19.3/20

UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FIME
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Selected Topics on Optimization

Computational Experience with Heuristics for the Multi-Knapsack Problem (MKP)

ID	Name	Career
2144189	Oscar Alexis Aldana Almanza	ITS
2226784	Samuel Alejandro Casaos Esquivel	ITS
2147342	Fernanda Lozano Ovalle	ITS
2147975	Jose Angel Martinez Herrera	ITS

Team B – Group 002

Dr. Roger Z. Rios Mercado

May 12, 2026

Introduction

What are Knapsack Problems?

Suppose a hitchhiker must fill up his knapsack by selecting from among various possible objects those which will give him maximum comfort. This knapsack problem can be mathematically formulated by numbering the objects from 1 to n and introducing a vector of binary variables $x_j (j = 1, \dots, n)$ having the following meaning:

$$x_j = \begin{cases} 1 & if\ object\ j\ is\ selected; \\ 0 & otherwise \end{cases}$$

Then, if p_j is a measure of the comfort given by object j, w_j its size and c the size of the knapsack, our problem will be to select from among all binary vectors x satisfying the constraint

$$\sum_{j=1}^{n} w_j x_j \leq c,$$

the one which maximizes the objective function

$$\sum_{j=1}^{n} p_j x_j$$

$s.t.\ x_j \in \{0,1\}\ for\ every\ j = 1, \dots, n.$ (Martello & Toth, 1990).

The Multiple Knapsack Problem (MKP)

The Multiple Knapsack Problem (MKP) extends the classical model by considering a set of m distinct knapsacks, each with its own capacity c_i. The objective is to assign a subset of n items, each defined by a profit p_j and a weight w_j, to these knapsacks to maximize the total profit.

To ensure a valid solution, the following conditions must be met:

- ✓ The total weight of items assigned to any single knapsack i cannot exceed its specific capacity c_i.
- ✓ Each item j can be assigned to at most one knapsack, meaning an item cannot be shared or duplicated across different containers.

Key Applications

- Logistics and Loading: Assigning cargo to a fleet of trucks or shipping containers, where each vehicle has a different weight or volume limit.
- Production Scheduling: Allocating tasks to multiple machines, each with a finite amount of available processing time.
- Capital Budgeting: Selecting which investment projects to fund across several different departments or fiscal periods, each with its own budget constraint.
- Cloud Computing: Distributing virtual machine requests across multiple physical servers to optimize CPU and RAM usage.

Problem description

Data

The Multiple Knapsack Problem (MKP) is defined as follows:

given a set $N = \{1, \dots, n\}$ of items and

a set $M = \{1, \dots, m\}$ of knapsacks $(m < n)$

each item j in N has an associated profit p_j and weight w_j, and each knapsack i in M has capacity c_i (Martello & Toth, 1990, p. 157).

Decisions

For each pair (i, j) with i in M and j in N, define the binary variable:

$x_{ij} \in \{0,1\} \qquad i \in M$, j ∈ N (6.4)

The variable x_{ij} equals 1 if item j is assigned to knapsack i, and 0 otherwise. This binary restriction enforces the 0–1 nature of the problem.

Objective Function

The objective is to maximize the total profit:

maximize $\quad z = \Sigma_{i=1}^{m}\Sigma_{j=1}^{n} p_j x_{ij} \quad (6.1)$

This expression sums the profits of all items assigned to the knapsacks.

Constraints

Capacity Constraints:

$\Sigma_{j=1}^{n} w_j x_{ij} \leq c_i \qquad i \in M = \{1, \ldots, m\} \quad (6.2)$

These inequalities ensure that the total weight assigned to each knapsack does not exceed its capacity.

Assignment Constraints:

$\Sigma_{i=1}^{m} x_{ij} \leq 1 \qquad j \in N = \{1, \ldots, n\} \quad (6.3)$

Each item can be assigned to at most one knapsack.

Additional Assumptions

p_j and c_i are positive integers $\quad (6.5)$

$w_j \leq \max_{i \in M}\{c_i\} \quad$ for $j \in N \quad (6.6)$

$c_i \geq \min_{j \in N}\{w_j\} \quad$ for $\; i \in M \quad (6.7)$

$\Sigma_{j=1}^{n} w_j > c_i \qquad$ for $\qquad i \in M \quad (6.8)$

These assumptions prevent trivial infeasible cases and ensure that items and knapsacks are meaningfully defined.

Complete Mathematical Model

maximize $z = \Sigma_{i=1}^{m}\Sigma_{j=1}^{n} p_j x_{ij}$ (6.1)

subject to

$\Sigma_{j=1}^{n} w_j x_{ij} \leq c_i \quad i \in M = \{1, \dots, m\}$ (6.2)

$\Sigma_{i=1}^{m} x_{ij} \leq 1 \qquad j \in N = \{1, \dots, n\}$ (6.3)

$x_{ij}\ 0\ or\ 1 \qquad i \in M\ ,\ j \in N$ (6.4)

Problem example

To illustrate the mechanics of the Multi Knapsack Problem (MKP), we present an example based on the loading problem described by Samuel Eilon and Nicos Christofides in 1971.

The loading problem is defined as the allocation of given items with a known magnitude to boxes with limited capacity.

Then, there are two objectives that can arise: The first one is to minimize the number of boxes, for instance we have 100 containers that we will have to accommodate in some ships, so the goal is to minimize the number of ships needed to put all the containers.

On the other hand, the second situation is to minimize the number of items not accommodated in the boxes or in other words, to maximize the total utility of the accommodated items. In this case, we have a fixed number of ships where we must put our containers. Then, we must make choices to select which containers will be in which ship.

Then, the problem can be formally formulated as follows: "Assign *n* items of given magnitude $w_j (j = 1,2, \dots, n)$ to *m* boxes, each box having a capacity C_i, so that the capacity constraints are not violated and the total profit of the assigned items is maximized.

A set of *n* items is available for shipment, characterized by their respective weights (w_j) and profits (p_j).

In this instance, the weights (w_j) and the capacity (C_i) are expressed in tons and the profit (p_j) represents the total revenue generated by transporting each item and are expressed in U.S Dollars (USD).

Data

We consider $m = 2$ vessels, each defined by its maximum capacity C_i.

For this instance, we set $C_1 = 15$ and $C_2 = 20$ tons.

These attributes are detailed in the following table:

Item (j)	Weight (w_j) [Tons]	Profit (p_j) [$]
1	7	315
2	7	1834
3	10	364
4	6	4610
5	8	209

In this case, not all items can be accommodated because $\sum C_i < \sum w_j$, it guarantees that we need an optimization strategy to maximize the total utility within the limited available space.

Decision Variables

We define the binary decision variable x_{ij}, where $x_{ij} = 1$ if the item j is allocated in the vessel i, and $x_{ij} = 0$ otherwise.

In such a way the problem works, we must select which items will be in the vessels without exceeding the limits of the vessels.

Then, to demonstrate the model's implementation, we propose a feasible allocation that maximizes the utilization of the available capacity.

Vessel 1 ($C_1 = 15$): We select item 1 ($w_1 = 7$) and item 4 ($w_4 = 6$). The sum of weights is 13 tons, so it does not exceed the limit capacity of vessel 1.

Vessel 2 ($C_2 = 20$): We choose item 2 ($w_2 = 7$) and item 3 ($w_3 = 10$). The sum of weights is 17 tons, so it does not exceed the limit capacity of vessel 2.

In this scenario, we selected the items that had a bigger profit, that is why we didn't select item 5.

The allocation described above is represented by the binary variable x_{ij}.

Item (j)	Vessel (i= 1)	Vessel (i= 2)
1	1	0
2	0	1
3	0	1
4	1	0
5	0	0

Each item must be at most in one vessel $\sum_{i=1}^{m} x_{ij} \leq 1$ ensuring that an item is not in more than one vessel simultaneously.

Constraints Verification

We demonstrate that it is a feasible solution comparing the total weight for each vessel with each capacity:

Vessel 1: $w_1 + w_4 = 7 + 6 = 13$ tons ≤ 15 tons.

Vessel 2: $w_2 + w_3 = 7 + 10 = 17$ tons ≤ 20 tons.

Objective Function

The total performance of this allocation is measured by the objective function Z, which represents the total revenue accrued.

$$Z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_j \, x_{ij}$$

We substitute the value of the items selected:

$$Z = 315 + 1834 + 364 + 4610$$

$$Z = \$\,7123$$

Results

The results of this study demonstrate that the allocation of the four selected items (1, 2, 3, and 4) provides a total revenue of $7,123 USD, while maintaining the integrity of the physical constraints. With capacity utilization exceeding 85% for both vessels, the solution proves to be highly effective for this specific instance.

Constructive Heuristic

The constructive stage of the algorithm is designed to generate an initial feasible lower bound for the 0-1 Multiple Knapsack Problem. The efficiency of this phase relies on a greedy sequential allocation strategy governed by the following mathematical framework.

1. Initial Sorting and Efficiency Criterion

Before the allocation begins, the set of n items must be reindexed according to their relative efficiency. This ensures that the greedy selection prioritizes items that maximize profit per unit of weight:

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}$$

where p_j and w_j denote the profit and weight of item j, respectively.

2. Formal Greedy Algorithm

The constructive process fills each knapsack $i \in \{1, \ldots, m\}$ one by one. The decision variables x_{ij} are determined as follows:

Step 2.1: Initialization

For the first knapsack ($i = 1$), the residual capacity is set to the total capacity: $\overline{c_i} = c_i$

All decision variables are initially $x_{ij} = 0$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

Step 2.2: Sequential Item Assignment

For each knapsack i from 1 to m, the algorithm scans all items j from 1 to n. An item j is assigned to knapsack i if and only if it satisfies both the availability and the capacity constraints:

Availability Constraint: The item has not been assigned to any previous knapsack:

$$\sum_{k=1}^{i-1} x_{kj} = 0$$

Role of index k: The index k serves as a counter representing the subset of knapsacks

$\{1, \ldots, i-1\} \subset M$ that has already been processed in the sequential allocation.

Capacity Constraint: The weight of the item does not exceed the current residual capacity of the knapsack:

$$w_j \leq \overline{c_i}$$

Step 2.3: State Update

If both conditions in Step 2.2 are met, the assignment is formalized:

$$x_{ij} = 1$$

The residual capacity for the current knapsack is immediately updated before evaluating the next item:

$$\overline{c_i} = \overline{c_i} - w_j$$

3. Objective Function Evaluation

Upon completing the iterations for all m knapsacks, the initial lower bound z is calculated using the global objective function:

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

4. Final Solution State

The output of this phase is a feasible assignment where:

The total weight in each knapsack i satisfies:

$$\sum_{j=1}^{n} w_j x_{ij} \leq c_i$$

Each item j is assigned at most once:

$$\sum_{i=1}^{m} x_{ij} \leq 1$$

Example of the Constructive Heuristic

Suppose we have:

- 2 knapsacks
- 6 items

Knapsack Capacities

$$c_1 = 7, c_2 = 6$$

Total available capacity:

$$7 + 6 = 13$$

Step 1 — Item Data

Item j	Profit p_j	Weight w_j	Efficiency $\frac{p_j}{w_j}$
1	24	2	12.0
2	20	3	6.67
3	18	4	4.5
4	15	5	3.0
5	12	4	3.0
6	10	3	3.33

After sorting by efficiency:

Order	Item
1	Item 1
2	Item 2
3	Item 3
4	Item 6
5	Item 4
6	Item 5

Step 2 — Initialization

Initially:

$$x_{ij} = 0$$

Residual capacities:

$$\bar{c}_1 = 7, \bar{c}_2 = 6$$

Step 3 — Fill Knapsack 1

Item 1

$$w_1 = 2 \leq 7$$

Assign:

$$x_{11} = 1$$

Update:

$$\bar{c}_1 = 7 - 2 = 5$$

Item 2

$$w_2 = 3 \leq 5$$

Assign:

$$x_{12} = 1$$

Update:

$$\bar{c}_1 = 5 - 3 = 2$$

Item 3

$$w_3 = 4 > 2$$

Cannot assign.

Item 6

$$w_6 = 3 > 2$$

Cannot assign.

Item 4

$$w_4 = 5 > 2$$

Cannot assign.

Item 5

$$w_5 = 4 > 2$$

Cannot assign.

State After Knapsack 1

Assigned items:

$$\{1,2\}$$

Residual capacity:

$$\bar{c}_1 = 2$$

Current profit:

$$24 + 20 = 44$$

Step 4 — Fill Knapsack 2

Initialize:

$$\bar{c}_2 = 6$$

Item 1

Already assigned:

$$\sum_{k=1}^{1} x_{k1} = 1$$

Cannot assign.

Item 2

Already assigned.

Cannot assign.

Item 3

Available and feasible:

$$w_3 = 4 \leq 6$$

Assign:

$$x_{23} = 1$$

Update:

$$\bar{c}_2 = 6 - 4 = 2$$

Item 6

$$w_6 = 3 > 2$$

Cannot assign.

Item 4

$$w_4 = 5 > 2$$

Cannot assign.

Item 5

$$w_5 = 4 > 2$$

Cannot assign.

Final Assignment Matrix

$$X = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Unassigned Items

The following items could not fit into any knapsack:

$$\{4, 5, 6\}$$

This happens because the remaining capacities after the greedy assignments were too small.

Step 5 — Objective Function

$$z = 24 + 20 + 18$$

$$\boxed{z = 62}$$

Final Solution Summary

Knapsack	Assigned Items	Total Weight	Total Profit
1	{1 , 2}	5	44
2	{3}	4	18

Unassigned items:

$$\{4, 5, 6\}$$

Final objective value:

$$\boxed{z = 62}$$

Local Search Heuristic

We have a set of items $I = \{1, \dots, n\}$ and a set of m knapsacks with capacities $W_1, \dots W_m$.

The solution is represented by a vector $y = (y_1, y_2, \dots, y_n)$, where for each item $j \in I$:

$$y_j = \begin{cases} 0 & \text{if item } j \text{ is currently unassigned} \\ k & \text{if item } j \text{ is assigned to knapsack } k \in \{1,2, \dots, m\} \end{cases}$$

Feasibility: For every knapsack k, the total weight of the items assigned to it must satisfy $\sum_{j:y_j=k} w_j \leq W_k$.

The items that are unassigned are represented as the set:

$$\bar{X} = \{j \in I \mid y_j = 0\}$$

Move

It consists of selecting an assigned item i (where $y_i = k$) and replacing it with an unassigned item j from the set $\bar{X}$ (where $y_j = 0$), as long as the capacity W_k is not exceeded.

Algorithm

Input: y (Initial feasible solution vector)

Output: y_{best} (Optimized assignment vector)

$improve \leftarrow 1$

While (improve)

Generate $N(y)$ using the 1-1 swap move

*(*The neighborhood $N(y)$ is the set of all possible swaps between an assigned item i and an unassigned item *j)*

$y^* \leftarrow$ best neighbor of $N(y)$

(y^* is the vector that provides the maximum $\Delta z = profit(j) - profit(i)$ while remaining feasible).

If $F(y^*)$ is better than $F(y)$ then

$y \leftarrow y^*$

Else

$improve \leftarrow 0$

End if

End while

Example of the Local Search Heuristic

Data

Suppose we have:

- 2 knapsacks
- 5 items

The capacity of each knapsack is:

$$c_1 = 27$$

$$c_2 = 28$$

The attributes are detailed in the following table:

Item (j)	Profit (p_j)	Weight (w_j)
1	9	26
2	12	18
3	10	15
4	8	21
5	14	40

Step 1. Initial Solution

$$y = (1,2,0,0,0)$$

$$z = 21$$

Step 2. Move

Select an assigned item i (where $y_i = k$) and replace it with an unassigned item j from the set $\bar{X}$ (where $y_j = 0$), as long as the capacity W_k is not exceeded.

Step 3. Get all neighbors

Item (i)	Item (j)	Feasibility	Δz
1	3	YES	1
1	4	YES	-1
1	5	NO	
2	3	YES	-2
2	4	YES	-4
2	5	NO	

Step 4. Get the best from the neighborhood

As we see, the only move that fits with the constraint and increments the objective function is the move (1,3)

Hence, our solution is:

$$y = (0,2,1,0,0)$$

At this point, we return to step 3 looking forward to a better solution.

Item (i)	Item (j)	Feasibility	Δz
2	1	YES	-3
2	4	YES	-4
2	5	NO	
3	1	YES	-1
3	4	YES	-2
3	5	NO	

Now, there are not feasible solutions where the objective function increases. Therefore, we have finished our algorithm so far.

It remains as follows the objective function:

$$z = 22$$

Experiments

For this experiment, we used two heuristics to solve the Multiple Knapsack Problem. All instances had a maximum value of 500, minimum value of 50, maximum weight of 100, minimum weight of 10, and 2 knapsacks. We tested 3 set sizes with 20 repetitions each.

Experiment #1	Number of repetitions	n	Maximum value	Minimum value	Maximum weight	Minimum weight	Number of knapsacks
Set_s	20	100	500	50	100	10	2
Set_m	20	500	500	50	100	10	2
Set_l	20	2000	500	50	100	10	2

In the small set, both heuristics ran in less than half a millisecond. Almost half of the tests showed no improvement from the Local Search. This is normal — with only 100 items, the first heuristic already finds a very good solution.

Set_s	CH	Time	LS	Time	AI	RI
Data1_s	18450	0.314 ms	18450	0.0954 ms	0	0.0000%
Data2_s	17014	0.3388 ms	17072	0.1132 ms	58	0.3409%
Data3_s	20202	0.4822 ms	20202	0.0657 ms	0	0.0000%
Data4_s	18257	0.3961 ms	18332	0.1064 ms	75	0.4108%
Data5_s	20166	0.3497 ms	20180	0.178 ms	14	0.0694%
Data6_s	16787	0.3228 ms	16787	0.114 ms	0	0.0000%
Data7_s	17793	0.3302 ms	17794	0.1166 ms	1	0.0056%
Data8_s	18045	0.3146 ms	18045	0.0674 ms	0	0.0000%
Data9_s	20739	0.4101 ms	20739	0.0696 ms	0	0.0000%
Data10_s	18375	0.3843 ms	18375	0.0761 ms	0	0.0000%
Data11_s	18488	0.3379 ms	18547	0.1804 ms	59	0.3191%
Data12_s	19765	0.3413 ms	19796	0.1125 ms	31	0.1568%
Data13_s	18477	0.4801 ms	18522	0.1474 ms	45	0.2435%
Data14_s	19433	0.3376 ms	19433	0.0975 ms	0	0.0000%
Data15_s	18605	0.4151 ms	18605	0.0667 ms	0	0.0000%
Data16_s	17652	0.3552 ms	17666	0.1206 ms	14	0.0793%
Data17_s	18544	0.4099 ms	18544	0.0996 ms	0	0.0000%
Data18_s	17515	0.3695 ms	17523	0.1781 ms	8	0.0457%
Data19_s	18671	0.3596 ms	18700	0.2524 ms	29	0.1553%
Data20_s	20791	0.4185 ms	20808	0.1685 ms	17	0.0818%

In the medium set, the Local Search started taking longer — about 7 times slower than the Constructive Heuristic. But it also started finding better solutions more consistently.

Set_m	CH	Time	LS	Time	AI	RI
Data1_m	97214	0.4489 ms	97292	4.792 ms	78	0.0802%
Data2_m	90745	0.4458 ms	90790	3.1995 ms	45	0.0496%
Data3_m	93416	0.4512 ms	93426	2.9996 ms	10	0.0107%
Data4_m	89197	0.4634 ms	89202	2.868 ms	5	0.0056%
Data5_m	94767	0.502 ms	94843	2.8954 ms	76	0.0802%
Data6_m	90747	0.4234 ms	90776	3.9612 ms	29	0.0320%
Data7_m	85025	0.4461 ms	85063	3.6979 ms	38	0.0447%
Data8_m	89442	0.4814 ms	89477	2.8704 ms	35	0.0391%
Data9_m	91055	0.4055 ms	91055	2.0511 ms	0	0.0000%
Data10_m	92219	0.5836 ms	92219	1.6985 ms	0	0.0000%
Data11_m	94230	0.3969 ms	94299	2.5836 ms	69	0.0732%
Data12_m	88580	0.4854 ms	88611	4.712 ms	31	0.0350%
Data13_m	91167	0.4376 ms	91208	3.9309 ms	41	0.0450%
Data14_m	95778	0.4174 ms	95814	4.0437 ms	36	0.0376%
Data15_m	93619	0.5434 ms	93619	1.4558 ms	0	0.0000%
Data16_m	93427	0.4378 ms	93434	2.8574 ms	7	0.0075%
Data17_m	95637	0.4073 ms	95637	1.4397 ms	0	0.0000%
Data18_m	91413	0.5245 ms	91425	2.851 ms	12	0.0131%
Data19_m	89473	0.4683 ms	89480	2.7415 ms	7	0.0078%
Data20_m	91731	0.4782 ms	91759	2.5825 ms	28	0.0305%

With 2,000 items, the Local Search was 55 times slower but still found improvements in most tests. The extra time was worth it at this size.

Set_l	CH	Time	LS	Time	AI	RI
Data1_l	357430	0.8672 ms	357433	44.9353 ms	3	0.0008%
Data2_l	391135	0.8572 ms	391135	21.6032 ms	0	0.0000%
Data3_l	371581	0.8728 ms	371613	83.4114 ms	32	0.0086%
Data4_l	364923	0.8785 ms	364974	41.074 ms	51	0.0140%
Data5_l	368544	0.8488 ms	368560	44.2338 ms	16	0.0043%
Data6_l	373126	0.83 ms	373177	43.7073 ms	51	0.0137%
Data7_l	360048	0.8255 ms	360048	25.1243 ms	0	0.0000%
Data8_l	369525	1.0135 ms	369538	41.9284 ms	13	0.0035%
Data9_l	362922	0.8327 ms	362963	44.532 ms	41	0.0113%
Data10_l	355437	0.8928 ms	355479	45.5947 ms	42	0.0118%
Data11_l	378208	0.9223 ms	378229	82.9861 ms	21	0.0056%
Data12_l	376810	1.0778 ms	376842	40.7697 ms	32	0.0085%
Data13_l	372277	0.8257 ms	372310	43.9479 ms	33	0.0089%
Data14_l	380224	0.8378 ms	380287	82.4994 ms	63	0.0166%
Data15_l	374994	0.8513 ms	374994	23.4131 ms	0	0.0000%
Data16_l	380861	0.8848 ms	380916	82.2586 ms	55	0.0144%
Data17_l	356769	0.925 ms	356778	47.8243 ms	9	0.0025%
Data18_l	384218	1.0142 ms	384255	51.8997 ms	37	0.0096%
Data19_l	373040	0.8432 ms	373040	22.5261 ms	0	0.0000%
Data20_l	379329	0.9489 ms	379338	43.2822 ms	9	0.0024%

Extra set (Set XL; n=100,000)

Set_xl	CH	Time	LS	Time	AI	RI
Data1_XXL	19304470	30.996ms	19304537	272964ms=4.55min	67	0.0003%

In conclusion, the results show that the Constructive Heuristic is very fast and provides good initial solutions, especially for small instances, while the Local Search heuristic can improve those solutions as the problem size increases. For Set S, the improvements were often minimal or nonexistent because the constructive approach was already close to the best solution; however, for Set M and Set L, Local Search produced more consistent improvements, although at the cost of higher execution times. The additional test with $n = 100{,}000$ confirmed that this trade-off becomes much more significant for very large instances, since Local Search required several minutes to obtain only a negligible improvement. Therefore, the Constructive Heuristic is more suitable when speed is the priority, while Local Search is more useful for medium and large instances when a better solution justifies the extra computational time.

References

Eilon, S. and Christofides, N. (1971) ‘The loading problem’, Management Science, Vol. 17, pp.259–268.

M. E. Lalami, M. Elkihel, D. El Baz and Boyer, V. (2012) ‘A procedure-based heuristic for 0-1 Multiple Knapsack Problems’, Int. J. Mathematics in Operational Research, Vol. 4, No. 3, pp.214–224

Martello, S., & Toth, P. (1990). Knapsack problems: Algorithms and computer implementations. Wiley.