# Genetic search and the dynamic layout problem

**Jaydeep Balakrishnan**
Faculty of Management,
University of Calgary,
Calgary, Alberta T2N1N4, Canada.
Ph: (403) 220 7844   Fax: (403) 282 0095
Email: balakris@ucalgary.ca

**Chun Hung Cheng**
Department of Systems Engineering and Engineering Management,
The Chinese University of Hong Kong,
Shatin, Hong Kong.
Email: chcheng@se.cuhk.edu.hk

Revised version published in *Computers and Operations Research*, 27, 2000, 587-593.

# Genetic search and the dynamic layout problem

## Scope and Purpose

Many organizations today are operating in a dynamic and market-driven environment. To stay competitive, their facilities must be adaptive to market fluctuations. The dynamic layout problem devises a multiperiod layout plan based on these demand fluctuations. In this paper, we propose an improved implementation of an existing genetic algorithm for solving the dynamic layout problem.

## Abstract

An improved genetic algorithm for the dynamic layout problem, based on a paper that was published in this journal, is formulated and tested in this research. Our genetic algorithm differs from the existing implementation in three ways: first, we adopt a different crossover operator, second, we use mutation, and third, we use a new generational replacement strategy to help increase population diversity. A computational study shows that the proposed GA is quite effective.

# Genetic search and the dynamic layout problem

## 1. Introduction

In a paper published in this journal, Conway and Venkataramanan [1] examined the suitability of genetic algorithms (GA) for the dynamic layout problem. A genetic search uses the mechanics of natural selection and natural genetics to evolve a population of initial solutions into a near-optimal solution [2,3]. This approach is suited to handle multiple and nonlinear objective functions as well as side constraints.

In their procedure, a string in the population consists of $n \times t$ digits, where $n$ is the number of departments in the layout and $t$ is the number of periods in the planning horizon. So a string represents every department in each period. To crossbreed, two strong strings (based on the fitness function) from the population are selected. Then a random splicing position is generated and the strings are split.The substrings are then swapped. Since these swaps may create infeasible solutions (eg. two occurences of department 5 in one period), additional digit swaps are done to ensure feasibility.The string with lower cost is allowed to survive to the next generation. In tests, where cross-breeding was done for up to 100 generations using population sizes of upto 800, the algorithm performed well compared to dynamic programming for six and nine department problems.

Our GA differs from the GA in [1] in three aspects:

1. We employ a different crossover operator to increase the search space.
2. Mutation is used to increase population diversity.
3. We use a generational replacement approach to increase population diversity.

Computational results show that the proposed GA generally performs better than the GA in [1] with respect to the cases tested..

## 2. The proposed Nested Loop GA

The suggested procedure uses a GA with nested loops. The inner loop uses a steady state replacement approach, and replaces the most "unfit" individual in each generation. The outer loop will replace a large number of "unlucky" individuals in a generation.

### Inner Loop

As in [1], let *p[i]* denote layout string *i*, an individual. When $n = 6$, and $p = 5$, individual *i* with *p[i]* = 243156 342651 342615 342615 342561 has department 2 assigned to location 1 in the first period, department 4 to location 2 and so on with department 1 being assigned to location 6 in period 5 . We will randomly generate a set of layouts as the initial population. From this population, a pair of parents are chosen randomly. Then a point-to-point crossover operator is used. In a point-to-point crossover, we no longer cut layout strings randomly at one position. Instead, we cut them at every position, starting from the first position to the last position in the string. The fitness function is identical to that in [1]. To illustrate the procedure, consider a six department, two period problem. We select two layouts from the population: layout A and layout B. Let the configurations of these layouts be:

layout A: *p[A]* = 214563 345621

layout B: *p[B]* = 123456 234561

If we swap the departments in the first positions of the two layouts, two new child layouts or strings will be formed, layout 1 and layout 2, as follows:

*p[1]* = 114563 345621

*p[2]* = 223456 234561   (split at the 1st position)

Now the layouts are split at position 2 and the departments in the second positions are swapped. This gives the following layouts 3 and 4.

*p[3]* = 124563 345621

*p[4]* = 213456 234561   (split at the 2nd position)

This is done for each position. After the final swap we get new layouts 21 and 22 as follows:

*p[21]* = 123456 234561

*p[22]* = 214563 345621  (split at the 11th position).

Note that we do not split at the 12th position as this is would result in the same strings denoted by the initial layouts A and B. Thus, the point-to-point crossover operator produces *2(nt-1)* child layouts. Although many child layouts are illegal (an illegal child layout is one with a duplication of any department in a period), at least one child layout is legal under the point-to-point crossover. For example, in this case we are guaranteed to have legal child layouts after a split at position 6. Here, out of 22 layouts, six child layouts (i.e., *p[3], p[4], p[11], p[12], p[21], p[22]* ) are legal. A feasibility test is applied each time to eliminate illegal child layouts. The minimum cost legal child layout will replace the maximum cost parent in the population.

Mutation may occur in the minimum cost legal child layout. However, the probability of this is quite small. We first randomly choose a period in which mutation will take place. Then within the period chosen, two departments will be randomly chosen for interchange. For example, mutation applied to p[22] = 214563 345621 will result in *p[22']* = 214563 365421 when the second period and departments 4 and 6 are chosen for swapping. This mutated child will then replace the maximum cost parent.

As the selection and mutation process continues, the population will evolve over successive generations. The loop terminates when the difference between the best child layouts in two successive generations is less than a very small threshold value. A flowchart of the procedure is presented in Figure 1b.

Outer Loop

The outer loop consists of periodically replacing a number of 'poor' parents in the population. The initial population is called the 'old' population. First, the inner loop replaces some of the individuals in this population. Then, in the outer loop process, a number of new layouts or individuals are generated randomly. These layouts then replace the poorest layouts in the 'old' population. This creates a 'new' population consisting of some parents from the 'old' population, the children created through the inner loop, and the randomly created new individuals. This new population will serve as the 'old' population for the next run of the inner loop. The outer loop process stops when the difference between the average cost of two successive outer loops is less than a minimum threshold value. The flow chart for the outer loop process is shown in Figure 1a.

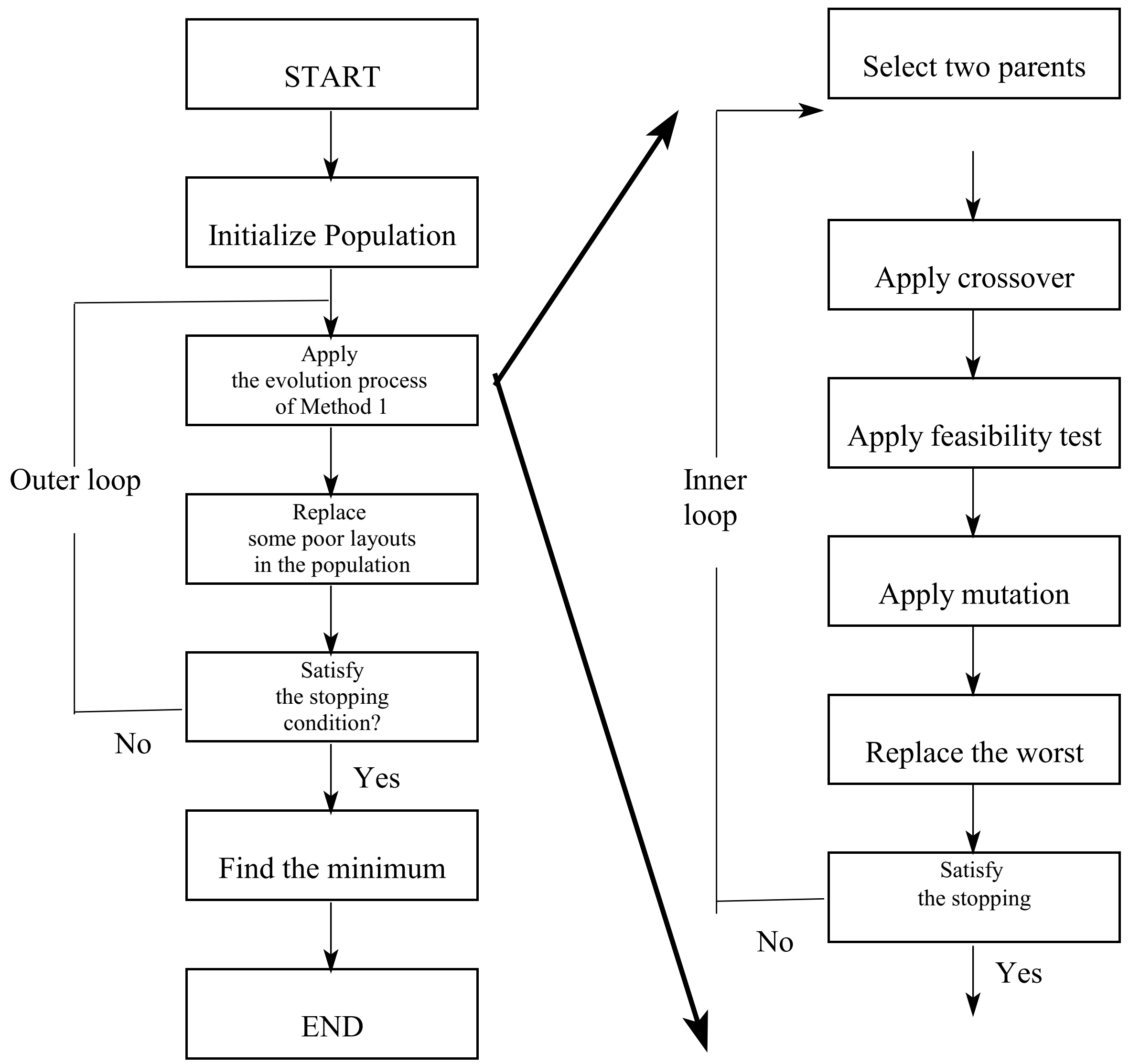

**Figure 1a**

**Outer Loop**

**Figure 1b**

**Inner Loop**

## 3. Rationale for the Nested-Loop Process

The point-to-point crossover of the inner loop process has some advantages over the GA in [1]. Since every individual gene is crossbred, the search space within each population is larger than in the case of the single-point crossover. The feasibility test is also simpler.

Mutation is useful, since it creates children with characteristics that would not normally occur by cross breeding. However, the frequency of mutation is deliberately kept low. Since mutation is done on the best child, frequent mutation would result in the loss of good children. Thus, low mutation rates result in occasion injection of new characteristics into the population without destroying the existing genetic characteristics of the population.

Since each inner loop works with the same population, the GA might end up in a local optimum if only an inner loop were done. The outer loops ensure that that inner loops work with different populations. This enlarges the search space and should lead to better solutions.

## 4. Computational Results

The computational experiment has the following factors:

1. Planning horizons: Two different planning horizons of 5 periods and 10 periods are used. This will give us an idea of the effectiveness of the algorithms over longer term planning horizons.
2. Sizes of layouts (departments or machines): Three layout sizes of 6, 15, and 30 departments are used. This will help us determine the effectiveness of the algorithms for larger layouts.

This gives us 2×3 or 6 different situations. Eight replications were performed for each situation. Each of the eight problems was randomly generated and is similar to the problem shown in [1]. For each problem, the sum of the flows in a layout during a period was constant during the entire horizon. This prevented any one period's cost from dominating the others. The average shifting cost for a department was set to be 15% of the average material handling cost for the department. The five period problems use the first five periods of data from the ten period problems. In each problem, some departments had high inflows as compared to the others. These high inflow departments were changed in every period to simulate demand changes.

A initial experiment was done to test the different parameters. We ran tests with population sizes of 100, 500, and 1000. We also performed experimental runs with mutation probabilities of 0.001, 0.002, and 0.003. Finally, we tested threshold values (for the stopping conditions) of 1%, 5%, and 10%. Based on the results, we found that the following parameter values gave the best results for the nested-loop GA: population size = 500, mutation probability = 0.002, and the threshold value for the stopping condition = 5%.

The comparison results are shown in Tables 1 - 6. Each table gives the minimal cost solution for each of the 8 problems using the following methods:

1. Genetic Search by Conway and Venkataramanan: CONGA
2. Nested-Loop Genetic Algorithm: NLGA

The better solution for each test problem is highlighted. In the 6 department situation (Tables 1 and 2), NLGA performs better than CONGA in fourteen out of the sixteen cases ( in one of the cases, the costs were identical). In Table 3 (15 department, 5 period) CONGA performs better than NLGA in six out of the eight cases. In Table 4 (15 department, 10 period), NLGA performs better than CONGA in all cases. Finally, in the 30 department problems (Tables 5 and 6), the NLGA performs better in all cases.

Given given our test problems, it appears that NLGA generally performs better than CONGA. It is interesting to note that for the three largest types of problems, NLGA performed better than CONGA in every problem tested. Similarly, the NLGA appears to be better than CONGA in the 10 period problems where it performed better on every problem.

## 5. Conclusion

In this paper we examined the use of genetic search in solving the dynamic layout problem. Our genetic algorithm differs from the existing algorithm in many ways including using a new crossover operator, a mutation approach, and a new generational replacement strategy. The computational results showed that the proposed algorithm is quite effective.

**Acknowledgements**

The authors wish to thank Professor D.G. Conway for the use of his GA code, to the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Research Grant Council of Hong Kong for their financial support..

**References**

1. D.G Conway and M.A.Venkataramanan, "A Genetic search and the dynamic facility layout problem", *Computers & Operations Research*, 21, 8, 955-960, 1994.
2. L.D. Chambers, *Practical Handbook of Genetic Algorithms, Volume I*, CRC Press, Boca Raton, FL (1995).
3. L.D. Chambers, *Practical Handbook of Genetic Algorithms, Volume II*, Applications, CRC Press, Boca Raton, FL (1995).

Table 1

6 Departments and 5 periods

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| CONGA | 108976 | 105170 | 104520 | 106719 | 105628 | 105606 | **106439** | 104485 |
| NLGA | **106419** | **104834** | **104320** | **106515** | 105628 | **104053** | 106978 | **103771** |

Table 2
6 Departments and 10 periods

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| CONGA | 218407 | 215623 | 211028 | 217493 | 215363 | 215564 | 220529 | 216291 |
| NLGA | **214397** | **212138** | **208453** | **212953** | **211575** | **210801** | **215685** | **214657** |

Table 3
15 Departments and 5 periods

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| CONGA | **504759** | 514718 | **516063** | **508532** | 515599 | **509384** | **512508** | **514839** |
| NLGA | 511854 | **507694** | 518461 | 514242 | **512834** | 513763 | 512722 | 521116 |

Table 4
15 Departments and 10 periods

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| CONGA | 1055536 | 1061940 | 1073603 | 1060034 | 1064692 | 1066370 | 1066617 | 1068216 |
| NLGA | **1047596** | **1037580** | **1056185** | **1026789** | **1033591** | **1028606** | **1043823** | **1048853** |

Table 5
30 Departments and 5 periods

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| CONGA | 632737 | 647585 | 642295 | 634626 | 639693 | 637620 | 640482 | 635776 |
| NLGA | **611794** | **611873** | **611664** | **611766** | **604564** | **606010** | **607134** | **620183** |

Table 6
30 Departments and 10 periods

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| CONGA | 1362513 | 1379640 | 1365024 | 1367130 | 1356860 | 1372513 | 1382799 | 1383610 |
| NLGA | **1228411** | **1231978** | **1231829** | **1227413** | **1215256** | **1221356** | **1212273** | **1245423** |