Coral Reef Optimization on Continuous Functions

By: Evelyn Kuo, Marina Sha, Semawit Gebrehiwot, Lydia Yang (SPACE)

Carnegie Mellon University

Date: May 6, 2022

Abstract

This paper presents an optimization algorithm, Coral Reef Optimization (CRO) and its performance on finding optima in continuous functions. Optimization and local search is crucial for the development of society; however, finding the best solution is often difficult and time consuming. Instead, we turn to nature to discover algorithms such as the CRO algorithm that find good approximate solutions to these problems. CRO simulates how corals populate a reef and compete with each other for space, allowing only the healthiest corals to survive. Thus, by using CRO to model how coral reefs grow, we can approximate an optimal solution for a given health function. In this paper, we implemented CRO and compared it to other benchmark algorithms that use local search such as the genetic algorithm (GA) and simulated annealing (SA). We found that although CRO has the potential to perform well against these benchmark algorithms (CRO performed well in one dimension, for example), the large amount of hyperparameters in CRO made it difficult to tune the parameters and thus difficult to determine the optimal performance of CRO.

1 Introduction

Throughout the last few decades, humanity has made numerous advances in science and technology that have allowed populations to flourish. However, due to rapid population growth, the issue of limited resources has become more pressing than ever. Thus, maximization and optimization of resources is a core problem in the development of modern society. From irrigation systems in agriculture to monetary distribution in economics, optimization has important applications in a wide range of systems.

In general, optimization seeks to maximize some value (or reward) over some cost. Although some optimization problems such as the global alignment problem can be solved efficiently, other optimization problems such as the traveling salesperson problem cannot be solved efficiently and are NP-hard [1]. Thus, in cases in which it is currently impossible or impractical to find exact solutions to optimization problems, it is important to find algorithms that give good approximate solutions.

We are constantly surrounded by optimization in nature. Over millions of years of evolution, nature has optimized itself to be as efficient as possible, and as a result, it contains many excellent approximation algorithms for optimization problems [2]. For example, ants use pheromones to find the shortest paths to their

food sources, and slime molds form networks that are both fault tolerant and have a small network size in order to maximize efficiency in distributing resources [3-4]. Thus, nature contains many useful approximation algorithms.

For this paper, we looked specifically at the coral reef optimization algorithm [5]. The coral reef optimization algorithm (CRO) describes how different corals reproduce and grow, competing with other corals for space in the reef. It simulates multiple aspects of coral growth, reproduction, and competition, such as broadcast spawning (sexual reproduction), brooding (self-fertilization), asexual reproduction, and larvae setting. In general, the Coral Reef Algorithm produced good results against other benchmark algorithms such as simulated annealing and genetic algorithms in one dimension, but for higher dimensions, the difficulty in parameter tuning made it challenging to determine how well CRO was performing (it was difficult to tell whether CRO was performing worse than the benchmark algorithms because of its parameters or because of the algorithm itself).

2 Background

2.1 Optimization

Optimization problems consist of maximizing or minimizing a real valued *objective function* where *inputs* or possible *solutions* to the function live in a set called the *solution space* [6]. Due to the mathematical property that minimizing the negative of a function is equivalent to maximizing the function, optimization problems are conventionally defined in terms of minimizations. The general statement of an optimization problem is as follows: given a set A and a function $f : A \to R$,

$$x^* = \text{argmin}_{x \in A} f(x)$$

where x^*, the optimal solution, has the property that for all $x \in A$, the optimal value $f(x^*)$ is such that $f(x^*) \leq f(x)$. The generality of the statement allows optimization to be useful for many fields, as it is often the case we seek answers as to what object is the best according to some evaluation criterion.

Some functions and some sets are relatively easy for optimization like continuous sets and convex functions whereas discrete sets often force optimization to conduct an exhaustive search of the entire set to find the optimal solution which can be computationally consuming if the set is very large [7]. As a result, we turn to approximation algorithms to optimize problems which seek to find solutions which may not be optimal, but have objective values which are close to the optimal value in a computationally efficient manner.

2.2 Local Search

A specific subset of approximation algorithms are those based on processes in nature or in the human world. These algorithms rely on the assumption that solutions which are "near" each other have objective

values (or "scores") which are similar, so they rely on locally searching around solutions which already have "good" (small) objective values to find more solutions which also yield "good" objective values which are hopefully better [8]. In general, local search algorithms start with some randomly generated solutions, then search for "neighbors" of solutions with better (lower) scores, typically stopping after some set number of iterations or reaching some solution which has a better score than all of its neighbors. In addition, many local search algorithms allow for some chance of moving to a worse solution to avoid falling into local minima. Thus, although local search algorithms may start with a set of poorly scoring solutions due to random selection, they will generally move towards better solutions as the number of iterations increases and thus eventually reach solutions with "good" scores. Some notable examples of local search algorithms are the genetic algorithm, simulated annealing, and the Coral Reef Optimization Algorithm, which are discussed below.

2.3 Benchmark Algorithms

In order to measure the effectiveness of the Coral Reef Optimization Algorithm, we tested it against two benchmark algorithms that use a similar process of local search to find their solutions: the genetic algorithm (GA) and the simulated annealing algorithm (SA).

The first benchmark algorithm, the simple genetic algorithm (GA), draws inspiration from natural selection and evolution of populations [9]. Due to natural selection and genetics, organisms with better fitness, whether it be having a stronger body or a specific color, tend to survive longer and produce offspring which also have good fitness. In addition, offspring are not carbon copies of their parents nor are they limited to the traits their parents have, as there is a random chance during DNA replication of a mutation which yields a new trait. The new trait may allow the offspring to be more fit than its parents, but also may make the offspring less fit. The class of genetic algorithms relies on the ideas of natural selection, genetics, and evolutions of populations by identifying individual organisms' genetic code as solutions from the given solution space with biological fitnesses associated with the objective values of the solutions (we implemented smaller objective values as better biological fitness); having solutions "mate" via a crossover operation which takes in two "parent" solutions and produces two "children" solutions in the solution space which are a mix of the two "parents"; having a mutation operation which mimics random mutation which takes in one solution and produces a mutated solution which is close to the original solution in the solution space; and simulating evolution by modeling the population of solutions over several time steps where at each time step, solutions with poor fitness have a high probability of being eliminated like what happens in the natural world due to natural selection. In theory, the best and average objective value of the population of solutions should decrease over time (which means the fitness increases over time) and the output of the algorithm is the solution within the population at the final time step with the best objective value. The vanilla genetic algorithm was analyzed due to its similarity to the coral reef optimization algorithm which is an example of a genetic algorithm as it

maintains an evolving population of corals which represent solutions which sexually reproduce, mutate, die off if the coral is not fit.

The second benchmark algorithm, simulated annealing (SA), draws inspiration from metal annealing [10-11]. Metals consist of metal grains which prior to treatment may be oriented in such a way as to cause undesirable properties like brittleness, hardness or inability to handle certain stresses [12]. By heating the metal enough to allow the metal grains to move and following a suitable cooling schedule, the metal grains are able to have enough time to settle and recrystallize in a formation which yields desirable properties. The cooling schedule is such that near the beginning, the metal is very hot which allows the metal grains to move a lot, but as the metal cools, the metal grains have less freedom of movement and can only settle in places near their current location. The algorithm starts off at a random initial solution and at a predefined initial temperature. At each time step, the temperature changes according to the cooling schedule and a random neighbor of the current solution is generated. Then, based on the current solution's objective value, the random neighbor's objective value, and the current temperature, the random neighbor may replace the current solution with some probability which is higher when the random neighbor's objective value is better than that of the current solution and also higher if the temperature is high. Note that the cooling schedule is a hyperparameter, and the variant of thermodynamic simulated annealing was used instead of vanilla simulated annealing which used laws of thermodynamics in order to replace the cooling schedule hyperparameter with a few constant hyperparameters instead. Simulated annealing was analyzed due to its similarity with the coral reef optimization algorithm between the cooling schedule and the competition for space in the reef as they both are such that at the beginning, the quality of solutions is allowed to be worse which allows for more exploration of the solution space, but as time goes on, the algorithm is more and more strict about the quality of the solutions.

2.4 Coral Reef Optimization (CRO)

2.4.1 CRO Overview

The Coral Reef Optimization Algorithm is a space optimization algorithm that represents how corals reproduce and compete with other corals for space in the reef, thus only allowing the corals with the highest "fitness" to survive (Sanz, et. al). The CRO algorithm encodes an optimization problem by identifying corals as solutions within the solution space; a reef by a N by M grid of such corals; the biological fitness/health function of the corals with the negative of the objective function (smaller values of the objective function are higher biological fitness) which determines competition between corals/solutions with how likely a coral can push out an existing coral of the reef which is how likely a new solution can replace an existing solution; and reef evolution by simulating several time steps of coral sexual and asexual reproduction and depredation. The algorithm simulates depredation of corals by removing corals with low fitness at each time step of the algorithm. The algorithm takes in as input N and M (the width and height of the grid), a set of "solutions"

(or corals), a health function (based on the objective function related to the optimization problem), and some additional parameters related to reproduction and depredation. It outputs a $N \times M$ grid (which represents the reef) occupied by the corals.

2.4.2 CRO Algorithm

A brief description of the Coral Reef Algorithm is as follows. Note that the algorithm is based on the algorithm as described in "The Coral Reefs Optimization Algorithm: A Novel Metaheuristic for Efficiently Solving Optimization Problems" by Sanz, et. al.

At the start of the algorithm, a $N \times M$ grid is initialized, and a fraction p_0 of the squares are randomly occupied by corals (the remaining squares are empty) which simulates an initial reef which may not be fully occupied by corals yet. Each of the corals has a corresponding health, which is determined by the health function.

After initialization, the algorithm simulates the reproduction of the corals, in which they produce both sexually and asexually and compete for space in the reef. Note that these steps are repeated until some stop condition is met (for example, after a set number of iterations or after a solution reaches a certain value).

First, the corals undergo broadcast spawning, in which a fraction F_b of the corals are selected to be broadcast spawners. Then, pairs (or couples) are randomly chosen from the broadcast spawners, and each pair will sexually reproduce to form a coral larvae (note that each coral will only be a parent once). Any corals not chosen to be broadcast spawners will reproduce via brooding, in which random mutations of the brooding coral are used to produce coral larvae (this represents internal sexual reproduction, in which the corals undergo self-fertilization). Afterwards, some fraction F_a of the corals will produce an identical copy of themselves, which represents asexual reproduction.

After the larvae are formed (either through broadcast spawning, brooding, or asexual reproduction), they will try to settle in the coral reef. Over k steps, each larva will randomly try to settle in some square in the reef. If the square is unoccupied, the larvae will occupy the square, otherwise it will only occupy the square if the square is occupied by a coral with lower health. Finally, at each step of the algorithm, a fraction F_d of the corals with the worst health will be removed from the reef with probability p_d, which represents depredation in the reef.

The coral reef optimization algorithm differs from existing algorithms by incorporating genetics and competition. Like genetic algorithms, the coral reef optimization algorithm exploits the existence of good solutions by producing similar child solutions from the good parent solutions via the crossover operation. The idea behind having the crossover operation is parents which are biologically fit, which in the context of optimization mean the objective values are small, are likely to produce offspring which are also biologically fit. Furthermore, like simulated annealing, the coral reef optimization has a component of variable competition to allow for ample exploration. Because the reef starts off relatively empty, biologically unfit corals,

which correspond to solutions with high objective values, are able to survive in the reef. The reef slowly gets more and more populated over time which will then increase the level of competition so that biologically unfit corals cannot survive anymore. The gradual increase in the level of competition between corals, which are solutions, allows for a period of exploration in the beginning to better explore the solution space and then eventually switches over to exploitation of solutions which are already known to be good. The key strength of the coral reef optimization algorithm is utilizing exploration and exploitation at suitable points of the algorithm and maintaining a large enough population of solutions at a given point of time so that the algorithm is able to explore multiple different areas of the solution space simultaneously.

3 Methods

3.1 Evaluation

We decided to evaluate the performance of the coral reef optimization algorithm (CRO) in comparison to a simple genetic algorithm (GA) and simulated annealing (SA), two similar optimization approximation algorithms. Due to the random nature of the initialization step where the initial population of solutions were generated randomly, we ran each of the algorithms multiple times and looked at the statistics of the output solutions of each of the algorithms. Because CRO and GA start with initial populations, the starting pool of initial solutions were shared. The number of iterations/evolution time steps between CRO and GA were kept constant. Furthermore, because CRO and GA maintain a population of solutions whereas SA only kept track of one solution at a time, in order to keep the number of objective function evaluations relatively similar, SA was run (number of iterations) $\times$ (population size of CRO or GA) number of time steps.

3.2 Rastrigin Functions

To test the performance of CRO, we applied our algorithms onto the problem of optimizing the Rastrigin functions [13-14], a collection of continuous benchmark functions defined for some positive real number x as:

$$f(x) = A \cdot n + \sum_{i=1}^{n} [x_i^2 - A \cdot \cos(2\pi x_i)]$$

where $A = 10$, n is the number of dimensions, and x is in the range $[-5.12, 5.12]^n$. The graph representation for the function (where $n = 1$) is depicted in figure 1.

We chose the Rastrigin function because it is more challenging to find optimal solutions for than other functions such as $f(x) = \sum_{i=1}^{n} x_i^2$. This is because as shown by the graph, the Rastrigin function has many local minima, so it is easier to fall into these local minima when applying local search algorithms.

Note that the Rastrigin function has a global minimum at $x = [0]^n$. Thus, since we took smaller values as having better scores, we want our solutions to be as close to $[0]^n$ as possible when using Rastrigin as the

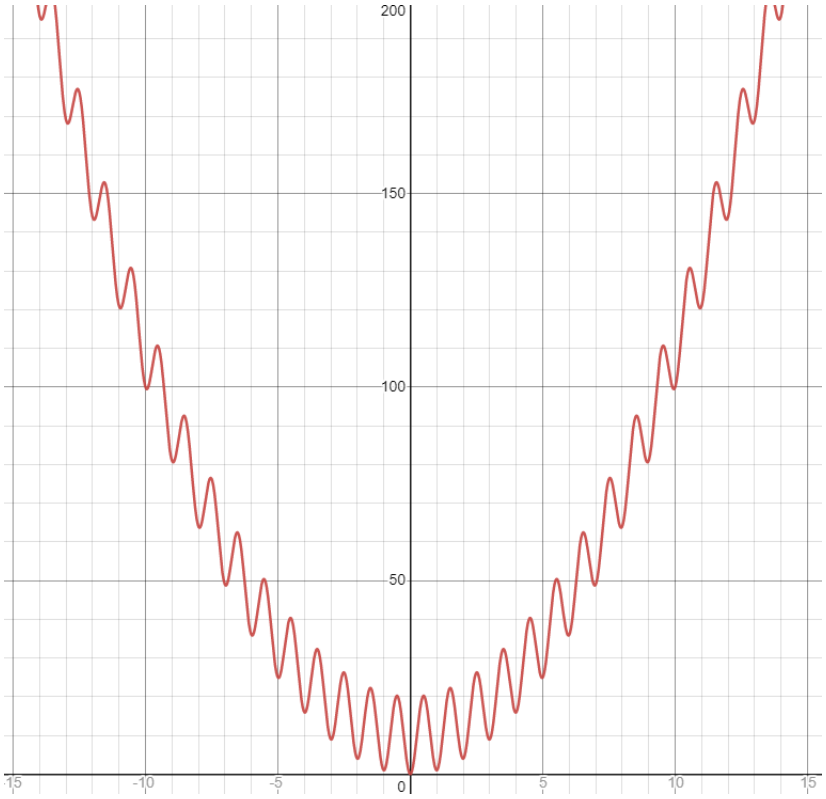

Figure 1: Rastrigin function for $n = 1$

objective function. Although it is technically possible for an initial randomly generated solution to be $[0]^n$ (the optimal solution), this is incredibly unlikely since the initial solutions (which are in $\mathbb{R}^n$) are randomly chosen from the interval $[-5.12, 5.12]^n$. Thus, it is much more likely for the initial solutions to be sub-optimal and thus have to be improved through local search to approach the optimal solution.

We defined crossover (used by CRO and GA for sexual reproduction) and mutation (used by CRO, GA, and SA for generating random neighbors) operators for the Rastrigin function optimization problem as follows:

- mutate(x): select a point using a Gaussian distribution with mean x and standard deviation 1
- crossover(parent$_1$, parent$_2$): for each coordinate, select a real number uniformly randomly between the parents' coordinate value

3.3 Discussion of Algorithms

3.3.1 Genetic Algorithms

Ultimately, we wanted to take into consideration best practices for genetic algorithms based on the literature while also taking into account the parameters for CRO so that the genetic and CRO algorithms can be fairly compared.

For population, we wanted a population size consistent with the size of the reef in the CRO algorithm. Thus, since the CRO algorithm had a 5×6 reef, we set the population size equal to 30. For the number of generations, we set it to $100n$ (where n is the number of dimensions) to be consistent with the number of iterations in the CRO algorithm. For crossover rate, the PatilV suggests a rate of about 60% is optimal but we found a rate of 70% worked well after parameter tuning [15]. For mutation rate, the literature suggests a rate of less than 5% and we found a rate of 3.2% to be best for this problem.

3.3.2 Simulated Annealing

The number of iterations of the SA algorithm was set to the number of generations $\times$ population size of the CRO and GA algorithms to ensure a similar number of objective function evaluations. The basic SA algorithm takes as input a cooling schedule which specifies at each iteration, what the temperature is, however because our group found this to be too big of a hyperparameter space to tune, we used a variant of the SA algorithm called thermodynamic simulated annealing which used laws of thermodynamics and the change in the solution quality between one timestep and the change in solution quality over the whole algorithm's runtime to dynamically set the temperature and in turn relied on two real hyperparameters instead: initial temperature and k_A which specifies how fast the temperature changes should be [10]. The last parameter was the probability function dictating whether worse solutions should replace good solutions, in which we used the Metropolis Hastings Algorithm probability which was used in the CRO paper and other literature on simulated annealing [10-11].

3.3.3 Coral Reef Optimization

The parameters for the Coral Reef Optimization algorithm are defined below.

The dimensions N and M of the reef define the population size of $N \times M$. The p_0 hyperparameter defines the ratio of the populated to unpopulated slots in the reef during initialization. F_b defines the fraction of broadcast spawners among the corals in the reef, and k defines the number of attempts larvae are allowed to settle into the reef (either by pushing out another coral with lower health or finding empty reef space). F_a defines the fraction of corals which asexually reproduce in the reef. F_d defines the fraction of worst-health corals which are at risk of depredation, and p_d is the risk of depredation for these corals.

For the Rastrigin optimization problem, we chose $p_0 = 0.4$, $F_b = 0.9$, $k = 2$, $F_a = F_d = 0.1$, and $p_d = 0.05$. We chose the parameters based on the parameters used in Sanz, et.al.

4 Results

The code for the implemented algorithms are at https://github.com/enkokoro/02251Project. We modeled our reefs as an array of corals with varying health.

In order to counteract the random initialization, we ran each of the algorithms 10 times and summarized the best, average, and standard deviations of the objective values across the 10 runs.

Table 1: Results for Ratrigin $n = 1$

Algorithm	Best Objective Value	Average Objective Value	Standard Deviation of Objective Value
CRO	0.0	$9.44398 * 10^{-4}$	$2.92788 * 10^{-3}$
GA	$8.34888 * 10^{-14}$	$3.21692 * 10^{-3}$	$7.36104 * 10^{-3}$
SA	$1.30951 * 10^{-7}$	$2.56079 * 10^{-5}$	$4.94849 * 10^{-5}$

Table 2: Results for Ratrigin $n = 5$

Algorithm	Best Objective Value	Average Objective Value	Standard Deviation of Objective Value
CRO	4.34655	8.68775	3.28546
GA	2.97608	5.01308	1.77544
SA	1.94231	3.91584	1.26146

Table 3: Results for Ratrigin $n = 10$

Algorithm	Best Objective Value	Average Objective Value	Standard Deviation of Objective Value
CRO	22.35716	37.06037	12.10945
GA	11.46621	15.73531	3.53195
SA	22.40993	27.20736	3.33707

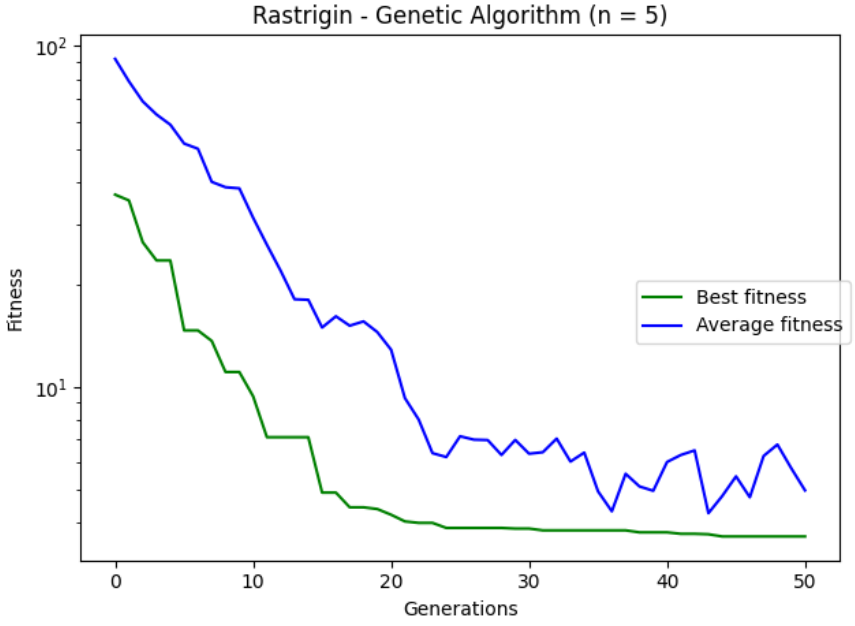

Figure 2: Genetic algorithm fitness evolution over the generations

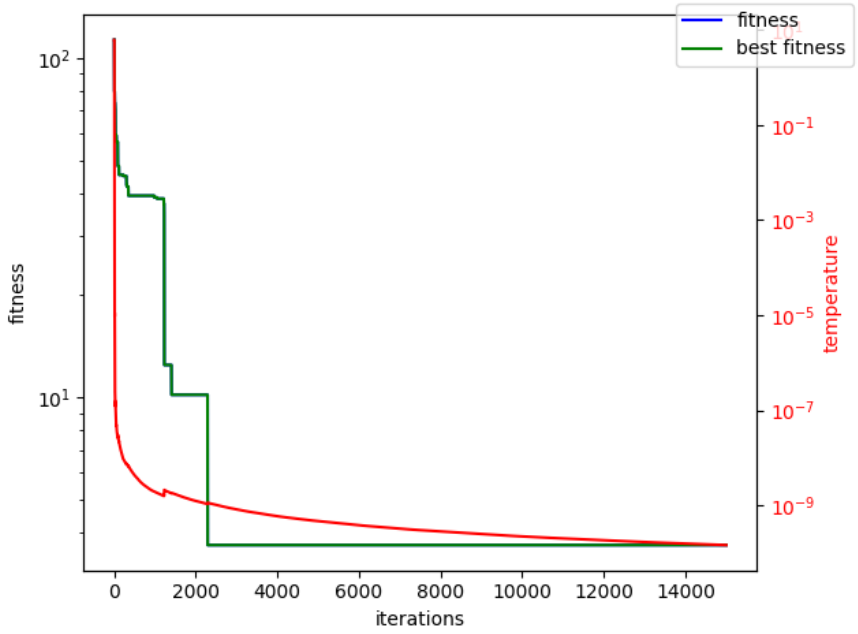

Figure 3: Thermodynamic simulated annealing fitness evolution over the iterations

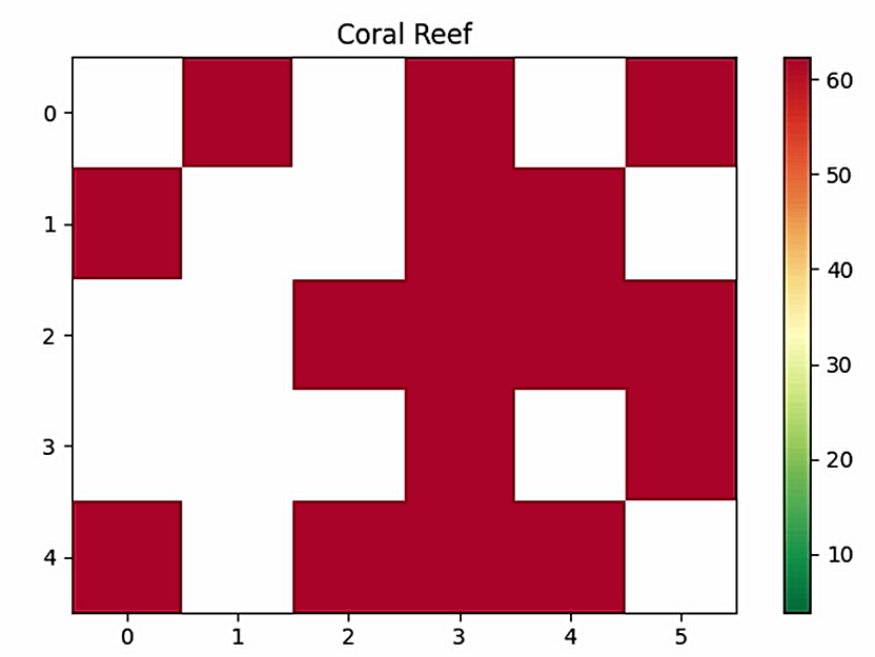

Figure 4: Coral reef optimization algorithm initial reef state consisting of empty reef space and low fitness solutions

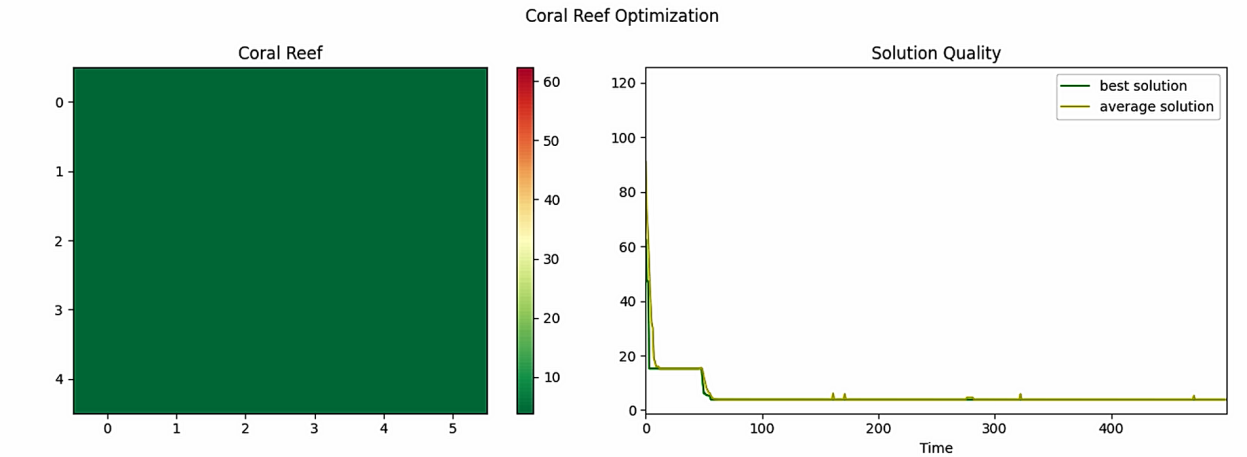

Figure 5: Coral reef optimization algorithm final reef state with how the fitness of the reef evolved over time

Example runs demonstrating how the fitness/objective value changes over the iterations are included for GA (figure 2) and SA (figure 3) for the Rastrigin function optimization problem with dimension $n = 5$. In addition, the initial (figure 4) and final (figure 5) states of the CRO algorithm for $n = 5$ are provided. All algorithms demonstrate the ability to find lower and lower objective values as time goes on and the coral reef optimization algorithm also demonstrates the ability to fill the reef with high fitness corals over time. The full set of videos and plots can be found on the Github page.

We found that with the parameters from the paper, CRO did not have significantly better performance than GA and SA. In fact, GA and SA performed better for $n = 5$ and $n = 10$ ($n = 1$ is too simple of a problem to compare the algorithm as all of the algorithms were able to reach very low objective values). We believe the lackluster performance is due to the large number of hyperparameters of the CRO algorithm as compared to GA and SA. We believe the coral reef algorithm should able to exhibit good performance, however there is a lot of effort and time needed to tune the hyperparameters which makes the algorithm less practical to use. We tried performing a parameter search for the parameters of CRO by holding other parameters constant while varying a selected parameter across a range (for example, varying the parameter p_0 in the range $(0, 1)$) and plotting the objective/fitness values obtained. Our idea was to generate the plots and choose values for the parameters which minimized the objective values. Selected graphs generated from our parameter search are included below (figure 6).

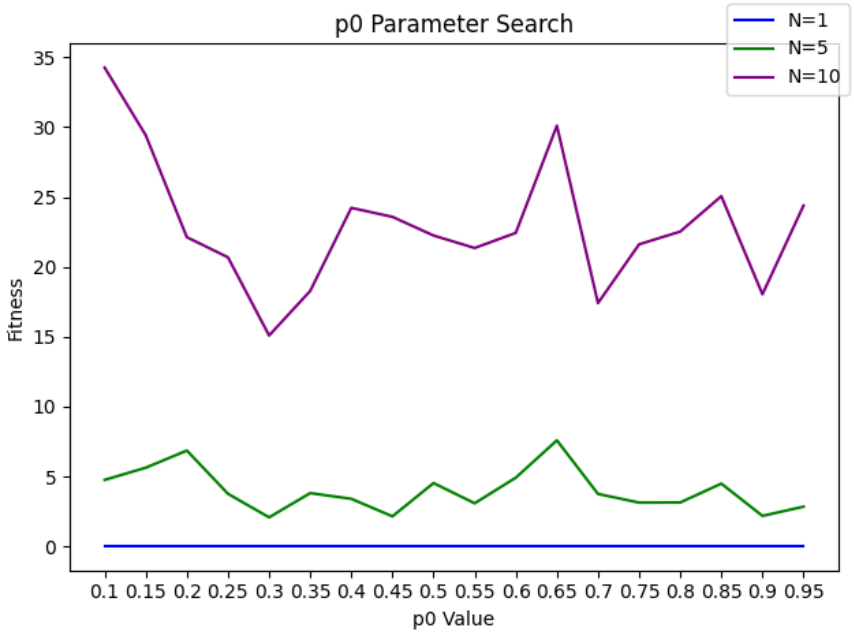

Figure 6: Parameter search graph for p_0 (occupied to free ratio of reef initialization)

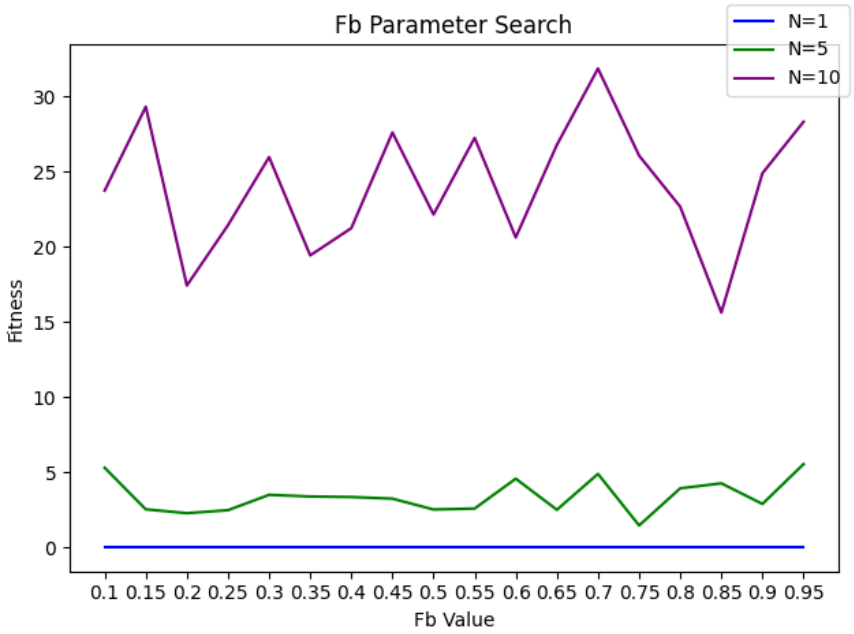

Figure 7: Parameter search graph for F_b (fraction of broadcast spawners)

As seen from the graphs, the plots show parameters are quite hard to tune, as the objective value does not monotonically increase as the value moves away from some minimum objective value. The plots exhibit lots of variation and show how the parameter choice is nontrivial and likely should more on combinations of parameter changes and interactions of parameters rather than optimizing parameters in isolation. It was

interesting to see how there was a meta optimization problem of optimizing hyperparameters for our given optimization problem.

5 Discussion

Coral Reef Optimization offers many possibilities for optimization problems. For instance, CRO can be modified to apply to combinatorial optimization problems, which aim to find an optimal set of solutions from a larger set of objects. There are multiple real world applications of combinatorial problems which CRO can be applied to. CRO by design allows us to split a search space into cell-like grids, so we can geographically divide a region into grids and apply CRO with some health function that determines the quality of the solutions. For instance, Sanchez et al have shown how CRO can be used to optimize wind farm design, which aims to find optimal wind turbine placement in a wind farm given a set of possible locations [16]. Thus, as a potential application of CRO, we propose a power plant optimization problem that utilizes CRO to identify the optimal locations to place power plants on a geographic grid.

Other possibilities for future research could include comparing CRO to other benchmark algorithms such as harmony search and differential evolution [17-18]. Each optimization algorithm has its own benefits and drawbacks, so comparing CRO to these algorithms could reveal its strengths and limitations. Overall, although we did not find that CRO had a better performance against other benchmark algorithms such as GA and SA, with more parameter tuning and optimized parameters, it may be possible for CRO to perform better than these algorithms and be an effective approximation algorithm.

6 References

1. Reinelt, G. (1994). The traveling salesman: computational solutions for TSP applications.
2. Fogel, D.B. An evolutionary approach to the traveling salesman problem. Biol. Cybern. 60, 139–144 (1988). https://doi.org/10.1007/BF00202901
3. Collett M, Chittka L, Collett TS. Spatial memory in insect navigation. Curr Biol. 2013 Sep 9;23(17):R789-800. doi: 10.1016/j.cub.2013.07.020. PMID: 24028962.
4. Tero, Atsushi & Takagi, Seiji & Saigusa, Tetsu & Ito, Kentaro & Bebber, Daniel & Fricker, Mark & Yumiki, Kenji & Kobayashi, Ryo & Nakagaki, Toshiyuki. (2010). Rules for Biologically Inspired Adaptive Network Design. Science (New York, N.Y.). 327. 439-42. 10.1126/science.1177894.
5. Salcedo-Sanz, S., & Del Ser, J., & Landa-Torres, I., & Gil-Lopez, S., & Portilla-Figueras, Antonio. (2014). The Coral Reefs Optimization Algorithm: A Novel Metaheuristic for Efficiently Solving Optimization Problems. TheScientificWorldJournal. 2014. 739768. 10.1155/2014/739768.
6. Escoffier, Bruno & Bonifaci, Vicenzo & Ausiello, Giorgio. (2011). Complexity and Approximation in Reoptimization. 10.1142/9781848162778_0004.
7. Boyd, Stephen P., & Vandenberghe, L. (2004). Convex Optimization. Cambridge University Press. p. 129. ISBN 978-0-521-83378-3.
8. Pirlot, Marc. (1992). General local search heuristics in combinatorial optimization: A tutorial, Belgian Journal of Operations Research. Statistics and Computer Science. 32. 8-67.
9. Gerges, F., & Zouein, G., & Azar, D., &(12 March 2018). "Genetic Algorithms with Local Optima Handling to Solve Sudoku Puzzles". Proceedings of the 2018 International Conference on Computing and Artificial Intelligence. ICCAI 2018. New York, NY, USA: Association for Computing Machinery: 19–22. doi:10.1145/3194452.3194463. ISBN 978-1-4503-6419-5.
10. De Vicente, J., & Lanchares, J., & Hermida, R. (2003). Placement by thermodynamic simulated annealing. Physics Letters A. 317. 415-423. 10.1016/j.physleta.2003.08.070.
11. Kirkpatrick, Scott & Gelatt, C. & Vecchi, M.. (1983). Optimization by Simulated Annealing. Science (New York, N.Y.). 220. 671-80. 10.1126/science.220.4598.671.
12. Laarhoven, P. J. M., & Aarts, E. H. L. (1987). Simulated annealing: Theory and applications. Dordrecht: D. Reidel.
13. Abiyev, R. H., & Tunay, M. (2015). Optimization of High-Dimensional Functions through Hypercube Evaluation. Computational intelligence and neuroscience, 2015, 967320. https://doi.org/10.1155/2015/967320
14. Mühlenbein, H. (1992). Parallel Genetic Algorithms in Optimization. In: Krönig, D., Lang, M. (eds) Physik und Informatik — Informatik und Physik. Informatik-Fachberichte, vol 306. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-77382-2_1
15. PatilV., P. (2015). THE OPTIMAL CROSSOVER OR MUTATION RATES IN GENETIC ALGORITHM : A REVIEW *.

16. Salcedo-Sanz, S., & Gallo-Marazuela, D., & Pastor-Sánchez,. & A., & Carro-Calvo, L., & Portilla-Figueras, A.; Prieto, L. (2014). Offshore wind farm design with the Coral Reefs Optimization algorithm. Renewable Energy, 63, 109-115. https://doi.org/10.1016/j.renene.2013.09.004
17. Gao, X.Z. & Govindasamy, V. & Xu, H. & Wang, Xianjia & Zenger, Kai. (2015). Harmony Search Method: Theory and Applications. Computational Intelligence and Neuroscience. 2015. 1-10. 10.1155/2015/258491.
18. Storn, R., & Price, K. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341–359 (1997). https://doi.org/10.1023/A:1008202821328

7 Appendix

7.1 Coral Reef Optimization

Input:
optimization problem (solution space, objective function, mutation and crossover operators)
number of generations
N, M : grid dimensions corresponding to population size
initial occupation rate p_0, broadcast spawning rate, asexual reproduction rate, predation factor, predation probability
Output: The optimal/most fit solution
Pseudocode
Reef Initialization
Assign squares to be occupied at random by corals/solutions such that $0 < p_0$ = free / occupied < 1
Reef Evolution for number of generations
1. Broadcast Spawning (External Sexual Reproduction)
2. Brooding (Internal Sexual Reproduction)
3. Larvae Setting
4. Asexual Reproduction
5. Depredation in Polyp Phase
Output most fit coral/lowest objective value solution of the current reef

7.2 Genetic Algorithm

Input:
optimization problem (solution space, objective function, mutation and crossover operators)
number of generations
population size
crossover rate, mutation rate
Output: The optimal/most fit solution
Pseudocode
Population Initialization
Randomly pick population size many elements from solution space as initial population
Population Evolution for number of generations
1. Select solutions from the population, with better fitness (lower objective value) solutions being selected with higher probability
2. Iterate through pairs of solutions of selected population and perform crossover with crossover rate probability to produce offspring
3. Iterate through offspring and perform mutation with mutation rate probability
Output solution with lowest objective value from the current population

7.3 Simulated Annealing

Input:
optimization problem (solution space, objective function, mutation and crossover operators)
number of iterations
cooling schedule, probability of replacement
Output: The optimal/most fit solution
Pseudocode
Initialization
randomly pick an initial solution from the solution space as the current solution
For number of iterations with temperature following the cooling schedule
1. Generate a neighbor solution of the current solution with the mutation operator
2. Based off of probability of replacement(current solution, neighbor solution, temperature) [if neighbor solution has strictly better/lower objective value: probability = 1; else probability of replacing with a worse solution is higher if temperature is higher], replace current solution with neighbor solution

Output current solution