INT. J. PROD. RES., 1999, VOL. 37, NO. 16, 3739–3758

An iterative facility layout algorithm

K.-Y. GAU† and R. D. MELLER†*

In this paper we present a facility layout algorithm that iterates between a genetic algorithm with a slicing tree representation and a mixed-integer program with a subset of the binary variables set via the genetic algorithm. The genetic algorithm is very good at finding low-cost solutions while maintaining shape constraints on the departments. The slicing tree representation and the mixed-integer program are compatible in terms of layout representation, with the mixed-integer program representation more general. The mixed-integer program allows us to relax the last remaining constraints of the slicing tree representation of the genetic algorithm. We present our genetic algorithm and the iterative algorithm, while illustrating their performance on test problems from the literature. In 10 of the 12 problems, new lower best-cost solutions are found.

1. Introduction

The facility layout problem is concerned with finding the most efficient non-overlapping planar arrangement of departments with unequal area requirements within a facility (see Bozer and Meller (1997) for a taxonomy of layout problem definitions). The definition of 'efficient' in the above statement is not clear because the solution has an impact on so many facets of the facility's efficiency. However, since 'between 20 to 50% of the total operating expenses within manufacturing are attributed to materials handling' (Tompkins *et al.* 1996), and an efficient layout can 'reduce these costs by at least 10 to 30%' (Tompkins *et al.* 1996), layout solutions are developed that attempt to minimize the material handling costs. Typically, this is interpreted as either minimizing a distance-based objective or maximizing an adjacency-based objective. Constraints for the layout problem are concerned with departmental area requirements, and shape and locational restrictions.

As we discuss further later, the facility layout problem is a very difficult optimization problem, with much of the work concentrated on heuristics (see the surveys by Kusiak and Hergau (1987) and Meller and Gau (1996)). Some of the most recent heuristics that provide good solutions include (in chronological order): LOGIC (Tam 1992a, b), SPIRAL (Goetschalckx 1992), MULTIPLE (Bozer *et al.* 1994), FLEX-BAY (Tate and Smith 1995), SABLE (Meller and Bozer 1996), BEND (Kamoun and Yano 1996), and GA-MIP (Banerjee *et al.* 1997). Layout heuristics are usually characterized by whether they require an initial layout (improvement-type approach) or not (construction-type approach). All of the above algorithms, except SPIRAL and BEND, are improvement-type approaches, yet, all but

Revision received February 1999.

† Department of Industrial and Systems Engineering, Auburn University, Auburn, AL 36849-5346, USA.

*To whom correspondence should be addressed; e-mail: rmeller@eng.auburn.edu; website: http://www.eng.auburn.edu/~rmeller

International Journal of Production Research ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/JNLS/prs.htm
http://www.taylorandfrancis.com/JNLS/prs.htm

MULTIPLE are used in a construction-type manner (random initial layouts can be used effectively). LOGIC, FLEX-BAY, SABLE, and GA-MIP are based on simulated annealing or genetic algorithm (GA) searches.

A notable exception to the above statement that much of the work is concentrated on heuristics is the mixed-integer program (MIP) introduced by Montreuil (1990) for the general facility layout problem. Unfortunately, as admitted by Montreuil, and explored by Meller *et al.* (1998), it is an extremely difficult problem to solve optimally. The original formulation by Montreuil is only solvable for problems where $n \approx 5$, where n is the number of departments. Even after considerable work to improve the solvability of the layout MIP, Meller *et al.* (1998) were only able to solve specially-structured problems where $n \approx 10$ ($n \approx 7$ for generally-structured problems).

To find optimal solutions to the facility layout problem, a general methodology, like MIP, must be used. However, since the layout MIP is unsolveable for any realistic-sized instance, some type of heuristic must be used. In the past, heuristics based on the MIP have used a pre-processing heuristic to determine the values of the binary variables in the MIP (Banerjee *et al.* 1992, Montreuil *et al.* 1993, Langevin *et al.* 1994, Banerjee *et al.* 1997). The resulting linear program (LP) is then solved and the algorithm terminates. At this point the user may go back to the pre-processing heuristic and re-solve the LP to iterate.

We consider a different iterative approach to the facility layout problem in this paper. As we show later, the slicing tree representation of a layout corresponds very well (although not exactly) to the binary variables in the layout MIP. We use a genetic algorithm based on the slicing tree representation to heuristically find a solution. We then use this solution to specify a subset of the binary variables in the MIP. Finally, after solving the MIP, we use any new information from the MIP to generate the initial population of the genetic algorithm, thus, closing the iterative loop.

In § 2 we discuss the background of slicing tree representation, the layout MIP, and motivate our iterative approach. In § 3 we present our genetic algorithm based on a slicing tree representation. In § 4 we present our iterative approach. In both § 3 and § 4 we present our results for problems found in the literature. Finally, in § 5 we present our conclusions.

2. Problem background

2.1. *Slicing tree representation*

Tam introduced the slicing tree representation for facility layout in two papers (Tam 1992, b). LOGIC is the name given to these two algorithms, where LOGIC stands for layout optimization with guillotine induced cuts (Tompkins *et al.* 1996) (one algorithm is a simulated-annealing based search and the other is based on a genetic algorithm). In either search version, LOGIC represents a layout as a slicing tree where the departments are placed in the tree in the 'leaf' positions and 'cuts' fill the remaining nodes. Figure 1 represents how the slicing tree in (*a*) is translated to the layout in (*b*) if departments on the right of a cut are placed relative to the departments on the left of the cut in the direction specified by the cut (N = North, E = East, etc.). Slicing trees can be represented with the *postorder method*, where in the postorder method we visit the left subtree, the right subtree, and then the root in a recursive fashion. For example, the slicing tree in figure 1(*a*) is

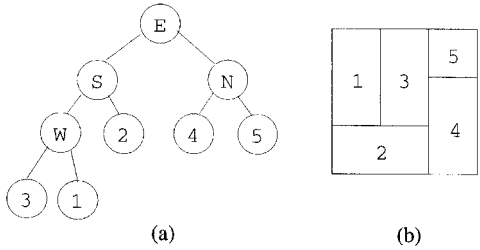

Figure 1. Slicing tree representation of a layout.

represented as: 1, 3, W, 2, S, 4, 5, N, E. Note that in computer algorithms, the cuts often take on numerical values (e.g., N = −1, S = −2, E = −3, and W = −4).

In LOGIC, an initial slicing tree is given and departments are fixed to their leaf positions throughout the execution of the algorithm. The algorithm proceeds by generating new solutions that exchange the positions of cuts in the tree. Clearly, the initial solution will significantly influence the final solutions generated since the initial slicing tree will limit which solutions can be represented within LOGIC.

2.2. *Facility layout MIP*

As mentioned earlier, Montreuil (1990) introduced the first MIP for facility layout. The formulation is also presented in a recent layout survey paper (Meller and Gau 1996). The decision variables of the layout MIP are the half-length l_i^x, half-width l_i^y, and centroid (c_i^x, c_i^y), for each (unfixed) department i. Clearly, no two departments can overlap in a valid layout; i.e. for each ij pair, the two departments' projections onto x and y can overlap in at most one of the two directions. Meller *et al.* (1998) reformulated the binary decision variables in the problem and their formulation more easily relates to our iterative approach. Thus, we refer to their definition of the binary decision variables instead of those in Montreuil's formulation.

The requirement of non-overlapping departments can be linearized using binary variables denoting the relative location of departments:

$$z_{ij}^x = \begin{cases} 1, & \text{if } i \text{ precedes } j \text{ in the } x\text{-direction} \\ 0, & \text{otherwise} \end{cases}$$

and

$$z_{ij}^y = \begin{cases} 1, & \text{if } i \text{ precedes } j \text{ in the } y\text{-direction} \\ 0, & \text{otherwise.} \end{cases}$$

Montreuil's original model is only solvable for $n \approx 5$. Results in Meller *et al.* (1998) indicate that, at present, problems with $n \approx 7$ are generally solvable and problems with $n \approx 10$ can be solved under certain conditions (sparse flows and loose area constraints). The authors indicate that more work is needed on this problem until realistically-sized problems can be solved.

2.3. *Iterative approach*

The slicing tree approach, as outlined by Tam, although very powerful, has two weaknesses. We attempt to address them with a new GA that utilizes a slicing tree representation. The MIP is even more powerful than the slicing tree in terms of

solutions it can represent and constraints that it can model. However, it remains a very difficult optimization problem.

Both approaches utilize a continuous layout representation and are able to accommodate fixed departments. The objective function for both approaches is a distance-based measure:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} c_{ij} d_{ij}, \tag{1}$$

where f_{ij}, c_{ij}, and d_{ij} represent the flow, per-unit cost, and distance from department i to department j. In both approaches the distance is measured rectilinearly between department centroids (although the GA can easily accommodate other distance measures).

In an attempt to capitalize on the advantages of both of these approaches, we combine them in an iterative approach. We first present our genetic algorithm, and compare it to LOGIC.

3. Genetic algorithm with a slicing tree representation (GAST)

There are two aspects of Tam's approach (Tam 1992a, b) that restrict the solution space:

(1) Departments are fixed to their initial positions in the slicing tree throughout the algorithm.
(2) The structure of the slicing tree remains fixed as in the initial slicing tree.

We will examine methods to remove each of these restrictions, and thus, enlarge the solution space for an initial layout.

To change a layout solution, as represented by a slicing tree, the departments can change locations with one another within the tree, i.e. in addition to the cuts being exchanged with one another, departments can also be exchanged. (Of course, departments cannot be exchanged with cuts if the slicing tree is to remain a valid layout.) Thus, if implemented correctly, allowing for department exchanges will increase the performance of a slicing-tree based algorithm by enlarging the solution space.

To further enlarge the solution space that a slicing-tree based algorithm considers, the structure of the slicing tree should be permitted to change. One way to achieve this is to exchange different *branches* of the slicing tree, i.e. to define a branch to be a sub-tree that is rooted in a cut that is not the root of the entire tree; e.g., in figure 1(*a*), the sub-tree containing departments 1, 2, and 3, and rooted with an S cut would be a branch. Further define the nodes that correspond to department positions as *leaves*. A leaf-for-branch exchange or a branch-for-branch exchange will fundamentally alter the structure of the slicing tree (note that leaf-for-leaf exchanges are merely the department exchanges discussed above). However, implementing such a routine is not straightforward.

There is another method that will achieve many of the same benefits as branch-for-branch exchanges. We can add dummy departments to the slicing tree. Dummy departments are departments with little or no area and no interaction with other departments. Note that adding dummy departments will, in essence, make the slicing tree larger. Further note that if departments are allowed to be exchanged with other departments, then the effect will be that multiple slicing tree structures can result from one initial slicing tree. For example, by adding one dummy department

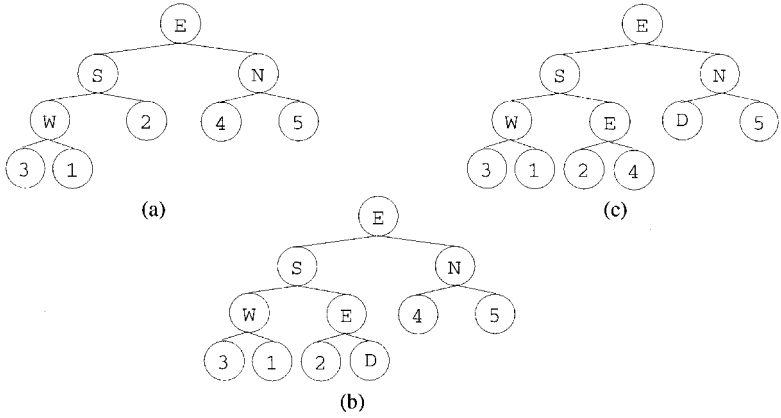

Figure 2. Adding dummy departments to a slicing tree representation.

(denoted by D) to the tree in figure 1(*a*) (repeated here in figure 2(*a*)) we can form the slicing tree in figure 2(*b*). Now consider the exchange of department 4 and dummy department D. The resulting layout is shown in figure 2(*c*). Such a layout could not have been accomplished without dummy departments unless the tree structure was changed.

Implementing the changes outlined above (exchanging departments and incorporating zero-area dummy departments) results in a new genetic algorithm with a slicing tree representation (GAST). The details of this new genetic algorithm follow.

3.1. *The search algorithm*

A genetic algorithm is a metaheuristic based on the evolutionary process of natural systems (Holland 1973). Genetic algorithms have been applied to many optimization problems, such as TSP, bin packing, scheduling, and VLSI design. It differs from other traditional search methods by starting with a population of solutions instead of a single solution. When a correct set of parameters is chosen (which is not a trivial task), genetic algorithms will perform like Darwin's 'survival of the fittest' principle; the higher the fitness of a member of the population, the higher the probability that it will reproduce and the overall population's fitness will be improved generation by generation. In our case, layouts with low costs as measured by (1) have high fitness values.

A simple genetic algorithm (GA) that produces good results in many optimization problems consists of the following three operators: (1) selection, (2) crossover, and (3) mutation (see Goldberg (1989) for more details on GAs).

Selection is a process that selects individuals from the previous generation. The selected individuals are then copied, unchanged, to the new generation. The probability of being selected to remain in the new generation is according to the fitness of the individual. The higher the fitness of the individual, the higher the probability that the individual will be selected.

Crossover is a process that randomly selects two parents from the old population according, usually, to increasing probability based on increasing fitness (see Bäck and Schwefel (1993) for a discussion of exceptions to this rule). The GA then randomly picks a section and generates offspring by exchanging one section of one parent with the corresponding section of the other parent. Once two valid offspring

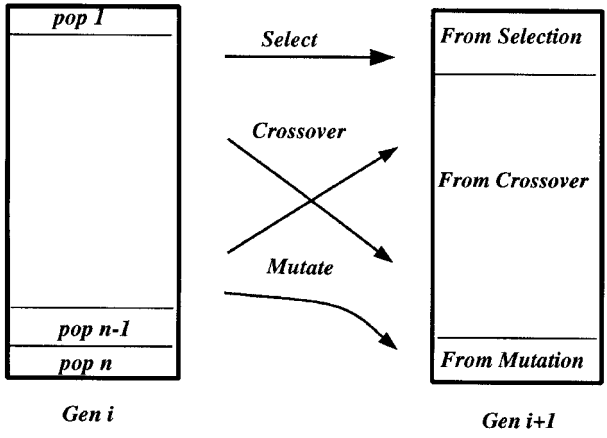

Figure 3. From *Gen i* to *Gen i* + 1.

are produced, they are copied into the new generation. In the algorithm there are two kinds of crossover operators: one operator is a crossover on the cuts, the other is on departments. Here we implement simple crossover mechanisms: a simple one-point crossover on cuts and the *partially matched crossover* on departments (see Goldberg (1989) for other possible crossover mechanisms).

Mutation is a process that randomly changes one component of the string during the crossover process (according to previous research (Goldberg 1989), the mutation rate should be low), i.e. there will be a crossover on either cuts or departments, then these offspring will randomly mutate on cuts or departments. When mutation occurs on cuts, it randomly selects one cut and randomly picks one of the other three cuts to replace it. When mutation occurs on departments, two randomly-selected departments are exchanged.

Figure 3 illustrates the concept of GA population operators from generation (*Gen*) to generation.

3.1.1. *Penalty function*

Since we use the objective function of minimizing the total flows times distances, if there is no shape constraint, all the departments will line up their centroids, and we will end up with a layout that suffers from all departments being long and skinny. We use the penalty method from Coit *et al.* (1996) to constrain department shapes. This penalty method is based on each department's aspect ratio; that is, the ratio of the longest side length to the shortest side length. (When implemented with non-rectangular departments, the aspect ratio is based on the department's minimum enclosing rectangle.) Note that we modify the penalty method slightly; instead of setting the severity factor equal to 2 (Coit *et al.* 1996), we set it equal to 1.

3.2. *The GAST algorithms*

In this section we will discuss three versions of our genetic algorithm with slicing tree (GAST). First we discuss our initial base GA, which we refer to as BASE-GA. We then consider a GA using multiple initial trees. We refer to this GA as MULT-GA. We then discuss how to extend the slicing tree representation with the use of dummy departments. We refer to this version as EXT-GA.

The following notation will be used in our three genetic algorithms.

Notation

S_t the number of different initial tree structures
P_o the population size
r_k the select (to keep) rate
r_c the crossover rate
r_m the mutation rate,
$r_k + r_c + r_m = 1.0$
p_c the probability of crossover on cuts
p_d the probability of crossover on departments $(= 1 - p_c)$
Gen the total number of generations.

The GAST procedure follows.

GAST algorithm

Step 1. Generate S_t initial trees by applying the postorder method.
Step 2. Set the control variables P_o, r_k, r_c, r_m, p_c, p_d, and *Gen*.
Step 3. According to the tree structure, generate a population of layouts with different department sequences and different cuts.
Step 4. Calculate the cost of each layout in the population. Sort the population according to each layout's cost.
Step 5. Keep the best solutions found in the previous generation according to the r_k rate. If considering non-feasible solutions, then half of the best solutions should be feasible and the other half may be infeasible.
Step 6. Randomly select two parent solutions from the previous generation according to a roulette wheel that is built based on their fitness.
Step 7. Based on p_c, choose a type of crossover to produce the offspring.
Step 8. Repeat steps 5–7 until the number of offspring totals $P_o \times (1 - r_m)$.
Step 9. Perform mutation according to the mutation rate, r_m, by doing the simple crossover on department cuts, then randomly changing one of the cuts or exchanging a pair of the departments. Mutation will produce the last $P_o \times r_m$ offspring.
Step 10. Repeat steps 4–9, until the number of generations equals *Gen*.
Step 11. Repeat steps 3–10, until every tree structure is evaluated.

According to Goldberg (1989), we need to choose the parameter set carefully. After testing some parameter sets with different test problems, we found that as long as we kept the mutation rate small (in our cases < 0.1), there is no significant difference in the results. Therefore, we choose $r_k = 0.02$, $r_c = 0.96$, $r_m = 0.02$, and $p_c = 0.5$ for our parameter set. We did not consider optimizing the GA parameter set (as noted previously, this is a significant problem in and of itself).

3.2.1. *BASE-GA*

Our initial base genetic algorithm, which we denote by BASE-GA, is formed by setting $S_t = 1$ in the GAST procedure. In this section we compare the results of BASE-GA with our version of LOGIC (BASE-GA with $p_c = 1.0$). In order to perform a fair comparison, we obtain the initial trees used from (Tam 1992b) for problems Tam15, Tam20 and Tam30. For AB20 (Armour and Buffa 1963), M11 (Bozer

	LOGIC		BASE-GA		Improvement (%)	
Data set	Avg	Minimum	Average	Minimum	Average	Minimum
Tam15	8 502.4	8 410.1	8 500.2	8 301.9	0.0	1.3
Tam20	11 633.5	11 405.8	10 402.9	9 927.9	10.6	13.0
Tam30	23 613.2	23 104.5	21 215.9	20 484.8	10.2	11.3
AB20	8 531.9	8 165.0	5 903.0	5 773.4	30.8	29.3
M11	1 551.3	1 542.0	1 306.4	1 243.3	15.8	19.4
M15	46 323.5	44 881.2	32 557.8	30 210.8	29.7	32.7

Table 1. Results for LOGIC and BASE-GA.

et al. 1994), and M15 (Bozer *et al.* 1994), instead of generating the initial tree from average linkage method (Tam 1992b), the initial trees are generated randomly. The tests were performed for a fixed population size (500) and number of generations (500). To maximize our comparisons with other approaches, we used an aspect ratio constraint of 5.0 for each non-fixed department. Each test problem was run ten times using the same tree each time. Full test results are presented in Gau (1998). When presenting results, the values in the tables will be based on the objective function value, equation (1), which we are attempting to minimize.

From the test results presented in table 1, we can see BASE-GA outperformed LOGIC, especially for those cases where the tree is randomly initialized. Therefore, it is critical when using LOGIC to use the average linkage method to initialize the slicing tree, whereas randomly initialized trees appear satisfactory for BASE-GA. In this way, BASE-GA appears more robust than LOGIC.

Since we illustrated that BASE-GA outperforms LOGIC, in future sections we do not focus our GA comparisons on LOGIC directly. However, we always note the best LOGIC solution found with our ten runs presented in table 1.

3.2.2. *MULT-GA*

In this section we introduce a multiple-tree structure approach to expand the solution space in BASE-GA, which is based on one tree structure. In MULT-GA, five different tree structures are chosen for each data set; $S_t = 5$ in the GAST procedure. We build trees with a different number of levels; that is, the longest path from leaf node to root node will change in each tree. This will increase the variety of trees. According to each tree, we apply our GA to find a better layout. Table 2 presents the minimum layout cost for each tree based on fixed population size (500) and number of generations (500). We compare the results of MULT-GA to our LOGIC results (Imp1 %), the best results presented in previously published papers (Imp2 %), and BASE-GA (Imp3 %). The best layout result and source by problem is: Tam15 (10 099.1) (Tam 1992b), Tam20 (9495.2) (Meller and Bozer 1996), Tam30 (20 420.7) (Meller and Bozer 1996), AB20 (5524.7) (Tate and Smith 1995), M11 (1372.7) (Meller and Bozer 1996), M15 (31 936.3) (Meller and Bozer 1996), M25 (1588.4) (Meller and Bozer 1996), Ma12 (416) (Kamoun and Yano 1996), B12 (8768.2) (Tate and Smith 1995), B14 (5052.0) (Tate and Smith 1995), V10-E (20 320.5; Euclidean distances) (Tate and Smith 1995), and V10-R (22 395.0; Rectilinear distances) (Banerjee *et al.* 1997).

In most cases, the best published results are based on the minimum of ten runs with a simulated annealing or genetic algorithm heuristic. As we can see from table 2,

Data set	Tree						Imp1 (%)	Imp2 (%)	Imp3 (%)
	T1	T2	T3	T4	T5	Min			
Tam15	8530.6	8493.3	8537.7	8527.5	8655.9	8493.3	−1.0	15.9	−2.3
Tam20	9591.2	9746.6	9734.4	9760.0	9752.3	9591.2	15.9	−1.0	3.5
Tam30	21 150.5	20 803.8	20 712.5	20 787.7	20 817.8	20 712.5	10.4	−1.4	−1.1
AB20	5648.8	5318.9	5953.7	5416.2	5386.8	5318.9	34.9	3.7	8.5
M11	1185.2	1264.2	1215.7	1223.9	1228.8	1185.2	23.1	13.7	4.9
M15	31 132.7	29 157.6	31 084.6	34 063.0	33 770.9	29 157.6	35.0	8.7	3.6
M25	1596.5	1608.9	1682.0	1670.4	1591.3	1591.3	N/A	−0.2	N/A
Ma12	355.8	337.8	338.6	363.0	361.0	337.8	N/A	18.8	N/A
B12	9486.3	9055.3	9817.9	9670.5	9273.5	9055.3	N/A	−3.3	N/A
B14	5151.9	4900.4	4925.4	5359.0	5485.0	5268.8	N/A	−4.3	N/A
V10-E	19 535.0	20 077.4	21 376.8	18 817.9	20 581.9	18 817.9	N/A	7.4	N/A
V10-R	22 319.1	22 548.6	22 352.6	22 879.7	21 926.4	21 926.4	N/A	2.1	N/A

Table 2. Results for MULT-GA.

MULT-GA improves the best-known solution from the literature for many of the data sets; in some cases it does so considerably: Tam15 (15.9%), M11 (13.7%), M15 (8.7%), Ma12 (18.8%), and V10-E (7.4%). In almost all cases, MULT-GA with five tree structures performs comparably to any other approach with ten starting points. Of course, some of the comparisons are not exact due to department representation (e.g., SABLE (Meller and Bozer 1996) is based on a discrete representation) or department shape constraints.

The results with multiple (different) tree structures improved over numerous runs with one single tree structure in almost all cases. The drawback is that we are then required to generate different tree structures *a priori*. If we can find a more general tree that can represent more than one tree structure, then we might be able to search only on this general tree. This leads us to the next section and our final GA.

3.2.3. *EXT-GA*

In order to increase the flexibility of a fixed tree structure, we add dummy departments to the slicing tree as discussed earlier in this section.

EXT-GA performs the same steps as the BASE-GA. The only difference is the addition of dummy departments to the initial slicing tree. According to the number of the real departments, we add a different number of dummy departments. Specifically, we add dummy departments such that the size of the largest tree is the next highest power of 2 (since the slicing tree is a binary tree). For example, for AB20, we test the number of dummy departments added from 4 to 12, which varies the total number of departments from 24 to 32. In general, the last tree structure for a data set is a full balanced tree.

The results from EXT-GA are presented in table 3. As we expected, adding more dummy departments, in general, helps to generate a better layout. However, it also shows that we cannot add too many dummy departments; obviously this will cause it to perform too many exchanges that will not make a difference in the layout cost (exchanging two dummy departments). The last two columns in table 3 illustrate the improvement over MULT-GA using the minimum over all extended slicing trees (Imp1 %) or the solution from the slicing tree with the largest number of dummy departments (Imp2 %). EXT-GA improves the best-known solution from the litera-

Data set	Tree T1 D	T1 Min	T2 D	T2 Min	T3 D	T3 Min	T4 D	T4 Min	T5 D	T5 Min	All trees min	Imp1 (%)	Imp2 (%)
Tam15	9	8360.2	9	8329.4	13	8334.0	13	8316.6	17	8233.6	8233.6	3.1	3.1
Tam20	4	9758.8	4	9513.5	8	9573.6	8	9681.0	12	9606.4	9513.5	0.8	−0.2
Tam30	2	21255.6	6	20658.0	10	20671.1	18	21069.4	34	20825.8	20658.0	0.3	−0.5
AB20	4	5471.8	4	5604.2	8	5731.1	8	5810.5	12	5275.5	5275.5	0.8	0.8
M11	3	1195.7	3	1188.2	5	1189.1	5	1187.8	6	1188.9	1187.8	−0.3	−0.3
M15	2	29580.1	10	29592.8	10	28269.6	14	28983.9	18	27228.0	27228.0	6.6	6.6
M25	4	1675.8	4	1620.7	6	1532.3	6	1550.9	8	1498.8	1498.8	5.8	5.8
Ma12	1	333.3	1	334.2	3	337.6	3	352.7	4	333.3	333.3	1.3	1.3
B12	7	8623.5	7	9163.1	11	9618.1	11	8965.5	15	8485.4	8485.4	6.3	6.3
B14	12	5077.3	8	5165.3	8	5163.0	12	5033.7	16	4804.1	4804.1	8.8	8.8
V10-E	2	20358.5	2	19753.3	4	19655.4	4	19753.3	6	18618.7	18618.7	1.1	1.1
V10-R	2	22614.2	2	23411.1	4	22986.7	4	22352.6	6	22282.0	22282.0	−1.6	−1.6

Table 3. Results for EXT-GA.

	Tam15	Tam20	Tam30	AB20	M11	M15	M25	Ma12	B12	B14	V10
BASE-GA	259	454	537	232	117	192	332	125	168	143	214
MULT-GA	1179	2270	2685	1160	585	960	1660	625	840	715	1070
EXT-GA(T1)	396	551	577	238	145	180	447	135	260	399	275
(T2)	385	535	650	261	150	290	409	138	296	289	277
(T3)	568	378	741	326	178	281	492	203	343	342	337
(T4)	579	451	930	339	191	422	379	181	370	257	330
(T5)	610	499	1397	392	207	450	501	215	412	396	391

Table 4. Runtime comparisons for three GAs (seconds).

ture for many of the data sets and in some cases it does so significantly: Tam15 (18.5%), AB20 (4.5%), M11 (13.4%), M15 (14.7%), M25 (5.6%), Ma12 (19.9%), B14 (4.9%), and V10-E (8.4%). In fact, improved best-known solutions have been found by one of the GAST algorithms for all data sets except Tam20 and Tam30.

Note that in general it does not require the addition of too many dummy departments to achieve comparable results between MULT-GA and EXT-GA. However, as one would expect, the runtime for EXT-GA could approach the runtime of MULT-GA with five tree structures if the number of dummy departments is too large. Table 4 presents runtime results to illustrate this point (runtimes are presented in seconds for a 133 MHz Pentium-processor personal computer). The number of dummy departments is indicated by the tree structure (see table 3). For example, the number of dummy departments for Tam20 is 4, 4, 8, 8, and 12 for T1, T2, T3, T4, and T5, respectively. In general we see that the runtime of EXT-GA for T5 is approximately one-half the runtime for MULT-GA. Also note that T5 may be too large in some cases and a smaller tree will have a shorter runtime.

3.3. *Conclusions regarding the algorithm*

In this section three different versions of GAST for facility layout were introduced (all tree structures for the test problems are presented in appendix A of Gau (1998)). Each version of GAST has found good solutions as compared to the results in the literature (our best solutions are presented in appendix B of Gau (1998)).

Among the three versions, EXT-GA has proved to be the most successful. From tables 1–4, we form the following conclusions with respect to these approaches.

- Our genetic algorithm with slicing tree representation, BASE-GA, is an improvement over LOGIC (Tam 1992b).
- MULT-GA with five tree structures improves the performance of BASE-GA, and improves the best-known solution for 10 data sets from the literature.
- As the number of dummy departments increases in EXT-GA, the solution quality improves.
- The results from EXT-GA are comparable to MULT-GA, and also produced new best-known solutions.
- Although the runtime for EXT-GA is longer than BASE-GA, it is less than MULT-GA.
- Instead of five different initial trees in MULT-GA, only one general tree is needed in EXT-GA; yet, EXT-GA still searches over more different tree structures than MULT-GA.

4. Iterative approach

In this section we consider how to iterate between our genetic algorithm (GA) and a mixed-integer program (MIP) for the facility layout problem. As discussed previously, the MIP for the layout problem with general flow structures cannot be solved when the number of departments is greater than seven (Meller *et al*. 1998). Therefore, to use the MIP on even a relatively small problem with 10–15 departments, almost all of the binary variables must be set *a priori*. In this case, the result from the MIP is a heuristic solution, even though it is optimal for the initial binary variable conditions.

In this section we present how we implemented an iterative approach for this problem. We begin with the iteration from GA to MIP. After presenting some results, we discuss how to iterate from MIP back to GA. We then discuss the limitations of this approach.

4.1. *Implementation: GA-to-MIP method*

As we mentioned previously, we obtain a final layout from our GA. Then, according to the relative location between each department pair in the slicing tree, we set the binary variable values in the MIP formulation as follows. (Note that the bottom left-hand corner is the origin of the layout $(0, 0)$.)

Cut	Binary variable value set
${}_i\mathrm{N}_j$	$z^y_{ji} = 1; z^y_{ij} = z^x_{ij} = z^x_{ji} = 0$
${}_i\mathrm{S}_j$	$z^y_{ij} = 1; z^y_{ji} = z^x_{ij} = z^x_{ji} = 0$
${}_i\mathrm{E}_j$	$z^x_{ji} = 1; z^x_{ij} = z^y_{ij} = z^y_{ji} = 0$
${}_i\mathrm{W}_j$	$z^x_{ij} = 1; z^x_{ji} = z^y_{ij} = z^y_{ji} = 0.$

In this section, the algorithm will proceed as follows:

Step 1. Run the EXT-GA algorithm from § 3.
Step 2. According to the final layout from EXT-GA, we specify the pairwise location arrangement for all department pairs.

No	Proportion of variables fixed (%)	Cost	Improvement/GA (%)	Time	Number of iterations	Nodes
1	100.0	1178.4	—	0.5	131	0
2	58.2	1177.6	0.0	318.0	32 814	1037
3	56.4	1170.3	0.7	128.4	23 209	857
4	54.5	1170.3	0.7	50.5	9 769	332
5	52.7	1170.3	0.7	152.3	30 254	1060

Table 5. M11 GA-to-MIP method results.

Step 3. Select a percentage in the range of 50%–100%. Fix this percentage of the MIP binary variables by starting at the root node and moving down the tree, one level at a time. Minor variations are performed to achieve the target percentage.
Step 4. Send the binary variable information to the MIP formulation.
Step 5. Solve the MIP to find an optimal solution given these specified cuts.

4.1.1. *Results: GA-to-MIP method*

As we have mentioned, the initial input data of this approach is from EXT-GA. We would like to see if this approach can improve over EXT-GA, which is likely to generate a near-optimal layout. Our results from GA to MIP are limited by the solvability of the MIP formulation even though we use the improved formulation from Meller *et al.* (1998) (with the following added valid inequalities: T3, B1, V1, B2, and S3). We only test this method with smaller-sized problems (M11, Ma12, B12, and B14). An example of our results is shown in table 5 for M11 (runtimes are given in seconds on a Sun Ultra workstation).

We solved the problem first by setting 100% of the binary values. This ensured that any improvements with less than 100% of the binary values specified would be due to layout improvements and not due to the area relaxation in the MIP. For example, the layout with the objective function value of 1187.8 became 1178.4 with 100% of the variables set due to the area approximation in the MIP.

Among the four test problems, a slight improvement is found for M11 (0.7%) and B14 (4.2% with 71.2% of the binary variables fixed; from 4626.7 to 4434.5) while no improvement is found for the other two data sets with approximately 50% of the variables not fixed. In some cases, if we did not set enough binary variables, we could not solve the problem; e.g., Ma12 is unsolvable with 53% of the binary variables specified.

The GA-to-MIP approach is somewhat limited due to two factors. The first is that the MIP is difficult to solve if too few binary variables are set, which, of course, limits the improvements the MIP might make. The second is that the solution from the GA itself may be good enough as to allow little room for improvement (i.e., near-optimal). In the next section we give the MIP a known non-optimal solution to investigate the latter factor.

4.1.2. *Modification to the MIP input*

In this section we test the GA-to-MIP approach by giving the MIP formulation input corresponding to a solution worse than the best-known solution. In this case, if we were using a heuristic that does not find a very good solution, we could use the

No	Proportion of variables fixed (%)	Cost	Improvement/GA (%)	Time	Number of iterations	Nodes
1	100	1370.3	—	0.5	128	0
2	74.5	1228.9	10.3	3.7	954	46
3	70.9	1228.9	10.3	6.2	1579	61
4	40.0	1211.1	11.6	180.0	26331	599

Table 6. M11 non-optimal GA-to-MIP method results.

MIP to obtain a better result. To limit our GA's ability, we set the number of generations for the test problems to be very small (*Gen*= 50). As before, we set the percentage of binary variables such that the MIP can be solved (e.g., for M11, 30–80%). Table 6 shows that for M11 an 11.6% improvement is achieved, although it takes 599 nodes. Note that the final solution here (1211.1) is not as good as the initial solution for EXT-GA as was used in the previous section (1178.4). Similar conclusions can be drawn for the other three data sets.

4.2. *Implementation: MIP-to-GA method*

In general, the output from the MIP will not conform to a slicing tree. Moreover, although it may conform to a slicing tree, it may not be the original slicing tree. Therefore, in almost every case, we cannot directly pass the MIP solution back to our GA via the slicing tree that was used to generate the original MIP input. In this section we describe two methods that may be used for the purpose of passing MIP results back to our GA.

Let us consider using EXT-GA with a limited number of generations as discussed in §4.1.2 for test problem M11. The layout generated by the GA is given in figure 4. Using this as input with 40.0% of the cuts fixed, the MIP produced the layout shown in figure 5. Note that the layout almost, but does not quite (due to the placement of two empty areas within the layout), conform to a slicing tree representation. If we

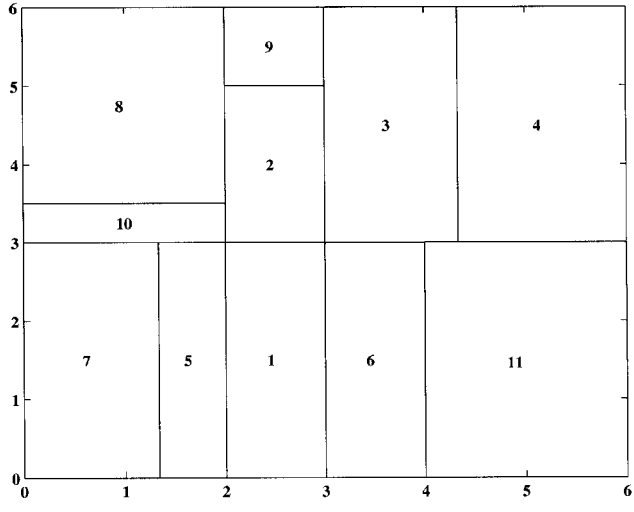

Figure 4. M11 (known) non-optimal layout from GA.

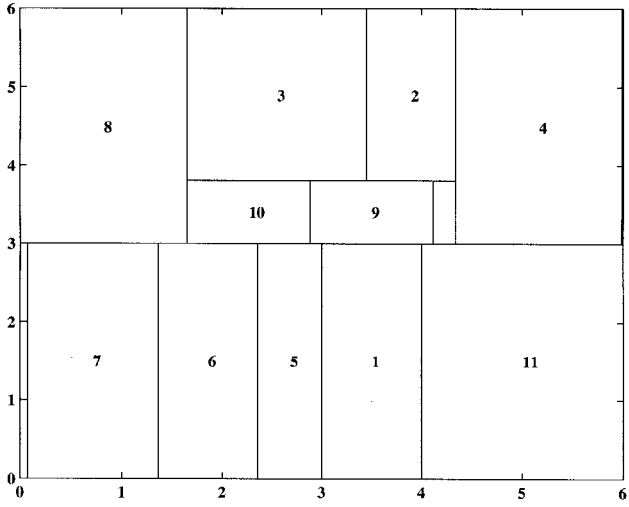

Figure 5. M11 layout from MIP.

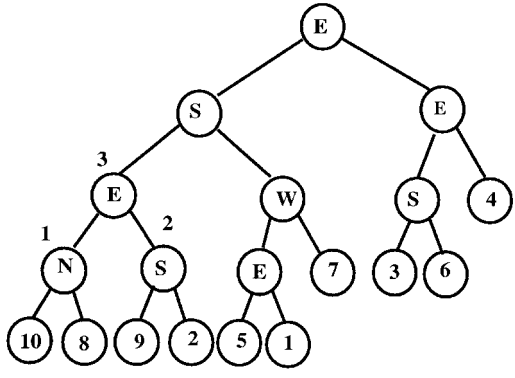

Figure 6. Slicing tree for the original GA layout.

ignore these empty areas in the layout, we can represent the layout by a slicing tree. For comparison, the slicing tree for the GA-generated layout (figure 4) is presented in figure 6 and the slicing tree for the MIP-generated layout (figure 5) is presented in figure 7. Clearly, the two slicing trees are not the same and iteration from MIP to GA is not straightforward.

One possible way to implement MIP to GA is as follows:

Step 1. Use the original slicing tree from GA as our new slicing tree as well.

Step 2. For our new slicing tree, keep the department sequence the same as in the original slicing tree.

Step 3. From the cut information in the MIP-generated slicing tree (figure 7), we try to set the corresponding cuts in our new slicing tree (which, recall, has the same structure as the slicing tree from our original GA solution).

Step 3.1. For example, we can set Cut 1 (between Departments 10 and 8) in our new tree (figure 8) to W since in figure 7 the cut between Departments 10 and 8 is W.

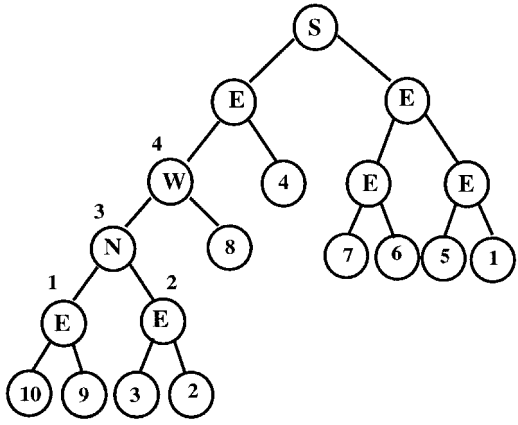

Figure 7. Slicing tree for the improved MIP layout.

Step 3.2. As for Cut 3 (Departments 10 and 8 to Departments 9 and 2) in our new tree, we can set it according to Cuts 1, 2, 3, and 4 in figure 7. This results in a 75% chance of the cut being E, and a 25% chance of the cut being N. (Note that at first examination it appears that the cut may take the value W due to Departments 2 and 8. However, note that when the proper orientation is considered, an E may need to be a W; likewise for N and S.)

Step 3.3. Continue with the rest of the cuts in a similar manner.

Step 4. Add an appropriate number of dummy departments (D), and add dummy cuts (C) if necessary to accommodate the dummy departments.

Step 5. See figure 8 for the final slicing tree with dummy departments and cut information (the probabilities are omitted).

Step 6. The cut probability information can be used in such a way that the initial population of the GA is generated according to the probabilities.

Step 7. We can then perform the EXT-GA as described previously in § 3.

The results for this method are presented in table 7, where we explicitly initialized the population with all sixteen cut combinations and then randomly generated the

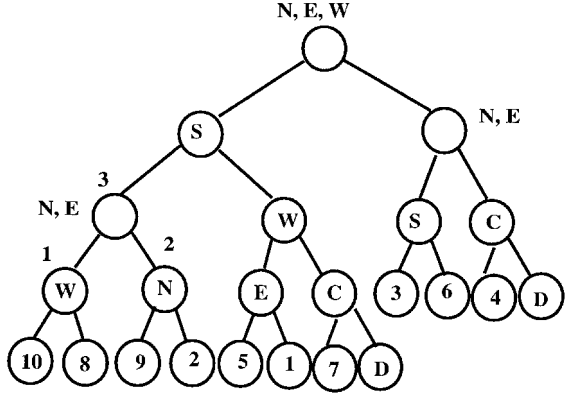

Figure 8. New slicing tree with different cut combinations.

	MIP to GA	EXT-GA (T1)	EXT-GA (T2)
Dummy departments	2	3	3
Average	1208.7	1261.3	1250.5
Improvement (%)		4.2	3.3
Minimum	1185.2	1195.7	1188.2
Improvement (%)		0.9	0.3

Table 7. M11 MIP-to-GA results with the first method.

other 484 population members. We compare these results with the results from two runs of EXT-GA that did not use the MIP-solution information. We can see that the MIP-to-GA method achieved an improvement (4.2% and 3.3%) in the average final population layout solutions. However, the best layout improvement is only 0.9% and 0.3% for the two trees, respectively. We believe this comparison is fair since in the MIP-to-GA method we used a slicing tree with one less dummy department than the two other slicing trees. Therefore, we conclude that starting our GA with the information from our MIP had a positive effect on our ending population.

This method may work in some cases, but in general, there are too many combinations of cuts to form an initial population of 500 members since each cut two levels above the leaf nodes could take on four values (there is one less cut than departments). For example, if there are five cut nodes that take on four cut values, we have exceeded our population size. Likewise, if there are nine cut nodes that take on two cut values, we have also exceed our population size. Therefore, another method is needed.

The next method of iterating from MIP to GA is based on the observation that instead of adding dummy departments to our new slicing tree arbitrarily, we might add them with a purpose. This purpose is to have one tree that represents both the original GA slicing tree and the MIP-generated slicing tree. Our general procedure is to:

(1) Take the slicing tree that conforms to the MIP-generated solution.
(2) 'Grow' the tree so that it can accommodate the original slicing tree from the GA solution by equalizing the number of levels in each branch of the two slicing trees.
(3) Use the generated tree in EXT-GA with the MIP-generated layout as one member of the initial GA population.

An example of the above method is presented for illustration. First, there are six levels to the MIP-generated slicing tree. Since there are five levels in the original GA-generated slicing tree, we do not need to add any levels to our new slicing tree. Now, for each leaf node in the GA-generated slicing tree, determine if the MIP-generated tree could accommodate it, i.e. if the branch in the MIP-generated tree is not long enough, then extend it.

For example, departments 10, 8, 9, and 2 in the original tree (figure 6) can be accommodated in the MIP-generated slicing tree. On the other hand, the arrangement of departments, 5, 1, and 7 cannot. Therefore, the branch in the MIP-generated slicing tree that terminates at the leaf node with department 4 must be extended. With this change alone, we arrive at the slicing tree in figure 9, which clearly represents the MIP-generated slicing tree. Note that some of the cuts (C) can be set

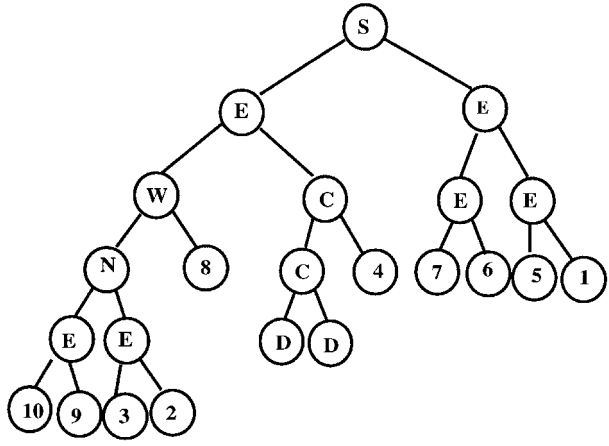

Figure 9. Slicing tree after adding dummy departments.

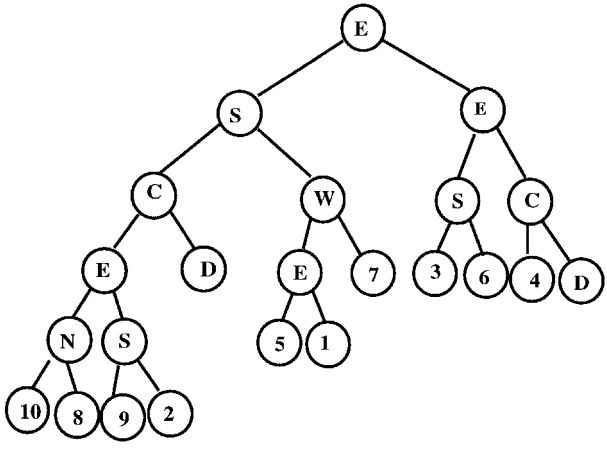

Figure 10. New MIP slicing tree that represents the original GA slicing tree.

arbitrarily in arriving at the final slicing tree. In figure 10 we arrange the department nodes and cut nodes to represent the layout from GA-generated layout (slicing tree in figure 6).

This method enables us to keep the information we gained from the improved MIP solution. The results for this procedure with data set M11 is presented in table 8. We can see that this MIP-to-GA method achieved a 5.2% and 4.4% improvement in the layout solution averages. However, the best layout improvement is only 0.6%, and 0.0%, for the two trees, respectively. As before, we believe this comparison is fair since we used a slicing tree with one less dummy department. Therefore, in this case as well, we conclude that starting our GA with one improved solution in the population from the MIP had a positive effect on our ending population.

It is not clear which MIP-to-GA method will do better in general. The strength of the second method is that at least one member of the initial population will conform

	MIP-to-GA	EXT-GA (T1)	EXT-GA (T2)
Dummy departments	2	3	3
Average	1195.4	1261.3	1250.5
Improvement (%)		5.2	4.4
Minimum	1188.2	1195.7	1188.2
Improvement (%)		0.6	0.0

Table 8. M11 MIP-to-GA results with the second method.

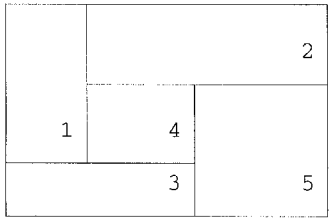

Figure 11. Feasible MIP solution that is infeasible for a slicing tree representation.

to the layout from the MIP. The limitation of this method, however, is that another slicing tree must be generated and it may require many dummy departments.

Since there are solutions that the MIP can represent and the slicing tree cannot, there will be cases when it is not clear how to represent the MIP solution in a slicing tree. For example, consider the layout in figure 11. Since there is not an initial guillotine cut in this layout, no slicing tree can represent it. Therefore, we need to find a cut that almost satisfies this property and initialize the GA population with a proportional number of solutions satisfying the two alternatives for this cut. For example, department 3 in figure 11 can be placed so that it is either under department 1 (and completely to the left of department 4) or under department 4 (and completely to the right of department 1). In either case, the resulting layout can then be represented by a slicing tree. Therefore, in our initial GA population, there would be an equal number of solutions with each cut and the above procedures can be used from that point forward to construct the rest of the GA population and/or new slicing tree.

4.3. *Iterative approach discussion*

The slicing tree structure gives us an easy way to specify the pairwise binary variables in the MIP formulation. Once some variables are set, through the MIP we can obtain an optimal solution given these fixed values. We present two methods for communicating the improvements back to our GA if the MIP solution can be represented or approximated with a slicing tree solution. Using such an iterative approach we are confident an overall good solution can be obtained.

Unfortunately, in most cases, it takes too long to solve the MIP formulation, and we do not see a very large improvement when many of the binary variables are fixed to ensure MIP solvability. However, the iterative approach suggests that once the MIP formulation is improved we will have an overall improved approach. Additional research on improving the solvability of the layout MIP formulation is needed.

5. Conclusions

In this paper we investigated an iterative approach for the facility layout problem. First, we developed three genetic algorithms based on a slicing tree representation (GAST): BASE-GA, MULT-GA, and EXT-GA. Our final results from GAST improved the best-known solution to 10 data sets from the literature. The final results from EXT-GA were combined with the layout mixed-integer programming (MIP) formulation. The iterative algorithm was developed and tested on small data sets.

Some further improvements can be made to EXT-GA. Instead of considering four cuts, we can consider additional cuts such that we can represent other department shapes (e.g., L-shaped departments). Also, instead of adding dummy departments, we may consider exchanges that change the structure of the tree directly, such as branch-for-branch or branch-for-node exchanges.

Although the performance of this iterative approach is limited by the MIP solvability, it can be used to help improve an algorithm that does not generate a near-optimal result. Moreover, the iterative approach can provide confidence for the results from layout heuristics since it will improve solutions that are not initially very close to being optimal.

Acknowledgement

This study was supported in part by NSF CAREER Grant DMII 9623605.

References

ARMOUR, G. C. and BUFFA, E. S., 1963, A heuristic algorithm and simulation approach to relative allocation of facilities. *Management Science*, **9,** 294–309.

BÄCK, T. and SCHWEFEL, H., 1993, An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, **1,** 1–23.

BANERJEE, P., MONTREUIL, B., MOODIE, C. L. and KASHYAP, R. L., 1992, A modelling of interactive facilities layout designer reasoning using qualitative patterns. *International Journal of Production Research*, **30,** 433–453.

BANERJEE, P., ZHOU, Y., KRISHNASAMI, K. and MONTREUIL, B., 1997, Genetically assisted optimization of cell layout and material flow path skeleton. *IIE Transactions*, **29,** 277–291.

BOZER, Y. A. and MELLER, R. D., 1997, A reexamination of the distance-based facility layout problem. *IIE Transactions on Design and Manufacturing*, **29**, 549–560.

BOZER, Y. A., MELLER, R. D. and ERLEBACHER, S. J., 1994 An improvement-type layout algorithm for single and multiple floor facilities. *Management Science*, **40,** 918–932.

COIT, D. W., SMITH, A. E. and TATE, D. M., 1996, Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal on Computing*, **8,** 173–182.

GAU, K.-Y., 1998, An iterative facility layout algorithm. PhD dissertation, Auburn University, Auburn, AL, USA (unpublished).

GOETSCHALCKX, M., 1992, An interactive layout heuristic based on hexagonal adjacency graphs. *European Journal of Operational Research*, **63,** 304–321.

GOLDBERG, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley).

HOLLAND, J. H., 1973, Genetic algorithms and the optimal allocations of trials. *SIAM Journal on Computing*, **2,** 88–105.

KAMOUN, M. and YANO, C. A., 1996, Facility layout to support just-in-time. *Transportation Science*, **30,** 315–329.

KUSIAK, A. and HERAGU, S. S., 1987, The facility layout problem. *European Journal of Operational Research*, **29,** 229–251.

LANGEVIN, A., MONTREUIL, B. and RIOPEL, D., 1994, Spine layout design. *International Journal of Production Research*, **32,** 429–442.

MELLER, R. D. and BOZER, Y. A., 1996, A new simulated annealing algorithm for the facility layout problem. *International Journal of Production Research*, **34,** 1675–1692.

MELLER, R. D. and GAU, K.-Y., 1996, The facility layout problem: recent and emerging trends and perspectives. *Journal of Manufacturing Systems*, **15,** 351–366.

MELLER, R. D., NARAYANAN, V. and VANCE, P. H., 1998, Optimal facility layout design. *Operations Research Letters*, **23**, 117–127.

MONTREUIL, B., 1990, A modelling framework for integrating layout design and flow network design. *Proceedings from the Material Handling Research Colloquium*, Hebron, KY, pp. 43–58.

MONTREUIL, B., VENKATADRI, U. and RATLIFF, H. D., 1993, Generating a layout from a design skeleton. *IIE Transactions*, **25,** 3–15.

TAM, K. Y., 1992a, A simulated annealing algorithm for allocating space to manufacturing cells. *International Journal of Production Research*, **30,** 63–87.

TAM, K. Y., 1992b, Genetic algorithms, function optimization, and facility layout design. *European Journal of Operational Research*, **63,** 322–346.

TATE, D. M. and SMITH, A. E., 1995, Unequal area facility layout using genetic search. *IIE Transactions*, **27,** 465–472.

TOMPKINS, J. A., WHITE J. A., BOZER, Y. A., FRAZEILE, E. H., TANCHOCO, J. M. A. and TREVINO, J., 1996, *Facilities Planning*, 2nd edn (New York: Wiley).