second that an ant uncommits from food source B.

The `step()` method controlled behavior based on state. In the `UNCOMMITTED` state, ants on food attempted to commit to A or B based on α_A or α_B; otherwise, they moved randomly. In the `COMMITTED` state, ants could uncommit with λ_A or λ_B, or attempt recruitment via tandem running. During tandem running, ants moved toward the feeder; upon arrival, they consumed food and remained committed.

I implemented movement using `get_neighborhood()` and grid move agent to choose and move to unoccupied cells. Tandem movement minimized distance to the feeder. I implemented recruitment by identifying uncommitted ants in the same cell and, with probability β_A or β_B, setting their `tandem_running` flag. At food sources, `any_food()` checked for availability, and `eaten()` reduced the supply. Committed ants continued recruiting while food remained.

Results: I ran the ant foraging simulation using ant model's `run_model()` and analyzed results from get model method [4], which tracked recruitment states and food depletion over time. Output messages confirmed state changes, with most ants recruited to `COMMITTED_A`, suggesting convergence on one dominant food source [6].

Visualization tools showed Feeder A's food level dropped exponentially after time step 800, while Feeder B declined more gradually. This clearly showed divergence, supporting the model's winner-takes-most dynamics. The initial delay before depletion suggested a modeled exploration phase.

In 10 sample runs, iteration counts ranged from 1169 to 1671, showing convergence times were consistent:

Run Number	Iterations to Complete
1	1250
2	1512
3	1408
4	1258
5	1197
6	1499
7	1169
8	1559
9	1344
10	1671

Table 2: Number of iterations to complete 10 sample runs

Despite occasional recruitment to Feeder B, Feeder A remained dominant in all runs. The stochastic recruitment process introduced variability, but convergence behavior was robust. The model did not simulate additional sources of randomness like environmental noise or switching between feeders. Visualizations omitted individual recruitment events, uncommitted counts, and switching behavior, limiting granularity.

Early model versions lacked feeder differentiation, which affected recruitment trends. After I fixed this, the system consistently favored Feeder A. The consistent convergence pattern demonstrated that positive feedback drove commitment behavior. Future improvements could add stochastic or environmental variation to better capture real-world ant behavior.

5 SKILLS, TECHNIQUES, AND KNOWLEDGE ACQUIRED

This project deepened my understanding of Genetic Algorithms (GAs) for solving complex problems like graph coloring, highlighting the critical role of parameter tuning to balance exploration and exploitation and avoid premature convergence. In particular, experimenting with population size, mutation rate, and generations taught me how these factors influence diversity and solution quality. Using the DEAP package [3] simplified implementation, but tuning for larger graphs showed the importance of extensive testing and experimentation.

Furthermore, modifying fitness functions to incorporate balanced coloring constraints revealed how subtle changes can drastically affect optimization. Learning to encode individuals effectively clarified how problem representation guides the search process. Through this, I also gained a clearer grasp of NP-complete problems and how GAs approximate solutions efficiently.

The project additionally strengthened my understanding of biocomputing concepts—selection, crossover, mutation, and fitness—as rooted in biological evolution. Implementing neutrality analysis taught me to compute and interpret neutral mutations, revealing landscape flatness or ruggedness crucial for understanding search spaces.

Moreover, attention to detail in encoding and decoding solutions was vital; debugging revealed how small errors could disrupt downstream processes. This reinforced the importance of precision in genetic algorithm implementation.

Working on the ant foraging ABM [4] further taught me to think critically about biological behaviors computationally. The connection between decentralized ant behaviors and applications like network repair became clear. I learned how movement mechanisms, such as random versus food-directed walks and Euclidean distance calculations, strongly impacted system efficiency and convergence.

During implementation, I encountered challenges managing state transitions to avoid unintended loops, which highlighted the need for careful probabilistic logic design. Small coding errors caused significant behavioral differences, emphasizing precision in agent-based modeling.

Reflecting on modeling choices, I questioned whether tandem running best represented real ants or if pheromone trails [6] might improve realism. This raised the broader issue that balancing biological accuracy and computational simplicity is a key consideration.

Interpreting stochastic yet deterministic system behavior deepened my appreciation of positive feedback and amplification effects. Initial assumptions about variability gave way to recognizing consistent patterns, refining my understanding of the model's dynamics.

In addition, visualization was crucial for interpreting results; the provided plot helped reveal key transitions and confirm data accuracy, displayed the importance of clear graphical representation in agent-based models.

Ultimately, I learned that effective modeling requires balancing simplicity and representational power—making thoughtful decisions about what details to include to capture essential system dynamics without unnecessary complexity.

6 CONCLUSION

These projects extended my understanding of Genetic Algorithms and agent-based modeling through practical applications in graph coloring and ant foraging simulations. By modifying fitness functions and integrating balance constraints, I explored how neutrality and balanced distributions impact solution quality and search space structure. Using the DEAP package, I implemented evolutionary algorithms that effectively found valid and balanced colorings, demonstrating the importance of careful parameter tuning to balance exploration and exploitation. Transitioning from ODE models to agent-based models with Mesa enabled me to capture individual ant behaviors and interactions, revealing how recruitment and commitment dynamics lead to emergent collective foraging patterns. Managing agent state transitions and probabilistic behaviors showcased the importance of precise logic design in dynamic environments.

ACKNOWLEDGMENTS

The project was completed as part of the coursework for CSE 568 under the guidance of faculty at Arizona State University.

REFERENCES

[1] J. Textor and M. P. H. Stumpf, "A comparative study of negative selection based anomaly detection in sequence data," *Pattern Recognition*, vol. 44, no. 1, pp. 21–30, 2011.
[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
[3] F.-A. Fortin, D. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "Deap: Distributed evolutionary algorithms in python," https://deap.readthedocs.io/en/master/, 2015, accessed: May 2025.
[4] Zybooks, "Modeling collective behavior in ants," https://learn.zybooks.com, 2025, available via course platform.
[5] D. Masad, J. Kazil *et al.*, "Mesa: Agent-based modeling in python," https://mesa.readthedocs.io/, 2025, accessed May 11, 2025.
[6] M. Goadrich, "Pheromone laying ants model," https://github.com/mgoadric/ants-mesa, 2015, gitHub repository, accessed May 11, 2025.