AN EVOLUTIONARY ALGORITHM FOR SOLVING THE GEOMETRICALLY CONSTRAINED SITE LAYOUT PROBLEM

H. Harmanani[1], P. Zouein[2], AM ASCE, A. Hajar[3]

ABSTRACT

Construction site layout has been recognized as an important activity in construction site planning by field practitioners and researchers alike. This problem involves coordinating the use of limited space to accommodate temporary facilities (such as fabrication shops, trailers, materials or equipment) so that transportation costs of resources are minimized. The layout problem considered in this paper is a static layout problem characterized by affinity weights used to model transportation costs between facilities and by geometric constraints between relative positions of facilities on site. This paper presents an investigation of applying an evolutionary approach to optimally solve the aforementioned layout problem. The proposed algorithm is two-phases: an initialization phase that generates an initial population of layouts through a sequence of mutation operations, and a reproduction phase that evolve the layouts generated in phase one through a sequence of genetic operations aiming at finding an optimal layout. The paper concludes with a number of examples illustrating the strength and limitations of the proposed approach.

INTRODUCTION

Construction site layout has been recognized as an important activity in construction site planning by field practitioners and researchers alike. This problem involves coordinating the use of limited space to accommodate temporary facilities (such as fabrication shops, trailers, materials, or equipment) so that they can function efficiently on site.

The layout problem is generally defined as the problem of 1) identifying the shape and size of the facilities to be laid out, 2) identifying constraints between facilities, and 3) determining the relative positions of these facilities that satisfy the constraints between them and allow them to function efficiently.

There are different classes of layout problems that have been studied in the literature. The variations stem from the assumptions made on the shape and size of facilities and on the constraints between them. Facilities may have a defined shape and size or a loose shape in which

[1] Assistant Professor, Department of Computer Science, Lebanese American University, Po.Box 36, Byblos, Lebanon

[2] Assistant Professor, Department of Industrial Engineering, Lebanese American University.

[3] Research Assistant, Department of Computer Engineering and Sciences, Lebanese American University

case they will assume the shape of the site to which they have been assigned (e.g., bulk construction material). The constraints can vary from simple non-overlap constraints to other geometric constraints that describe orientation or distance constraints between facilities.

In the layout problem, addressed here, the shape and size of facilities are fixed. Facilities can have 2-dimensional geometric constraints on their relative positions along with affinity weights describing the level of interaction or flow between them. While weights are treated as soft constraints between layout objects, the geometric constraints are treated as hard constraints that should be satisfied for the layout to be feasible.

This paper presents an evolutionary algorithm for optimally solving the site layout problem as characterized by affinity weights and 2-dimensional geometric constraints between facilities. The algorithm is tested on different problems where the size and the number of constraints were varied. The paper concludes with a discussion of the capabilities and limitations of the proposed approach.

LITERATURE REVIEW

The layout problem is an NP-complete combinatorial optimization problem that has applications in many contexts including construction site planning, architectural space planning, manufacturing cell layout and VLSI design. Hence, layout planners often resort to using heuristics to reduce their search for acceptable solutions. These heuristics comprise strategic knowledge prescribing the order in which to select objects and to meet constraints, and they have been modeled with various degrees of truthfulness, detail and success in operations research and artificial intelligence applications for space planning (Heragu and kusiak 87, Yeh 1995, Tommelein et al. 91, Zouein 95, Zouein and Tommelein 99).

The application of Genetic Algorithms (GAs) to layout problems is relatively recent. GAs are different from normal optimization and search procedures in many ways. Indeed, steepest descent algorithms and simulated annealing algorithms (which "occasionally" accept non-improving solutions) work with one solution at a time. In contrast, GAs work with a family of solutions, known as the "current population", from which we obtain the "next generation" of solutions. When the algorithm is designed properly, we obtain progressively better solutions from one generation to the next. That is, good or part of good solutions propagate from one generation to the next and lead to better solutions as we produce more generations. Furthermore, GAs work with a coding of the parameter set, not the parameters themselves and use payoff (objective function) information, not derivatives or another auxiliary knowledge. Last GAs use probabilistic transition rules, not deterministic rules to evolve from one solution set to another.

Hence, the main advantage of using an evolutionary approach or a GA is in the fact that it only needs an objective function with no specific knowledge about the problem space. The challenge, however, remains in finding an appropriate problem representation that results in an efficient and successful implementation of the algorithm.

GAs have been applied to solving the facility layout problem (FLP) in the area of production facilities (Tanaka and Yoshimoto 95, Tate and Smith 93 and 95) and to solving the "construction site-level facility layout" (Li and Love 98). In both applications, the layout problem considered was modeled as a location-allocation problem, which consists of allocating a set of predetermined facilities into a set of predetermined sites where the smallest site can accommodate the largest facility. GAs have not been used however to solve the geometrically constrained site layout problem as defined in this paper.

THE GEOMETRICALLY-CONSTRAINED SITE LAYOUT PROBLEM

The layout problem, modeled in this paper, is characterized by rectangular layout objects with fixed dimensions representing the facilities to be positioned on site. Facilities can be positioned in one of two orientations only: a 0° or 90° orientation. In addition, facilities can have 2-dimensional constraints on their relative positions: namely minimum and maximum distance, orientation, and non-overlap constraints. Minimum and maximum distance constraints limit the distance between the facing sides of two facilities in the X or Y direction to be greater than or less than a predefined value respectively. Distance constraints can be used to model equipment reach or general clearance requirements. Orientation constraints limit an facility's position to be to the North, South, East, or West of another reference facility. These constraints can be used to locate access roads or gates with respect to the main facility. Non-overlap constraints are the default constraints that restrict the positions of any two facilities from overlapping.

The objective is to find a feasible arrangement for all layout objects within the site space that minimizes the sum of the weighted distances separating the layout objects: Z:

$$Z = \sum \sum_{j<i} (w_{ij} \times d_{ij}) \qquad \text{eq. 1}$$

Where w_{ij} is the affinity weight between objects i and j which could be used to represent the flow or the unit transportation cost between i and j. d_{ij} is the rectilinear distance separating objects i and j. A feasible arrangement is obtained by finding positions for all layout objects that satisfy the 2-dimensional constraints between them.

EVOLUTIONARY APPROACH

The basic notion of evolutionary computation is to mimic some principles of natural evolution in order to solve optimization problems of high complexity. A group of randomly initialized points of the search space (*individuals*) is used to search the problem space. Each individual encodes all necessary problem parameters (*genes*) as bit strings, vectors, or graphs. The population evolves following a parody of Darwinian principle of the survival of the fittest. Individuals are selected according to their quality, measured by a fitness function, to produce offsprings and to propagate their genetic material (parts of the partial solution found so far) into the next generation. Each offspring undergoes a sequence of operators (e.g., *mutation*, *inversion*, *crossover*) with a certain probability to provide diversity of the population avoiding premature convergence to a single local optimum. The iterative process of selection and combination of "good" individuals should yield even better ones, until a solution is found or a certain stop criterion is met.

GENETIC CODING

A population is a collection of chromosomes where each chromosome represent a layout solution. Every chromosome is coded as a vector whose length is equal to the number of facilities that exists on site *n*. Each facility *i* is represented by the coordinates of its position on site: X_i, Y_i; its dimensions: L_i, W_i, and by a series of pointers pointing at the facilities that surrounds it in the four directions: North, South, East, and West. These pointers will be used to facilitate the check for overlap. The fitness function used is Z.

GENETIC OPERATORS

The genetic operators used are the following:

1. *Mutation*: selects a random location (X, Y) and calls the FindRange function to determine the range at that location where the block can be inserted and not overlap with other neighboring blocks. If the block cannot be inserted, i.e., no range is found then the operation is aborted.

2. *Inversion*: flips a block's position across a horizontal or vertical axis passing through the centroid of the site. This operator also uses the FindRange function to determine if a range exists at the new position. If none exists then the operation is aborted.
3. *Swap*: exchanges the positions of two randomly selected blocks in a chromosome. It is possible that either block may not fit in the place of the other block. If at least one block fits, then the other block is mutated or else, the operation is aborted.
4. *Move*: moves a randomly selected block in a chromosome d units in the x or y direction. The value of d is either Gaussian or unity depending on the number of iterations in which the best solution did not change.
5. *Rotation*: rotates a block 90° anti-clockwise about its upper-left corner.
6. *Flip2Edge*: This operator is similar to inversion with one difference: the axes pass through the centroid of a randomly selected reference block and not the site.
7. *Add Missing Blocks*: acts on chromosomes that contain missing blocks, i.e., blocks for which no feasible position was found in the initialization phase (see next section for details). This may happen as a result of an inefficient use of the site space. This operator attempts to find feasible positions for the missing blocks in a chromosome by applying a sequence of mutations.
8. *Fix Blocks*: fixes the positions of two blocks that have distance or orientation constraints between them.
9. *Aging*: destroys chromosomes whose fitness value did not change after x iterations where x is set to 50. This operator is intended to age solutions that will not lead to the optimal one and allow young chromosomes to evolve into better solutions.

Note that operators 1 through 7 all call the FindRange function to determine if the block operated on can be placed at or in the neighborhood of the new suggested location (P) by the operator. The FindRange function returns the range at P in both the X and Y direction (Rx and Ry respectively) where the block (B_i) with dimensions (L_i, W_i) can be positioned without overlapping with other blocks. This function uses the following procedure to determine Rx and Ry:

a. Let MinRange = $L_i/2$ ($W_i/2$).
b. Create a double integer array of size L (W).
c. Fill the array with the distances from P to the nearest blocks right and left of P (above and below P).
d. Let d_R (d_T) and d_L (d_B) be the minimum distance between P and the nearest blocks to its right (top) and to its left (bottom) respectively.
e. If Min (d_R, d_L) >= MinRange then set Rx = Min (d_R, d_L), else set Rx = 0.
f. Repeat steps a) to d) using the values between parenthesis to determine Ry. Ry = Min (d_T, d_B), if Min (d_T, d_B) >= MinRange and 0 otherwise.

If the FindRange function fails to return a range, i.e., either Rx or Ry is null then the operation calling the FindRange function will be aborted.

INITIALIZATION PHASE

Chromosomes in the initial population are generated randomly by applying a sequence of mutation operations. First, blocks with user-defined fixed positions on site are added to a chromosome; these blocks cannot be moved by the above GA operations. Next, mutation is applied on the remaining blocks by selecting randomly a block at a time. If applying mutation on a given block results in an infeasible range for block insertion then mutation is repeated on the same block up to 100 times to find a feasible position for that block. If after 100 trials no

feasible position is found, this block is left out of the chromosome. Hence, this method may result in chromosomes representing partial layout solutions. The operator Add-Missing-Blocks will be responsible for completing such partial solutions in the evolutionary phase.

EVOLUTIONARY PHASE

Following the generation of the initial population, the genetic operators are applied to evolve the initial population into better ones as shown in the pseudo-code below:

```
Evolutionary_Placement (Site, Facilities, Constraints)
{   generate_initial_population();
    evaluate();
    keep_the_best();
    do {
          select_chromosomes();
          apply_genetic_operators();
          calculate_fitness();
          update_population();
          keep_the_best();
        } while(improvement_obtained or time_not_exceeded);
return; }
```

Every operator has a probability associated with it. The product of the operator's probability and the population size determines the maximum number of times this operator can be fired in one generation. Operator's probabilities and values of other parameters were determined experimentally and are as shown in Table 1 below.

Table 1: Operators and Probabilities

Parameter	Value	Parameter	Value
Population size	100	Probability of Rotation	0.2
Probability of Mutation	0.1	Probability of Swap	0.1
Probability of Move	1	Probability of Flip2Edge	0.2
Probability of Inverse	0.1	Max Age for Aging	50

Also experimentally, it was found that the pure survival of the fittest does not always work for this type of layout problems. Hence, some"bad" chromosomes (i.e., chromosomes representing partial solutions or chromosome with high fitness value) were preserved in the next generation as this could in the long run help the algorithm converge to a global optimum. The following criteria was used in selecting chromosomes for reproduction:

1. Choose 5% from the best chromosomes.
2. Randomly choose 10% from "bad" chromosomes with density fitness less than 60% or more that 150% of the best chromosome. The density fitness is the ratio between fitness value and number of blocks available in the chromosome. Note that a chromosome's fitness value can exceed 100% if the chromosome represents a partial solution.
3. Randomly choose 80% from those chromosomes with density fitness between 80% to 120% of the best chromosomes.
4. Choose randomly the remaining chromosomes.

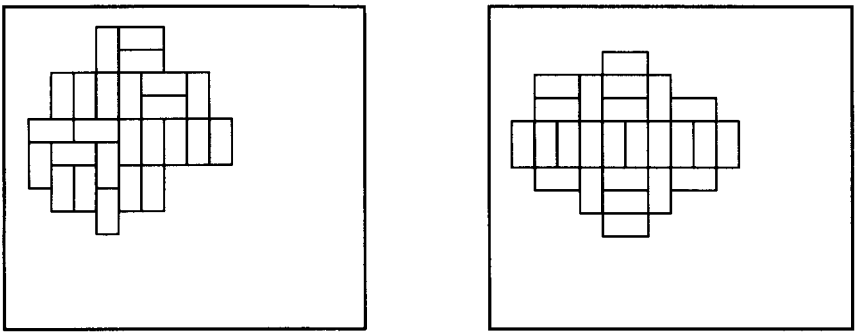
EXAMPLES APPLICATION

A C++ program was developed on a Personal Computer as the implementation of the proposed GA. The GA was run on a variety of test problems from which we chose: a loosely constrained

problem and a tightly constrained problem where the total-objects-to-site-area-ratio was varied to illustrate the GA's performance under both settings.

EXAMPLE 1: A LOOSELY CONSTRAINED PROBLEM

This example is intended to test the GA's performance on a loosely constrained problem with a large number of layout objects to be positioned. To this end, we solved a problem with 25 layout objects of equal size (2x1 unit area) and a total-objects-to-site-area ratio less than 20%. The layout objects have no geometric constraints on their relative positions but do have proximity weights between them. A weight of 1 is defined between all pairs of layout objects. The GA stopped after 51 seconds or 200 generations. The layout solution obtained has a fitness value equal to 1444 and is shown in Fig. 1(a). This problem was then solved using the algorithm developed by (Zouein 95) which is a suboptimal algorithm based on a linear programming formulation and the best layout returned by this method had a fitness value equal to 1416 and is shown in Fig. 1(b).



(a) (b)

Figure 1: (a) The GA Solution After 200 Generations (b) A better Solution

EXAMPLE 2: A CONSTRAINED PROBLEM

This example is intended to test the GA's performance on a layout problem with a relatively smaller number of layout objects but a greater number of constraints and a higher demand for site space. The total-objects-to-site-area ratio is around 50% as compared to 20% in the previous example, and the layout objects have 2-dimensional geometric constraints between them in addition to the proximity weights. Table 2 summarizes the layout objects and the site dimensions. Table 3 shows the proximity weights between them.

Table 2: Input data

Object	(L, W)	Area	Object	(L, W)	Area
Cons. Site	(30, 20)	600	Object-6	(9, 4)	36
Object-1	(7, 5)	35	Object-7	(9, 5)	45
Object-2	(4, 4)	16	Object-8	(10, 4)	40
Object-3	(8, 5)	40	Object-9	(2, 2)	4
Object-4	(2, 1)	2	Object-10	(4, 4)	16
Object-5	(7, 7)	49			

Table 3: Proximity Weights

Object-i	Object-j	W_{ij}	Object-i	Object-j	w_{ij}
Object-1	Object-2	25	Object-6	Object-7	100
Object-2	Object-3	50	Object-8	Object-9	25
Object-4	Object-5	50	Object-9	Object-10	25

In addition, Object-6 is constrained to be to the North of Object-7 and there is a minimum distance constraint of 5 units in the Y-direction between object-9 and object-10. This problem is solved by the proposed GA which stops after 93 seconds or 250 iterations. The best layout solution found has a fitness value of 1450 and is shown in Fig. 2 below. Note that the gray-shaded rectangle represents Object-4.

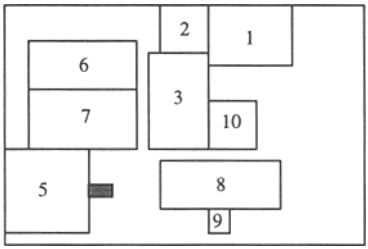


Figure 2: The GA Solution for the Problem of Table s

There are alternative optimal solution for this problem. Fig. 3 shows only two such optimal layouts with optimal fitness value equal to 1262.5.

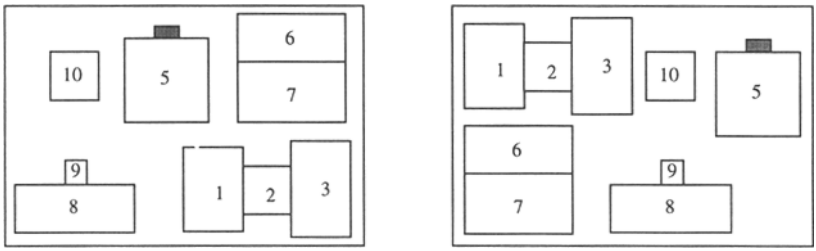


Figure 3: Alternative Optimal Layout Solutions for the Problem of Table 1

As it can be seen from the above results, the GA found close to optimal solutions in both cases of a loosely constrained problem with large number of layout objects to be positioned and a constrained problem with a relatively congested site. The algorithm was tested on a number of other problems and it was noted that it performed better in loosely constrained ones with small total-objects-to-site-area ratio even when the number of blocks was large. This result is expected because of the very nature of GAs.

CONCLUSION AND DIRECTIONS FOR FUTURE RESEARCH

This paper presented the results of an initial investigation on the application of genetic algorithms for solving the geometrically constrained site layout problem. Key features of this algorithm are that it uses a large number of different GA operators to vary positions of objects around the site and that it maintains in each generation chromosomes representing partial layout

solutions. These so-called "bad" chromosomes were kept to help the evolution process get out of local optima.

The algorithm was tested on different problems with a different number of blocks and constraints. In most cases where the total-objects-to-site-area ratio did not exceed 60%, the algorithm returned close to optimal solutions in a reasonable time (less than 2 minutes) after 200 generations. In problems with higher total-objects-to-site-area ratio the algorithm failed to find feasible solutions. Finally, the relationship between computational time and the number of layout objects was found to be approximately linear.

Future research will focus on experimenting more with different population sizes and probabilities of GA operators to study their effect on the rate of convergence. Another research direction will investigate the viability of using constraint satisfaction to code sets of possible positions of objects and use them to instantiate positions of objects in a chromosome. This approach will ensure the generation of complete and feasible chromosomes.

References

Goldberg, D.E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Boston.

Kusiak, A.and Heragu, S.S., 1987, The Facility Layout Problem, *European J. of Operational Research.*, Vol. 29, pp. 229-251.

Li, H. and Love, P.E.D., 1998, Site-Level Facilities layout Using Genetic Algorithms, *J. of Comp. in Civ. Engrg.*, ASCE, Vol. 12, 4, pp. 227-231.

Tanaka, H. and Yoshimoto, K., 1993, Genetic Algorithm Applied to the Facility Layout Problem, *Dep. of Industrial Engrg. and Management*, Waseda Univ., Tokyo.

Tate, D.M. and Smith, A.E., 1993, Genetic Algorithm Optimization Applied to Variations of the Unequal Area Facilities Layout Problem, *Proc. of the 2nd Industrial Engrg. Research Conf.*, Los Angeles, CA, pp.335-339.

Tate, D.M. and Smith, A.E., 1995, A Genetic Approach to the Quadratic Assignment Problem, *Computers and Operations Research*, Vol. 22, No. 1, pp.73-83.

Tommelein, I.D., Levitt, R.E., and Confrey, T., 1991, SightPlan Experiments: Alternate Strategies for Site Layout Design, *J. of Comp. in Civ. Engrg.*, ASCE, Vol. 5, 1, pp.42-63.

Yeh, I.-C., 1995, Construction-site Layout Using Annealed Neural Network, *Journal of Computing in Civil Engineering*, ASCE, Vol. 9, 3, pp.201-208.

Zouein, P.P., 1995, *MoveSchedule: A Planning Tool for Scheduling Space Use on Construction Sites*, Ph.D Dissertation, Dep. of Civil and Env. Engrg., Univ. of Michigan, Ann Arbor, MI, Dec.

Zouein, P.P. and Tommelein, I.D., 1999, Dynamic Layout Planning Using a Hybrid Incremental Solution Method, *J. of Const. Engrg. and Management*, ASCE, Nov/Dec.