A Genetic Algorithm Approach to Facility Layout Planning

James Houghton and Sarah Rheault

Industrial and Systems Engineering Independent Study, Spring 2017

Table of Contents

1. Introduction and Background

The facility layout problem is a fundamental problem in manufacturing engineering and operations research. The problem seeks a solution which defines the positions of machines or departments within the facility such that material handling costs are minimized. An optimal solution to this problem will be invaluable to the company it serves as it will reduce time wasted in material handling between departments, thereby increasing the efficiency of the operation, reducing cycle time, and increasing throughput. There are several methodologies that have been developed to find optimal solutions for various forms of the facility layout problem, including Muther's Systematic Layout Problem, the Quadratic Assignment Model, and Genetic Algorithms.

This report will briefly introduce each of the aforementioned approaches to solving a facility layout problem, and then describe and evaluate the effectiveness proposed approach using a numerical example. The performance of the proposed approach will be evaluated in comparison to benchmarks established in El-Baz (2003) and Mak et al. (2003).

1.1 Muther's Systematic Layout Problem

An early, fairly primitive approach for allocating space in facility is Muther's Systematic Layout Planning (MSLP). This method assigns categories of importance for adjacency for every department. Ranking is based on a "Proximity Ratio" determined by the number of interactions, or flow, between departments. Departments who interact very frequently receive the highest preference, "A", meaning that their adjacency is "Absolutely Essential". Departments with the least interaction are labeled "X" meaning their adjacency is not desirable. A block diagram can be built based on the relationships described in MSLP. This system describes subgroups that should be located near each other in the overall layout. Next, the practical limitations and necessary modifications are considered to develop a feasible layout. The layout is evaluated and the process reiterates until a final design is reached. A major limitation of this model is that it is heavily influenced by the user's opinions. All relationships in each subcategory are treated equally when created space relationship diagrams. Manually choosing the location of each department is slow. In addition, there no way to objectively evaluate the quality of different layout configurations. To do so, a Quadratic Assignment model must be used.

1.2 Quadratic Assignment Model

A mathematical based, *Quadratic Assignment*, model was developed for layout problems of a larger scale than that for which MSLP could be implemented. This model quantifies employee and material traffic as "flow" between departments. A distance matrix between each available location is also constructed. The model's objective function calculates the total flow x distance when assigning locations for each department. Equation 1 shows a general model in the canonical form:

$$\text{Min } z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{jh} f_{jk} X_{ij} X_{hj}$$

$$\textit{Subject to}:$$

$$\sum_{i=1}^{n} X_{ij} = 1 \; for \; j = 1, 2, \ldots, n$$

$$\sum_{j=1}^{n} X_{ij} = 1 \; for \; i = 1, 2, \ldots, n$$

$$X_{ij} \in (0,1) \; for \; i, j = 1, 2, \ldots, n$$

$$\text{Where } d_{jh} \in \text{D distance matrix and } f_{jk} \in \text{F flow matrix}$$

Equation 1: Quadratic Assignment Model

The variables d_{jh} and f_{jk} capture information about the distance between locations and the flow between departments. X_{ij} and X_{hj} are binary and control the decision of where each department will be located. Implementing a Quadratic Assignment model uses the same input as MSLP regarding flow. Parallel implementation of these methods can be used efficiently when solving relatively small problems. As number of decision variables increases the solution space grows exponentially. Thus, a more powerful approach is required. One such approach for efficiently solving problems of a larger scale is implementation of a Genetic Algorithm.

2. Genetic Algorithm

A Genetic Algorithm is a type of Evolutionary Algorithm that can be used to solve optimization and search problems using methods inspired by evolution and natural selection. The solution space evolves to yield progressively more optimal solutions throughout generations.

2.1 Basic Methodology/Approach

The vocabulary used when discussing Genetic Algorithm is similar to that used in describing natural selection and genetics. The “population” is the set of feasible solutions or “individuals” that are combined with one another and mutated to increase the “fitness” of the overall population. The fitness of an individual is determined by a method similar to that used in the quadratic assignment model, according to the absolute position of each chromosome, as well as its position relative to others. Feasible solutions are enumerated by a sequence of numbers to form a string. These numbers are considered the “chromosomes” that make up a solution’s Genome.

The first step for implementing a Genetic Algorithm for Layout Planning is to define several basic parameters that will control its behavior. These include the population size, maximum number of generations, and probability of mutation. The population size will determine how many solutions will be evaluated in each iteration. The maximum number of generations will provide a stopping point for the algorithm. The probability of mutation will control how new solutions will be generated in the next iteration. Additionally, the rectilinear distance between departments and the flow between departments must be defined so that the fitness of each individual may be determined. The flowchart in Figure 1 provides a general overview of the algorithm's operating logic.

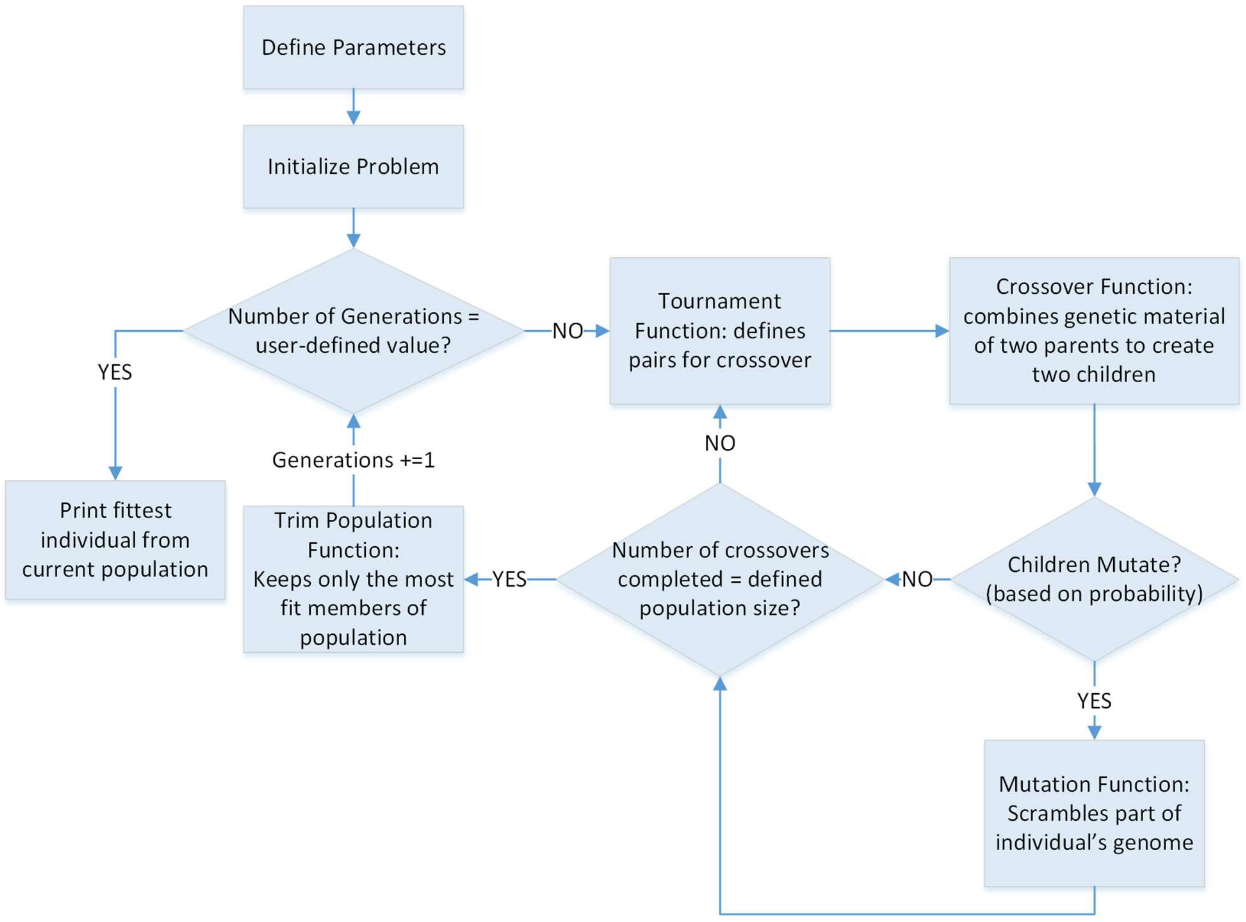
Define Parameters
Initialize Problem
Number of Generations = user-defined value?
YES
NO
Tournament Function: defines pairs for crossover
Crossover Function: combines genetic material of two parents to create two children
Print fittest individual from current population
Generations +=1
NO
Trim Population Function: Keeps only the most fit members of population
YES
Number of crossovers completed = defined population size?
NO
Children Mutate? (based on probability)
YES
Mutation Function: Scrambles part of individual's genome

Figure 1: Flowchart of Genetic Algorithm Logic

2.2 Enumerating the Problem

Using a Genetic Algorithm requires that all possible solutions are represented by a string of numbers or characters. The most enumeration can be done when the number of departments and available locations are equal. For these types of problems, the size of each department is assumed to be equal. The distances between each available location are measured rectilinearly. The distance matrix corresponding to a problem with 6 locations will be encoded as followed:

Table 1: Distance matrix for a 6-location layout problem

	Location 1	Location 2	Location 3	Location 4	Location 5	Location 6
Location 1	0	1	2	1	2	3
Location 2	1	0	1	2	2	3
Location 3	3	2	0	3	3	2
Location 4	2	2	3	0	1	2
Location 5	3	2	2	1	0	1
Location 6	3	2	1	2	1	0

In a Genetic Algorithm each location is assigned a number. These numbers are used to form solution strings. The number in the first position represents the location assignment of the first department. The second position in the string reflects the location of the second department and so on.. Thus, the solution string “413652” corresponds to the layout shown below.

Table 2: Layout represented by solution "413652"

Department 2	Department 6	Department 3
Department 1	Department 5	Department 4

2.3 Initialize Problem

The problem is initialized by a function that creates the initial population. The initial population is a list of *n* individuals, each individual being a list of the numbers 1-6 in random order, and *n* being the population size as defined by the user. The initial population provides the original "parents" for each subsequent generation.

2.4 Tournament Function

The tournament function is used to pair up members of the population for crossover. The tournament has size *k,* meaning it takes a random sample of *k* individuals from the population, and selects the most fit to be the first parent for the crossover. It then takes a second random sample of *k* individuals, and selects the most fit from that sample to be the second parent for crossover. The value of *k* affects the selection pressure, or the degree to which fitter individuals are favored by the selection method. The higher the value of *k*, the greater the selection pressure, and the more the fitter individuals are favored.

2.5 Crossover Function

The crossover function is the algorithm equivalent of sexual reproduction in nature. The two parents, selected based on their fitness by the tournament function, combine their genetic material to create two children by means of a two-point crossover. The two-point crossover calls for two points to be defined on the parent chromosome strings. The genetic material between those two points is swapped and copied to the two children, and the remaining parental chromosomes are copied to the children as they appear in the parents. Figure 2 illustrates the two-point crossover.

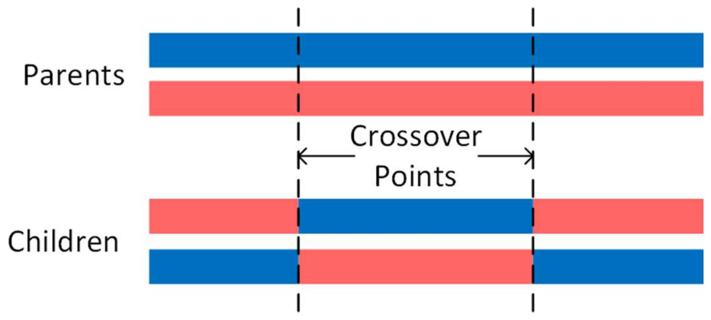

Figure 2: Two-point crossover method

Because the layout problem requires each feasible solution to contain each department number exactly once, the crossover function was slightly modified to prevent department numbers from occurring more than once in each solution. The modification gives priority to the chromosomes between the crossover points, eliminating any duplicate numbers that come from outside of the crossover points, and replacing them with whichever number in the sequence from 1-6 is missing from the child solution.

After the crossover is performed, the children are either added into the rest of the population, or mutated and then added to the population. In each generation, *n* number of crossovers are performed, where *n* is the user-defined population size.

2.6 Mutation

After crossover, child solutions are mutated with probability P(m), a user-defined parameter. A random number between 0 and 1 is generated and compared with P(m) to determine whether a child will be mutated. If the random number is less than or equal to P(m), the mutation function is called. Just as in nature, mutation adds some genetic diversity to the population. Chromosomes or sequences of chromosomes which may not have been introduced to the population if children were created exclusively by crossover can be introduced by mutation.

The mutation function behaves similarly to the crossover function. For mutation, the function defines two points on the child chromosome string. Chromosomes between those two points remain untouched, but chromosomes on either side of the two points are scrambled in place. After a child is mutated, it is added to the population.

2.7 Selection

Each time *n* crossovers are completed, the population is trimmed back down by removing all but the *n* most fit individuals. This ensures that each generation starts with the same number of individuals in the pool of potential parents. After the maximum number of generations has been reached, the most fit individual in the current population is identified and considered the optimal solution.

Both the population trimming and the final selection of the optimal individual use a fitness evaluation function to assign a fitness value to each individual and rank the individuals by their fitness. The fitness evaluation function sorts the user-provided distance matrix according to the order of the individual's chromosomes. For example, the unsorted distance matrix and the matrix sorted for an individual solution of "263154" are shown in Table 3.

Table 3: Unsorted vs. sorted distance matrices

Unsorted distance matrix						
	1	2	3	4	5	6
1	0	802	156	389	249	300
2	802	0	759	482	608	584
3	156	759	0	182	150	401
4	389	482	182	0	179	179
5	249	608	150	179	0	100
6	300	584	401	179	100	0

Matrix sorted for solution "263154"						
	2	6	3	1	5	4
2	0	584	759	802	608	482
6	584	0	401	300	100	179
3	759	401	0	156	150	182
1	802	300	156	0	249	389
5	608	100	150	249	0	179
4	482	179	182	389	179	0

After the distance matrix is sorted, the fitness of the individual is computed by multiplying corresponding entries in the sorted distance matrix and the flow matrix, ($flow_{ij}$ * $sorted\_distance_{ij}$) and summing those products, similar to the method used in the quadratic assignment model. The fitness evaluation function then creates a dictionary containing each solution and its corresponding fitness value. This dictionary can be sorted based on the fitness values and the most fit individual selected or the least fit individuals eliminated.

2.8 Output

After as many generations as were defined by the user have been synthesized, the most fit individual in the remaining population is the optimal solution as found by the algorithm. More generations offer more opportunities for the true optimal solution to be created by crossover or mutation. For process visibility, the program prints a table showing the best solution in each generation, followed by a statement showing the optimal final solution and the fitness of that solution.

3. Numerical Example

The proposed Genetic Algorithm approach was tested using an example taken from M. El-Baz (2003). In this example, the flow between machines and material handling costs are used to find the best placement of nine machines arranged into a 3x3 grid. The distances between machines, flow of materials between each machine, and material handling costs between machines for this example can be found in Table 4.

Table 4: Distance, material handling cost, and flow matrices for El-Baz example

Distance (From/To)	[0,1,2,1,2,3,2,3,4] [1,0,1,2,1,2,4,2,3] [2,1,0,3,2,1,4,3,2] [1,2,3,0,1,2,1,2,4] [2,1,2,1,0,1,2,1,2] [3,2,1,2,1,0,3,2,1] [2,3,4,1,2,3,0,1,2] [3,2,3,2,1,2,1,0,1] [4,3,2,3,2,1,2,1,0]	Material Handling Cost (From/To)	[0,1,2,3,3,4,2,6,7] [0,0,12,4,7,5,8,6,5] [0,0,0,5,9,1,1,1,1] [0,0,0,0,1,1,1,4,6] [0,0,0,0,0,1,1,1,1] [0,0,0,0,0,0,1,4,6] [0,0,0,0,0,0,0,7,1] [0,0,0,0,0,0,0,0,1] [0,0,0,0,0,0,0,0,0]	Flow (From/To)	[0,100,3,0,6,35,190,14,12] [0,0,6,8,109,78,1,1,104] [0,0,0,0,0,17,100,1,31] [0,0,0,0,100,1,247,178,1] [0,0,0,0,0,1,10,1,79] [0,0,0,0,0,0,0,1,0] [0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,12] [0,0,0,0,0,0,0,0,0]

The proposed approach is compared with two other approaches using guidelines proposed by Mak et al., who also evaluated their work with this example. Their work tested 19 different combinations of genetic parameters. Table 6 below shows the levels for Population Size and Generations used for each test. El-Baz considered a cutoff point of 1.5 times the average population fitness when determining the portion of solutions that will remain in the population. Mak et al. carries the top 5% of solutions into the next generation. The Probability of Crossover, P(c) and Probability of Mutation, P(m) used in the proposed approach, El-Baz (2003), Mak et al. (1998) can be found in Table 5.

Table 5: Probabilities of crossover and mutation used by each approach

	Proposed Approach	El-Baz 2003	Mak et al.
P(c)	1	0.9	0.6
P(m)	0.4	0.1	0.001

Table 6: Levels of population size and generations used for tests

Test	Population Size	Generations	Test	Population Size	Generations
1	20	10	11	40	40
2	40	10	12	100	40
3	100	10	13	200	40
4	200	10	14	20	10
5	500	10	15	40	10
6	20	20	16	100	10
7	40	20	17	20	20
8	100	20	18	40	20
9	200	20	19	10	500
10	20	40			

The varying levels of population size and maximum number of generations aimed to determine the best settings for population size and number of generations. In general, increasing population size and number of generations increases the size of the solution space and should yield more optimal results. However, it is desirable to set P and G as low as possible such that the genetic algorithm still reaches an optimal solution, to reduce computation time. Mak et al. used the exhaustive search method to confirm an optimal material handling cost. The 8 possible configurations that will result in an optimal material handling cost of $4818 are shown in in Figure 3 below.

7	1	6	6	2	5	4	3	7	7	3	4
3	9	2	1	9	8	8	9	1	1	9	8
4	8	5	7	3	4	5	2	6	6	2	5
5	8	4	5	2	6	4	8	5	6	1	7
2	9	3	8	9	1	3	9	2	2	9	3
6	1	7	4	3	7	7	1	6	5	8	4

Figure 3: Configurations resulting in optimal material handling cost

Each combination of settings was tested 10 times and the best and average solutions recorded, as well as the number of runs where one of the 8 optimal solutions was reached. The results of the 19 trials are summarized in Table 7, in a comparison between the approaches of El-Baz, Mak et al., and the approach proposed in this paper.

4. Results & Comparisons

Table 7: Results of 19 trials comparing methodology performance

			Proposed Approach			El-Baz			Mak et al.		
Test No.	P	G	Best	Avg	# optimal	Best	Avg	# optimal	Best	Avg	# optimal
1	20	10	4818	4961	2	5039	5310.1	0	5233	5504.4	0
2	40	10	4818	4838	9	4818	5231.9	1	5040	5286.7	0
3	100	10	4818	4824.7	9	4818	4961	2	4818	5024.8	1
4	200	10	4818	4825.3	9	4818	4895.9	5	4818	4891.4	2
5	500	10	4818	4818	10	4818	4822	9	4818	4833.2	7
6	20	20	4818	4862.1	7	4872	5172.9	0	5225	5481.2	0
7	40	20	4818	4818	10	4818	5052	1	4927	5174.6	0
8	100	20	4818	4818	10	4818	4855.2	4	4818	4889.1	4
9	200	20	4818	4818	10	4818	4842.1	6	4818	4846.5	5
10	20	40	4818	4827.3	8	4818	5074.1	2	5225	5462.2	0
11	40	40	4818	4848	5	4818	4979.5	2	4927	5163.8	0
12	100	40	4818	4818	10	4818	4842.8	7	4818	4871.4	4
13	200	40	4818	4818	10	4818	4842.1	6	4818	8440	5

14	20	100	4818	4818	10	4818	4940.9	5	5225	5453	0
15	40	100	4818	4821.2	9	4818	4862.7	6	4818	5141.6	1
16	100	100	4818	4818	10	4818	4826.8	8	4818	4866	5
17	20	200	4818	4818.6	9	4818	4893.6	6	4818	5303.9	1
18	40	200	4818	4818	10	4818	4858.3	7	4818	5141.4	1
19	10	500	4818	4837.6	9	4818	4983.7	4	4818	5184.3	1
Totals					166			81			37

With twice as many total occurrences of the optimal solution as El-Baz's approach, and more than four times as many than in Mak et Al's approach, the approach proposed in this paper is superior to the two others in finding the optimal solution more often. A comparison of the performance of each method at each setting can be seen in Figure 4.

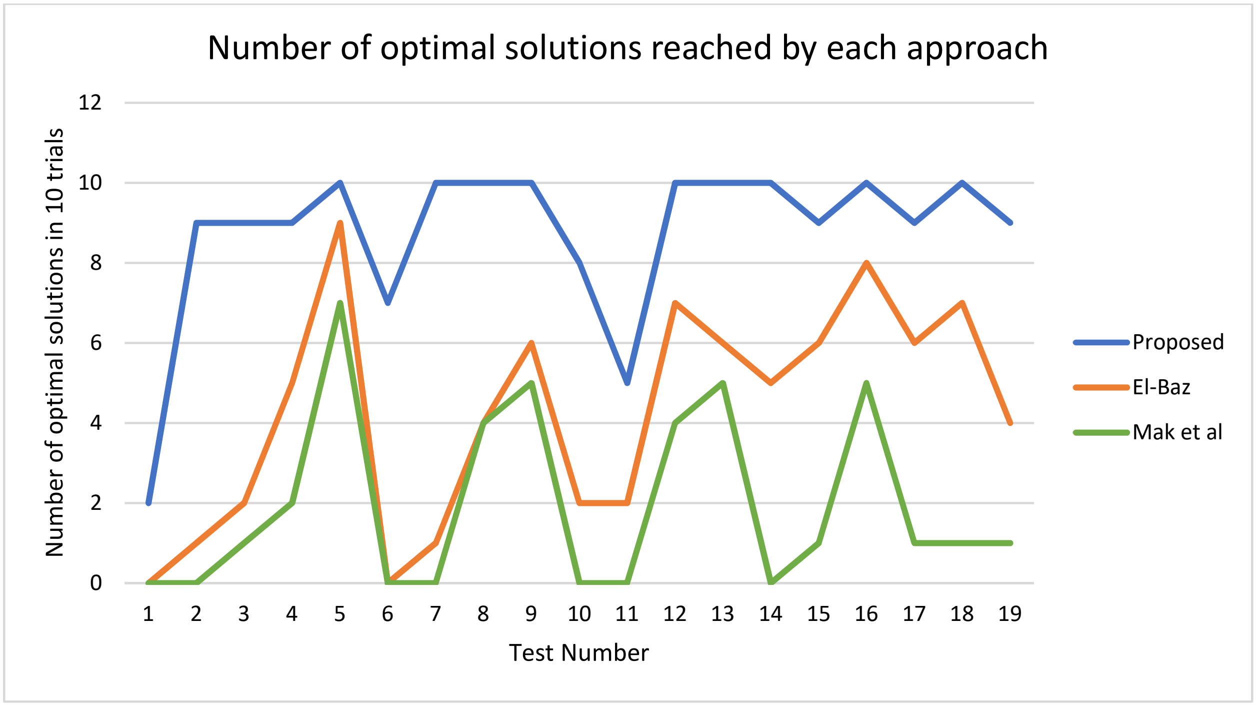

Figure 4: Comparison of methodology performance

These results show that the method proposed in this paper is superior to El-Baz's and Mak et al.'s method in finding more optimal solutions with smaller values of P and G. It can also be seen that there are many combinations for values of P and G for which the genetic algorithm will always or almost always find the optimal solution. For the proposed method, it seems that (P,G) = (40, 20) (trial 7) is a good choice for relatively low settings and high rate of success. The proposed method outperformed both of the alternative methods at every setting of P and G. It can be seen that some combinations of P and G are universally undesirable for every method, namely (P,G) = (20, 20), (40, 40), and (10, 500) (trials 6, 11, and 19).

5. Conclusions

This paper proposes a Genetic Algorithm approach for solving balanced Facility Layout Problems. In all 19 tests the proposed approach obtained an optimal solution more frequently than previous benchmarks set by El-Baz and Mak et al. Based on this comparison, the proposed approach can be used to solve facility layout problems.

6. Future Work

The genetic algorithm can be used to find solutions for countless optimization and search problems with minor adjustments. Any problem whose feasible solutions can be enumerated as strings or lists of numbers can apply a genetic algorithm to find the solution that maximizes or minimizes some value. However, it should be noted that genetic algorithms are rather limited by the complexity of the problem. With very many decision variables, and therefore very long chromosome strings, the size of the search space can increase exponentially. At a certain point, the search space will be too large for the genetic algorithm to be useful. This means that extremely complex problems like the design of an engine, for example, could only be solved by a genetic algorithm if broken into smaller subproblems. For optimization and search problems of average complexity, the genetic algorithm is effective.

One search problem that can be effectively solved using a genetic algorithm is the traveling salesman problem. Similar to the layout problem, a solution to the traveling salesman problem is essentially an assignment. Rather than assigning departments to locations, it would be assigning geographical locations to an order in which they should be visited. For example, a solution to a travelling salesman problem could be “3517426”, meaning that location 3 should be the first stop, location 5 the second, and so on. The application potential for the genetic algorithm as a method for finding optimal solutions to search and optimization problems is extensive.

7. Works Cited

El-Baz, M. A. (2004). A genetic algorithm for facility layout problems of different manufacturing environments. Computers & Industrial Engineering, 47(2-3), 233-246. doi:10.1016/j.cie.2004.07.001