Exploration or Convergence? An Iterative Control Mechanism for GAs

Abstract

Genetic algorithm based optimizers have to balance extensive exploration of solution spaces to find good solutions with convergence to generate solutions quickly. Many optimizers use a two phase approach where the first phase explores the solution space and the second converges on a set of potential regions. This paper describes an algorithm GA_ITER that iteratively applies a GA based optimizer with a bias towards either exploration or convergence. The optimizer is executed with a very small number of evaluations which leads to fast generation of solutions. The iterative approach of GA_ITER with the bias has been shown to lead to fast generation of good solutions. Experiments in two real-world domains have shown that GA_ITER can improve the performance of an existing GA without compromising the quality of the solution.

1 Introduction

Optimizers using genetic algorithms have been successfully used to solve many real world optimization problems. This is in part because Genetic algorithms (GA) have proved to be effective at searching a large space of possible solutions and finding good solutions. The issue of how much time or resources an optimizer should expend to find good solutions has been studied for a long time. A consequence of large solution spaces is that optimizers have to deal with the problem of convergence on locally optimal but globally sub-optimal solutions. There is a possibility that the population will converge on individuals within one particular region of the space. Optimizers need to decide between sampling unexplored regions, *i.e.*expanding the search, and converging on a subspace. This is a difficult design choice. If an optimizer does not sample sufficient points, it is likely to find itself converging upon a local optimum. On the other hand, an optimizer may over sample the space and it could take a long time before it converges to a good solution. This tradeoff is especially difficult in problems in high dimensional spaces because of the large solution spaces.

This paper describes a new algorithm GA_ITER that iteratively applies genetic algorithms to both find good solutions and also converge quickly. On each iteration, the algorithm decides whether to explore (search) or converge (focus) and applies the GA appropriately. The algorithm has been evaluated in two real-world domains, (1) finding the optimal *gape* (cross-sectional area) of a snake jaw and (2) finding the best set of parameters to model the swimming motion of a pumpkinfish. The first problem involves a search in an eight dimensional space while the second requires

a search in a fourteen dimensional space. This paper describes the algorithm along with the experimental results that shows the GA_ITER algorithm significantly improves the performance of an existing GA.

2 Problem Description

Morphology is the study of how changes in structure can affect function. The jaw of a snake is a complex object composed of many bones and the lengths of the various bones can affect both the size and shape of the prey that can be swallowed. One method of studying the effect is to construct a computational model of the jaw. This section briefly describes the structure of a snake jaw based on a specimen of the gopher snake (*Pituophis melanoleucus*) and the computational issues involved in studying the morphology.

A snake jaw is composed of ten elements, four bones on each side connected symmetrically by elements at the top and bottom. The four bones on each side are the *supratemporal, quadrate, compound* and *dentary*. The elements at the top and bottom are the *braincase* and *symphyseal*. Figure 1 is a representation of a snake jaw showing how the elements are connected. In the figure, the *braincase* and *symphyseal* are perpendicular to the side view. The *symphyseal* is not actually a bone but is an elastic element and the amount that it can be stretched varies from species to species. The joints connecting the elements allow movement in two perpendicular planes, the frontal and the sagittal, for each joint. These movements are constrained by restrictions on the maximum and minimum angles in each plane.

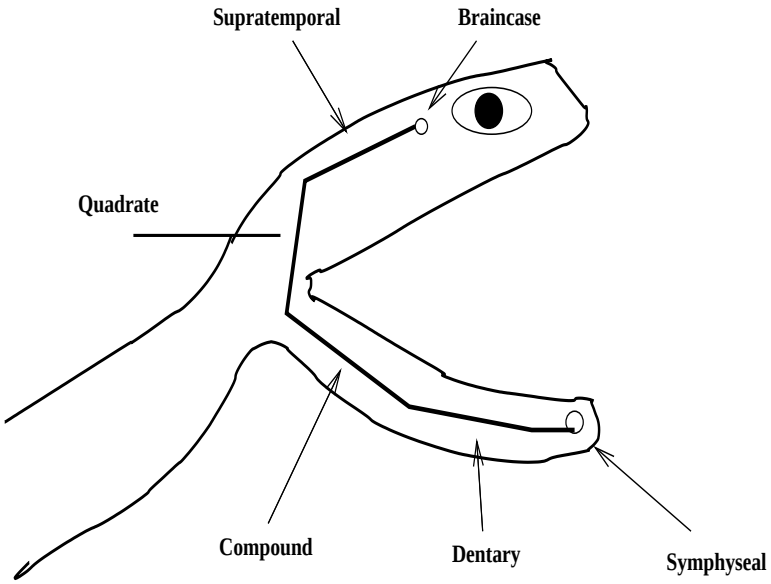

Figure 1: Side View of a Snake Jaw

Herpetologists are interested in the size and shape of the largest prey that a snake can swallow.

They believe that this is determined by the absolute and relative dimensions of the bones that make up a snake jaw. A snake jaw specification is composed of a set of dimensions for each of the bones. Finding the maximum size of the gape for a jaw specification consists of determining the corresponding set of values for each of the joint angles. This is treated as an optimization problem in eight real-valued dimensions, one for each joint angle subject to (1) constraints on the maximum and minimum values of each joint angle and (2) constraints that the configuration is realizable, *e.g.*the upper jaw does not overlap the lower jaw. A more detailed description of the problem and the system used to solve it can be found in [2].

The constraints render many potential solutions infeasible. Figure 2 is a surface plot that shows how the *fitness* of the solution changes as two of the angles (quadrate-frontal and compound-sagittal) are varied. The plot also shows that there are many infeasible points in the two dimensional region (the ones with values less than zero) that make it hard to find good solutions in the eight dimensional space.

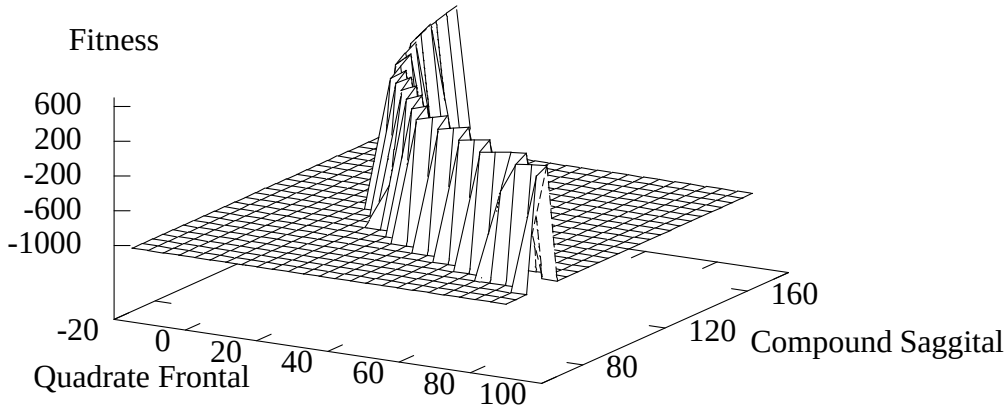

Figure 2: The Effect of Changes in Quadrate-Frontal and Compound-Sagittal Angles on Fitness

Several GA based optimization packages including the well known and referenced GALib package [13] were evaluated on the snake jaw problem. The GADO (Genetic Algorithm-Based Design Optimization) package [11, 3], an optimization package that was originally developed for the design of supersonic nozzle inlets, was eventually selected. The GADO package had the best performance for both time and solution quality. Figure 3 shows a comparison of the performance of GADO and GALib on an example problem from the snake jaw problem.

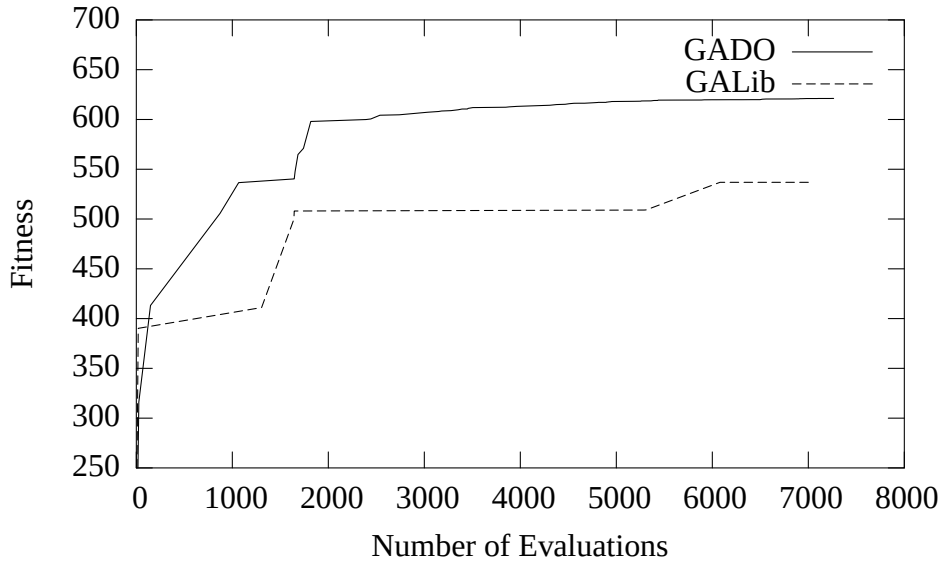

Figure 3: Performance of GALib vs GADO in the Snake Jaw Domain

3 Convergence versus Exploration

The tradeoff between convergence and exploration arises when there is a decision as to whether to generate new (subsequent) points from an unexplored region (exploration) or near a known good point (convergence). A GA should converge when it is currently in what it believes to be a region that would lead to better solutions. The GA should explore when it believes that the current region will not yield signifi cantly better solutions. Two factors that greatly affect the decision as to whether to explore or converge are (1) the total number of points to be evaluated (indirectly, this affects the number of generations of the population) and (2) the size of the population. A GA typically uses random sampling (exploration) to fi ll the initial population pool and then uses a combination of (1) mutation, (2) adaptation, and (3) sampling to generate new individuals for subsequent generations. If the size of the population is too small, the GA will have insuffi cient diversity to effectively use its crossover operators. On the other hand if the population size is too large, the GA wastes computational resources in managing and analyzing the individuals in the population. The total number of points (individuals) to be evaluated also directly affects how quickly a GA can or should converge upon a solution. If the total is relatively large compared to the population size, the GA can initially spend more resources (time) examining (sampling) points from unexplored regions rather than combining individuals from the population (mutation and cross-over). A larger number of sample points would help the GA avoid sub-optimal regions. On the other hand a large total number of evaluations requires more computation time, an important consideration when the cost of generating each individual is relatively high or when the solution space is large.

There have been many studies [7, 6, 4] that have explored various combinations of controlling

parameters to generate the best performance in GAs. Using the best set of parameters, *i.e.*the ones that generate the best solutions, usually results in a GA evaluating many points. This can be computationally infeasible if (1) it requires a substantial amount of computation to generate each point and (2) there are many dimensions (parameters) to the problem so the search space is large. The snake jaw problem is an example of such a problem. It takes $3 \sim 5$ seconds to generate each point and there are eight dimensions in the problem. The default settings for GADO ($number\_of\_evaluations = 8000, population\_size = 80$) require four hours to generate a solution for one specification of bone dimensions. The computational cost can be reduced by lowering the total number of evaluations but this can negatively impact the solution quality. Table 1 shows the impact of the total number of evaluations on the solution quality.

Total Number of Evaluations	Fitness Value
250	420.92
500	489.09
750	449.42
1000	552.56
2500	605.73
5000	624.75
8000	625.1

Table 1: Effect of Total Number of Evaluations on GADO (Snake Jaw Problem)

Figure 4 shows how the fitness of the solutions changes as the GA generates successive new individuals. The most interesting observation that can be made from the figure is that there are several places where the GA "plateaus", *i.e.*successive individuals do not have any impact on the overall fitness. This occurs after about 200 solutions (individuals) have been generated and again after 2000 and 3500 solutions. In addition, after 3500 points, the GA has essentially plateaued and is generating successive points in the same region without significantly improving the quality of the solution.

The performance can be improved if the GA can avoid regions with low potential and instead focus on searching for good solutions in yet unexplored regions. A region rapidly loses potential if successive points generated from that region are not significantly better. The plot in Figure 4 illustrates two related problems with the current approach. Firstly, many solutions have to be generated before the GA can converge on a good solution. Secondly, there are regions where the GA might avoid generating unproductive solutions by changing its focus. What is needed is an algorithm that (1) generates better solutions faster and (2) will not converge prematurely and generate lower quality solutions by dynamically changing the focus between exploration and convergence. The GA_ITER algorithm that we have developed has these desired qualities and is described in the next section.

Other approaches to dealing with the tradeoff between exploration and convergence include swarming [9, 8, 1, 5] and scouting-inspired evolutionary algorithms (SEA) [10]. Swarming focuses more on using the initial sample of points to find good regions of convergence by incrementally

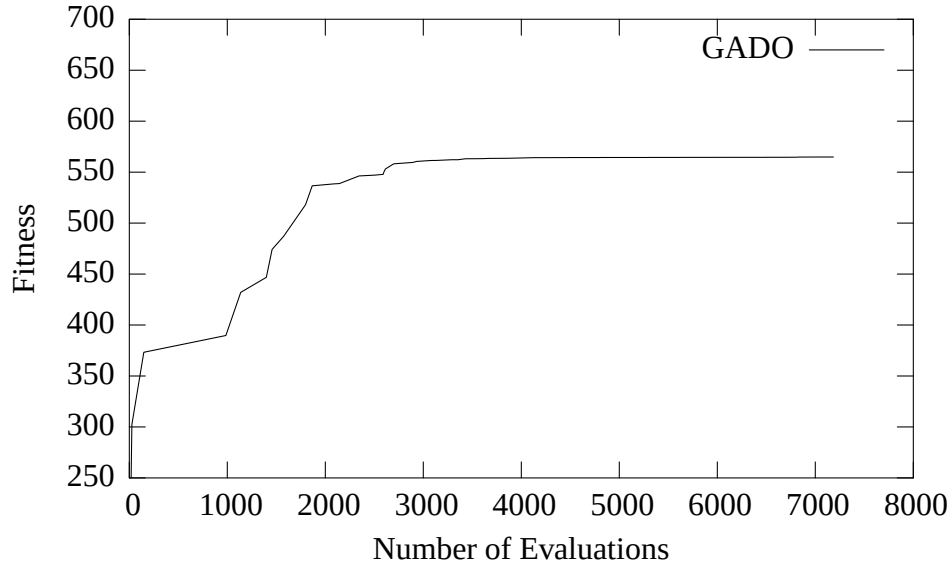

Figure 4: Behavior of GADO on a Snake Jaw Problem

searching from a given location. The SEA approach is similar in many ways to our approach but SEA deals more with trying to avoid locally optimal regions automatically, by modulating the search dynamics based on previously generated individuals.

4 Algorithm Description

The main idea behind our algorithm is an iterative application of the GA where each iteration requires a decision as to whether to converge or explore. The decision is implemented by "seeding" each iteration with individuals from the previous iteration to bias the GA. The result at the end of each iteration, where each iteration is one complete invocation of the GA, is compared with the result from the previous iteration. A subset of the points in the population at the end of the iteration is then used to seed the next iteration. The number of points used to seed is dependent upon whether the improvement in results between the current and previous iterations is above a given threshold. If the improvement is greater than (or equal to) the threshold, the system will continue in the same direction and will seed with the entire population (excluding infeasible points). If the improvement is below the threshold, the GA is has found a plateau and the number of points used to seed the next iteration is reduced. This causes the GA to fi ll the population by sampling points at random. The number of points used to seed the next iteration biases how much exploration is performed. If the GA is still improving the solution by more than the threshold, the search is strongly biased in the current direction by using the entire population. If the GA has found a plateau, the search is biased in the current direction slightly (it might contain the best solution) by using a much smaller number of seed points.

A second key idea is that the the number of points to be evaluated on each iteration is set at a very low value (just slightly larger than the population size). This causes the GA to converge quickly, even to a sub-optimal region. The subsequent iterations will determine whether the algorithm will continue in this region or explore other regions. This focus on a small number of evaluations on each iteration results in good solutions showing up in a small number of iterations and thus a small number of evaluations.

4.1 The GA_ITER Algorithm

The basic GA_ITER algorithm is given below:

1. Set the parameters for the GA to the recommended default values.

2. execute GADO, evaluate the result and calculate the improvement from the result of the previous execution of GADO. If the improvement is greater than or equal to the *threshold*, then set the value of *number_of_seed_points* to *converge_seed_points*, seed the next GADO execution with *number_of_seed_points* from the best clusters (see Section 4.2 for a description of clustering) found in this execution. Repeat step2.

3. If the improvement is less than the *threshold*, then GADO has found a plateau. If there has been no improvement greater than the threshold in the last ten iterations, the algorithm terminates. Otherwise, set the value of *number_of_seed_points* to *explore_seed_points*, seed the next GADO execution with points from the best clusters in this iteration and repeat step 2.

The GA parameters used for our experiments in the snake jaw domain are shown in Table 2.

GA Parameter	Value
population_size	80
number_of_evaluations	200
converge_seed_points	80
explore_seed_points	20

Table 2: GA Parameters for Snake Jaw Problem

The population size is dependent upon the number of dimensions in the problem and the default value in GADO is ten times the number of dimensions.

4.2 Clustering and Seeding

There are many ways in which points generated from one iteration can be used to seed the next iteration. The best points from the population could be selected but this has the problem of limiting

Percent of Best Solution	GA_Iter (Avg)	Best	Worst	Std Deviation
50%	100%	100%	100%	0
75%	48.7%	25.9%	77.7%	17.8
90%	49.8%	29.5%	65.9%	12.2
95%	62.1%	45.9%	86.8%	15.8
100%	89.9%	81.7%	102.5%	11.1

Table 3: Evaluation of GA_ITER on Snake Jaw Problem

diversity in that the best points could come from one region or cluster of points and the next iteration would be strongly biased towards that cluster. Instead the algorithm uses clusters of points within the population to seed the next iteration. The clusters are defined as points that are separated from each other by a distance that is less than a *clustering_threshold*. This approach has the advantage of not having a pre-defined number of clusters. The value of the variable *clustering_threshold* is currently set at 0.15, a value that has been experimentally determined to give good results. Once the clusters have been computed, the clusters are ordered by the best fitness value from each cluster and then the points within each cluster are rank ordered by their fitness values. When selecting the points to seed the next iteration, the best point is removed from each cluster in order of their ranking and placed in the pool. The process is repeated until the pool contains *number_of_seed_points* points.

5 Experimental Results

The performance of GA_ITER was evaluated in two different real-world domains. The first domain is the snake jaw problem described in Section2. The second domain is to calculate the best values for the parameters of both a fish and fluid model that best describes the swimming motion of a fish.

The first set of experiments involved evaluating GA_ITER on many different variations of the snake jaw problem. Each bone's dimensions was varied by increasing and decreasing it by 10% up to a maximum of 30%. The maximum stretch of the symphysis was also varied between 1 and 2 times the length of the braincase in increments of 0.2. The evaluation of GA_ITER is shown in Table 3.

Table 3 shows the number of evaluations required by GA_ITER as a percentage of the number of evaluations required by GADO to find the different levels of solution quality over all the snake problems. The experiments were carried out with 25 different variations of the snake problem. The first column represents the solution quality, *e.g.*the first line refers to solutions that are 50% of the best known solution. The second column represents the average number of evaluations required by GA_ITER as a percentage of the evaluations required by GADO. The third and fourth columns show the best and worst case performance (percentage wise) and the fifth column gives the standard deviation.

The data shows that GA_ITER finds good solutions much faster than GADO. This is especially important for problems where (1) the solution space is large and (2) the cost of generating a solution

is high. For those problems, the computational time required for a GA to find the best solution would be too great and instead a satisficing solution would be sufficient. Figure 5 shows a plot of the performance of the two genetic algorithms on a problem instance from the snake jaw domain.

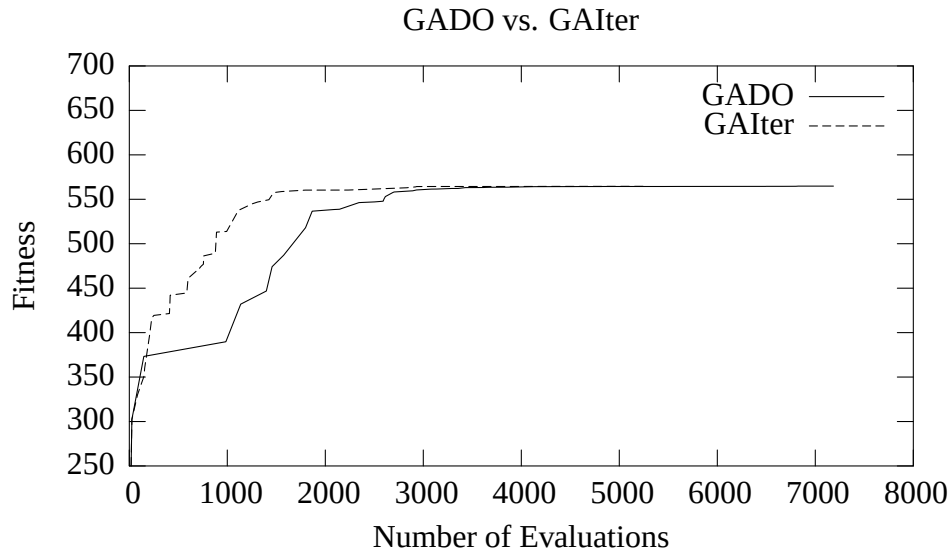

Figure 5: Plot of GA_ITER vs GADO on Snake Jaw Problem

5.1 Fish Locomotion Models

The GA_ITER algorithm was evaluated on a second real-world domain, that of modeling the swimming motion of a fish, specifically a pumpkinfish. The swimming model [12] actually consists of two computational models representing (1) the fish and (2) the fluid. Each model has an effect on the other and information must be exchanged between the models for an accurate simulation. The goal is to use the models to compute a swimming motion that most closely resembles the swimming motion of an actual fish over a specified time interval. The fish model is an approximation based on a sequence of stiff rectangular plates connected through hinges. Currently, the model uses eleven plates and twelve hinges

Images of a swimming pumpinfish were recorded over a short time interval at the rate of thirty frames per second. The coordinates of several points on the fish body were extracted from each frame and are used to show the body shape changes (changes in the coordinates) as the fish swims. An iterative loop is used to calculate the coordinates of the corresponding points on the fish model. Each iteration corresponds to an image frame. The coordinates of positions of the digital fish body are then extracted for each time interval of the simulation. The computed coordinates are then compared with coordinates extracted from the video of the actual fish. The goal of the optimization is to minimize the sum of the least squares difference between the actual and computed coordinates over the entire time interval (all iterations).

The behaviors of the two models are controlled by setting values for fourteen parameters, *e.g.*the initial positions, velocities and angles for the plates and hinges and the viscosity of the fluid The problem can thus be defined as an optimization problem in fourteen dimensions where the goal is to determine the *best values* for the controlling parameters of the models such that the difference in motion between the actual and digital fish is minimized. The evaluation of the GA_ITER algorithm will be extended to cover more complex models of the fish and fluid that require an optimization with forty two parameters.

GA Parameter	Value
population_size	140
number_of_evaluations	350
converge_seed_points	140
explore_seed_points	20

Table 4: GA Parameters for Fish Problem

The GA parameter settings used for this problem are specified in Table 4.

Percent of Best Solution	GA_Iter
50%	8%
75%	9%
90%	10%
95%	8%
100%	12.1%

Table 5: Evaluation of GA_ITER on Fish Model

The GA_ITER algorithm performed very well in this second domain. Table 5 shows the data that was obtained for this second problem in comparing GA_ITER and GADO. The GA_ITER algorithm is particularly useful in this problem because of the computational requirements. Each point (potential solution) takes thirty seconds to generate and using the default parameters for GADO requires a fourteen day computational run. The GA_ITER algorithm finds a good solution (within 5% of the best known) in less than ten percent of the time that GADO takes.

5.2 Discussion

The experimental results from the two domains show that the GA_ITER algorithm performs very well in comparison to GADO. The GA_ITER algorithm converges on good solutions much faster than GADO and the quality of the solutions is just as good. Over all our experiments, there were some problem instances where GADO had better solutions and other problem instances where the inverse was true. In all cases, the best solution from GA_ITER was within 2% of the best solution of GADO.

6 Conclusion

This paper has described an iterative algorithm GA_ITER for controlling GA's that provides a better ability to reason and control the tradeoff between exploration and convergence. The algorithm generates good solutions more efficiently, an important factor with problems that (1) have large solution spaces and (2) require significant computational resources to generate solutions. The algorithm has been evaluated on two such real-world problems, one of modeling snake jaws and the second the modeling of the swimming motion of fish. In both domains the algorithm has performed very well in finding good solutions more efficiently than previous systems. In addition, the GA_ITER algorithm can be easily combined with existing GAs.

References

[1] ANGELINE, P. Evolutionary computation versus particle swarm optimization: Philosophy and performance. In *Proceedings of the 1998 Conference on Evolutionary Computation* (1998).

[2] AUTHOR. Using a genetic algorithm to optimize the gape of a snake jaw. In *Proceedings of Twenty Fourth ACM Symposium on Applied Computing* (2004).

[3] BLAIZE, M., KNIGHT, D., AND RASHEED, K. Automated optimal design of two dimensional supersonic missile inlets. *The Journal of Propulsion and Power 14*, 6 (1998).

[4] CICIRELLO, V., AND SMITH, S. Modeling ga performance for control parameter optimization. In *Proceedings of The Genetic and Evolutionary Computation Conference 2000* (2000).

[5] EBERHART, R., AND SHI, Y. Comparison between genetic algorithms and particle swarm optimization. In *Proceedings of the 1998 Conference on Evolutionary Computation* (1998).

[6] EIBEN, A. E., HINTERDING, R., AND MICHALEWICZ, Z. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 3*, 2 (1999).

[7] GREFENSTETTE, J. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics 16*, 1 (1986).

[8] KENNEDY, J., EBERHART, R., AND SHI, Y. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[9] KENNEDY, J., AND EBERHART, R. C. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks* (1995).

[10] PFAFFMANN, J. O., BOUSMALIS, K., AND COLOMBANO, S. A scouting-inspired evolutionary algorithm. In *Proceedings of Congress on Evolutionary Computation 2004* (2004).

[11] RASHEED, K., HIRSH, H., AND GELSEY, A. A genetic algorithm for continuous design space search. *Artificial Intelligence in Engineering 11*, 3 (1997).

[12] ROOT, R. G., PSEMENEKI, T., CORTEZ, R., WATTS, P., AND JR; J. H. L. Heads or tails? anterior thrust generation in numerically-simulated carangiform fi sh. *Journal of Morphology 260* (2004).

[13] WALL, M. The galib-2.4.2 genetic algorithm library. `http://lancet.mit.edu/ga`.