Maximizing the number of mixed packages subject to variety constraints

David J. Robb*, Dan Trietsch

Department of Management Science and Information Systems, The University of Auckland, Private Bag 92019, Auckland, New Zealand

Received July 1997; received in revised form May 1998

Abstract

We develop a polynomial-time algorithm to optimise a variant of the one-dimensional bin-packing problem with side constraints. We also develop a pseudo-polynomial procedure to actually implement that optimal solution. The specific application is the allocation of excess of a population of various types of cards (e.g., left over from a previous selling season) to fixed-sized “variety packs” which guarantee a given level of variety (i.e., no more than $k$ of any type of card). Some card types with large numbers (perhaps the most unpopular from the previous season) may have to be discarded to preserve the variety constraint. The method developed employs a test for feasibility of a given number of packs and includes a simple allocation procedure. A numerical example is provided along with (worst-case) complexity calculations. In addition, we solve a practical problem in which an organisation marketing Christmas cards sought to determine the impact of pack size and variety level on the level of unallocated cards.

Scope and purpose

Organisations involved in the stocking and sale of seasonal “style” items such as greeting cards, may periodically face excess stock of various items (e.g., from previous seasons), and may decide to market the items in packs guaranteed to contain a certain degree of variety. Our objective in this paper is to show how to maximise the number of card packs that can be formed from an assortment of excess stock, where there is a marketing-based variety constraint restricting the number of each type of card in each pack. The solution procedure developed is intuitively appealing and can be easily implemented on a spreadsheet. In addition to presenting a numerical example, we provide results from the application and implementation of the method in the card sales operations of a charitable organisation. The method could also have broader application in other settings where variety is sought – such as in groups or teams. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Bin packing; Polynomial algorithm; Pseudo-polynomial; Style goods; Seasonal products

*Corresponding author. Tel.: +64-9-373-7599x.5990; fax: +64-9-373-7430.
E-mail address: d.robb@auckland.ac.nz (D.J. Robb)
1. Introduction

The management of style goods which have limited appeal after a season has attracted much attention during the past decades (e.g. [1, 2]). In particular, newsboy models have been widely investigated as methods to provide optimal tradeoffs between lost sales and overstocking. In the case of managing multiple items, disposition becomes an annual problem as it is unlikely that any season will end without excesses in numerous items. One policy exploited, particular for low value items, is to sell excess stock at a discounted price in packs or bundles. In the case of products such as festival cards (Christmas, Easter, Hanukkah, etc.) marketers may seek to mitigate the risk of a customer inheriting too many of an “unpopular” card by promoting variety (e.g., insisting that each pack of cards has “no more than $k$ of any given kind”, or “at least $y$ different types of card”).

The impact of “variety” constraints is to force some degree of “wastage”, i.e., items that cannot be allocated to any packs. However, organisations forming packs in an ad hoc fashion, may well be incurring more wastage than necessary. Here we present a polynomial method (which we call the optimality procedure) to minimise the level of wastage (equivalently maximise the number of packs that can be formed) and describe exactly how to fill the packs (note that we do not force each pack to be identical in terms of content) – by a pseudo-polynomial allocation procedure. The method used is easily explained and has been implemented on a spreadsheet. Following a small numerical example, we describe how it was used as a decision tool in a charitable organisation which derives a significant portion of its income from card sales.

The problem is a variant of a bin-packing problem – where one is concerned with the partitioning of a set of items into equal subsets. In our case we permit discarding those items which cannot fit and we assume duplication within each bin to be subject to a (side) constraint. Our problem also has connections with cutting stock (trim) and knapsack problems which have been extensively researched ([3, 4]).

We employ the following nomenclature (note that $p$, $k$, $n$, and $c_i$ constitute the input parameters to the problem):

**Notation**

- $p$: number of items in each package (prescribed)
- $k$: maximum number of items of any single type in a package (the variety constraint)
- $n$: number of types of items
- $c_i$: number of items of type $i$ ($i = 1, \ldots, n$), sorted in non-increasing order from left to right (we may refer to these as “columns”)
- $T_i = \sum_{j=1}^{n} c_j$, i.e., the number of items in the tail to the right of item $i$, not inclusive (the total number of items is given by $T_0$)
- $N$: a trial value for the number of packages
- $N^*$: the optimal (maximal) number of packages
- $N^U$: an upper bound on $N$ (not necessarily feasible)
- $N^L$: a lower bound on $N$ (guaranteed to be feasible)
- $m$: the number of types that have too many items for some $N$ (i.e., $c_m > kN$ and $c_{m+1} \leq kN$) alternatively, $m = |\{i: c_i > kN\}|$
\( m^U \) an upper bound on \( m \) (associated with \( N^L \))
\( m^L \) a lower bound on \( m \) (associated with \( N^U \))
\( r \) the minimal number of unallocated cards (i.e., after the allocation procedure)
\( x_i \) total number of type \( i \) cards used
\( y_{ij} \) the number of type \( i \) card in package \( j \)

2. A formal algorithm

The problem can be formulated in terms of two separate and sequential sub-problems: the first to determine the maximal feasible number of packages (i.e., to optimise), and the second to allocate items to such packages. We present (in Section 2.1) an integer programming formulation, and then describe efficient procedures to solve the optimality (Section 2.2) and allocation (Section 2.3) subproblems.

2.1. Integer programming formulation

To determine the (maximal) number of packages used \( (N) \) and the number of cards of type \( i \) \( (x_i) \) used, we have to solve the following integer program:

\[
(P1) \quad \text{max } N
\]
\[
\text{s.t. } \quad x_i \leq Nk, \quad i = 1, 2, \ldots, n \ (\text{variety}),
\]
\[
\quad x_i \leq c_i, \quad i = 1, 2, \ldots, n \ (\text{availability}),
\]
\[
\quad \sum_{i=1}^{n} x_i = Np \quad (\text{create full packages}),
\]
\[
\quad x_i, N \geq 0, \text{ integer.}
\]

Once we have \( N^* \) and \( x_i^* \), we must solve the following problem allocating cards to packages, where \( y_{ij} \) is the number of cards of type \( i \) allocated to package \( j \).

\[
(P2) \quad \sum_{j=1}^{N^*} y_{ij} = x_i^*, \quad i = 1, 2, \ldots, n \ (\text{piece usage}),
\]
\[
\quad \sum_{i=1}^{n} y_{ij} = p, \quad j = 1, 2, \ldots, N^* \ (\text{package size}),
\]
\[
\quad y_{ij} \leq k, \quad i = 1, 2, \ldots, n \quad j = 1, 2, \ldots, N^* \ (\text{variety}),
\]
\[
\quad y_{ij} \geq 0 \quad \text{integer.}
\]
2.2. Optimality procedure

Theoretically, solving the LP relaxation of P1 provides a tight upper bound for the IP. Furthermore, let $\lfloor N_{LP} \rfloor$ be the truncated LP solution, then $\lfloor N_{LP} \rfloor = N^*$. These claims are true if and only if the solution of the LP relaxation with $N = \lfloor N_{LP} \rfloor$ in the first $n$ constraints (i.e., $x_i \leq \lfloor N_{LP} \rfloor k$) and the last constraint (i.e., $\sum x_i = \lfloor N_{LP} \rfloor p$) is feasible and integer. Feasibility here implies that the optimal objective function value will be $N^*$ (and not less) as defined by the truncation. Note now that instead of the two constraints $x_i \leq kN$ and $x_i \leq c_i$ we can write the single constraints $x_i \leq \min\{kN, c_i\}$. Therefore, there exists an optimal LP relaxation solution that takes into account exactly one of these two constraints for each $x_i$ (i.e., either variety or availability). The IP then becomes an instance of the transportation problem (with $n$ sources and two destinations including a dummy destination) and therefore the existence of an integer optimal solution is assured. Since $\lfloor N_{LP} \rfloor$ is feasible for the LP relaxation $N^* = \lfloor N_{LP} \rfloor$ must also be feasible (in terms of the individual constraints on $x_i$). The constraint $\sum x_i = N^* p$ then guarantees that the objective function will indeed yield $N^*$ exactly. (This result holds even after generalising the model as discussed in Section 2.6 below, but we omit the details.)

Since LP is polynomial, we obtain a polynomial solution for our sub-problem. Note, however, that although there exists an equivalent transportation problem for the IP we must solve the LP relaxation by a general LP algorithm because we do not know in advance which constraints apply to each $x_i$. [Note, in the numerical example given in Section 2.4 the LP requires 27 variables ($N$ itself is a variable) and 53 constraints, and the solution takes more than 70 simplex iterations (a slightly altered version with 25 variables and 49 constraints took 68 iterations to converge using a generic solver)].

By exploiting the special structure of the problem however, we can do much better. Our specialised polynomial algorithm for the optimality procedure is based upon a simple observation: when attempting to get $N$ packages, if there exists any column $i$ such that $c_i > kN$, then we can utilize at most $kN$ of these items; the remainder can be discarded. This leads to the observation that $N$ is feasible if and only if

$$\frac{\sum_{i=1}^{n} \min\{kN, c_i\}}{p} \geq N. \quad (3)$$

A search procedure, starting with $N^U = \lfloor T_0/p \rfloor$ (where we ignore the variety constraint) can now be used to find the maximal feasible $N$, i.e., $N^*$. A simple approach is to conduct the search by updating $N^U$ iteratively until it becomes feasible and thus optimal. But first, for convenience, we replace Eq. (3) by

$$N \text{ is feasible if and only if } \frac{mkN + T_m}{p} \geq N, \quad (4)$$

where the numerator is the total number of usable items for $N$ packages: the first $m$ items contribute $kN$ items each, and the tail to their right is added to the product. In this connection, by definition, $m$ is the number of types that have more than $kN$ items. A new iteration is started whenever the (continuously updated) value of $N^U$ is still infeasible. $N^U$ is then updated by

$$N^U = \lfloor \frac{mkN + T_m}{p} \rfloor. \quad (5)$$
Nonetheless, a slight modification yields lower complexity, and this is the version we present in more detail. It involves bounding \( m \) and \( N \) and searching for the largest feasible \( N \) (i.e., \( N^* \)), within its bounds, using the bisection method. Within each iteration the same method is used to find \( m \). The bounds for \( N \) become tighter in progressive iterations and the distance between the bounds for \( m \) is monotone non-increasing.

Initially, sort the \( c_i \) values in reverse size order and reindex them such that the first will be largest and the last smallest (this takes \( O(n \log[n]) \)). (Actually, it is enough to sort the first \( p \) largest values. The sequence beyond column \( p \) is immaterial to our needs.) Let \( N^U = \lfloor T_0/p \rfloor \). Let \( m^L \) be the index of the last column such that \( c_m > kN^U \) (if no such column exists, \( N^* = N^U \), and we may proceed directly to the allocation procedure). To find \( m^L \), start by checking whether \( m^L \geq 1 \), which is true if and only if \( c_1 > kN^U \). Henceforth, we assume that this condition is met (or, as mentioned, we already have feasibility for \( N^* = N^U \)).

Next, it is necessary to search for \( m^L \) over the range 1 to \( p - 1 \) (\( m \) can never exceed \( p - 1 \), or we could create extra packages from the discarded items), which we do by iteratively halving the search range and comparing the median (or close to median) column in the current range, say column \( t \), to \( kN^U \). If \( c_t > kN^U \), then \( m^L \geq t \); otherwise, \( m^L < t \). The search ends when two consecutive median columns are adjacent to each other, and \( m^L \) is the largest one of them for which the column exceeds \( kN^U \) (this procedure takes \( O(\log[p]) \) to complete).

Find \( N^L \) from

\[
N^L = \text{Max} \left\{ c_p + \frac{T_p}{p}, \left\lfloor \frac{p/k}{c[p/k] + T[p/k]} \right\rfloor \right\}.
\] (6)

(To validate this bound note that both elements reflect feasible solutions that meet the criterion of Eq. (2).) Find \( m^U \), between \( m^L \) and \( p \), as the \( m \) value that is associated with \( N^L \) (similar to the way in which \( m^L \), associated with \( N^U \), was found previously). At this stage we have the initial values of \( N^L \), \( N^U \), \( m^L \) and \( m^U \), and we can proceed to iterate (we assume that \( N^U > N^L \), since otherwise they coincide with each other and with \( N^* \), so there is no problem). At this stage, \( N^U \) is untested, since we do not know whether it is feasible. (If \( N^U \) is updated during the procedure it becomes infeasible from then on).

**Optimality Procedure:**

1. If \( N^U - N^L > 1 \) or if \( N^U - N^L = 1 \) but \( N^U \) is untested, set \( N = \lfloor (N^U + N^L)/2 \rfloor \) and go to step 2; else (since \( N^U - N^L = 1 \) and \( N^U \) is infeasible) set \( N^* = N^L \), \( m = m^U \) and go to the allocation procedure.
2. If \( m^L = m^U \) set \( m = m^L \); else, find \( m \) by the bisection method, starting with \( m = \lfloor (m^L + m^U)/2 \rfloor \). Go to step 3 with \( m \).
3. Check the feasibility of \( N \) by Eq. (2). If \( N \) is feasible, set \( N^L = N \) and \( m^U = m \); else (\( N \) is infeasible) set \( N^U = N \) (and \( N^U \) is now known to be infeasible) and \( m^L = m \); return to step 1.

2.3. Allocation procedure

To solve P2 we propose the following method. Start with the items set next to each other in the same sorted sequence utilized in the optimality procedure. Next, we calculate whether there is
a remainder, \( r \)
\[
r = [mkN^* + T_m]_{\text{Mod}p}.
\] (7)

If \( r > 0 \) discard \( r \) items of any type or combination of types (on the one hand, one may prefer to use this opportunity to reduce further the number of items in the first \( m \) types; on the other hand, by discarding other types the items discarded may become more useful in the future. As discussed in Section 2.6, a weighting scheme may be employed to decide which items to discard). Next, combine all the items into one long array, in their sorted order. Allocate the first \( N^* \) items to the first bin in such a way that the first item in the array is on the bottom of the bin and the \((N^*)\)th item is on top. Repeat by allocating the next \( N^* \) items to the second bin, etc. Finally, create packages by picking the top item from each bin. (To prevent series of adjacent identical items in each package, it may be desirable to shuffle the bins before collecting items to packages.)

2.4. Numerical example

Let \( n = 26, p = 10 \) and \( k = 2 \). The 26 types are denoted by the letters A–Z (for simplicity they are already sorted in that order). Specifically, there are 23 A’s, 13 B’s, 7 C’s, 4 D’s, 2 E’s, 2 F’s, and one each of the remaining types, G–Z. Fig. 1 shows this as a histogram.
Table 1
Tabulation of $T_i$ (the number of items to the right of item $i$)

| $i$ | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | ...
|-----|----|----|----|----|----|----|----|----|----|----|----|----|
| $c_i$ | 23 | 13 | 7  | 4  | 2  | 2  | 1  | 1  | 1  | 1  | 1  | ...
| $T_i$ | 71 | 48 | 35 | 28 | 24 | 22 | 20 | 19 | 18 | 17 | 16 | ...

It is convenient to tabulate $T_i$ for $i = 0, 1, \ldots, p$. The result, starting with $T_0 = 71$, is given by Table 1.

$N^U = \lceil T_0/p \rceil = \lceil 71/10 \rceil = 7$, and it is untested. Column 1 has excessive items for this value ($c_1 = 23 > kN^U = 2 \times 7$), and therefore $n^L \geq 1$. Checking ($c_1 > kN^U$) consecutively for columns 5 (the median of 1 and $p - 1 = 9$), 3 (the median of 1 and 5) and 2, we find that none of them need to be reduced; therefore we conclude that $m^L = 1$. $N^L = 3$, as per

$$N^L = \text{Max}\left\{\frac{c_p + T_p}{p}, \left\lceil \frac{p/k}{c_{[p/k]} + T_{[p/k]}} \right\rceil \right\}$$

$$= \text{Max}\left\{1 + \frac{16}{10}, \left\lceil \frac{10/2}{2 + 22} \right\rceil \right\} = \left\lceil \frac{32}{10} \right\rceil = 3 \quad (8)$$

and checking consecutively for columns 5, 3 and 4 we find $m^U = 3$. Going into step 1 with these values we try $N = 5$ (the median of $N^L = 3$ and $N^U = 7$), and we find that it is feasible (with $m = 2$), since $(mkN + T_2)/p = (2 \times 2 \times 5 + 35)/10 = 55/10 = 5.5 \geq 5$. Therefore we set $N^L = 5$ and $m^U = 2$.

In the next iteration we try $N = 6$ (yielding $m = 2$) and find that it is infeasible, as $(2 \times 2 \times 6 + 35)/10 = 5.9 < 6$. Since $N^L$ and $N^U$ are now adjacent, and $N^U$ is infeasible, $N^L$ must be optimal. So, the optimality procedure terminates with $N^* = N^L = 5$ and $m = 2$. Note that here we are discarding 13 A’s ($c_1 - kN^* = 23 - (2 \times 5) = 13$), and 3 B’s ($c_2 - kN^* = 13 - (2 \times 5) = 3$).

At this stage we enter the allocation procedure. Our final total is $55 (71 - (13 + 3))$. Therefore, $r = 55_{\text{mod}}10 = 5$. In response, let us remove 2 A’s and 3 B’s (any five items could be removed, however), leaving us with 8 A’s, 7 each of B and C, 4 D’s, 2 each of E and F, and one each of the remaining types, G–Z (see Fig. 2).

A
A B C
A B C
A B C
A B C D
A B C D
A B C D E F
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Fig. 2. Numerical example after the first stage of the allocation procedure.
We arrange these in the array:

AAAAAAAABBBBBBBCCCCCCCDDDDDEEEFFGHIJKLMNOPQRSTUVWXYZ

Next, we reorganize the array into 10 bins in the following format (Fig. 3):

A B C D F K P U Z
A B C D F J O T Y
A A B C D E I N S X
A A B C C E H M R W
A A B C C D G L Q V

Fig. 3. Completed allocations for numerical example.

And our (five) packages are given by the rows, e.g., A B B C D F K P U Z.

2.5. Complexity calculations

In terms of complexity, there are three major parts of consider: (i) sorting the items in decreasing order; (ii) the optimality procedure; and (iii) the allocation procedure. The sorting takes $O(n \log[n])$. Step 1 of the optimality procedure may have to be repeated $O(\log[T_0/p])$ times, i.e., $O(\log[T_0])$. Step 2 may take $O(\log[p])$ for each iteration of step 1; therefore the combined complexity of the optimality procedure is $O(\log[T_0] \log[p])$. (Note: the search for $m$ is $O(\log[p])$ because $m$ cannot exceed $p - 1$, since if $m \geq p$ we discard at least $p$ items of different types so we can compose at least one additional package from the discarded items.)

As for the allocation procedure, arguably, it is really part of the packaging rather than solving the maximisation problem. Be that as it may, it takes $O(N*p) \leq O(T_0)$ to allocate the items to bins – since we must count through all of them for that purpose. The time to actually package the items also has a similar complexity, since each item must be picked and put into a package. In conclusion, the complexity of the initial sorting may dominate the whole combined procedure, while the complexity of the optimality procedure is $O(\log[T_0]\log[p])$ and that of the allocation procedure is linear in the number of cards (and thus pseudo-polynomial in terms of the input string). But note that each of these complexities is determined by a different combination of inputs, so the one that dominates may vary between instances.

To elaborate, while the optimality procedure is polynomial, the allocation procedure is inherently a function of the size of the input variables, i.e., is only pseudo-polynomial. This is because our problem is “high multiplicity” (see [5, 6]), meaning that a short input string (e.g., 23 items of type A) describes a potentially large number of identical items. During allocation we must physically handle all these items one-by-one (if you take the cards in groups you will immediately violate the variety constraint, and the time required for this is not bounded by a polynomial function of the length of the input string. Obviously, the complexity of the allocation procedure cannot be reduced below $O(Np)$, but in practice this pseudo-polynomiality is not a limitation.
2.6. Generalising the algorithm

The model developed in this paper may be extended to cope with a number of modifications as follows:

- **A lower bound on the number of pieces/cards in each pack.**
  Let \( k_i^L \) be the minimum number required of item \( i \) in each pack. We could formulate this in the IP above by adding the constraints \( x_i \geq Nk_i^L \). In our own optimality procedure we can solve the problem by the normal means and if none of these constraints is binding then there is no problem. If some are binding then we simply set
  \[
  N^* \leq \min_i \left[ \frac{c_i}{k_i^L} \right]
  \]
  and then employ the allocation procedure in the regular fashion.

- **An upper bound on the number of pieces/cards in each pack.**
  Again, this can be modelled in the LP above by modifying the first constraint to \( x_i \leq Nk_i^U \), where \( k_i^U \) is the maximum permissible number of item \( i \) in each pack. For our own algorithm, at the start of the optimality procedure we would not sort the items in decreasing order \( c_i \), but in decreasing order of
  \[
  \frac{c_i}{k_i^U}.
  \]
  Some of the formulae presented in Section 2 would then need to be adapted.

- **Different weights (values) prescribed for different pieces/cards.**
  Using the IP formulation all that is required here is to replace the objective function with maximising a weighted linear combination of \( x_i \). Our own procedure can readily handle this – simply by ensuring that any discarding prioritizes items of lowest value. Note that there are two types of discarding – one due to the *variety constraints*, and the other due to conditions of *integrality* (full packs). It can be shown that weights have no impact on items discarded for variety reasons.

3. Practical illustration

A branch of a New Zealand charitable organisation has 8365 Christmas cards (in 37 types) remaining unsold from the previous year. This year they wish to market packets of these “obsolete” cars at a discounted price (40–50%), ensuring that customers get as much variety as possible (customers are wary of purchases being loaded with too many of a given obsolete or “unpopular” card). The numbers for each type of card are (650, 530, 490, 480, 430, 410, 380, 370, 360, 320, 290, 285, 265, 260, 220, 220, 210, 185, 182, 180, 180, 165, 160, 155, 145, 120, 90, 90, 85, 85, 85, 66, 65, 60, 45, 40, 12).

Utilising a spreadsheet implementation of the optimality procedure, we generated the largest number of mixed packages for various combinations of pack size \( p \), and variety constraint \( k \). Fig. 4 depicts the response surface of card “wastage” (i.e., those left after the allocation procedure).
For a given population of cards, wastage monotonically increases with the pack size, but monotonically decreases with the maximum number permitted of any kind of card in any pack. These findings are intuitively appealing as they correspond to the costs of tightening and relaxing constraints on systems.

The most attractive option to the organisation was a pack size of 20, which (to their surprise) had no more than two cards of any kind (i.e., at least 10 different types of card in each pack), with only five unallocated cards ($N^* = 418$, and the five being an unavoidable remainder!). In previous years, the organisation had spread all the cards around a room, and had volunteers wander around gathering cards in an ad hoc manner. Here, our proposed method is implemented by arranging the 37 varieties in order of population size, then flipping the cards over into one long array and “cutting” the array into 20 piles of 418 each. Each pack of 20 is then formed by selecting one card from each of the piles.

4. Conclusion

We have demonstrated how to determine the maximum number of packs that can be formed when variety constraints are placed on the formation of packs of products from existing (excess) stocks. The solution procedure is easily implemented, as is the allocation of cards to the packs. We show how the model can be extended to incorporate variable diversity constraints, minimum quantity constraints for some items, and weighting schemes to support discarding decisions. Naturally there are “broader” questions that may be asked. For instance, incorporating financial
aspects such as pricing (one could conceive of modelling price as a function of \( p \) and \( k \)), picking/packing (lower \( k \) is presumably cheaper), and the “cost” (lower revenues) of wastage. One could also consider the possibility of variable values of \( k \).

Related applications of this work beyond the inventory management of seasonal cards and style goods are bound to exist; e.g., the desirability of heterogeneity in teams [7] and variety in assigning tasks to individuals and/or time periods. The procedures outlined here may also be appropriate when collecting donations (e.g., food and clothing) for disaster relief packages where there is a desire to give victims aid packages of roughly equal volume with diversity as an important factor.

Acknowledgements

The authors are grateful for comments from the referees and Edward A. Silver which have led to improvements in the paper. In particular, one referee noted that the true complexity of our solution is pseudo-polynomial and another suggested generalisations and an alternative optimality procedure as well as the formal IP models. Professor Silver drew our attention to the use of Eq. (3) as an equivalent, but simpler, way to present Eq. (4).

References


David Robb is a Senior Lecturer in Operations Management at The University of Auckland. He received MBA and PhD degrees in Operations Management from The University of Calgary, where he served as Assistant Professor until 1994. His research and consulting interests are in inventory management and operations strategy.

Dan Trietsch is a Senior Lecturer in Operations Management at The University of Auckland. He received a B.S.M.E from Technion in 1973, an M.B.A. from Tel Aviv University in 1978, and a Ph.D. in operations research from Tel Aviv University in 1983. His research and consulting interests include quality management and JIT techniques.