The Genetic Algorithm and Local Optimizer Hybrid Approach for the Advanced Layout Problem

Abstract

Spatial layout problems affect architects, physicists and many other professionals in their attempts to streamline factory designs, find optimal molecular configurations or arrange components for micro-circuitry design. Although genetic algorithms have been used successfully in many of these areas, certain aspects of these problems still represent a major challenge for the GA community. This paper discusses the problems of adding heterogeneous objects, continuous placement and forbidden zones to the general layout problem. The factory floor layout problem is used as an example of the effect and problems of dealing with these extensions. Results are achieved through the use of an iterative local optimizer and a genetic algorithm.

1 Introduction

Spatial layout problems occur in many fields including component placement in micro circuitry design, modeling of theoretical physics problems such as the C60 Bucky Ball [1] and operations research problems like the factory floor layout problem. This paper will review the use of heterogeneous objects, continuous placement and forbidden zones in the general spatial layout problem and the specific instantiation of the factory floor layout problem that uses these extensions.

The Advanced Layout Problem (ALP), defined below, poses significant problems to the factory layout community. Currently layouts are only used as a guide for architects and engineers that are designing factory floor plans. These guides are used to give additional information about the interdependencies of machines that is not very easily seen in a factory design. This information can then be used by the human designers to improve the flow of materials, people and resources through the new plant. Although the ALP is still used mainly as a guide for designers, the ability to deal with extensions such as specific continuous placement, heterogeneous footprint information and forbidden zones make ALP more capable of directly generating practical floor plans that could be used as an advanced starting point for the design specialist.

1.1 Definition of the Advanced Layout Problem

The objective of the Advanced Layout Problem is to position N circular objects with heterogeneous radii into an area of length L and width W such that there are no overlaps between the circles. The distances between all pairs of entities are optimized according to a given weight matrix A. The problem can be more succinctly stated as

$$\sum_{i}^{N}\sum_{j}^{N} A_{i,j} * \parallel X - Y \parallel$$

where the vectors X and Y are encoded in a genetic algorithm and indicate the cartesian coordinates of each individual entity.

The weight matrix A in the above equation is NxN with each entry corresponding to the specific *priority* of relationship between two objects. Each priority is indicated by a particular class type and each class type can be either positive or negative. Negative relationships are those that keep entities apart and positive relationships are those that keep entities together. In the factory floor layout problem, these negative relationships might be due to hazardous chemical interactions, noise pollution to another machine or any such difficulty in keeping two machines in close proximity. Typical positive relationships are due to optimization of material flow between machines in a sequence. Table 1 shows an example matrix for the twelve entity problem.

1.2 Related Problems

Although the Advanced Layout Problem adds to the functionality of the traditional facility layout problem, a brief review of the base problem is necessary. One of the most thorough reviews on the subject of factory

	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11
m2	X										
m3	X	U									
m4	I	X	X								
m5	I	X	X	U							
m6	X	X	U	U	U						
m7	X	I	I	X	X	U					
m8	X	U	U	X	X	U	I				
m9	I	X	X	U	U	U	X	X			
m10	U	U	X	X	U	U	U	U	U		
m11	U	U	U	U	U	I	U	U	U	U	
m12	U	U	U	U	U	U	U	U	U	I	U

Table 1: Weight matrix for twelve entity problem where I = 20, U = 0, X = -20.

layouts is one by Meller and Gau [2] which explains methods such as the Quadratic Assignment Problem, graph theoretic approaches and other methods, as well as reviewing commercial packages.

The Quadratic Assignment Problem (QAP) is the most used method. Classically, the QAP attempts to allocate N objects in N discretely partitioned areas, one object per location. The input relationships are similar to that of the Advanced Layout Problem and the goal is also to optimize these relationships by placing the N objects onto the designated area. Evolutionary methods have been applied to this problem with great success [3][4]. The Advanced Layout Problem, however, places the objects in a continuous space instead of the discrete partitions of the QAP.

The Factory Layout Problem is similar to the Quadratic Assignment Problem but must also find clustering information, that is place groups such as departments, into a typically unrestricted floor plan. This area has been examined in detail and genetic algorithms have been shown to be superior to traditional methods [5].

One of the better Factory Layout Problem algorithms, SPIRAL, attempts to place objects in a graph with arcs extending only in hexagonal directions [6]. The algorithm is much like the ALP in that it is used to search for an optimal layout of machines. SPIRAL, however, has a fixed arc length so that placement is not continuous. SPIRAL also uses an interactive approach and requires additional user post processing. Although the purposes of SPIRAL and the Advance Layout Problem are not the same, an attempt is made in the **Results** section to compare the two methods even though comparisons might be not be fair to either method, and so should be considered only as a general guideline.

2 Methods

A genetic algorithm [7][8] was used in conjunction with a local optimizer to find optimal spatial layouts. A single population was used in both a twelve machine layout and a more complex forty-five machine layout.

2.1 Encoding

One of the main difficulties in encoding this problem lies in the differences of footprints and relationships for individual objects. These differences force each object's location to be specifically identified in the chromosome instead of allowing an encoding which could take advantage of similarities among objects. Because the complexity of a problem is related to its chromosome size and the encoding of the chromosome is related to the number of objects, the complexity increases in proportion to the number of objects for the particular problem.

Many encodings of the objects' locations into the chromosome were tested. These encodings included absolute encoding, where each object had its X and Y position directly encoded on the string, and relative encodings where the X and Y positions could be determined either relative to another object's position or as an absolute value. Overall, however, a simple absolute encoding was found to be the least deceptive for the GA. Our particular encoding used nine bits for each object's X and Y position. More bits might be necessary for higher resolution, but nine bits worked well for the examples included in this paper. These 9 bits were decoded to a number which was scaled to the respective length and width of the area minus the object's radius, so that objects did not extend past the area of confinement. These coordinates were then offset by the object's radius and tested as the position for the center of the object's footprint.

2.2 Local Optimizer

One problem often encountered in genetic algorithm solutions to practical problems is the inability of a GA to easily *finish* a problem, that is find the final solution. GAs are very good at finding relatively good neighborhoods of solution in a complex search space, but not as good at finding the final setting of all the parameters in the chromosome to *the* final solution. One good approach to solve this problem is to use a local optimizer (typically a kind of hill climber) to rate the fitness of a GA generated solution based on what it *would* be given the application of the local optimizer. This allows the GA to get good feedback on solutions without having to completely solve the problem, something the local optimizer can do more easily and more efficiently. We use such an approach in this problem.

The particular optimizer used for this problem uses an iterative technique. For each relationship in the matrix A an amount of error from the desired distance is calculated and weighted according to the relationship matrix. The error is defined as:

$$e = \frac{(Desired - Actual)}{Desired} * \frac{|A_{i,j}|}{MaxWeight}$$

The weighted error is then attributed to each member of the relationship pair by adding the error to the X and Y delta vectors as follows:

$$\delta x = e * (x_i - x_j)$$
$$\delta y = e * (y_i - y_j)$$

This indicates the cumulative desired change for all objects with this given layout. These delta vectors are cutoff at a small maximum threshold to lessen the effect of any one set of changes and then they are added to the X and Y vectors respectively.

If objects are moved off of the edge of the mapped area they are bounced back into the area based on the deflection off the walls of the area. Many other possible ways were investigated in dealing with this adjustment including toroidal mapping but were found to create too discontinuous a search space, and thus too deceptive of a problem.

These changes are iteratively computed either for a specific number of iterations or until the total sum of changes to the X and Y vectors is below some minimal threshold. For simple problems, the minimal threshold is usually reached with fewer than twenty iterations, but for larger problems in a small space, the objects tend to bounce around chaotically. Thus for tight placements, this algorithm tends to converge on less fit local optima more often than for more open space placements.

2.3 Separation Algorithm

Finally, a separation algorithm is used between each iteration of the local optimizer and at the end of the local optimizer to prevent overlap of the machines. The separation algorithm first sets a priority schedule according to distance of objects from the center of the floor. Those objects close to the center of the floor are considered volatile and thus are given lowest priority, while those in corners of the floor are considered fairly stable. This ranking is also similar to measuring the distance that the machine has changed since the last iteration. Both work well, but the first method is used because it is independent of the local optimizer.

Next, objects are placed according to the priority schedule, with highest priority being placed first. If the object overlaps with another machine it is moved in the direction in which the two machines overlap and is moved as little as possible to remove the overlap. This process is iterated until the object is clear of any other machines. After all objects are placed safely, the algorithm terminates.

2.4 Fitness Function

The fitness function takes the distance between each pair of machines and finds the percentage of correctness of that distance relative to the optimal distance indicated by the relationship matrix. The correctness is simply (1 - error). This correctness is then multiplied by the relationship weight and added to the total fitness for the given layout. The total fitness after all relationships have been measured is then divided by the total weight represented in the relationship matrix. The final fitness is guaranteed to be from zero (theoretical worse fitness) to one (theoretical best fitness). Usually the theoretical best fitness is not attainable because every relationship cannot be optimized, due to competing relationships.

One of the most basic biases that was overcome in this problem was that of the negative relationship. This bias arises from the fact that the fitness score can be affected more by a large distance with a negative weight than a small distance times a positive weight of the same magnitude, when the fitness of a particular relationship is determined by multiplying the weight of the relationship by the distance between the two objects. One possible solution to this bias would be to scale the negative weights, but this method would leave room for error as it would be very difficult to estimate a proper scaling. Another possible solution, used in this encoding, is to invert the distances of negatively weighted relationships, and then test the weight as a positive quantity. The equation used to invert the distance of negative relationships is given as

$$d = (MaximumDistance - d^*) + MinimumDistance$$

where d* is the original distance between the two objects.

Negative relationships are made more comparable to positive relationships by reversing the actual distance between the two objects. The actual distance is subtracted from the maximum allowable distance and the new distance is then treated for calculation purposes as equal to those of positive relationships. During calculation, the absolute value of the weight is taken as well to negate the inverted distance. Typical maximum allowable distances are the diagonal, width or length of the floor.

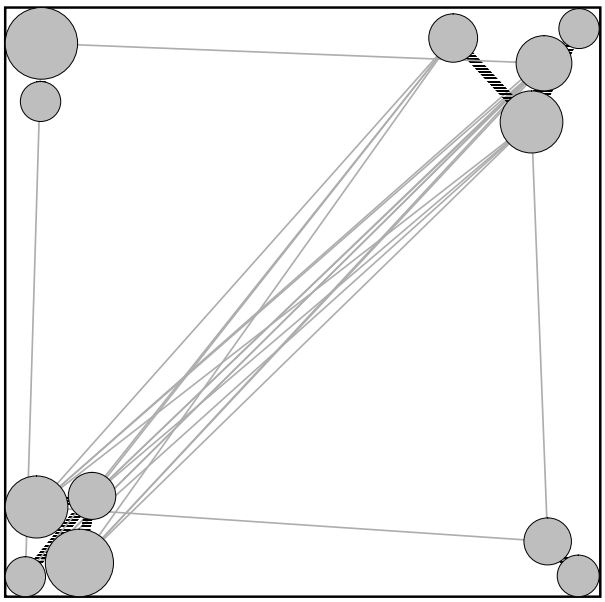

Figure 1: Output of twelve entity problem

3 Results

The genetic algorithm was run on three different problems. All problems were run using GALOPPS [9] which is available from the Michigan State University web site. Two point crossover was used at a rate of eighty percent and a mutation rate of ten percent. The high mutation rate was used because convergence was typically problematic after approximately fifty generations and mutation was able to help avoid such premature convergence. A tournament selection of size two and elitism of one were used to keep selection pressure strong.

The first problem, the twelve entity layout problem, is a theoretical problem in which there are two independent near global maxima without rotation. If rotation is included then this number is doubled due to symmetry. The term near optimal is used because small adjustments can be made to reach a true optima. One example of these two near global maxima is demonstrated by Figure 1, which was found by the genetic algorithm. Darker lines represent stronger relationships and dotted lines represent negative relationships.

The other global optimum is similar but has the two independent pairs of machines grouped in one corner. The genetic algorithm with the local optimizer converges on one of these two near global optima. The second optima can be said to be not globally optimal but is within a small epsilon of the global optimum.

The twelve machine problem was run both with and without the local optimizer. The results of sixteen trials each with a population size of twenty were averaged for each method and are given in Table 2. A predetermined near global optimum was based on best received results and the run was terminated after this mark was met, or after a thousand generations. The number that reached the near optimal goal are indicated in the first column. The second column, the best fitness, indicates the average of the best fitnesses over the sixteen runs. The last column indicates the average number of generations to reach the fitness given.

	Optimal	Best Fitness	Gens
Locally Optimized	(16 /16)	0.845813625	137.125
Non Optimized	(0 / 16)	0.7482265625	1000

Table 2: Results of the twelve entity layout problem.

The genetic algorithm with the local optimizer clearly performed much better than the genetic algorithm alone. The local optimizer is the reason for much of the success in this project, which can be seen simply from the fact that most of the fitnesses in the random population with the local optimizer are close to the best value achieved without the local optimizer. These locally optimized values, however, are not fit enough to be global optima even in a large scale search, as is evident by the number of generations taken to reach the optimal mark.

The second problem that the genetic algorithm was tested on was the forty-five entity problem, which is again a theoretical problem but one that more closely resembles the typical attributes of a large scale factory layout. Figure 2 shows the solution found by the genetic algorithm in conjunction with the local optimizer.

This problem was run similar to the twelve entity problem with a predefined optimal fitness value. Due to the increased complexity of the problem, however, the maximum number of generations was increased to two thousand and the number of individuals in the population was increased to forty. Table 3 shows the results of the genetic algorithm with and without the local optimizer, as well as the results of the SPIRAL method and the hand placed results of an expert in the field for comparison of basic layout capabilities. Again, the method with the local optimization performs significantly better than the genetic algorithm without the local optimization. The results of the genetic algorithm with local minimization were marginally better than the field expert's and significantly better than the SPIRAL method's. The SPIRAL method left too much space between machines and would need an algorithm to optimize the distances between machines to

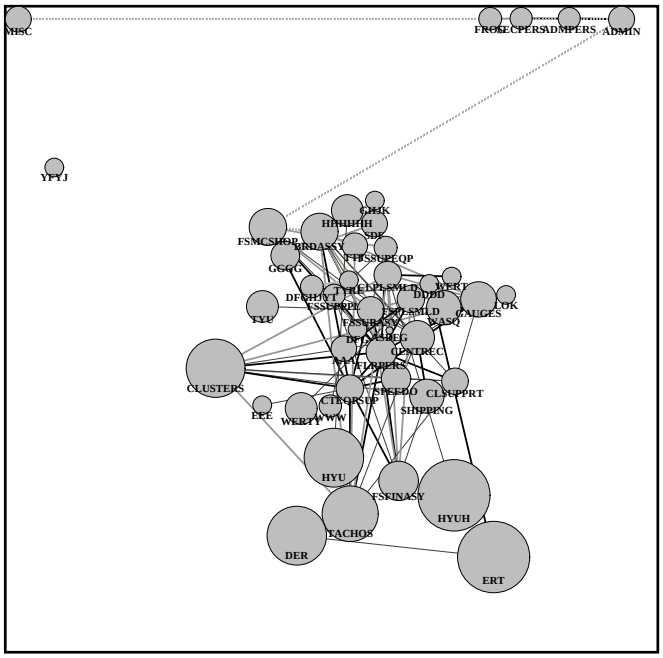

Figure 2: Output of forty-five entity layout problem

compete with the locally minimized algorithm. The genetic algorithm without the local optimizer was not even able to get to the level of the SPIRAL method.

Even though the genetic algorithm with the local optimizer improved over the other methods in comparison, it is evident from Figure 2 that a small amount of optimization is still possible. The negative relationship between the machines labeled "Admin" and "FSMCSHOP" could be extended to the far opposite corner. This advancement however would only yield an improvement of 0.005 in the fitness score. The positioning of a single object with many relationships tends to affect the total score more than one single relationship as this. Such minor improvements are usually improved by larger populations.

	Optimal	Best Fitness	Gens
Locally Optimized	(16 / 16)	0.9123779375	506.125
Non Optimized	(0 / 16)	0.8252123125	2000
Hand Optimized	N/A	0.907205	N/A
SPIRAL	N/A	0.852815	N/A

Table 3: Results of the forty-five entity layout problem.

The final problem took the forty-five entity layout problem and introduced nine obstacles or forbidden zones. These zones can represent walking paths, preallocated areas, support columns, or any such common obstruction. For purpose of calculation, in this example, the zones did not affect the distance between any two objects. If the zone represented a wall and the relationship of the two objects represented the pathway between the two objects, as in parts distribution, then the algorithm would have to be slightly modified to reflect this change in distance. However, such a change would be minimal.

An obstacle separation routine was included in the separation algorithm so that before an individual object was considered for placement it was moved outside of any obstacles that it might currently be on in a fashion similar to the separation equations used for overlap of two machines. When two machines were separated due to overlap, the new location was checked again for obstacles and separated if necessary.

The genetic algorithm was run once with parameters similar to before. A mutation rate of 2 percent was used with a maximum of two thousand generations and a population size of one hundred.

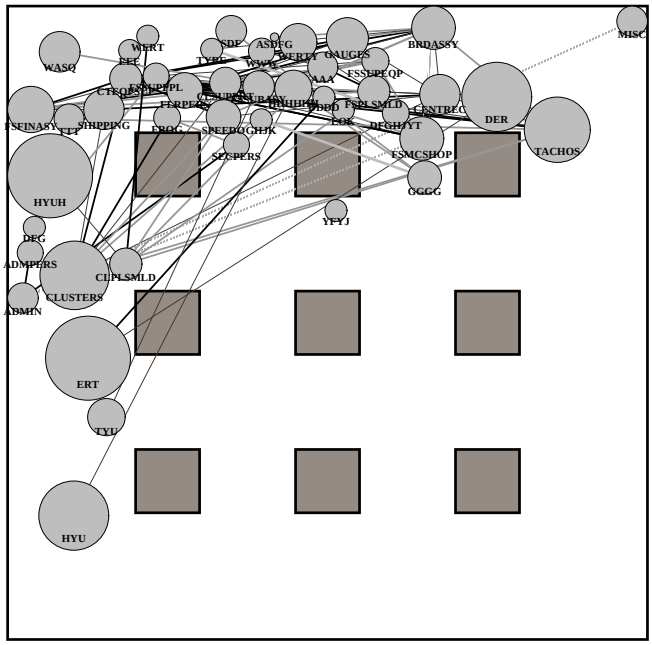

Figure 3: Output of forty-five entity layout problem with obstacles

4 Conclusions and Future Work

The Advanced Layout Problem adds features, such as continuous placement, heterogeneous footprints, and

forbidden zones, to the original Spatial Layout Problem to give it more flexibility, with the additional cost of increased complexity. A local optimizer helps reduce the complexity of the search space and the total search time, and thus allows perhaps even more advancement in the future.

In comparison with commercially available techniques that can be used for this problem, the local optimizer and genetic algorithm performed significantly better although room for small improvements still existed. In comparison with a field expert's results, the hybrid approach yielded slightly better results.

Possible enhancements could include using irregular footprints that match the machine more closely and using irregular placement areas. The forbidden zones could also be extended to include restricted as well as multi-level factories or even multi-dimensional volumes for placement of objects. Fixed objects could also be easily implemented to make such things as docking bays practical.

Finally, the layout problem could perhaps be simplified by a better encoding or a better local optimizer, so that more complex problems with higher object-to-area ratios could be solved more efficiently.

Acknowledgments

This research was supported in part by the Michigan State University Manufacturing Research Consortium. An additional thank you is also due for the entire Genetic Algorithms Research and Applications Group, which provided insight along the way.

References

1 D. M. Deaven and K. M. Ho, "Molecular Geometry Optimization with a Genetic Algorithm," in *Physical Review Letters*, 10 July 1995

2 Russell Meller and Kai-Yin Gau, "The Facility Layout Problem: Recent and Emerging Trends and Perspectives," in *Journal of Manufacturing Systems*, vol. 15, no. 5, 1996, pp. 351-366.

3 I. De Falco, R. Del Balio and E. Tarantino, "Testing Parallel Strategies on the Quadratic Assignment Problem," in *1993 IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, 1993, pp. 254-259.

4 P. P. C. Yip and Y.-H. Pao, "A Guided Evolutionary Simulated Annealing Approach to the Quadratic Assignment Problem," in *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 9, September 1994, pp. 1383-1387.

5 K. Kado, P. Ross and D. Corne, "A Study of Genetic Algorithm Hybrids for Facility Layout Problems," in *Proceedings of the 6th International Conference on Genetic Algorithms*, 1995, pp. 498-505.

6 M. Goetschalckx, "An interactive layout heuristic based on hexagonal adjacency graphs." in *European Journal of Operational Research*, vol. 63, 1992, pp. 304-321.

7 J. H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, 1975.

8 D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, 1989.

9 E. Goodman, "An Introduction to GALOPPS–The 'Genetic ALgorithm Optimized for Portability and Parallelism' System", Tech. Report, Michigan State University, 1994. (http://isl.cps.msu.edu/GA)

10 Volker Schnecke, "Hybrid Genetic Algorithms for Solving Constrained Packing and Placement Problems" Doctoral Dissertation, Department of Math and Computer Science, University of Osnabruck, October 1996.