CHAPTER 1 Optimization Algorithm

Within the scope of evolutionary algorithms, genetic algorithm [GA] and particle swarm optimization [PSO are the dominant algorithms when the problem demands robustness and performance. With the overarching objective of integrating the optimization algorithm with a complex and time-intensive objective function, the comparison between PSO and GA must be carefully considered to ensure that the chosen solver can meet the unique demand of its objective function. In this section the core functionality of each algorithm will be discussed and then compare them against one another in a case study to statistically determine the optimal solver for the problem.

1.1. Genetic Algorithm

The GA is a kind of evolutionary algorithm that mimics the general concept of evolution. Natural selection is often mentioned in the context of evolution since it is the strong individuals that survive in each environment. Being the strongest is a generalization that is defined by the objective function applied to the optimization problem. The structure of a population subject to the GA is visualized in figure 1.1 encapsulating a fixed number of chromosomes, which themselves encapsulate genes.

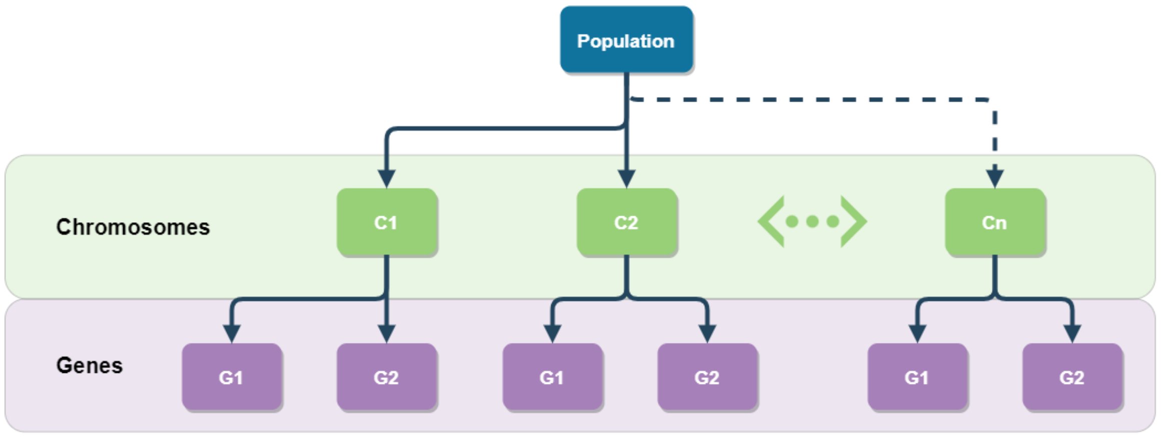


Fig. 1.1. Layout of a genetic algorithm with an arbitrary number of chromosomes and genes per population.

To understand the function of a gene, the application of the algorithm must be defined since the genes are merely input variables to the model that requires solving. If the optimization problem were a 2-dimensional surface plot minimization, the inputs to the model would be an arbitrary 2-dimension coordinate. Each dimension of this coordinate is considered a gene through the nomenclature of the GA.

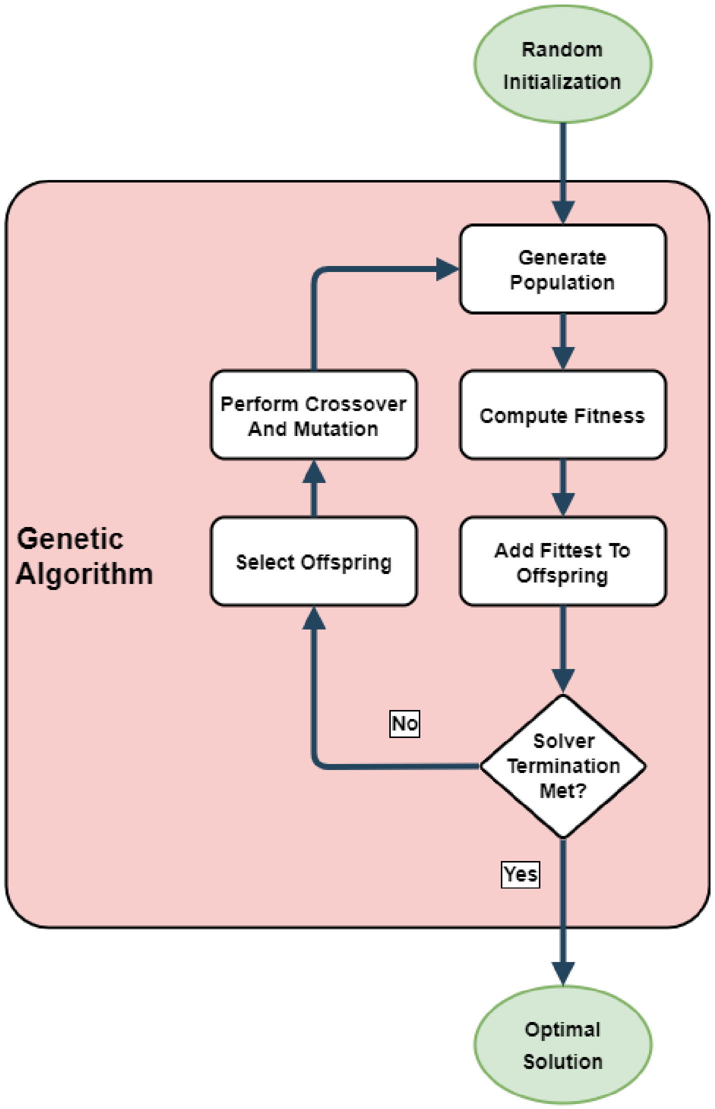


Fig. 1.2. Layout of a genetic algorithm execution loop.

Throughout each iteration of the solver a new population is produced through the means of selection, crossover, and mutation. This iterative loop ensures that the algorithm favors the desirable solutions while maintaining robustness through some degree of randomized search throughout the optimization domain.

1.2. Particle Swarm Optimization

Like GA, the PSO mimics the natural phenomenon of the power of a collective. This is often seen in swarms of insects such as bees which constantly communicate with one another to determine the optimal direction of the entire swarm. If the swarm's objective were to find a new location to establish a hive, each bee plays a critical role to gather information and relay it throughout the swarm so that the collective can weigh the signals and converge on decisions in real time. Instead of the population, chromosomes, genes, and offspring terminology, the PSO uses swarm size, particles, and leaders.

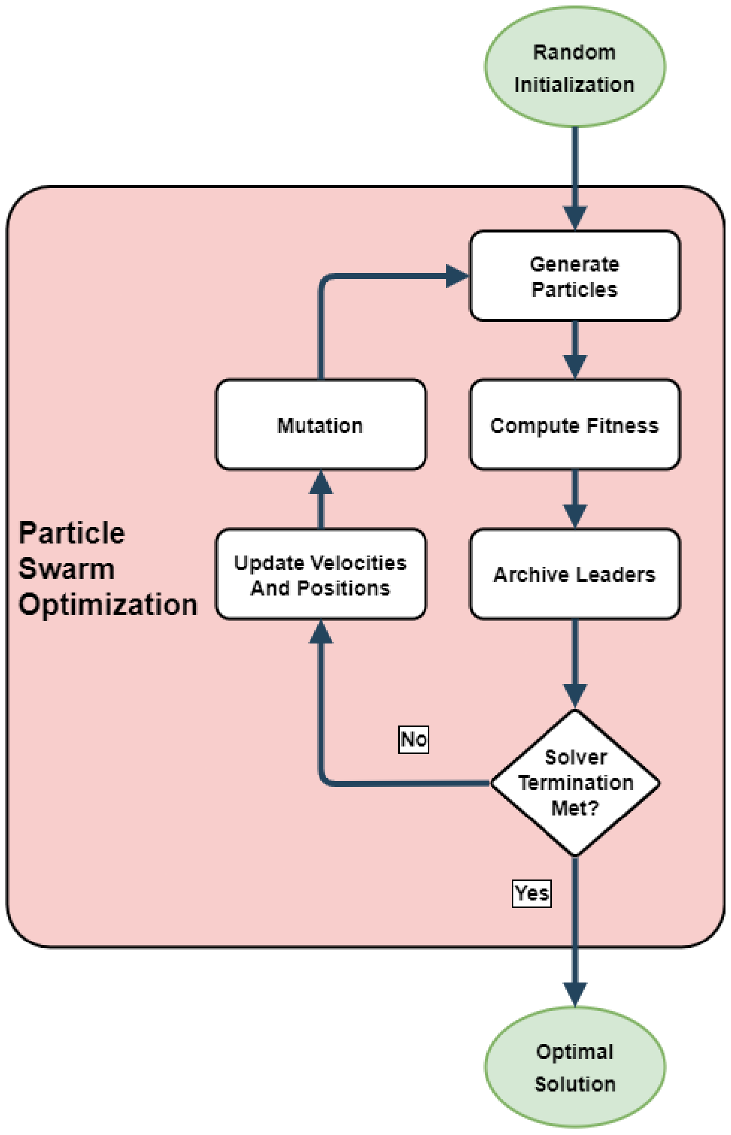


Fig. 1.3. Layout of a particle swarm optimization algorithm optimization loop.

The optimization loop of the PSO shows the process of updating velocities and positions per particle in the swarm as elaborated in equations 1 and 2.

$$v_{ij}(it+1) = \mathrm{w}v_{ij}(it) + r_1 c_1 \left(x_{min_{ij}}(it) - x_{ij}(it)\right) + r_2 c_2 \left(G_{min_j}(it) - x_{ij}(it)\right) \quad (1)$$

$$x_{ij}(it+1) = x_{ij}(it) + v_{ij}(it) \quad (2)$$

The current and successive iterations are denoted as it and $it+1$ respectively, where the local $x_{min_{ij}}$ and global G_{min_j} best solutions are determined prior to updating positions x_{ij} and velocities v_{ij}. The inertial weight coefficient w, local weight coefficients c_1 and r_1, and global weight coefficients c_2 and r_2 are integral in determining the relative influence the swarm has on the particle and vice versa.

TABLE 1.1
PSO SCALAR RANGES

Constant	**Range**
R	$\in [0.0,1.0]$
C	$\in [1.5,2.0]$
W	$\in [0.1,0.5]$

Referring to the optimization loop, the final step before calculating the objective function on the updated particles is to subject each particle to a mutation algorithm with a designated probability that the mutation executes. This allows for variation of the swarm and increasing the robustness of the solver to avoid convergence on local minima and maxima.

1.3. Schwefel Function Minimization Case Study

A case study was conducted to determine the optimal optimization algorithm among the subset of EAs through the Schwefel test function. A test function is used to test the ability of an optimization algorithm to converge on a solution that is the global maximum or minimum rather than the function's local maxima or minima. The Schwefel function was chosen since it has a plethora of local maxima and minima which can stall solvers prior to converging on the solution. The function is defined as:

$$f(x) = 418.9829d - \sum_{i=1}^{d} x_i sin\left(\sqrt{|x_i|}\right)$$

where d is the number of input dimensions and x_i is the function input per dimension i. The global minimum $f(x^*) = 0$ is located at $x^* = (420.9687, \ldots, 420.9687)$ inside of the hypercube $x_i \in [-500,500]$ for all $i = 1, \ldots, d$.

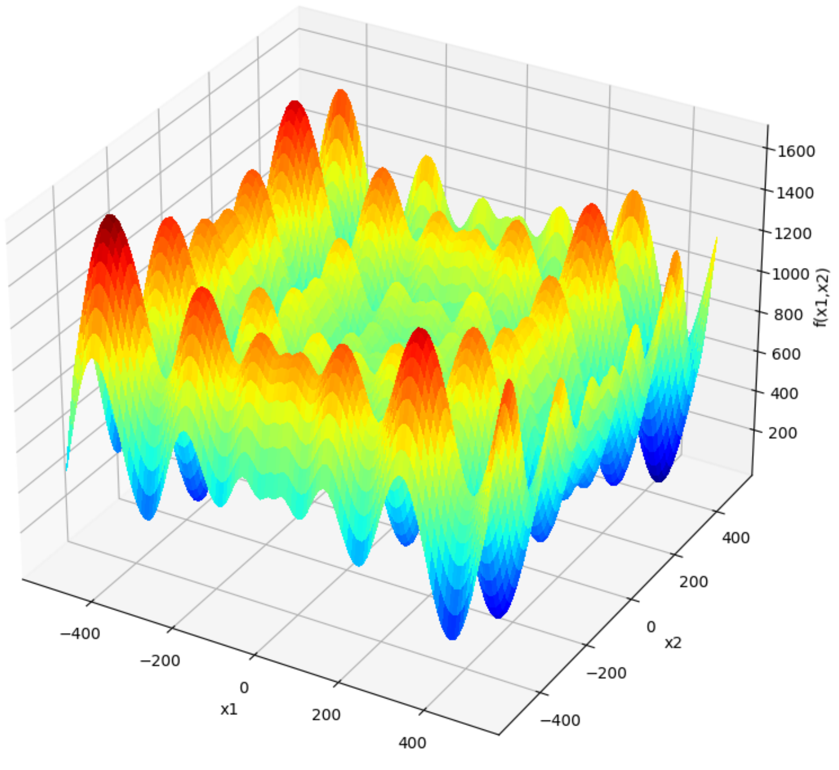


Fig. 1.4. Surface plot of the Schwefel function on the $x_{1,2} \in [-500,500]$ input range.

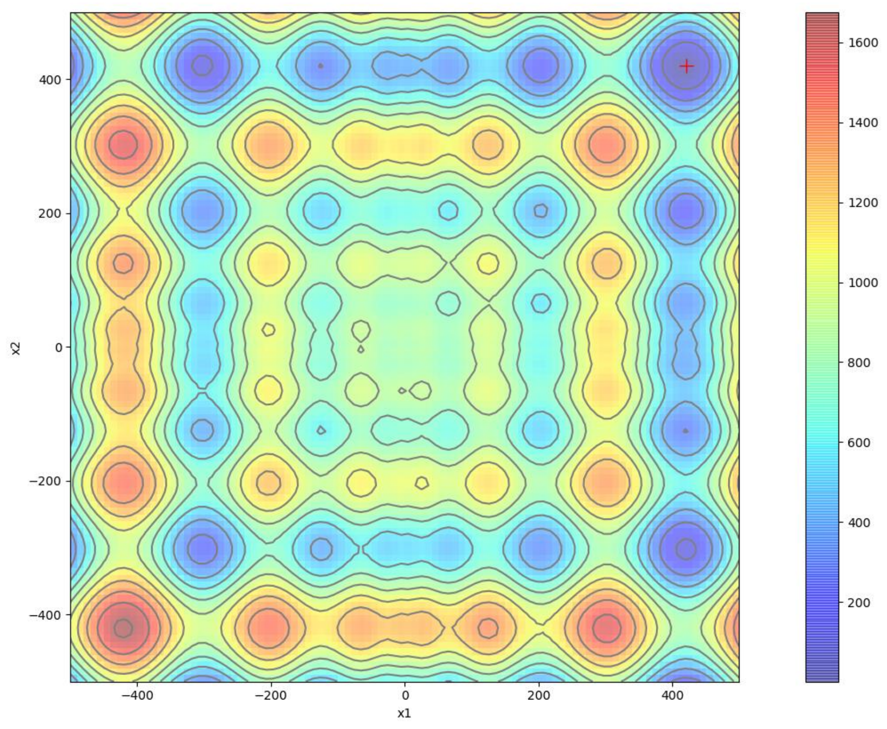


Fig. 1.5. Contour plot of the Schwefel function on the $x_{1,2} \in [-500,500]$ input range highlighting the global minimum with a red cross.

To couple a solver to this test function, a new input x^* is generated by the solver per iteration. These inputs are used to calculate and minimize the objective value through the Schwefel function until convergence on a solution. To ensure that each optimization algorithm is fairly compared in this case study, common solver parameters are used to configure each algorithm which can be found in table 1.2. Every algorithm will iterate over its population or swarm with the only solver termination criteria being the max number of stall iterations reached. Other solver termination criteria like reaching objective tolerance, timeout, and maximum iterations were omitted in this case study to isolate each solver through a consistent test domain. Additionally, the optimization process is conducted 5 times per algorithm to determine the average performance to ensure that an outlier does not significantly impact the decision making. Table 1.3 compares the EAs: PSO and GA through performance parameters like execution time and error. The solver robustness is the principal performance parameter, while the solver time holds less value as a performance parameter.

TABLE 1.2
OPTIMIZATION ALGORITHM CONFIGURATION

PSO		GA	
Population/Swarm Size	200	**Population Size**	200
Max Leader Size	100	**Offspring Size**	100
Comparator Key	Objective Value	**Crossover Percentage**	30%
Mutation Percentage	10%	**Mutation Percentage**	10%
Algorithm Stall Iterations	25	**Algorithm Stall Iterations**	25
Global Upper Bound	[500, 500]	**Global Upper Bound**	[500, 500]
Global Lower Bound	[-500, -500]	**Global Lower Bound**	[-500, -500]

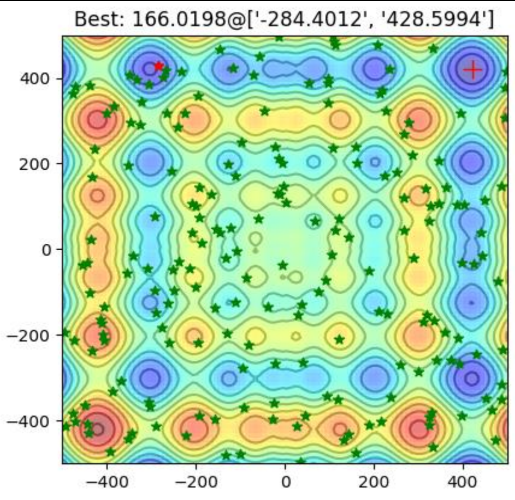
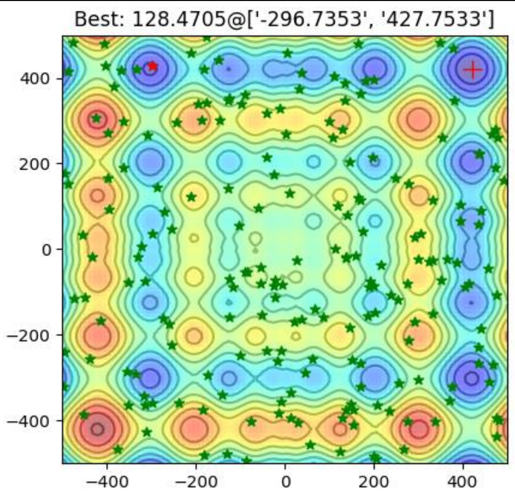
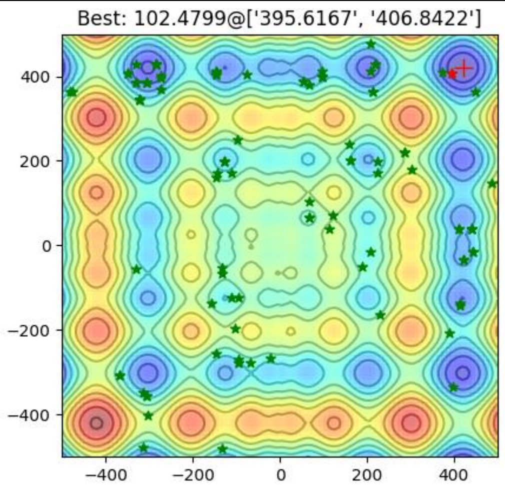
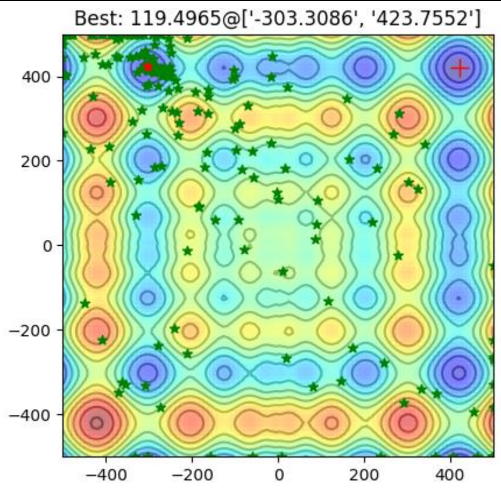
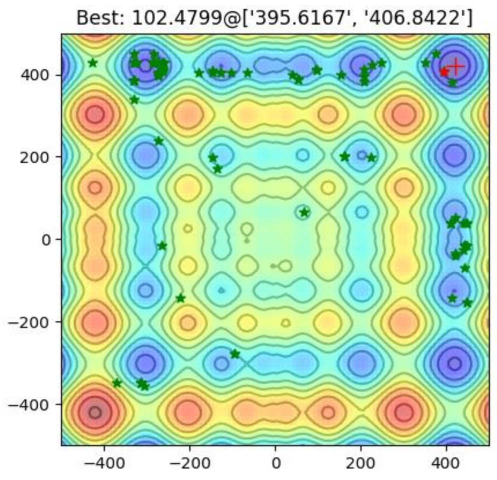
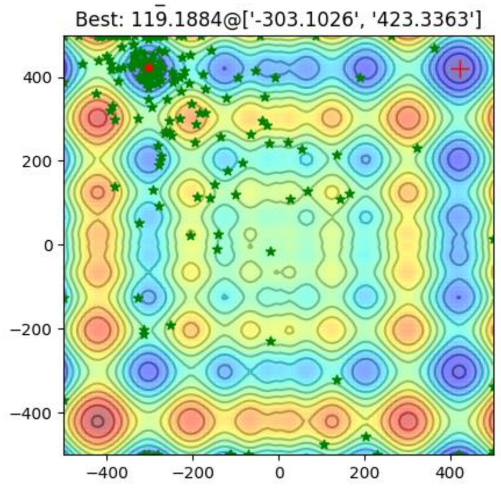
From the data found in table 1.3 it is evident that both the GA and PSO converge on the global minimum across 5 trials. The error in the final coordinate and the error in the resulting objective value at the coordinate were considerably low, although GA was not able to search the peaks as well as PSO. The characteristics of the algorithm’s ability to search the space can be visualized by plotting the swarm or population for a given solver iteration. A comparison between GA and PSO searching the space on the contour plot shown in figure 1.5 is achieved by selecting the early iterations of each solver. This comparison is found in table 1.2 which highlights the meta differences between GA and PSO.

TABLE 1.3
AVERAGE OPTIMIZATION ALGORITHM RESULTS

Algorithm	PSO	GA
Time (s)	1.5246	1.4546
Objective Function Executions	10760	4171
Solver Iterations	57	116
Value of X1 Solution	420.9728	420.9522
Error of X1 Solution (%)	0.5053	1.6543
Value of X2 Solution	420.9669	420.9729
Error of X2 Solution (%)	0.1810	0.4172
Value of Final Objective	0.0001	0.0002
Error of Final Objective (%)	0.0052	0.0195

The GA tends to cluster in the minima that it finds after the first iteration and spawn offspring that allows it to search those minima further. This continues until the population produces enough generations at better minima, reducing the number of offspring centered around the local minima. Contrasting this with PSO, the swarm finds the global optimal solution after the first iteration and begins to orient the velocities towards the swarm's global minimum (different from the domain's global minimum). When particles find other local minima, they will slightly affect the swarm's orientation unless it is the swarm's new global minimum, in this case the swarm begins to reorient towards this point with much greater influence.

TABLE 1.4
ALGORITHM CONVERGENCE VISUALIZATION

	GA	PSO
Iteration 1		
Iteration 3		
Iteration 5		

The time of termination, after 25 stall iterations were reached, for each algorithm was approximately the same due to the lack of computation intensity this optimization problem requires. However, the number of solver iterations i.e., the number of new swarms or populations produced, were much greater in the GA although this is not a concern. The method in which the swarms and populations are produced are time efficient and only significantly hinder computation time when the swarm or population are significant in size. When comparing the objective function executions required to converge on a solution, this is where there is a clear difference between the GA and PSO. The number of function executions is more than double that of the GA which is not intuitively a problem. The visualization of the problem is introduced in figure 1.6 showing the divergence of the GA and PSO solver times when the objective function execution time increases.

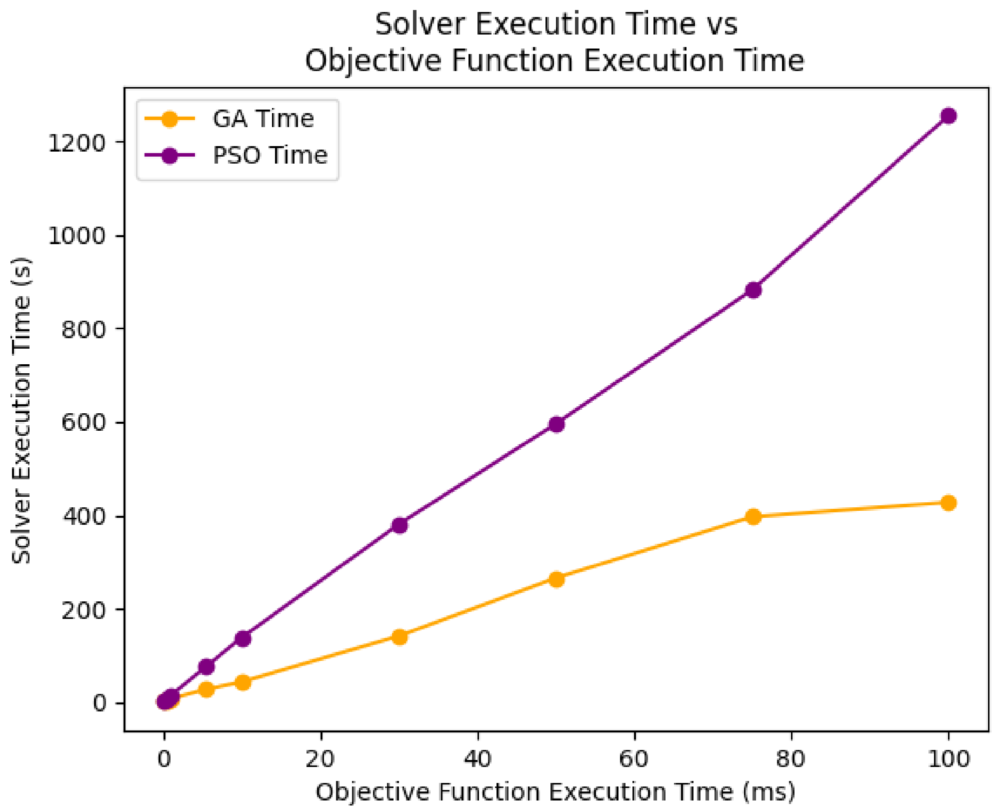


Fig. 1.6. Comparison of the average solver execution time between GA and PSO until 25 stall iterations are achieved using the Schwefel test function at different artificial objective function execution times.

The Schwefel function was artificially slowed down from the original 0.0086ms to 100ms in steps shown in the plot. Solving the Schwefel test function at each of these steps and logging the time it takes each algorithm to converge proves that the GA is much more efficient for slower objective functions. This is a very important decision variable when choosing between GA and PSO since the objective function will need to be solved multiple times per iteration of the motor optimization problem. If the data in the plot were extrapolated to seconds or even minutes in duration, then the difference in solver execution time between GA and PSO would be much more apparent. In summary, the GA is chosen as the optimal optimization algorithm for the Schwefel test function which will act as the foundation for the solver in the motor optimization problem.

1.4. NSGAII Configuration

Without modification, the GA cannot optimize multi-objective problems and requires a modified implementation that produces non-dominated solutions. The non-dominated sorting genetic algorithm II [NSGAII] is a modified implementation of the GA which will be implemented for the motor optimization problem. There are many core functionalities that are required for the NSGAII to successfully navigate a problem's constrained space and optimize towards a solution. This is no simple task and a misconfiguration of just one core function can result in an instable solver. The classification of NSGAII's core functionality can be segregated into selection of dominant parents and variation for searching the domain in a robust fashion.

1.5. Selection

Selection is a core solver function that identifies the strongest parents among the population through comparison of performance. This identification process is achieved with a fitness function, which is application specific, coupled with a maximization or minimization definition. Since the population size must remain constant, the weakest parents are removed from the current population and discarded. The remaining parents are then subject to variation which will be discussed in the next section. There are many robust selection algorithms that will find the highest performing parents such as Roulette Wheel and Rank selection, although in this paper the focus will be on Tournament selection. The basic principle is that a sample of parents are selected to compete against one another in a tournament-style comparison of their objective values. The likelihood of a parent being selected is dependent on the selection pressure which is a probabilistic measure of a candidate's likelihood of participation in a tournament. This parameter is an indicator of a solver's ability to converge since higher selection pressure relates to a higher convergence rate.

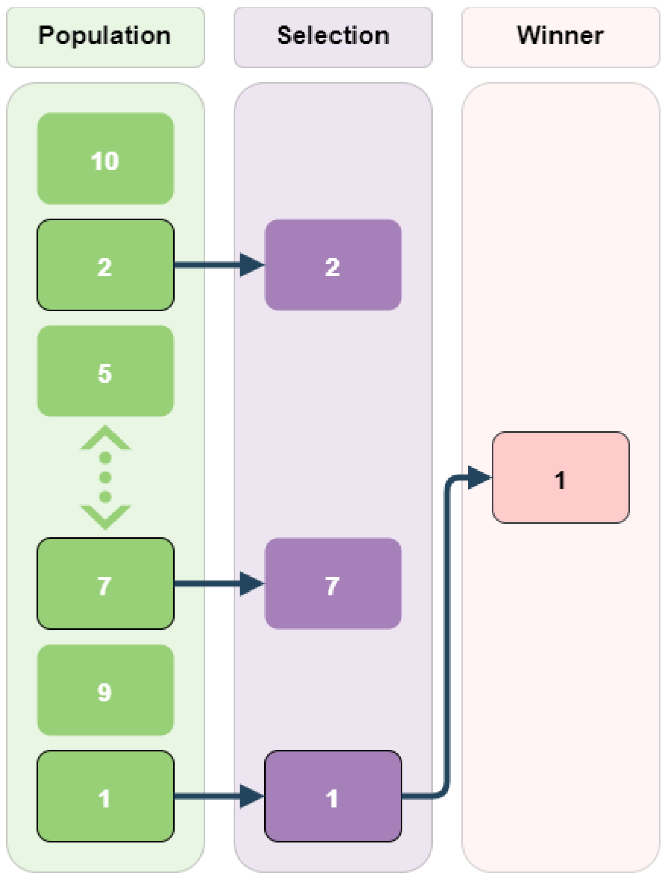


Fig. 1.7. Layout of a Tournament selection algorithm using arbitrary objective values to highlight the winning decision based on a minimization problem.

1.6. Variation

Like real life, the NSGAII has core functions that are appropriately named after events in the natural process of evolution. Crossover is one of these functions. It allows parents to exchange their qualities and produce children while the remaining qualities are subject to some form of randomized initialization. The number of variables that are subject to be overwritten is defined by a crossover point as visualized in figure 1.8. Note that the values of the variables were limited to binary for simplicity, but the true values can contain any format such as integers and real numbers. Since the crossover point determines the percentage of variables shared among parents, it is important to not choose too small or large of a ratio due to solver robustness. If a small percentage of variables from the parents were crossed over then the solver may become stuck in local minima or maxima rather than the desired global alternative. Alternatively, a large percentage of variables crossed over between parents will have large variations in the solution and can cause an instability in the solver.

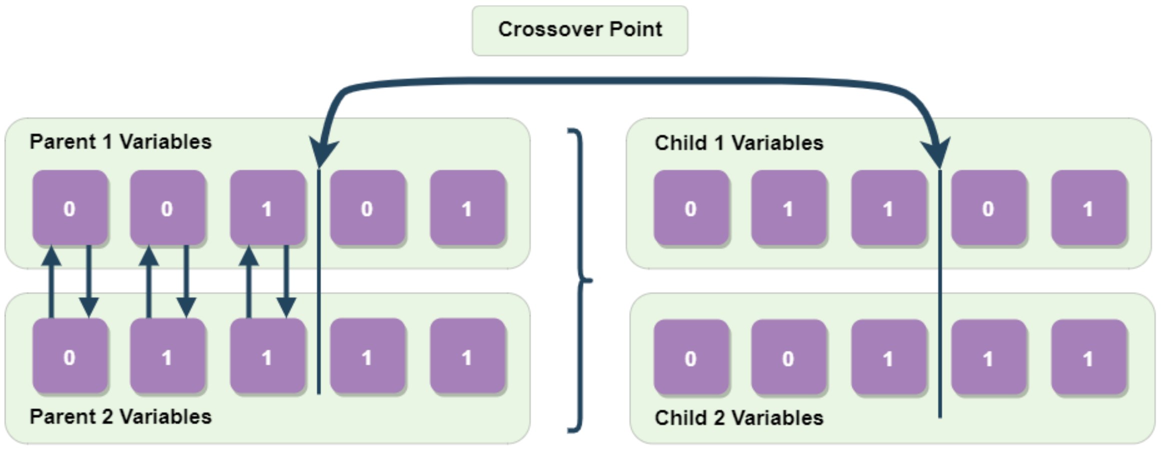


Fig. 1.8. Visualization of crossover between two parent variables to produce two child variables governed by the crossover point.

The frequency that the crossover is applied is also an important configuration consideration. This is defined as the probability that crossover will occur between parents and is integral in the solver's robustness. Like the crossover point, if the probability of crossover is set too high then the parents will often share variables when producing children which is susceptible to finding local minima or maxima rather than the desired global alternative. Contrasting this with a low probability of crossover between parents and the solver may become unstable. This is due to the children's variable initialization relying on some form of randomized initialization which will resist solver convergence.

Mutation is another important function of the EAs which is responsible for manipulating the values of randomly selected variables within a parent. The probability for mutating a parent's variables shall remain low to maintain solver robustness rather than introducing instability. The general concept of mutation is visualized in figure 1.9, which highlights the variables that were randomly selected for mutation within the parent.

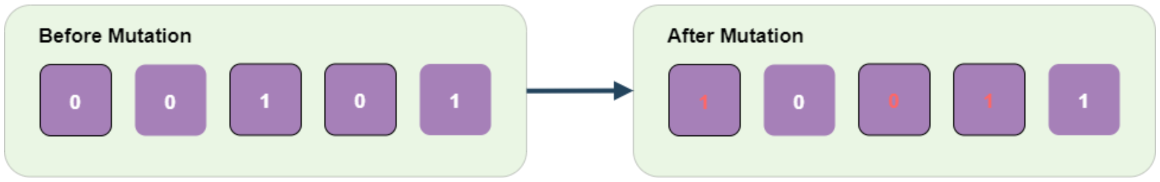


Fig. 1.9. Layout of a genetic algorithm with an arbitrary number of chromosomes and genes per population.

Note that the values of the variables were limited to binary for simplicity, but the true values can contain any format such as integers and real numbers.