ASSIGNING COMPONENTS TO ROBOTIC WORKCELLS FOR ELECTRONIC ASSEMBLY

Arvind Rajan and Moshe Segal

Arvind Rajan *and* ***Moshe Segal*** *are members of the Operations Research Department at AT&T Bell Laboratories in Holmdel, New Jersey, where Mr. Rajan is a member of technical staff and Mr. Segal is supervisor of the Operations Research Methods group. Mr. Rajan, who joined AT&T in 1986, is responsible for research and internal consulting in applied operations research, particularly in the areas of combinatorial optimization, mathematical programming, and more recently, manufacturing and marketing science. He holds a B.Tech in mechanical engineering from the Indian Institute of Technology, Madras, and M.S. and Ph.D. degrees in operations research from Northwestern University, Evanston. Mr. Segal is responsible for research in queueing and optimization methods and for developing related* (continued on page 102)

In a multiproduct assembly line, circuit packs are routed to robotic workcells for component insertion. Each type of circuit pack is routed only to the workcells having one or more of the needed components. We describe a problem of partitioning a given set of components among a group of identical workcells. The assignment should ensure load balance among the workcells and minimize the total workcell visits by circuit packs. We model this problem as an integer program and present an iterative heuristic algorithm that solves a sequence of network-flow models. Our experimental results show that the algorithm performs well on medium to large test problems constructed from real data.

Introduction

In recent years, the gap between research and practice in optimization has been closing. Even when the results of the research are discouraging—indicating, for example, that efficient methods for deriving exact solutions are unlikely to exist—the insights gained can still suggest heuristic algorithms to approximate the optimal solution. This point is illustrated here by a class of set-partitioning problems arising in electronic assembly and other manufacturing applications, for which we derive an effective heuristic based on mathematical programming techniques.

In a circuit pack assembly process, robotic workcells are used to assemble components that are unusually shaped or infrequently used. A robotic workcell (Figure 1) consists of a robot and several customized component feeders. Each component type is in its own special feeder located within the robot's reach. The products made on the line consist of several different circuit pack *codes* (types). Each code has a list of components to be inserted, with some components appearing in more than one code.

The number of feeders—therefore, the number of component types a robot may insert—is limited by the size of the robot's working

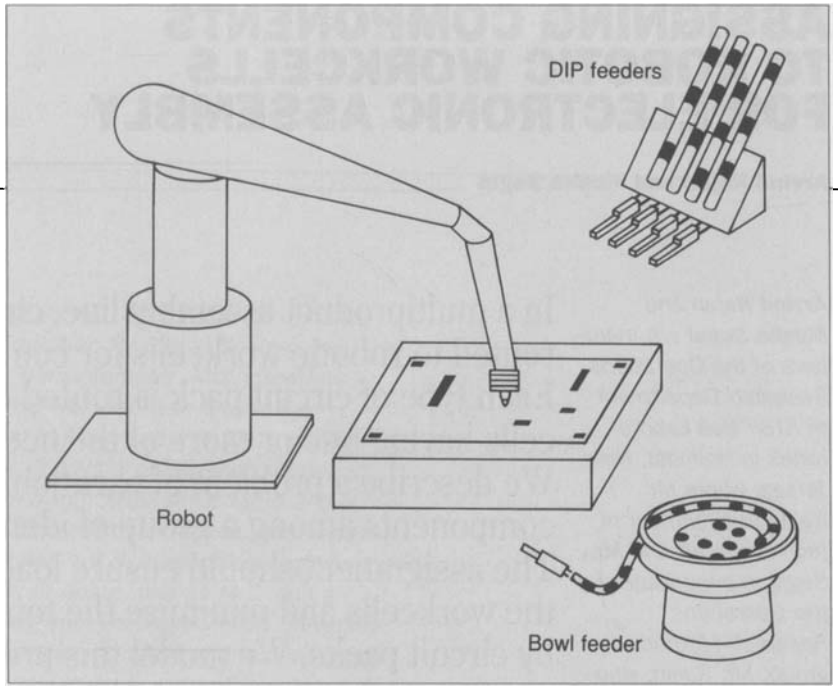

Figure 1. A robotic workcell.

envelope (i.e., by its reach). Each feeder must initially be fixtured and its location registered with the robot. The robot must then be trained to pick up components from the feeder and place them at the proper location on a circuit pack. This setup is done for each component type and each code; the task may take 30 minutes to an hour (by contrast, each pick and place operation takes 4 to 8 seconds). It is preferable that each component be assigned—either permanently or on a long-term basis—to a feeder in a given workcell, since reassigning a component means incurring another setup. The mix and volume of codes are assumed to remain uniform over a reasonable interval (e.g., a few weeks).

A typical robotic assembly process might include a number of such workcells, arranged so they can be accessed independently by the material handling system. The circuit packs travel in *magazines*, each carrying perhaps 10 or 20 circuit packs of the same code. Each code would need to visit *only* those workcells that have been assigned one or more components the code requires. Figure 2 shows a schematic illustration with six components, four codes (K1 to K4), and two workcells. Two questions related to the workcell and component arrangement are:

- How many workcells are needed?
- Given J workcells, how should the components be assigned, i.e., partitioned, among the workcells?

In practice, the constraints on the number of feeders and insertion rate of the robot considerably narrow the allowable range of values of J. Suppose that for each J in this allowable range, we could somehow assign all the components in the best possible way. The assignment would specify the subset of workcells that each code must visit. We could then analyze the corresponding configurations to determine what choice (of J) would let us best carry the workload without wasting capacity. Thus, the real challenge we focus on in the remainder of this paper is assigning components to a *fixed* number of workcells.

The component assignment should meet two objectives:

- It should ensure load balance, i.e., that each workcell performs approximately the same number of component insertions. This prevents workcells from becoming bottlenecks or being starved. It also reduces the production interval and work-in-process inventory.

Panel 1. Terms and Acronyms in This Paper

CONNET	constrained network
CPU	central processing unit
FFD	first fit decreasing
FMS	flexible manufacturing system
LP	linear program
NP	non-deterministic polynomial

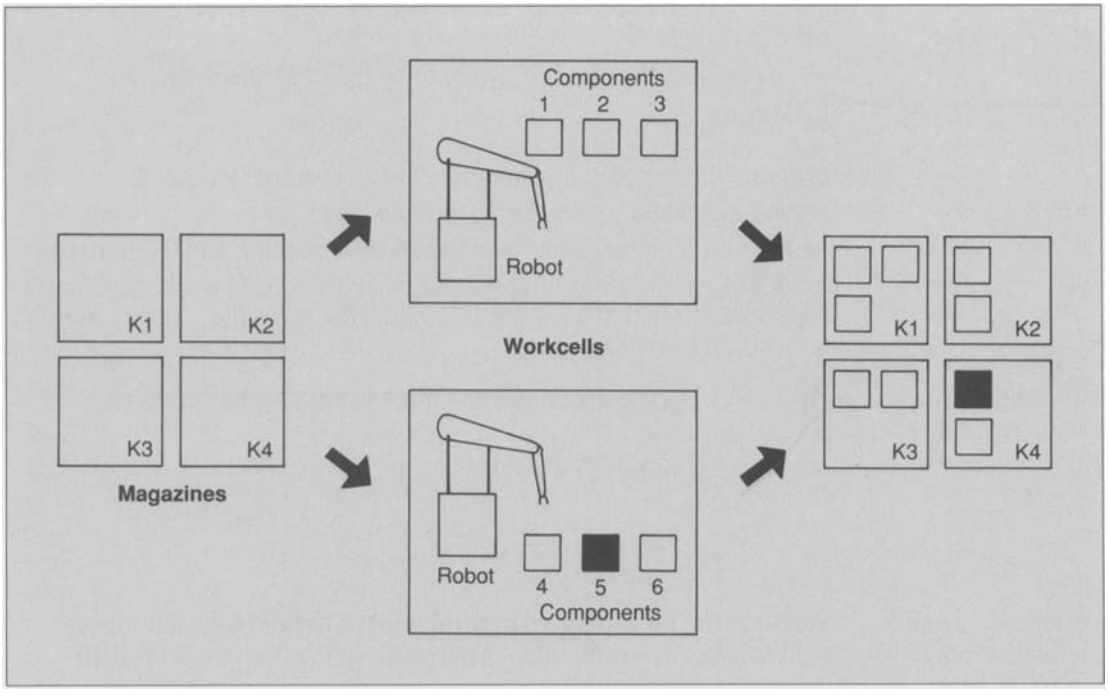

Figure 2. The robotic assembly process.

- It should minimize the number of visits by circuit pack magazines to workcells. This minimizes the amount of processing that adds no value, such as transport, magazine loading and unloading, and information transactions.

In Figure 2, for example, each code needs to visit only one workcell because of the way the components have been assigned.

We propose an integer programming model that meets both these objectives by modelling the load balance as a constraint, while incorporating minimization of the number of workcell visits in the objective function. Making load balance a constraint is preferable because we can calculate the average load and estimate the maximum allowable (or desired) variation in insertion volume for each workcell. The number of workcell-visits, on the other hand, is a very difficult quantity to estimate.

We also present a new algorithm to solve the problem. The algorithm modifies and relaxes the integer program to obtain a smaller linear program (LP). We then solve a sequence of such LPs, whose solutions converge to a solution of the original integer program. The solution at each step is used to construct the next LP in the sequence. The constraints of each LP correspond to a single commodity network-flow problem with additional linear side constraints. The structure makes the computational effort for each step relatively small. A greedy heuristic procedure is used to give the algorithm a good starting solution.

We have implemented the algorithm as a program written in FORTRAN 77. A special purpose LP code known as the Constrained Network (CONNET) that solves network-flow problems with additional constraints, is called as a subprogram.[1] The algorithm performs well on a range of test problems ranging from small experimental examples to large problems derived

Panel 2. Problem Parameters and Variables

$k = 1, \cdots, K$	Index of codes.
$i = 1, \cdots, I$	Index of components.
$j = 1, \cdots, J$	Index of workcells.
b_k	Total production volume of code k.
a_{ik}	The ikth entry of component-code incidence matrix A, i.e., it has value 1 if component i must be inserted on code k, 0 otherwise.
m_{ik}	Multiplicity (number of units) of component i to be inserted on a circuit pack of code k.
v_i	The volume of components of type i to be inserted, given by $v_i = \sum_{k=1}^{K} m_{ik} b_k$.
c	Maximum number of component feeders allowed per workcell.
p	Allowable percentage deviation in workcell load.
x_{ij}	Takes value 1 if component i is assigned to workcell j, and 0 otherwise.
y_{jk}	Takes value 1 if the assignment requires code k to visit workcell j, and 0 otherwise. Note that y_{jk} is 1 if and only if for some i, a_{ik} and x_{ij} are both strictly positive.

In addition, we define $V_{\max}$ and $V_{\min}$, the maximum and minimum allowable workcell loads respectively:

$$V_{\max} = (1/J)\ (1 + p/100) \sum_{i=1}^{I} v_i$$

$$V_{\min} = (1/J)\ (1 - p/100) \sum_{i=1}^{I} v_i$$

where $V_{\max}$ and $V_{\min}$ represent the range of insertion volumes permitted for each robot (equals average volume per workcell plus or minus p percent). In practice we set p at 10 to 20 percent.

from real data. We report on our computational experience and discuss the solutions obtained.

Problem Formulation

We now give a mathematical programming formulation of the assignment problem when J is fixed, and the robots are assumed to be identical. Notation for the problem is displayed in Panel 2. The basic integer problem, which is referred to throughout the remainder of the article, is displayed in Panel 3.

Observations on the Basic Problem

- Inequality (1) represents the constraint on the maximum number of feeders per workcell.
- Equation (2) is the requirement that each component be assigned *once* to a workcell. (Note that when a component's insertion volume is large, it may be necessary to assign it to multiple feeders or even to multiple workcells. This can be accommodated in our framework by creating multiple dummy component types to represent that component. Then we preprocess the incidence matrix A by partitioning the set of codes that carry that component and assigning a dummy component to each of the partitions. However, this amounts to suboptimizing the original problem.)
- Inequalities (3) and (4) are upper and lower bounds on the total number of insertions performed by each robot to ensure that their workloads are balanced. Unlike (1), constraints (3) and (4) are "soft" and amenable to designer adjustment.
- Inequality (5) summarizes the relationship between the codes and workcells. That is, it forces a code to

Panel 3. Statement of the Integer Problem

Objective:

Minimize $$\sum_{k=1}^{K} b_k \sum_{j=1}^{J} y_{jk} \quad \text{(IP)}$$

subject to $$\sum_{i=1}^{I} x_{ij} \le c, \quad j = 1, \cdots, J \quad (1)$$

$$\sum_{j=1}^{J} x_{ij} = 1, \quad i = 1, \cdots, I \quad (2)$$

$$\sum_{i=1}^{I} v_i x_{ij} \le V_{\max}, \quad j = 1, \cdots, J \quad (3)$$

$$\sum_{i=1}^{I} v_i x_{ij} \ge V_{\min}, \quad j = 1, \cdots, J \quad (4)$$

$$y_{jk} - a_{ik} x_{ij} \ge 0,$$
$$i = 1, \cdots, I;\ j = 1, \cdots, J;\ k = 1, \cdots, K \quad (5)$$

$$x_{ij}, y_{jk} \in \{0,1\},$$
$$1, \cdots, I;\ j = 1, \cdots, J;\ k = 1, \cdots, K \quad (6)$$

visit a workcell if any component associated with it is assigned to that cell.

Define $w_{jk} = \sum_{i=1}^{I} a_{ik} x_{ij}$, the number of component-types inserted on code k at workcell j. Observe that $w_{jk} = \sum_{i=1}^{I} a_{ik} x_{ij} \le \sum_{i=1}^{I} x_{ij} \le c$ by (1), so that $w_{jk}/c \le 1$. We use this to aggregate the constraints (5) and obtain the equivalent formulation (5′) below.

$$y_{jk} \ge \frac{\sum_{i=1}^{I} a_{ik} x_{ij}}{c}, \quad j = 1, \cdots, J;\ k = 1, \cdots, K \quad (5')$$

Note that y_{jk} will be 0 (code k does not visit workcell j) when $w_{jk} = 0$ because $b_k \ge 0$ and because we are minimizing. In short, formula (5′), together with the integrality condition on the y_{jk}, is equivalent to (5″) below, where $\lceil . \rceil$ represents rounding up to the next higher integer:

$$y_{jk} = \left\lceil \frac{\sum_{i=1}^{I} a_{ik} x_{ij}}{c} \right\rceil, \quad j = 1, \cdots, J;\ k = 1, \cdots, K \quad (5'')$$

Observe that these JK constraints (5′) or (5″), are present only to define the y_{jk}. In this form [with (5′)], the formulation (IP), though smaller than the original, will still be difficult to solve for realistic values of I, J, and K.

Literature Review

Models closely related to the one described appear in the production literature under the topic of

flexible manufacturing systems (FMS). Lofgren and McGinnis[2] describe a generalized version of our problem and formulate it as a nonlinear integer program. In a later paper with Ammons,[3] they develop a solution procedure for the generalized version and related problems, based on graph-theoretic heuristics. Ammons, Lofgren, and McGinnis point out that the model applies not only to electronic assembly, but also to a variety of other manufacturing applications, including:

- *FMS workcenter loading.* Jobs move among workcenters, and have the appropriate operations performed on them. Assign operations to the workcenters to minimize visits by jobs to workcenters and to balance the workloads among workcenters.
- *Part-picking systems.* Given a warehouse with an order profile and aisles serviced by a crane, find an assignment of parts to aisles that minimizes the average number of aisles that must be traveled to fill an order and balance activity across all aisles.
- *Kitting systems.* Components that form kits for an FMS are stored on trays in a miniload system. These are circulated to different robotic kitting stations by automatic guided vehicles. Assign components to trays to minimize the number of trays required to make a kit.

Other related work appears in the group technology literature. The most common solution approach is to approximate the relationships between the codes and the components—e.g., by using similarity coefficients—and apply graph heuristics. Other techniques discussed range from syntactic pattern recognition to q-analysis. For a full review, see King and Nakornchai.[4]

It is well known that large integer programs are generally difficult to solve in practice and are intractable in the theoretical sense. It can be demonstrated that (IP) is no exception; that is, it is in the class of nondeterministic polynomial (NP)-hard problems, as defined by Garey and Johnson.[5] The literature contains a number of papers on the constrained-set partitioning problem, to which our problem is related. Similar problems include capacitated plant/facility location problems as described in Aikens' survey[6] and in Mulvey and Beck's work on capacitated clustering problems.[7] These are all mixed integer programs with special structure. Known solution techniques range from standard greedy and interchange heuristics to more sophisticated mathematical programming techniques such as Lagrangian and dual ascent algorithms.

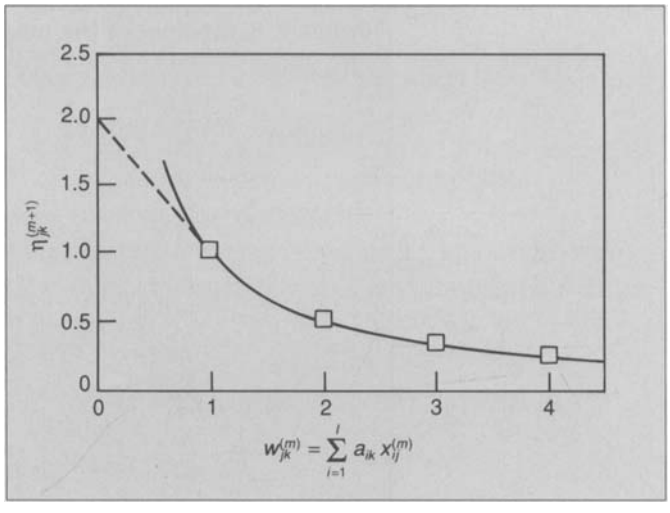

Figure 3. The iteration adjustment factor.

Development of Algorithm

We modify (IP) to obtain a sequence of linear programs (LP^m) for ($m = 1, 2, \cdots$), defined as follows:

$$\text{Minimize} \quad \sum_{k=1}^{K} b_k \sum_{j=1}^{J} \eta_{jk}^{(m)} \sum_{i=1}^{I} a_{ik} x_{ij} \qquad (LP^m)$$

which is subject to constraints (1), (2), (3), (4), and

$$x_{ij} \geq 0, \quad j = 1, \cdots, J; \; i = 1, \cdots, I \qquad (6')$$

To obtain (LP^m), we have substituted (5″) into the objective function to eliminate the y_{jk}, removed

(relaxed) the "ceiling," and put in its place the linear coefficient $\eta_{jk}^{(m)}$. Here, $\eta_{jk}^{(m)}$ is termed an *iteration adjustment factor*. Thus, (LP^m) minimizes a modified linear relaxation of the number of workcell visits, in contrast to (IP), which measures them exactly. To solve (IP), we propose to solve the linear programs (LP^m) in sequence. Iteration m consists of solving (LP^m) and computing $\eta_{jk}^{(m+1)}$, which depends on the optimal solution to (LP^m).

The iteration adjustment factor $\eta_{jk}^{(m)}$ is defined as follows: Let $x^{(m)}$ be the optimal solution to (LP^m), and define $w_{jk}^{(m)} = \sum_{i=1}^{I} a_{ik} x_{ij}^{(m)}$, the value of w_{jk} at the end of iteration m. Then, $\eta_{jk}^{(m+1)}$, where ($m = 0, 1, 2, \cdots$) is defined by (see also Figure 3):

$$\eta_{jk}^{(m+1)} = \frac{1}{w_{jk}^{(m)}} \qquad \text{if } w_{jk}^{(m)} \neq 0,$$

$$= q \text{ (a parameter)} \qquad \text{if } w_{jk}^{(m)} = 0 \qquad (7)$$

An initial solution $x^{(0)}$ (and the associated $w_{jk}^{(0)}$ values) are obtained through a starting heuristic, described in the next section. Note that the problems (LP^m) are reduced in size from the original (the JK constraints are gone), and that the constraints yield a "transportation" matrix except for constraints (3) and (4). They are, therefore, relatively easy to solve.

The Significance of $\eta_{jk}^{(m+1)}$. The factor $\eta_{jk}^{(m+1)}$ adjusts the cost associated with the workcell-code pair j–k at each iteration based on the present solution. Let $x^{(m)}$ be a solution (vector) of (LP^m) and suppose that it is integral (0/1). Then the value of the innermost summation in the objective function is $w_{jk}^{(m)}$, the number of component types inserted on code k at workcell j. Notice that $\eta_{jk}^{(m+1)}$ is low when $w_{jk}^{(m)}$ is high. Therefore, in moving to a new solution in the $(m+1)$th iteration, it tends to preserve those code-workcell visits from the previous solution that result in many component insertions, while trying to eliminate those code-workcell visits that result in few insertions.

Recall that $w_{jk}^{(m)} = \sum_{i=1}^{I} a_{ik} x_{ij}^{(m)}$ in (LP^m) is a surrogate for y_{jk} in the original (IP) whose value is 1 whenever $w_{jk}^{(m)}$ is strictly positive. Thus, $\eta_{jk}^{(m+1)} = 1/w_{jk}^{(m)}$ provides exactly the required adjustment factor when $w_{jk}^{(m)} > 0$ (depicted by the solid curve in Figure 3). On the other hand, when $w_{jk} = 0$, we set $\eta_{jk}^{(m+1)} = q$, a parameter that determines how we treat those code-workcell pairs having no components in common in the solution to the mth iteration. We will say more about the choice of q later.

Computational Features

In this section we describe some features of our implementation. These include the starting heuristic, the method for integerization and fine-tuning of fractional solutions, and our choice of the user-defined parameter q in the algorithm.

Starting Heuristic. We designed a starting heuristic to produce $x^{(0)}$, which in turn is used to define $\eta_{jk}^{(1)}$ by (7) above. Our starting heuristic is inspired by the successful FFD (first fit decreasing) heuristic for bin packing, described by Johnson,[8] and also discussed by Garey and Johnson in their previously cited research. It proceeds as follows:

1. Order the codes by decreasing volume.
2. Assign all components corresponding to the first code to as few workcells as possible, given the constraints on the number of feeders and on insertion volume.
3. Assign the components of the code with the second highest volume (that are as yet unassigned) to as few workcells as possible, starting with the last partially filled workcell.
4. Proceed in this fashion down the list of codes, cycling back to the first workcell when no more empty workcells remain.
5. Stop when all components have been assigned.

This procedure works well in practice, especially when the codes do not share many components. Occasionally, the heuristic may fail to produce a feasible solu-

Table I. Algorithm Performance for Small Test Problems

No.	No. of robots	% dev. in vol.	# feeders per cell	Starting solution	Best solution	$\frac{Z_H}{Z_1}$
	J	p	c	Z_1	Z_H	
1	5	20	12	30419	18613	0.61
2	5	15	12	28328	21575	0.76
3	5	10	12	27614	20451	0.74
4	4	20	12	22833	18613	0.81
5	4	15	15	25947	19737	0.76
6	4	10	15	25232	21727	0.86

NOTE: Results are for 42 components and 5 codes. Objective values are in "Total number of magazine visits to workcells."

tion for a particular problem instance. When this occurs, a special feature gives the user enough information to nudge the solution into feasibility on the next try.

Integerization and Fine-Tuning of Solutions. Another issue is that of non-integral values of variables in the optimal solutions of (LP^m). A pure network-flow problem always has an integer optimal solution. But our formulation, which has a few linear non-network constraints, produces solutions that may have some fractional valued variables.

The value of a variable x_{ij} always lies between 0 and 1, and from equation (2), $\sum_{j=1}^{J} x_{ij} = 1$, where $1, \cdots, I$.

100 For each such i, we simply choose the j with the largest x_{ij} and set that x_{ij} to 1, and all the others to 0. This rounding procedure always gives a 0/1 integer solution that satisfies constraint (2). The solution may, however, violate constraint (1), (3), or (4).

Such infeasibilities are handled by a fine-tuning heuristic, which adjusts a solution that is close to feasibility but violates constraint (1), (3), or (4) for one or more j. If (1) or (3) is violated for some j, workcell j has, respectively, too many components or too much insertion volume assigned to it. The heuristic successively reassigns components originally assigned to j to another workcell until j is feasible. The new workcell is chosen to minimize any additional visits incurred due to the reassignment.

On the other hand, if constraint (4) is violated for some j, workcell j has too few insertions to perform. The fine-tuning heuristic chooses appropriate components originally assigned to other workcells and reassigns them to j, always preserving feasibility at the other workcells so the resulting number of additional visits is minimized. This is repeated until the solution is feasible. However, should the procedure fail to reach a feasible solution, the user can intervene and relax one of the "soft" constraints, either (3) or (4). In practice, this was never necessary.

The Parameter q. The parameter q has a profound influence on the algorithm's behavior. Choosing $q > 1$ has a stabilizing effect, because it tends to preserve the status quo for those code-workcell pairs k and j that have no components in common ($ax = 0$) by increasing the cost of the coefficients associated with these pairs. Thus, improvements in the objective are usually brought about by squeezing each code's components onto fewer workcells without radically changing the assignment. In practice the algorithm is quite insensitive to a whole range of values of q ranging from 1.5 to 20. We chose the value $q = 2$, since the tangent to the curve for $\eta_{jk}^{(m+1)}$ versus w_{jk} at the point $w_{jk} = 1$ has an intercept of 2 (depicted by the line in Figure 3).

On the other hand, setting q at or near 0 gives us a more "global" procedure. Here, workcell-code pairs with no components in common are not penalized. This allows the solution to alter radically from iteration to iteration. The resulting algorithm is more erratic and time-consuming, but can sometimes help produce solutions far away from the previous one.

Experimental Results

Prototype software was prepared to implement the algorithm. It consists of the FORTRAN 77 program ROBOTS, which interacts with the user and the LP package CONNET. CONNET is a FORTRAN 77 software package that solves single commodity network-flow problems with side constraints. The interactive prototype program ROBOTS has the starting, integerization, and fine tuning heuristics. A major iteration of the algorithm consists of running CONNET followed by ROBOTS. The alternate calls to the program ROBOTS and CONNET

Table II. Algorithm Performance for Large Test Problems

No.	No. of robots	% dev. in vol.	# feeders per cell	Starting solution	Best solution	$\frac{Z_H}{Z_1}$
	J	p	c	Z_1	Z_H	
1	12	20	13	58665	51133	0.87
2	12	15	13	59103	49946	0.84
3	12	10	13	58475	53891	0.92
4	11	20	13	55228	46014	0.83
5	11	15	13	57218	48558	0.85
6	11	10	13	58969	52040	0.88
7	10	20	13	57241	42392	0.74
8	10	15	13	53258	45863	0.86
9	10	10	13	53501	46370	0.87
10	9	20	13	48287	42178	0.87
11	9	15	13	46347	44282	0.95
12	9	10	15	47051	43238	0.92

NOTE: Results are for 100 components and 20 codes. Objective values are in "Total number of magazine visits to workcells."

are made by a UNIX® system `shell` procedure.

Tables I and II document the runs made on a set of problems based on real data. The problems in Table I have 42 components, 5 codes and 4 to 5 workcells. Those in Table II contain 100 components, 20 codes and 9 to 12 workcells. The runs represent various values for the parameters J, p, and c. The parameter q was set at 2. It is worth noting that for the cases solved in Table II, the original formulation (IP) would have 1,080 to 1,440 variables and 18,127 to 24,136 constraints. The formulation (5′) reduces the number of constraints to 307 to 376 through aggregation. Finally, each of the (LP^m) has only 109 to 112 variables and 127 to 136 constraints.

The algorithm performed well on the test problems, and succeeded in improving the solutions produced by the initial heuristic by 5 percent to 26 percent for the large problems and 14 percent to 39 percent for the small problems. This represents a substantial improvement over our starting heuristic. The FFD heuristic we used is among the most effective known for the standard bin-packing problem that is related to our problem (see References 5 and 8). We do not have a reliable estimate of distance from the optimum for our problem, for we did not have good lower bounds against which to compare our algorithm. However, based on an elementary counting argument, we were able to prove that we actually obtained an optimal solution in two runs on the smaller problems (Nos. 1 and 4 of Table I). Good lower bounds may be obtained by solving LP relaxations of the original problem and its dual. This work remains to be done.

Once in a while reducing p actually produced solutions with fewer workcell visits (e.g., Table II, Nos. 1 and 2). Normally, in LP models, tightening a constraint will worsen the value of the objective function. Our problem is a discrete one, and the heuristic for finding the initial solution can sometimes produce a better one when a constraint is tightened. Also the fine-tuning and integerization routines are called more often and perform local optimization while correcting infeasibilities. This explains the occasional nonmonotonicity of the solution as p is decreased.

The problems were solved on an Amdahl 5890 computer running UNIX System V, Release 5.2.6a. The average central processing unit (CPU) time per iteration for the large problems was 1.6 seconds for ROBOTS (maximum 13.1 seconds) and 12.3 seconds for CONNET (maximum 14.1 seconds). The smaller problems ran much faster—a total of between 1 and 2 seconds per iteration with ROBOTS and CONNET combined. The best objective value was always reached in a few iterations, after which the algorithm tended to stop moving or to cycle among a few solutions. At no time were more than 12 iterations necessary before this phenomenon occurred. The algorithm, therefore, was stopped after 12 major iterations.

Conclusions and Future Work

The algorithm developed here proved effective in solving the special IP model. Large problems of real-life dimensions were solvable in reasonable time. The solu-

tions provided substantial improvements over those obtained from a standard heuristic.

A more comprehensive computational study would solve more (and different) large problems. Linear programming and dual relaxations could be used to obtain good lower bounds to further validate the algorithm's empirical performance. A comparison against other standard techniques, such as Lagrangean relaxation and dual ascent, might be interesting. Finally, a provable theoretical performance measure would be worthwhile to obtain, even for a simplified version of the problem.

Acknowledgment

We would like to thank Bruce Macrae for the time he took to introduce us to and discuss the problem of robotic insertion, and for the data he made available to us.

References

1. S. Chen and R. Saigal, "A Primal Algorithm for Solving a Capacitated Network-Flow Problem with Additional Linear Constraints," *Networks*, Vol. 7, No. 1, Spring 1977, pp. 59-79.
2. C. B. Lofgren and L. F. McGinnis, "Production Planning for Flexible Manufacturing," *Material Handling Research Center (MHRC) Technical Report*, Georgia Institute of Technology, MHRC-TR-82-01, 1983.
3. J. C. Ammons, C. B. Lofgren, and L. F. McGinnis, "A Workcenter Loading Problem in Flexible Manufacturing Systems," *MHRC Technical Report*, Georgia Institute of Technology, MHRC-TR-85-13, 1985.
4. J. R. King and V. Nakornchai, "Machine-Component Group Formation in Group Technology: Review and Extension," *International Journal of Production Research*, Vol. 20, No. 2, March/April 1982, pp. 213-232.
5. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, N.Y., 1979.
6. C. H. Aikens, "Facility Location Models for Distribution Planning," *European Journal of Operational Research*, Vol. 22, No. 3, December 1985, pp. 263-279.
7. J. M. Mulvey and M. P. Beck, "Solving Capacitated Clustering Problems," *European Journal of Operational Research*, Vol. 18, No. 3, December 1984, pp. 339-348.
8. D. S. Johnson, *Near Optimal Bin Packing Algorithms*, Doctoral Thesis, Department of Mathematics, Massachusetts Institute of Technology, 1973.

Biographies (continued)

software and applications for communications and manufacturing systems. He has B.Sc. and Ingenieur degrees in mechanical engineering from the Technion—Israel Institute of Technology, Haifa, and a Doctor of Engineering in operations research from The Johns Hopkins University, Baltimore. Mr. Segal joined AT&T in 1961.

(Manuscript received December 19, 1988)