CLEANING PROCESS OPTIMIZATION

Adarsh Krishnamurthy

Wei Li

INDEX

INTRODUCTION

Components that are manufactured by sand casting have to be cleaned after they are cast to remove sand particles that are embedded on the surface of the components. This cleaning process is essential for critical parts like engine blocks, which should not have any foreign debris in them. There are usually different methods used for cleaning cast components depending on their size. Smaller components are usually tumble cleaned by rotating the components in a tumbler. The mechanical vibrations and collisions remove the embedded sand particles. On the other hand, big components like engine blocks cannot be cleaned by this method. They are usually cleaned by blasting them with high-pressure water jet. The water removes all the embedded sand particles and does not harm the surface, as the pressure involved is low for actual part erosion. Figure 1 shows a cast component before and after cleaning. It is essential that the sand particles be removed from the surface of such component to prevent the premature failure of the component.

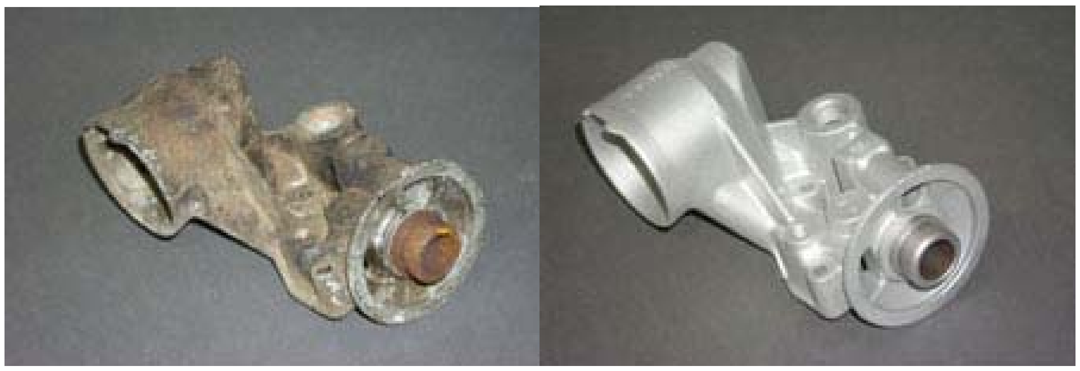


Figure 1: Cast component before and after cleaning

The radius of the water jet nozzle used for the cleaning process is much smaller than the size of the part to be cleaned. Hence, these nozzles are rotated while being simultaneously moved along the surface of the part. Rotating the nozzles increases the area covered by the nozzles or the water accessible regions. This makes the actual cleaning process using water jet complex. Figure 2 shows the schematic of cleaning of a complex part using rotating water jet. The waterjet is rotated as well as moved simultaneously above a complex part and parts cleaned are shown as dots on the surface of the part.

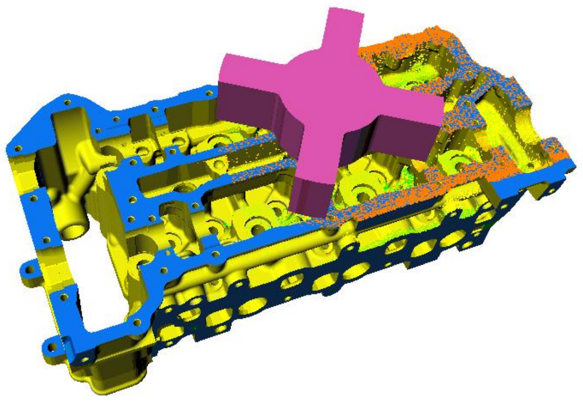


Figure 2: Schematic of the actual cleaning process

SIMPLIFIED PROCESS DESCRIPTION

Since the actual process is too complex to model, we simplified the process so that it can be optimized in a smaller design space. It involves moving single non-rotating waterjet nozzle over the surface of the part to be cleaned. Various parameters like the water pressure, standoff distance, angle of attack etc. affect the final cleaning effectiveness. We also considered only a flat square plate of a fixed dimension for cleaning.

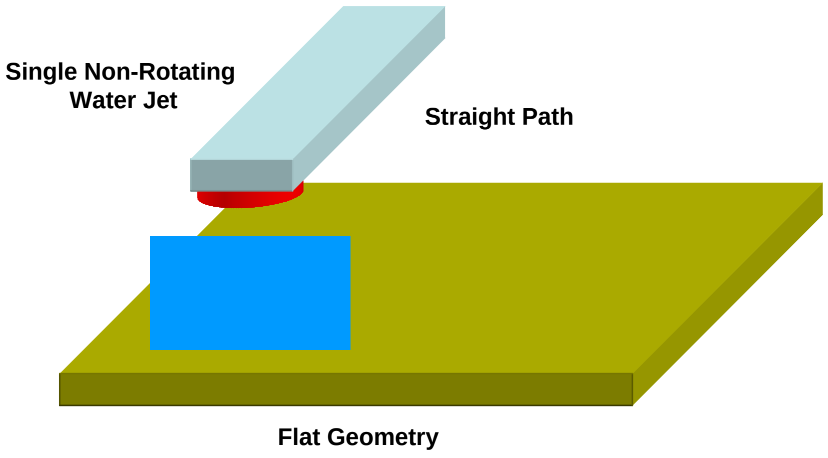


Figure 3: Simplified cleaning process

The simplified cleaning process is shown in Figure 3. It shows a single nozzle which is positioned at a particular standoff distance from a flat surface. The water jet is moved in a straight path in discrete steps to simulate the motion of the nozzle. In addition, the angle of attack as well as the radius of the nozzle can be changed to optimize the process.

Simulation Model

The water from the water jet is approximated by a set of rays as shown in Figure 4. The rays originate from the nozzle and have a pressure distribution that decays along the radial direction away from the center of the nozzle. The impact pressure is calculated at the points on the plate at which the rays hit the surface. Then based on the impact pressure, a quantity called the cleaning effectiveness is calculated.

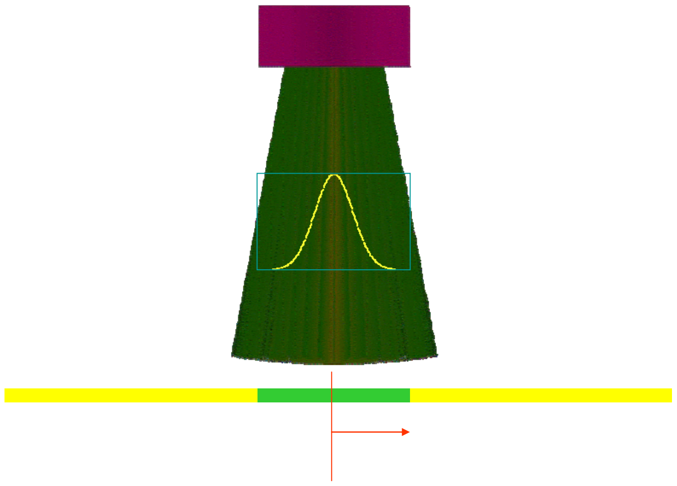


Figure 4: Waterjet approximated as a set of rays

A model for calculating the pressure distribution of a stationary waterjet [1] is established using equation(1). The nomenclature for the equation is listed in Table 1. This equation gives the impact pressure of the waterjet at any point on the surface of the plate.

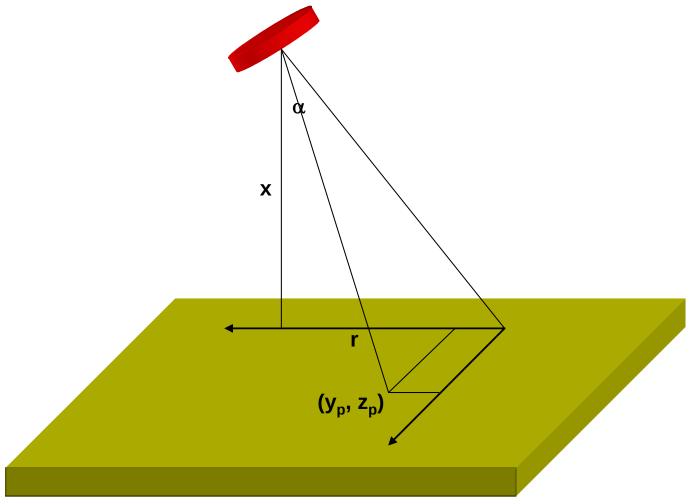


Figure 5: Waterjet model

$$P = 7.95\lambda k\psi\sqrt{P_0\rho}\left(\frac{r_0}{Cx}\right)^2\left[1-\left(\frac{r}{Cx}\right)^{1.5}\right]^3 \tag{1}$$

Table 1: Nomenclature

P	Impact pressure
λ	Stress coefficient
k	Flow resistance coefficient of water system
ψ	Sound speed in water
P_0	Water pressure from the nozzle
ρ	Density of water
r_0	Radius of nozzle
r	Distance of the point of consideration from the waterjet centerline
C	Jet spreading coefficient
x	Standoff distance

An example of the pressure distribution is shown in the figure below. In this case, the nozzle has an attack angle of 45°.

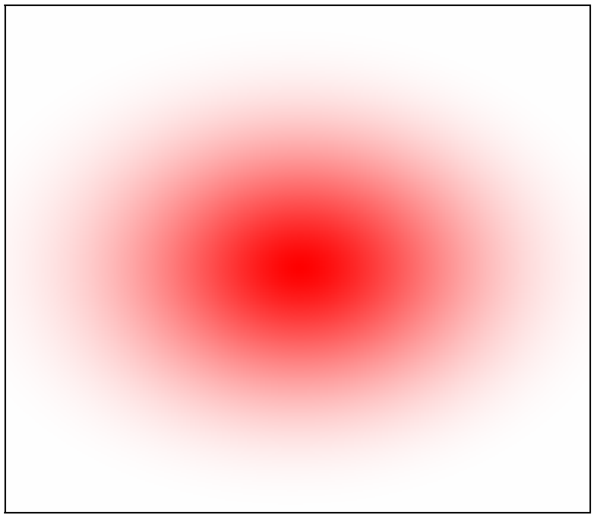


Figure 6: Pressure distribution on a flat surface due to a waterjet at 45° angle

Cleaning Effectiveness

The ability of the water jet for cleaning the surface is correlated directly to the impact pressure of the waterjet on the surface. Higher impact pressures correlate to higher cleaning effectiveness. We use a cumulative normal distribution to correlate the pressure of the water jet with cleaning effectiveness as shown in Figure 7.

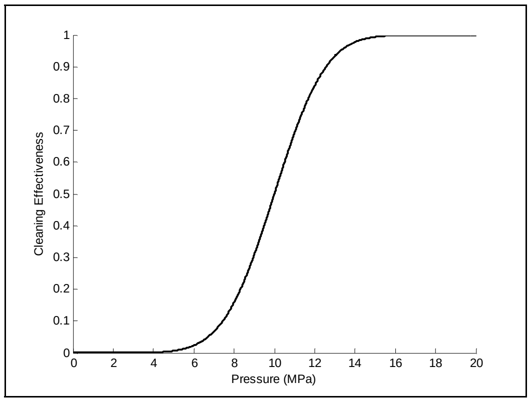


Figure 7: Correlation between impact pressure and cleaning effectiveness

The simulation program calculates impact pressure on the surface due to all the positions of the nozzle on a set of discrete but sufficiently dense points on the flat surface. Based on this impact pressure, the cleaning effectiveness is calculated. Figure 8 shows the simulation of the waterjet on a flat plate. The colors on the plate indicate the cleaning effectiveness at a particular point.

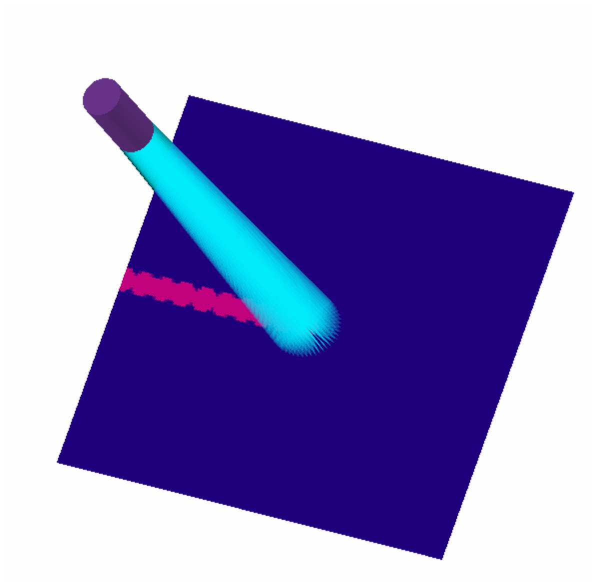


Figure 8: Simulation of a moving nozzle over a flat surface

PROBLEM FORMULATION

We have a two-objective optimization problem. The first objective is to maximize the sum of the average cleaning effectiveness, averaged over the set of sample points *S* on the surface for all nozzle positions, *N* and the coverage fraction, which is the fraction of the total area accessible by water. This is given in equation (2) and is obtained as an output of the simulations, where C is the coverage fraction and e is the effectiveness calculated at each sample point of *S*.

$$E = \sum_N \int e\,dA + \int c\,dA \approx \frac{1}{S}\sum_S \sum_N e + \frac{C}{S} \tag{2}$$

The second objective function is to minimize the number of positions of the waterjet, which directly corresponds to the feed rate of movement of the nozzle. If there are more positions, then the cleaning effectiveness will be high but on the other hand, the time for the process will also be high.

$$t = k_t \cdot n \tag{3}$$

where k_t is the constant time taken for each position and *n* is the number of positions. We assume that the distance moved in each step is constant and is inversely proportional to the number of steps. For the purpose of the simulations, we also assume k_t to be unity.

DESIGN VARIABLES AND PARAMETERS

We are optimizing the cleaning process for the standoff distance (*x*), angle of attack (*α*), nozzle-radius (r_0) and the number of steps (*n*). The number of steps is considered as a discrete variable as it can take only integer values. On the other hand, even though the available nozzle sizes are discrete, we can approximate it as a continuous variable to simplify the optimization. It can then be rounded off to the nearest available nozzle size after the optimization is completed.

The simulation model also takes in some fixed parameters. These parameters include the waterjet pressure P_0, jet-spreading coefficient *C* and flow resistance factor *k*. The model also needs some constants like density of water and speed of sound for the simulations.

CONSTRAINTS

One of the main constraints for the simulation is that the impact pressure should not be very high that it damages the surface of the part being cleaned. There are some physical constraints on the maximum and minimum standoff distances. In addition, the total time or the number of steps cannot exceed a particular threshold. Similarly, there are some constraints on the angle of attack in the sense that it cannot be too obtuse.

OPTIMIZATION

MULTI-OBJECTIVE FORMULATION

The complete formulation for the multi-objective formulation is given below. Most of the constraints except the pressure constraint are direct constraints on the design variables.

Maximize E (Cleaning effect given in equation (2))
Minimize t (Cleaning time given by equation (3))

subject to constraints

$P \leq P_{max}$	Maximum pressure
$n \in \{1,50\}$	Number of positions of the nozzle
$x \leq 0.1$	Maximum standoff distance
$-x \geq 0.05$	Minimum standoff distance
$R_0 \leq 0.008$	Maximum nozzle radius
$-R_0 \leq 0.0001$	Minimum nozzle radius
$\alpha \leq 45$	Maximum attack angle
$-\alpha \leq 0$	Minimum attack angle

SINGLE-OBJECTIVE FORMULATION

Since we wanted to generate the Pareto frontier for the multi-objective formulation, we decided to implement it as a single-objective formulation using the ε-constraint method. We used the second objective as a constraint and optimized the first objective. The complete formulation is given below. The second constraint directly gives the value for the number of steps for the simulations and it is taken as a constant during single objective optimization.

Maximize E (Cleaning effect given in equation (2))

subject to constraints

$P \leq P_{max}$	Maximum pressure
$t = \varepsilon$	Total time for cleaning.
$x \leq 0.1$	Maximum standoff distance
$-x \geq 0.05$	Minimum standoff distance
$R_0 \leq 0.008$	Maximum nozzle radius
$-R_0 \leq 0.0001$	Minimum nozzle radius
$\alpha \leq 45$	Maximum attack angle
$-\alpha \leq 0$	Minimum attack angle

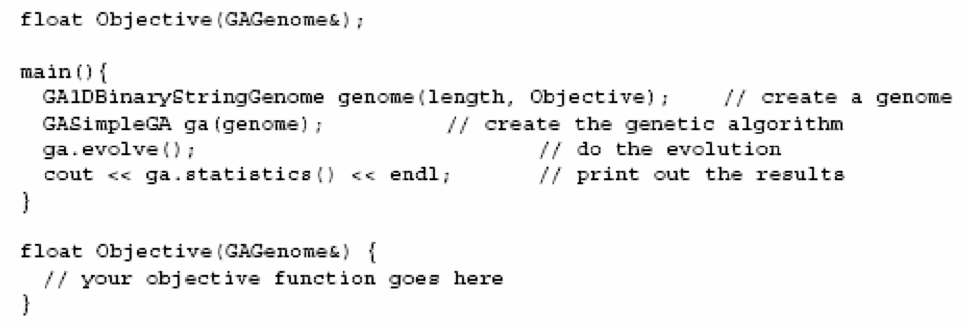
OPTIMIZATION METHOD

From Equation (1) and (2), we can see that the objective function for the optimization problem is a complex and nonlinear one. Since the form of the objective function does not have an explicit expression, a black-box optimization technique is required to solve it. We decided to use Genetic Algorithm (GA) because of the following reasons.

1. GA does not require gradient information. Because of the complexity of our objective function, it is difficult to get its gradient information.
2. GA can handle both continuous and discrete variables simultaneously. In our problem, the step number is a discrete variable, but attack angle, nozzle radius and standoff distance are continuous variables.
3. Most of the constraints in our problem are very simple – they are just boundary limits. This overcomes the difficulty of handling constraints in GA.
4. There are many free GA libraries available.

IMPLEMENTATION

We used a C++ genetic algorithm library – GAlib [3] for the implementation. The library includes tools for using GA to do optimization in any C++ program using user-defined genetic operators. We chose GAlib because it is written by C++, which is object oriented and hence can be seamlessly integrated with our simulation program. A simple optimization program using GAlib can be as short as three lines of codes and a user-defined objective function as shown in Figure 9. Furthermore, GAlib can handle simple constraints such as boundary limits, and supports continuous and discrete design variables, which make it ideal for our optimization problem.



```
float Objective(GAGenome&);

main(){
  GA1DBinaryStringGenome genome(length, Objective);    // create a genome
  GASimpleGA ga(genome);         // create the genetic algorithm
  ga.evolve();                            // do the evolution
  cout << ga.statistics() << endl;       // print out the results
}

float Objective(GAGenome&) {
  // your objective function goes here
}
```

Figure 9: Simple GA optimization program using GAlib

The general workflow of GAlib to solve an optimization problem is shown in Figure 10. Some of the important classes in GAlib are explained below.

- **GAGeneticAlgorithm** represents a genetic algorithm. Four different flavors of genetic algorithms based on how new population is created are supported: the standard simple genetic algorithm, steady-state genetic algorithm, incremental genetic algorithm and deme genetic algorithm.
- **GAPopulation** represents a population, which is a container for the genomes. Each population contains a scaling object that is used to determine the fitness of its genomes.
- **GAGenome** represents a genome (chromosome). Three different data types of genomes are supported: binary, string, and float. In addition, three different configurations of genomes are supported: array, tree and list. Use can customize the crossover and mutator for genomes.

- **GAAlleleSet** represents a container for the different values that a gene may assume. An allele set may be enumerated, unbounded, bounded or bounded with discretization. By using GAAlleleSets, we can define simple constraints in optimization problems.

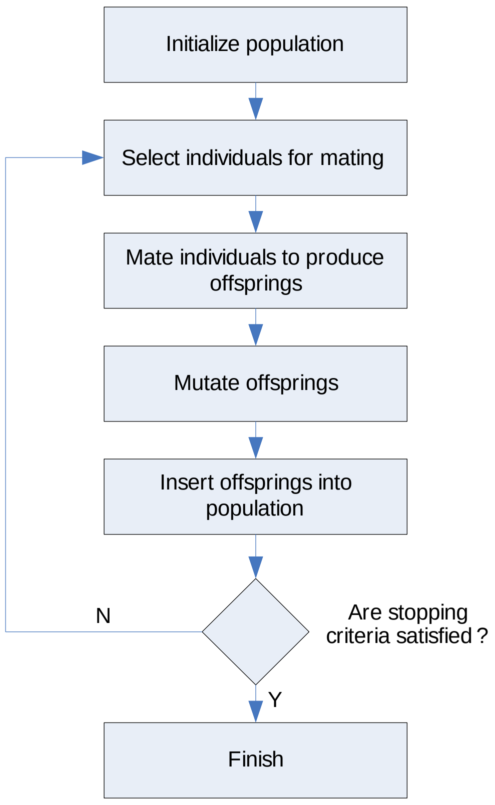

Initialize population
Select individuals for mating
Mate individuals to produce offsprings
Mutate offsprings
Insert offsprings into population
N
Are stopping criteria satisfied ?
Y
Finish

Figure 10: General workflow of GAlib

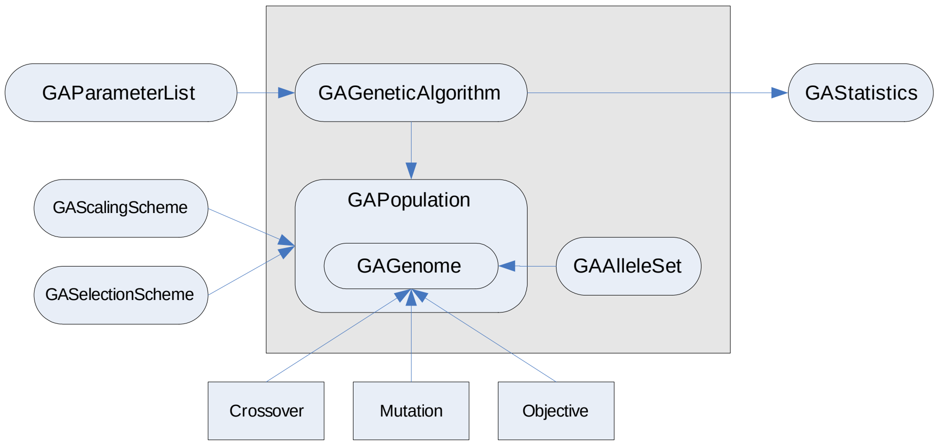


Figure 11: Relationships between major classes in GAlib

The relationships between the major classes of GAlib are shown in Figure 3. They can be categorized into three groups: input (*GAParameterList*, *GAScalingScheme*, *GASelectionScheme*, *Crossover* function, *Mutation* function, and *Objective* function), core (*GAGeneticAlgorithm*, *GAPopulation*, *GAGenome*, and *GAAlleleSet*), and output (*GAStatistics*).

GENETIC ALGORITHM IMPLEMENTATION

In the following section, we discuss some implementation details of using GAlib for the cleanability optimization. We use the "simple genetic algorithm," which is implemented in *GASimpleGA* class in the library. This algorithm uses non-overlapping populations and optional elitism. Each generation the algorithm creates an entirely new population of individuals. The parameters defined for the algorithm are listed in Table 2. We used the following parameters after experimenting with them. The parameters that were chosen finally usually guaranteed convergence.

Table 2: GA parameters

Parameter Name	**Value**
Population Size	100
Number of Generations	200
Mutation Probability	0.001
Crossover Probability	0.9
Generations to Convergence	20
Convergence Percentage	0.99

In this table, *Number of Generations* gives the maximum number of generations until which the GA is evolved. Evolution will stop if the number of generations exceeds this setting. *Generations to Convergence* and *Convergence Percentage* are used when convergence is used as the stop criteria. The *convergence percentage* is defined as the ratio of the *N*th previous best-of-generation score to the current best-of-generation score. *N* is defined by the *Generations to Convergence* parameter.

Objective function

Since the simple genetic algorithm cannot enforce complex constraints, the constraints were incorporated in the fitness function itself. We tried different fitness functions and finally used the function given in equation (4) for the optimization.

$$f = \max(e + c - m, 0) \tag{4}$$

where e is the cleaning effectiveness as calculated from equation (2)

c is the coverage effect or the fraction of the area accessible to the water jet

m is the pressure effect which is 1 if $P > P_{max}$ and 0 if $P \leq P_{max}$

The max function is used to keep the objective functions always positive. This was one of the requirements of GAlib as it is not capable of handling negative objective functions.

Genome

We use genomes of the type of 1D-float arrays, which is implemented in the *GARealGenome* class. The size of the array is 4, and each element in the array represents the *attack angle*, *standoff distance*, *nozzle radius* and *step number* respectively. Since currently we do not have any knowledge about the property of the cleaning process, the population is initialized with randomized designs in the feasible region.

Genome Operators

For all the three-genome operators – selection, crossover and mutation, we use the predefined default behaviors of *GARealGenome*. For the selection, the default scheme is RouletteWheel, which is implemented in *GARouletteWheelSelector* class. It looks through the members of the population using a weighted roulette wheel. Likelihood of selection is proportional to the fitness score.

For the crossover, the default scheme is Uniform Crossover, which is implemented in *GARealUniformCrossover* class. Uniform crossover is radically different to 1-point crossover. It randomly takes bits from parents. For each bit, we flip a coin to see if that bit should come from the mother or the father. For the mutation, the default scheme is Gaussian Mutation, which is implemented in *GARealGaussianMutator* class. The Gaussian mutator picks a new value based on a Gaussian distribution around the current value and respects the bounds.

Constraints

By means of *allele set*, we can set up boundary constrains for the design variables. Allele set is implemented in GAAlleleSet. It acts as a container for the different design values that a gene can assume. It may be bounded, unbounded, or discrete. The allele sets we used for the four design variables are listed in Table 3.

Table 3: Allele constraint sets for the design variables

Design Variable	**Allele Set**
Attack Angle	GAAlleleSet(0, 45)
Standoff Distance	GAAlleleSet(0.05, 0.1)
Nozzle Radius	GAAlleleSet(0.0001, 0.008)
Step Number	GAAlleleSet(1, 50, 1)

Fitness Scaling

Given a particular chromosome, the fitness function returns a single numerical "fitness," which is supposed to be proportional to the ability of the individual for mating. It is a possibly transformed rating based on the raw objective score. In our program, we use the default linear scaling scheme, which is implemented in *GALinearScaling*. Objective scores are converted to fitness scores using the relation

$$f = a \bullet obj + b \tag{5}$$

where a and b are calculated based upon the objective scores of the individuals in the population.

Stop Criteria

GAlib supports three different stop criteria: *TerminateUponGeneration*, *TerminateUponConvergence* and *TerminateUponPopConvergence*. *TerminateUponGeneration* compares the current generation to the specified number of generations. *TerminateUponConvergence* compares the current convergence to the specified convergence value. *TerminateUponPopConvergence* compares the population average to the score of the best individual in the population. In our program, we used the first two criteria, and both give the same optimum solution.

RESULTS

Using the ε-constraint method, we calculated the Pareto front for the optimization problem. Figure 12 shows the results of the optimization along with the Pareto front. The scale on y-axis is inverted to better interpret the results. One of the main notable features is that the exact values obtained from the optimization are not smooth and do not form a smooth curve. This can be attributed to numerical errors in the simulation program where the data is either truncated or approximated.

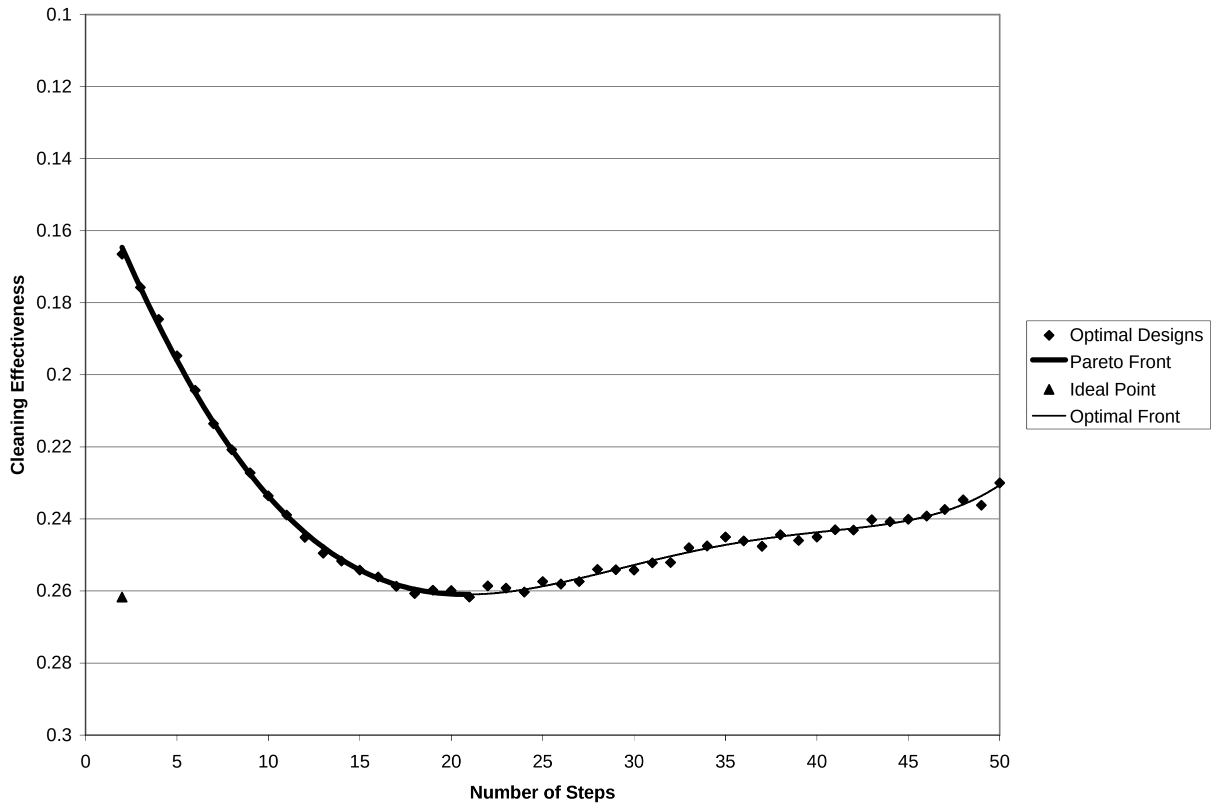


Figure 12: Results of the optimization showing the Pareto front (Note inverted scale on y-axis)

Each point on the above graph was averaged over three different GA runs. Since the simulation was slow, an average of only three runs was taken. The simulation took about an hour for each point in the graph and took approximately 50 hours to construct the complete Pareto front.

Table 4: Pareto optimal design points

Steps	Fitness	Angle	Standoff	Radius	L^2 Norm	L^1 Norm	L^{inf} Norm
2	0.1665	45	100	1.75	0.3173	0.3173	0.3173
3	0.1757	45	100	1.88	0.2874	0.3067	0.2867
4	0.1846	45	100	1.81	0.2601	0.2970	0.2570
5	0.1947	45	100	2.08	0.2313	0.2833	0.2233
6	0.2043	45	100	2.06	0.2074	0.2713	0.1913
7	0.2136	45	100	2.07	0.1890	0.2603	0.1603
8	0.2208	45	100	1.79	0.1816	0.2563	0.1363
9	0.2272	45	100	1.64	0.1812	0.2550	0.1400
10	0.2336	45	100	1.55	0.1854	0.2537	0.1600
11	0.2389	45	100	1.61	0.1954	0.2560	0.1800
12	0.2451	45	100	2.15	0.2075	0.2553	0.2000
13	0.2495	45	100	1.92	0.2237	0.2607	0.2200
14	0.2517	45	100	1.90	0.2423	0.2733	0.2400
15	0.2542	45	100	1.84	0.2612	0.2850	0.2600
16	0.2561	45	100	1.96	0.2806	0.2987	0.2800
17	0.2587	45	100	2.05	0.3002	0.3100	0.3000
18	0.2607	45	100	2.14	0.3200	0.3233	0.3200
19	0.2598	45	100	2.01	0.3401	0.3463	0.3400
20	0.2599	45	100	1.98	0.3600	0.3660	0.3600
21	0.2617	45	100	2.15	0.3800	0.3800	0.3800

Table 4 lists all the Pareto optimal design points. There are different methods to choose a particular design from the above table. We can use the ideal point as a reference and use it to choose the final design based on L^2, L^1 or the L^{inf} Norm.

The ideal point for the optimization is (2, 0.2617). Based on L^2 Norm, the best design obtained is shown in yellow in the above table with number of steps as 9. On the other hand, the best design based on L^1 Norm has 10 steps and the best design based on L^{inf} Norm has 8 steps. The difference in the best optimal design based on these three Norms is not very much. In practice, a best design can be chosen based on some other criteria also.

POST-OPTIMIZATION ANALYSIS

We performed some post-optimization analysis on the results to verify that the solutions obtained are accurate.

OPTIMALITY CONDITIONS

To verify whether the particular solution obtained was optimal, we used Monte-Carlo simulations. We give the details of the simulation for one of the cases with the optimal values of nozzle radius, standoff distance and angle of attack being 1.98 mm, 100 mm and 45° respectively. The number of steps was fixed at 20 for this simulation. The optimal value for the cleaning effectiveness for this design was 0.2599.

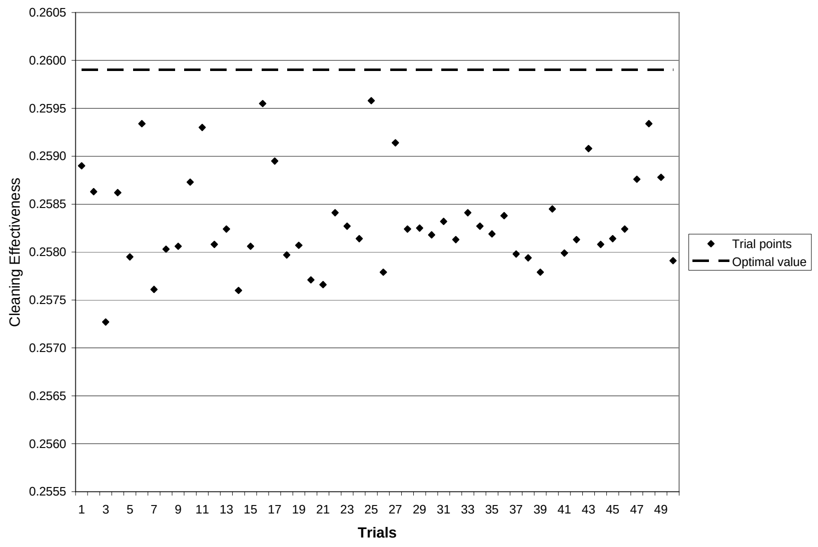


Figure 13: Plot of different trials of the Monte-Carlo simulation

Figure 13 shows the result of the value of different cleaning effectiveness values for 50 random designs. It was verified that the design obtained from GA is the best design in the neighborhood as the values as the cleaning effectiveness for all the other designs were less than that of the optimal design.

This analysis only proves that the design obtained is a local maximum. There can be still some other design that could be a global maximum. However, since the GA was initially seeded with random values, the probability of such a case existing is very less. Moreover, several different GA runs also converged to the same design. Hence, we can safely assume this is a global maximum.

SENSITIVITY ANALYSIS

The parameters for the cleanability optimization problem include the properties of water, surface size, surface material and water pressure at the nozzle. We analyzed the sensitivity of three design variables (standoff distance, attack angle and nozzle radius) with respect to the water pressure. The default water pressure used was 200 MPa. We solved a series of optimization problems under different given water pressures that ranged from 90% to 110% of the default value, and tracked the change in design variables with respect to the water pressure.

The analysis results show that when water pressure changes, the best standoff distance and attack angle remains unchanged (the maximum feasible value), and the nozzle radius decreases linearly. The sensitivity of the optimum design variables to the water pressure are given below.

$$\begin{aligned} &\frac{\partial \alpha}{\partial P_0} = 0 \\ &\frac{\partial x}{\partial P_0} = 0 \\ &\frac{\partial r_0}{\partial P_0} = -0.003\text{mm/MPa} \end{aligned} \tag{6}$$

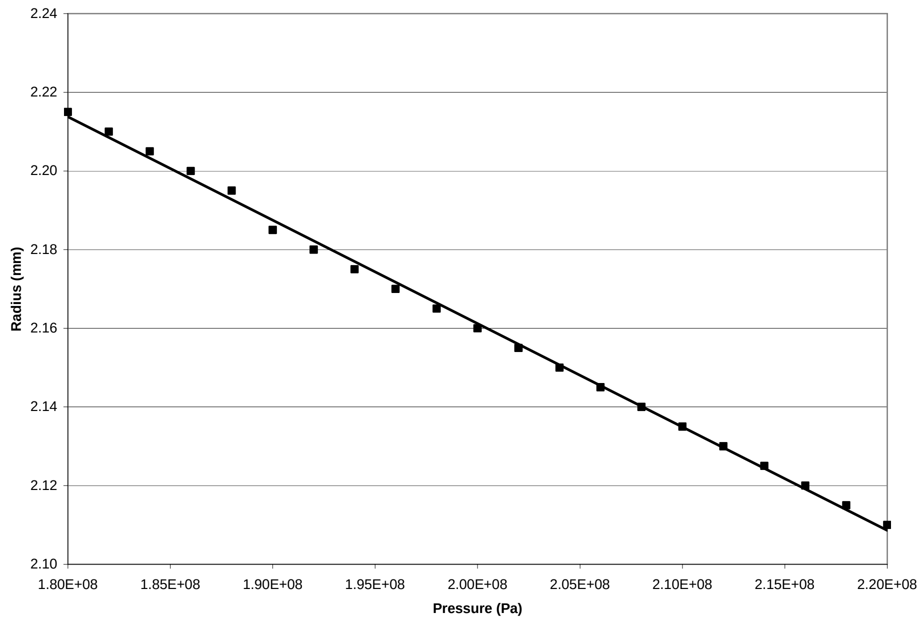


Figure 14: Variation of optimal nozzle radius with pressure

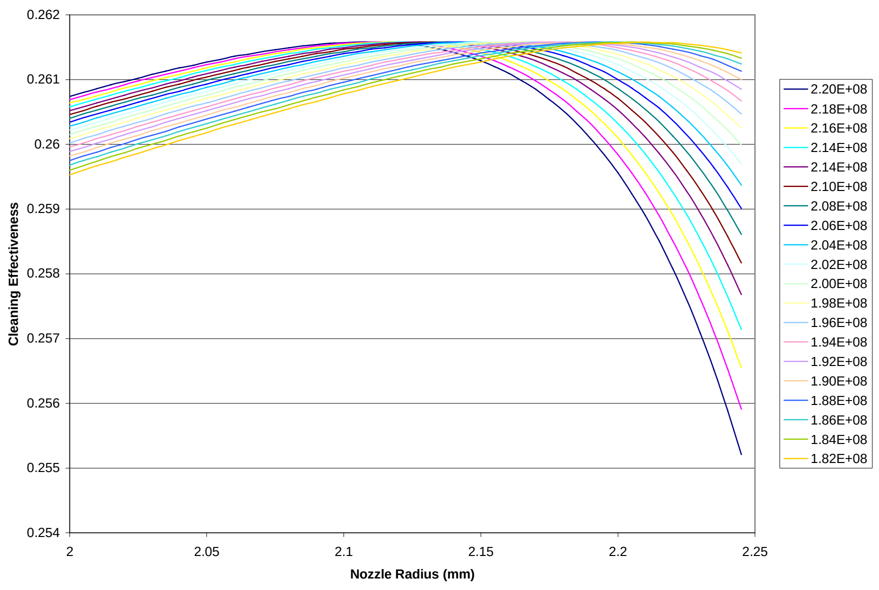


Figure 15: Variation of cleaning effectiveness with nozzle radius for different pressures

Figure 14 shows the variation of the optimal nozzle radius with respect to water pressure. As you can see the optimal nozzle radius decreases with increase in water pressure. The cleaning effect under different water pressure and different nozzle radii is shown in Figure 15. The optimal cleaning effect does not change with change with water pressure. However, the cleaning effect for a particular nozzle radius either increases or decreases depending on weather the particular nozzle radius is bigger or smaller than 2.15 mm respectively.

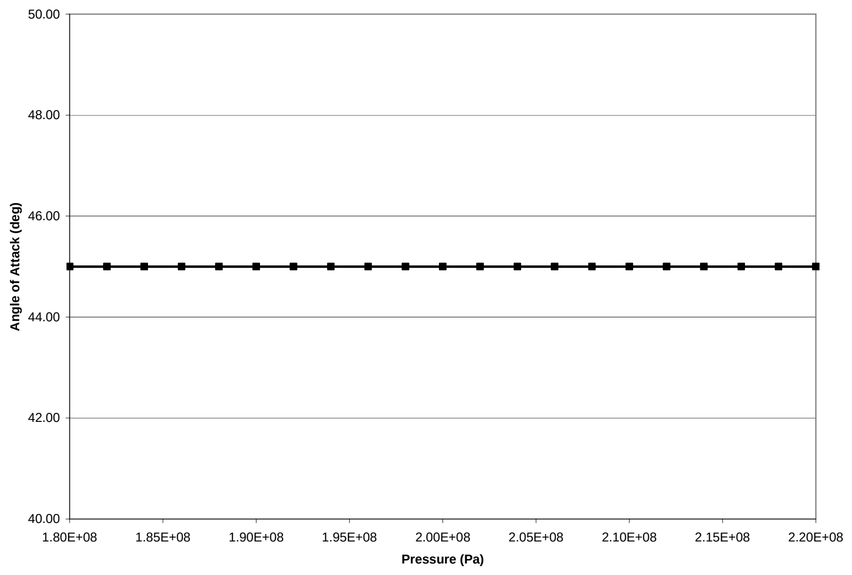


Figure 16: Optimum angle of attack for different pressures

Figure 16 and Figure 17 show that the water pressure does not affect the optimal standoff distance or the angle of attack. This is because these values are at edges of constraints and hence their optimum values do not change with the water pressure.

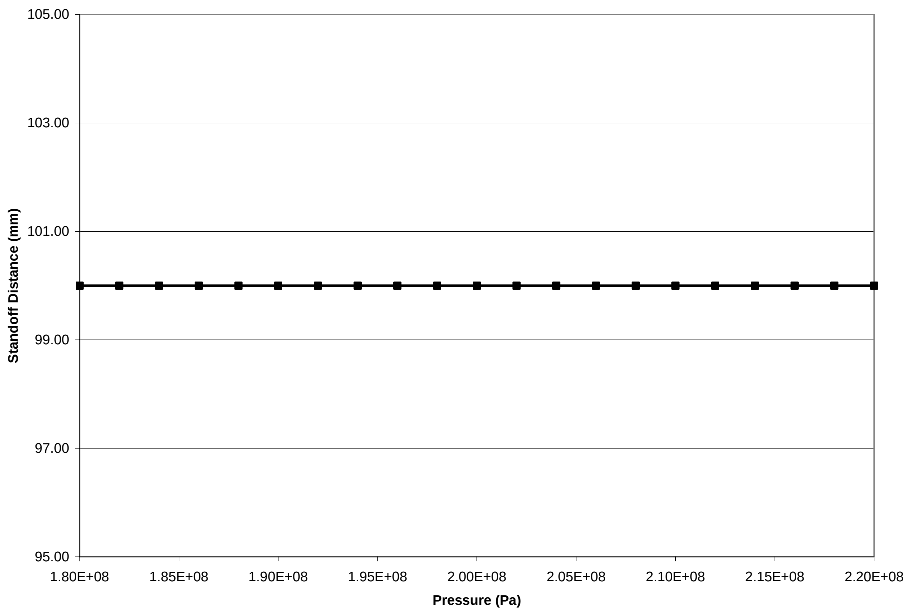


Figure 17: Optimum standoff distance for different pressures

CONCLUSIONS

A simplified cleaning process was optimized in a reduced design space using Genetic Algorithms. Analysis of the results indicate that the obtained solution is a really an optimum. The sensitivity of the optimum design with respect to one of the input parameters was obtained. Similarly, the sensitivity of the optimal design with respect to other parameters can also be obtained.

The simulation used for evaluating the fitness functions is computationally intensive. Moreover, the solutions obtained using GA were not exactly accurate due to numerical errors in the simulation. We can perform an error analysis on the simulation and can give bounds to the errors. Then the step size for different parameters in the GA algorithm can be correctly chosen. Currently we use the default step size for the design variables, which is not the best method, as it is not required to vary the design variables in a finer step than the error in the simulation.

Finally, the GA algorithm has to be fine-tuned to obtain accurate convergence. The parameters of the GA itself have to be chosen properly to make sure that the GA converges to the correct design fast. This is especially important in problem similar to the one in this project where the computation cost in evaluating the objective function is very high. Moreover, the running time also depends on the implementation of GA. The algorithm should be intelligent enough to reuse old data and should not re-compute the data.

REFERENCES

1. M.C. Leu. P. Meng, E.S. Geskin, L. Tismeneskiy, "*Mathematical Modeling and Experimental Verification of Stationary Waterjet Cleaning Process*," Journal of Manufacturing Science and Engineering, 1998, Vol 120, pp. 571-579.
2. George S. Springer, "*Erosion by Liquid Impact*," 1976, John Wiley
3. http://lancet.mit.edu/ga/