Appendix: Computer codes

A.1 INTRODUCTION

The diskette included in the volume contains the Fortran implementations of the most effective algorithms described in the various chapters. Table A.1 gives, for each code, the problem solved, the approximate number of lines (including comments), the section where the corresponding procedure (which has the same name as the code) is described, and the type of algorithm implemented. Most of the implementations are exact branch-and-bound algorithms which can also be used to provide approximate solutions by limiting the number of backtrackings through an input parameter (notation Exact /Approximate in the table).

Table A.1 Fortran codes included in the volume

Code	Problem	Lines	Section	Type of algorithm
MT1	0-1 Knapsack	280	2.5.2	Exact
MT1R	0-1 Knapsack	300	2.5.2	Exact (real data)
MT2	0-1 Knapsack	1400	2.9.3	Exact/Approximate
MTB2	Bounded Knapsack	190 (+1400)*	3.4.2	Exact/Approximate
MTU2	Unbounded Knapsack	1100	3.6.3	Exact/Approximate
MTSL	Subset-Sum	780	4.2.3	Exact/Approximate
MTC2	Change-Making	450	5.6	Exact/Approximate
MTCB	Bounded Change-Making	380	5.8	Exact/Approximate
MTM	0-1 Multiple Knapsack	670	6.4.3	Exact/Approximate
MTHM	0-1 Multiple Knapsack	590	6.6.2	Approximate
MTG	Generalized Assignment	2300	7.3	Exact/Approximate
MTHG	Generalized Assignment	500	7.4	Approximate
MTP	Bin Packing	1330	8.5	Exact/Approximate

* MTB2 must be linked with MT2.

All programs solve problems defined by integer parameters, except MT1R which solves the 0-1 single knapsack problem with real parameters.

All codes are written according to PFORT, a portable subset of 1966 ANSI Fortran, and are accepted by the PFORT verifier developed by Ryder and Hall (1981) at Bell Laboratories. The codes have been tested on a Digital VAX 11/780 and a Hewlett-Packard 9000/840.

With the only exception of MTB2 (which must be linked with MT2), the codes are completely self-contained. Communication to the codes is achieved solely through the parameter list of a "main" subroutine whose name is that of the code.

The following sections give, for each problem and for each code, the corresponding comment and specification statements.

A.2 0-1 KNAPSACK PROBLEM

A.2.1 Code MT1

SUBROUTINE MT1 (N, P, W, C, Z, X, JDIM, JCK,
XX, MIN, PSIGN, WSIGN, ZSIGN)

This subroutine solves the 0-1 single knapsack problem

$$\begin{aligned} \text{maximize } Z = \; & P(1)\,X(1) + \ldots + P(N)\,X(N) \\ \text{subject to } \; & W(1)\,X(1) + \ldots + W(N)\,X(N) \le C, \\ & X(J) = 0 \text{ or } 1 \text{ for } J=1, \ldots, N \end{aligned}$$

The program implements the branch-and-bound algorithm described in Section 2.5.2, and derives from an earlier code presented in S. Martello, P. Toth, "Algorithm for the solution of the 0-1 single knapsack problem", *Computing*, 1978.

The input problem must satisfy the conditions

(1) $2 \le N \le JDIM - 1$;

(2) P(J), W(J), C positive integers;

(3) $\max (W(J)) \le C$;

(4) $W(1) + \ldots + W(N) > C$;

(5) $P(J)/W(J) \ge P(J+1)/W(J+1)$ for $J = 1, \ldots, N-1$.

MT1 calls 1 procedure: CHMT1.

The program is completely self-contained and communication to it is achieved solely through the parameter list of MT1.
No machine-dependent constant is used.

MT1 needs 8 arrays (P, W, X, XX, MIN, PSIGN, WSIGN and ZSIGN) of length at least N + 1.

Meaning of the input parameters:

N = number of items;

P(J) = profit of item J (J = 1, ..., N);

W(J) = weight of item J (J = 1, ..., N);

C = capacity of the knapsack;

JDIM = dimension of the 8 arrays;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the optimal solution if Z > 0,
= error in the input data (when JCK = 1) if Z < 0:
condition −Z is violated;

X(J) = 1 if item J is in the optimal solution,
= 0 otherwise.

Arrays XX, MIN, PSIGN, WSIGN and ZSIGN are dummy.

All the parameters are integer. On return of MT1 all the input parameters are unchanged.

```
INTEGER P(JDIM), W(JDIM), X(JDIM), C, Z
INTEGER XX(JDIM), MIN(JDIM)
INTEGER PSIGN(JDIM), WSIGN(JDIM), ZSIGN(JDIM)
```

A.2.2 Code MT1R

```
SUBROUTINE MT1R  (N, P, W, C, EPS, Z, X, JDIM, JCK,
                  XX, MIN, PSIGN, WSIGN, ZSIGN, CRC, CRP)
```

This subroutine solves the 0-1 single knapsack problem with real parameters

maximize Z = P(1) X(1) + ... + P(N) X(N)

subject to W(1) X(1) + ... + W(N) X(N) ≤ C,

X(J) = 0 or 1 for J = 1, ..., N.

The program implements the branch-and-bound algorithm described in Section 2.5.2, and is a modified version of subroutine MT1.

The input problem must satisfy the conditions

(1) $2 \leq N \leq JDIM - 1$;

(2) P(J), W(J), C positive reals;

(3) max (W(J)) $\leq$ C;

(4) W(1) + ... + W(N) > C;

(5) P(J)/W(J) $\geq$ P(J + 1)/W(J + 1) for J = 1,..., N − 1.

MT1R calls 1 procedure: CHMT1R.

The program is completely self-contained and communication to it is achieved solely through the parameter list of MT1R.
No machine-dependent constant is used.

MT1R needs 10 arrays (P, W, X, XX, MIN, PSIGN, WSIGN, ZSIGN, CRC and CRP) of length at least N + 1.

Meaning of the input parameters:

N = number of items;

P(J) = profit of item J (J = 1,..., N);

W(J) = weight of item J (J = 1,..., N);

C = capacity of the knapsack;

EPS = tolerance (two positive values Q and R are considered equal if ABS(Q − R)/max (Q, R) $\leq$ EPS);

JDIM = dimension of the 10 arrays;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the optimal solution if Z > 0,
= error in the input data (when JCK = 1) if Z < 0:
condition −Z is violated;

X(J) = 1 if item J is in the optimal solution,
= 0 otherwise.

Arrays XX, MIN, PSIGN, WSIGN, ZSIGN, CRC and CRP are dummy.

Parameters N, X, JDIM, JCK, XX and ZSIGN are integer. Parameters P, W, C, Z, MIN, PSIGN, WSIGN, CRC, CRP and EPS are real. On return of MT1R all the input parameters are unchanged.

```
REAL P(JDIM), W(JDIM)
INTEGER X(JDIM)
INTEGER XX(JDIM), ZSIGN(JDIM)
REAL MIN(JDIM), PSIGN(JDIM), WSIGN(JDIM), CRC(JDIM), CRP(JDIM)
```

A.2.3 Code MT2

SUBROUTINE MT2 (N, P, W, C, Z, X, JDIM, JFO, JFS, JCK, JUB,
IA1, IA2, IA3, IA4, RA)

This subroutine solves the 0-1 single knapsack problem

$$\text{maximize } Z = P(1)\,X(1) + \ldots + P(N)\,X(N)$$

$$\text{subject to} \quad W(1)\,X(1) + \ldots + W(N)\,X(N) \le C,$$

$$X(J) = 0 \text{ or } 1 \text{ for } J = 1, \ldots, N.$$

The program implements the enumerative algorithm described in Section 2.9.3.

The input problem must satisfy the conditions

(1) $2 \le N \le JDIM - 3$;

(2) P(J), W(J), C positive integers;

(3) $\max (W(J)) \le C$;

(4) $W(1) + \ldots + W(N) > C$;

and, if JFS = 1,

(5) $P(J)/W(J) \ge P(J+1)/W(J+1)$ for $J = 1, \ldots, N-1$.

MT2 calls 9 procedures: CHMT2, CORE, CORES, FMED, KP01M, NEWB, REDNS, REDS and SORTR.

The program is completely self-contained and communication to it is achieved solely through the parameter list of MT2.
No machine-dependent constant is used.

MT2 needs 8 arrays (P, W, X, IA1, IA2, IA3, IA4 and RA) of length at least N + 3.

Meaning of the input parameters:

N = number of items;

P(J) = profit of item J (J = 1,..., N);

W(J) = weight of item J (J = 1,..., N);

C = capacity of the knapsack;

JDIM = dimension of the 8 arrays;

JFO = 1 if optimal solution is required,
= 0 if approximate solution is required;

JFS = 1 if the items are already sorted according to decreasing profit per unit weight,
= 0 otherwise;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if Z > 0,
= error in the input data (when JCK = 1) if Z < 0: condition −Z is violated;

X(J) = 1 if item J is in the solution found,
= 0 otherwise;

JUB = upper bound on the optimal solution value (to evaluate Z when JFO = 0).

Arrays IA1, IA2, IA3, IA4 and RA are dummy.

All the parameters but RA are integer. On return of MT2 all the input parameters are unchanged.

```
INTEGER P(JDIM), W(JDIM), X(JDIM), C, Z
DIMENSION IA1(JDIM), IA2(JDIM), IA3(JDIM), IA4(JDIM)
DIMENSION RA(JDIM)
```

A.3 BOUNDED AND UNBOUNDED KNAPSACK PROBLEM

A.3.1 Code MTB2

```
SUBROUTINE MTB2  (N, P, W, B, C, Z, X,
                  JDIM1, JDIM2, JFO, JFS, JCK, JUB,
                  ID1, ID2, ID3, ID4, ID5, ID6, ID7, RD8)
```

This subroutine solves the bounded single knapsack problem

$$\begin{aligned} &\text{maximize } Z = && P(1)\,X(1) + \ldots + P(N)\,X(N) \\ &\text{subject to} && W(1)\,X(1) + \ldots + W(N)\,X(N) \le C, \\ & && 0 \le X(J) \le B(J) \quad \text{for } J = 1, \ldots, N, \\ & && X(J) \text{ integer} \quad \text{for } J = 1, \ldots, N. \end{aligned}$$

The program implements the transformation method described in Section 3.2.

The problem is transformed into an equivalent 0-1 knapsack problem and then solved through subroutine MT2. The user must link MT2 and its subroutines to this program.

The input problem must satisfy the conditions

(1) $2 \le N \le JDIM1 - 1$;

(2) P(J), W(J), B(J), C positive integers;

(3) $\max (B(J)W(J)) \le C$;

(4) $B(1)W(1) + \ldots + B(N)W(N) > C$;

(5) $2 \le N + (LOG2(B(1)) + \ldots + LOG2(B(N))) \le JDIM2 - 3$;

and, if JFS = 1,

(6) $P(J)/W(J) \ge P(J+1)/W(J+1)$ for $J = 1, \ldots, N-1$.

MTB2 calls 4 procedures: CHMTB2, SOL, TRANS and MT2 (external).

Communication to the program is achieved solely through the parameter list of MTB2.
No machine-dependent constant is used.

MTB2 needs

4 arrays (P, W, B and X) of length at least JDIM1;

8 arrays (ID1, ID2, ID3, ID4, ID5, ID6, ID7 and RD8) of length at least JDIM2.

Meaning of the input parameters:

N = number of item types;

P(J) = profit of each item of type J (J = 1, ..., N);

W(J) = weight of each item of type J (J = 1, ..., N);

B(J) = number of items of type J available (J = 1, ..., N);

C = capacity of the knapsack;

JDIM1 = dimension of arrays P, W, B, X;

JDIM2 = dimension of arrays ID1, ID2, ID3, ID4, ID5, ID6, ID7, RD8;

JFO = 1 if optimal solution is required,
= 0 if approximate solution is required;

JFS = 1 if the items are already sorted according to decreasing profit per unit weight (suggested for large B(J) values),
= 0 otherwise;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if $Z > 0$,
= error in the input data (when JCK = 1) if $Z < 0$: condition $-Z$ is violated;

X(J) = number of items of type J in the solution found;

JUB = upper bound on the optimal solution value (to evaluate Z when JFO = 0).

Arrays ID1, ID2, ID3, ID4, ID5, ID6, ID7 and RD8 are dummy.

All the parameters but RD8 are integer. On return of MTB2 all the input parameters are unchanged.

```
INTEGER P(JDIM1), W(JDIM1), B(JDIM1), X(JDIM1), C, Z
INTEGER ID1(JDIM2), ID2(JDIM2), ID3(JDIM2), ID4(JDIM2)
INTEGER ID5(JDIM2), ID6(JDIM2), ID7(JDIM2)
REAL RD8(JDIM2)
```

A.3.2 Code MTU2

```
SUBROUTINE MTU2 (N, P, W, C, Z, X,
                 JDIM, JFO, JCK, JUB,
                 PO, WO, XO, RR, PP)
```

This subroutine solves the unbounded single knapsack problem

$$\begin{aligned} \text{maximize } Z = \; & P(1)\,X(1) + \ldots + P(N)\,X(N) \\ \text{subject to } \; & W(1)\,X(1) + \ldots + W(N)\,X(N) \leq C, \\ & X(J) \geq 0 \text{ and integer for } J = 1, \ldots, N. \end{aligned}$$

The program implements the enumerative algorithm described in Section 3.6.3.

The input problem must satisfy the conditions

(1) $2 \leq N \leq JDIM - 1$;

(2) P(J), W(J), C positive integers;

(3) max $(W(J)) \leq C$.

MTU2 calls 5 procedures: CHMTU2, KSMALL, MTU1, REDU and SORTR.
KSMALL calls 8 procedures: BLD, BLDF, BLDS1, DETNS1, DETNS2, FORWD, MPSORT and SORT7.

The program is completely self-contained and communication to it is achieved solely through the parameter list of MTU2.
No machine-dependent constant is used.

MTU2 needs 8 arrays (P, W, X, PO, WO, XO, RR and PP) of length at least JDIM.

Meaning of the input parameters:

N = number of item types;

P(J) = profit of each item of type J (J = 1, ..., N);

W(J) = weight of each item of type J (J = 1, ..., N);

C = capacity of the knapsack;

JDIM = dimension of the 8 arrays;

JFO = 1 if optimal solution is required,
= 0 if approximate solution is required;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if $Z > 0$,
= error in the input data (when JCK = 1) if $Z < 0$:
condition $-Z$ is violated;

X(J) = number of items of type J in the solution found;

JUB = upper bound on the optimal solution value
(to evaluate Z when JFO = 0).

Arrays PO, WO, XO, RR and PP are dummy.

All the parameters but XO and RR are integer. On return of MTU2 all the input parameters are unchanged.

```
INTEGER P(JDIM), W(JDIM), X(JDIM)
INTEGER PO(JDIM), WO(JDIM), PP(JDIM), C, Z
REAL RR(JDIM), XO(JDIM)
```

A.4 SUBSET-SUM PROBLEM

A.4.1 Code MTSL

SUBROUTINE MTSL (N, W, C, Z, X, JDN, JDD, ITMM, JCK,
WO, IND, XX, WS, ZS, SUM,
TD1, TD2, TD3)

This subroutine solves the subset-sum problem

$$\text{maximize } Z = W(1)\,X(1) + \ldots + W(N)\,X(N)$$

$$\text{subject to} \quad W(1)\,X(1) + \ldots + W(N)\,X(N) \le C,$$
$$X(J) = 0 \text{ or } 1 \qquad \text{for } J = 1, \ldots, N.$$

The program implements the mixed algorithm described in Section 4.2.3.

The input problem must satisfy the conditions

(1) $2 \le N \le JDN - 1$;

(2) W(J), C positive integers;

(3) max (W(J)) $<$ C;

(4) W(1) + ... + W(N) $>$ C.

MTSL calls 8 procedures: CHMTSL, DINSM, MTS, PRESP, SORTI, TAB, UPSTAR and USEDIN.

If not present in the library of the host, the user must supply an integer function JIAND(I1, I2) which sets JIAND to the bit-by-bit logical AND of I1 and I2.

Communication to the program is achieved solely through the parameter list of MTSL.
No machine-dependent constant is used.

MTSL needs

2 arrays (W and X) of length at least JDN;
6 arrays (WO, IND, XX, WS, ZS and SUM) of length at least ITMM;
3 arrays (TD1, TD2 and TD3) of length at least JDD × 2.

Meaning of the input parameters:

N = number of items;

W(J) = weight of item J (J = 1, ..., N);

C = capacity;

JDN = dimension of arrays W and X;

JDD = maximum length of the dynamic programming lists (suggested value JDD = 5000);

ITMM = (maximum number of items in the core problem) + 1; ITMM = JDN in order to be sure that the optimal solution is found. ITMM < JDN (suggested value ITMM = 91) produces an approximate solution which is almost always optimal (to check optimality, see whether Z = C);

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if Z > 0,
= error in the input data (when JCK = 1) if Z < 0: condition −Z is violated;

X(J) = 1 if item J is in the solution found,
= 0 otherwise.

Meaning of the internal variables which could be altered by the user:

IT = length of the initial core problem (suggested value IT = 30);

ID = increment of the length of the core problem (suggested value ID = 30);

M2 = number of items to be used for the second dynamic programming list; it must be $2 \le$ M2 $\le$ min (31, N − 4) (suggested value M2 = min (2.5 ALOG10 (max (W(J))), 0.8 N)). M1, the number of items to be used for the first dynamic programming list, is automatically determined;

PERS = value used to determine $\bar{c}$ according to the formula given in Section 4.2.2 (suggested value PERS = 1.3).

Arrays WO, IND, XX, WS, ZS, SUM, TD1, TD2 and TD3 are dummy.

All the parameters are integer. On return of MTSL all the input parameters are unchanged.

```
INTEGER W(JDN), X(JDN), C, Z
INTEGER WO(ITMM), IND(ITMM), XX(ITMM)
INTEGER WS(ITMM), ZS(ITMM), SUM(ITMM)
INTEGER TD1(JDD,2), TD2(JDD,2), TD3(JDD,2)
```

A.5 BOUNDED AND UNBOUNDED CHANGE-MAKING PROBLEM

A.5.1 Code MTC2

SUBROUTINE MTC2 (N, W, C, Z, X, JDN, JDL, JFO, JCK, XX, WR, PR, M, L)

This subroutine solves the unbounded change-making problem

$$\text{minimize } Z = X(1) + \ldots + X(N)$$

$$\text{subject to } W(1)\,X(1) + \ldots + W(N)\,X(N) = C,$$

$$X(J) \geq 0 \text{ and integer for } J = 1, \ldots, N.$$

The program implements the enumerative algorithm described in Section 5.6.

The input problem must satisfy the conditions

(1) $2 \leq N \leq JDN - 1$;

(2) W(J), C positive integers;

(3) $\max(W(J)) < C$.

MTC2 calls 5 procedures: CHMTC2, COREC, MAXT, MTC1 and SORTI.

The program is completely self-contained and communication to it is achieved solely through the parameter list of MTC2.
No machine-dependent constant is used.

MTC2 needs

5 arrays (W, X, XX, WR and PR) of length at least JDN;
2 arrays (M and L) of length at least JDL.

Meaning of the input parameters:

N = number of item types;

W(J) = weight of each item of type J (J = 1, ..., N);

C = capacity;

JDN = dimension of arrays W, X, XX, WR and PR;

JDL = dimension of arrays M and L (suggested value JDL = max (W(J)) − 1; if the core memory is not enough, JDL should be set to the largest possible value);

JFO = 1 if optimal solution is required,
= 0 if approximate solution is required
(at most 100 000 backtrackings are performed);

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if Z > 0,
= no feasible solution exists if Z = 0,
= error in the input data (when JCK = 1) if Z < 0:
condition −Z is violated;

X(J) = number of items of type J in the solution found.

Arrays XX, M, L, WR and PR are dummy.

All the parameters are integer. On return of MTC2 all the input parameters are unchanged.

```
INTEGER W(JDN), X(JDN), C, Z
INTEGER XX(JDN), WR(JDN), PR(JDN)
INTEGER M(JDL), L(JDL)
```

A.5.2 Code MTCB

```
SUBROUTINE MTCB (N, W, B, C, Z, X, JDN, JDL, JFO, JCK,
                 XX, WR, BR, PR, M, L)
```

This subroutine solves the bounded change-making problem

$$\begin{array}{lll} \text{minimize } Z = & X(1) + \ldots + X(N) & \\ \text{subject to} & W(1)\,X(1) + \ldots + W(N)\,X(N) = C, & \\ & 0 \le X(J) \le B(J) & \text{for } J = 1, \ldots, N, \\ & X(J) \text{ integer} & \text{for } J = 1, \ldots, N. \end{array}$$

The program implements the branch-and-bound algorithm described in Section 5.8.

The input problem must satisfy the conditions

(1) $2 \le N \le JDN - 1$;

(2) W(J), B(J), C positive integers;

(3) $\max(W(J)) < C$;

(4) B(J) W(J) $\leq$ C for J = 1, ..., N;

(5) B(1) W(1) + ... + B(N) W(N) > C.

MTCB calls 3 procedures: CHMTCB, CMPB and SORTI.

The program is completely self-contained and communication to it is achieved solely through the parameter list of MTCB.
No machine-dependent constant is used.

MTCB needs

7 arrays (W, B, X, XX, WR, BR and PR) of length at least JDN;
2 arrays (M and L) of length at least JDL.

Meaning of the input parameters:

N = number of item types;

W(J) = weight of each item of type J (J = 1, ..., N);

B(J) = number of available items of type J (J = 1, ..., N);

C = capacity;

JDN = dimension of arrays W, B, X, XX, WR, BR and PR;

JDL = dimension of arrays M and L (suggested value JDL = max (W(J)) − 1; if the core memory is not enough, JDL should be set to the largest possible value);

JFO = 1 if optimal solution is required,
= 0 if approximate solution is required
(at most 100 000 backtrackings are performed);

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if Z > 0,
= no feasible solution exists if Z = 0,
= error in the input data (when JCK = 1) if Z < 0:
condition −Z is violated;

X(J) = number of items of type J in the solution found.

Arrays XX, M, L, WR, BR and PR are dummy.

All the parameters are integer. On return of MTCB all the input parameters are unchanged.

```
INTEGER W(JDN), B(JDN), X(JDN), C, Z
INTEGER XX(JDN), WR(JDN), BR(JDN), PR(JDN)
INTEGER M(JDL), L(JDL)
```

A.6 0-1 MULTIPLE KNAPSACK PROBLEM

A.6.1 Code MTM

SUBROUTINE MTM (N, M, P, W, C, Z, X, BACK, JCK, JUB)

This subroutine solves the 0-1 multiple knapsack problem

$$\begin{aligned}\text{maximize } Z = \ & P(1)\,(Y(1,1) + \dots + Y(M,1)) + \\ & \dots + \\ & P(N)\,(Y(1,N) + \dots + Y(M,N))\end{aligned}$$

$$\begin{aligned}\text{subject to} \quad & W(1)\,Y(I,1) + \dots + W(N)\,Y(I,N) \le C(I) \quad \text{for } I = 1,\dots,M, \\ & Y(1,J) + \dots + Y(M,J) \le 1 \quad \text{for } J = 1,\dots,N, \\ & Y(I,J) = 0 \text{ or } 1 \quad \text{for } I = 1,\dots,M,\ J = 1,\dots,N.\end{aligned}$$

The program implements the enumerative algorithm described in Section 6.4.3, and derives from an earlier code presented in S. Martello, P. Toth, "Algorithm 632. A program for the 0-1 multiple knapsack problem", *ACM Transactions on Mathematical Software*, 1985.

The input problem must satisfy the conditions

(1) $2 \le N \le MAXN$ and $1 \le M \le MAXM$, where MAXN and MAXM are defined by the first two executable statements;
(2) P(J), W(J) and C(I) positive integers;
(3) $\min (C(I)) \ge \min (W(J))$;
(4) $\max (W(J)) \le \max (C(I))$;
(5) $\max (C(I)) < W(1) + \dots + W(N)$;
(6) $P(J)/W(J) \ge P(J+1)/W(J+1)$ for $J = 1,\dots,N-1$;
(7) $C(I) \le C(I+1)$ for $I = 1,\dots,M-1$.

MTM calls 5 procedures: CHMTM, PAR, PI, SIGMA and SKP.

The program is completely self-contained and communication to it is achieved solely through the parameter list of MTM.
No machine-dependent constant is used.

MTM needs

5 arrays (C, F, PBL, Q and V) of length at least M;
8 arrays (P, W, X, UBB, BS, XS, LX and LXI) of length at least N;
3 arrays (B, PS and WS) of length at least N + 1;
3 arrays (BB, XC and XL) of length at least M $\times$ N;
1 array (BL) of length at least M $\times$ (N + 1);
5 arrays (D, MIN, PBAR, WBAR and ZBAR) of length at least N (for internal use in subroutine SKP).

The arrays are currently dimensioned to allow problems for which M $\leq$ 10 and N $\leq$ 1000. Changing such dimensions also requires changing the dimension of BS, PS, WS, XS, LX and LXI in subroutine SIGMA, of BB, BL, XL, BS, PS, WS and XS in subroutine PI, of BB, LX and LXI in subroutine PAR, of D, MIN, PBAR, WBAR and ZBAR in subroutine SKP. In addition, the values of MAXN and MAXM must be conveniently defined.

Meaning of the input parameters:

N = number of items;

M = number of knapsacks;

P(J) = profit of item J (J = 1, ..., N);

W(J) = weight of item J (J = 1, ..., N);

C(I) = capacity of knapsack I (I = 1, ..., M);

BACK = −1 if exact solution is required,
= maximum number of backtrackings to be performed, if heuristic solution is required;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if Z > 0,
= error in the input data (when JCK = 1) if Z < 0: condition −Z is violated;

X(J) = 0 if item J is not in the solution found (Y(I, J) = 0 for all I),
= knapsack where item J is inserted, otherwise (Y(X(J), J) = 1);

JUB = upper bound on the optimal solution value (to evaluate Z when BACK $\geq$ 0 on input).

All the parameters are integer. On return of MTM all the input parameters are unchanged except BACK (= number of backtrackings performed).

```
INTEGER P(1000), W(1000), C(10), X(1000), Z, BACK
INTEGER BB(10,1000), BL(10,1001), XC(10,1000), XL(10,1000)
INTEGER B(1001), UBB(1000), F(10), PBL(10), Q(10), V(10)
INTEGER BS, PS, WS, XS
COMMON /SNGL/ BS(1000), PS(1001), WS(1001), XS(1000)
COMMON /PUB/ LX(1000), LXI(1000), LR, LRI, LUBI
```

A.6.2 Code MTHM

SUBROUTINE MTHM (N, M, P, W, C, Z, X, JDN, JDM, LI, JCK,
CR, MIN, XX, X1, F)

This subroutine heuristically solves the 0-1 multiple knapsack problem

$$\begin{aligned}
\text{maximize } Z = \; & P(1)\,(Y(1, 1) + \dots + Y(M, 1)) \; + \\
& \qquad\qquad \dots \qquad\qquad\quad + \\
& P(N)\,(Y(1, N) + \dots + Y(M, N))
\end{aligned}$$

$$\begin{aligned}
\text{subject to } \; & W(1)\,Y(I, 1) + \dots + W(N)\,Y(I, N) \le C(I) \\
& \qquad\qquad\qquad\qquad \text{for } I = 1, \dots, M, \\
& Y(1,J) + \dots + Y(M,J) \le 1 \qquad \text{for } J = 1, \dots, N, \\
& Y(I,J) = 0 \text{ or } 1 \quad \text{for } I = 1, \dots, M, \; J = 1, \dots, N.
\end{aligned}$$

The program implements the polynomial-time algorithms described in Section 6.6.2, and derives from an earlier code presented in S. Martello, P. Toth, "Heuristic algorithms for the multiple knapsack problem", *Computing*, 1981.

The input problem must satisfy the conditions

(1) $2 \le N \le JDN - 1$ and $1 \le M \le JDM - 1$;

(2) P(J), W(J) and C(I) positive integers;

(3) $\min(C(I)) \ge \min(W(J))$;

(4) $\max(W(J)) \le \max(C(I))$;

(5) $\max(C(I)) < W(1) + \dots + W(N)$;

(6) $P(J)/W(J) \ge P(J+1)/W(J+1)$ for $J = 1, \dots, N - 1$;

(7) $C(I) \le C(I+1)$ for $I = 1, \dots, M - 1$.

MTHM can call 6 subroutines:

CHMTHM to check the input data;
MGR1 or MGR2 to find an initial feasible solution;
REARR to re-arrange a feasible solution;
IMPR1 and IMPR2 to improve on a feasible solution.

The user selects the sequence of calls through input parameters.

The program is completely self-contained and communication to it is achieved solely through the parameter list of MTHM.
The only machine-dependent constant is used to define INF (first executable statement), which must be set to a large positive integer value.

MTHM needs

6 arrays (P, W, X, MIN, XX and X1) of length at least JDN;
2 arrays (C and CR) of length at least JDM;
1 array (F) of length at least JDM $\times$ JDM.

In addition, subroutine MGR2 uses

7 arrays of length 5;
1 array of length 201;
1 array of length 5 $\times$ 200.

Subroutine MGR2 is called only when M $\leq$ 5 and N $\leq$ 200.

Meaning of the input parameters:

N = number of items;

M = number of knapsacks;

P(J) = profit of item J (J = 1, ..., N);

W(J) = weight of item J (J = 1, ..., N);

C(I) = capacity of knapsack I (I = 1, ..., M);

JDN = dimension of arrays P, W, X, MIN, XX and X1;

JDM = dimension of arrays C, CR and F;

LI = 0 to output the initial feasible solution,
= 1 to also perform subroutines REARR and IMPR1,
= 2 to also perform subroutines REARR, IMPR1 and IMPR2;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if Z > 0,
= error in the input data (when JCK = 1) if Z < 0:
condition −Z is violated;

X(J) = 0 if item J is not in the solution found
(i.e. if Y(I, J) = 0 for all I),
= knapsack where item J is inserted, otherwise
(i.e. if Y(X(J), J) = 1).

Arrays CR, MIN, XX, X1 and F are dummy.

All the parameters are integer. On return of MTHM all the input parameters are unchanged.

```
INTEGER P(JDN), W(JDN), X(JDN), C(JDM), Z
INTEGER MIN(JDN), XX(JDN), X1(JDN), CR(JDM)
INTEGER F(JDM, JDM)
```

A.7 GENERALIZED ASSIGNMENT PROBLEM

A.7.1 Code MTG

```
SUBROUTINE MTG (N, M, P, W, C, MINMAX,
                Z, XSTAR, BACK, JCK, JB)
```

This subroutine solves the generalized assignment problem

$$\begin{aligned} \text{opt } Z = {} & P(1, 1)\,X(1, 1) + \ldots + P(1, N)\,X(1, N) + \\ & \ldots + \\ & P(M, 1)\,X(M, 1) + \ldots + P(M, N)\,X(M, N) \end{aligned}$$

(where opt = min if MINMAX = 1, opt = max if MINMAX = 2)

$$\text{subject to } W(I, 1)\,X(I, 1) + \ldots + W(I, N)\,X(I, N) \le C(I) \quad \text{for } I = 1, \ldots, M,$$

$$X(1, J) + \ldots + X(M, J) = 1 \quad \text{for } J = 1, \ldots, N,$$

$$X(I, J) = 0 \text{ or } 1 \quad \text{for } I = 1, \ldots, M, \; J = 1, \ldots, N.$$

The program implements the branch-and-bound algorithm described in Sections 7.3–7.5.

The input problem must satisfy the conditions

(1) $2 \le M \le$ JDIMR;

(2) $2 \le N \le$ JDIMC (JDIMR and JDIMC are defined by the first two executable statements);

(3) $M \le$ JDIMPC (JDIMPC, defined by the third executable statement, is used for packing array Y, and cannot be greater than (number of bits of the host) -2; if

a higher value is desired, subroutines YDEF and YUSE must be re-structured accordingly);

(4) P(I, J), W(I, J) and C(I) positive integers;

(5) W(I, J) $\leq$ C(I) for at least one I, for J = 1, ..., N;

(6) C(I) $\geq$ min (W(I, J)) for I = 1, ..., M.

In addition, it is required that

(7) (maximum level of the decision-tree) $\leq$ JNLEV. (JNLEV is defined by the fourth executable statement.)

MTG calls 24 procedures: CHMTG, DEFPCK, DMIND, FEAS, GHA, GHBCD, GHX, GR1, GR2, HEUR, KPMAX, KPMIN, PEN0, PEN1, PREPEN, SKP, SORTI, SORTR, TERMIN, TRIN, UBFJV, UBRS, YDEF and YUSE.

If not present in the library of the host, the user must supply an integer function JIAND(I1, I2) which sets JIAND to the bit-by-bit logical AND of I1 and I2. Such function is used in subroutines YDEF and YUSE.

Communication to the program is achieved solely through the parameter list of MTG.
No machine-dependent constant is used.

MTG needs

17 arrays (C, DD, UD, Q, PACKL, IP, IR, IL, IF, WOBBL, KQ, FLREP, DMYR1, DMYR2, DMYR3, DMYR4 and DMYR5) of length at least M;
25 arrays (XSTAR, XS, BS, B, KA, XXS, IOBBL, JOBBL, BEST, XJJUB, DS, DMYC1, DMYC2, DMYC3, DMYC4, DMYC5, DMYC6, DMYC7, DMYC8, DMYC9, DMYC10, DMYC11, DMYC12, DMYC13 and DMYCR1) of length at least N;
4 arrays (PS, WS, DMYCC1 and DMYCC2) of length at least N + 1;
6 arrays (E, CC, CS, TYPE, US and UBL) of length at least JNLEV;
7 arrays (P, W, A, X, PAK, KAP and MIND) of length at least M $\times$ N;
5 arrays (D, VS, V, LB and UB) of length at least JNLEV $\times$ M;
1 array (Y) of length at least JNLEV $\times$ N;
2 arrays (MASK1 and ITWO) of length at least JDIMPC.

The arrays are currently dimensioned to allow problems for which

M $\leq$ 10,
N $\leq$ 100,
JNLEV $\leq$ 150,

on a 32-bit computer (so, in the calling program, arrays P and W must be dimensioned at (10,100)). Changing such limits necessitates changing the dimension of all the arrays in subroutine MTG and in COMMON /PACK/ (which is included in subroutines MTG, YDEF and YUSE), as well as the four first executable statements.

Meaning of the input parameters:

N = number of items;

M = number of knapsacks;

P(I, J) = profit of item J if assigned to knapsack I (I = 1, ..., M; J = 1, ..., N);

W(I, J) = weight of item J if assigned to knapsack I (I = 1, ..., M; J = 1, ..., N);

C(I) = capacity of knapsack I (I = 1, ..., M);

MINMAX = 1 if the objective function must be minimized,
= 2 if the objective function must be maximized;

BACK = −1 if exact solution is required,
= maximum number of backtrackings to be performed, if heuristic solution is required;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if $Z > 0$,
= 0 if no feasible solution exists,
= error in the input data (when JCK = 1) if $Z < 0$: condition $-Z$ is violated;

XSTAR(J) = knapsack where item J is inserted in the solution found;

JB = lower bound (if MINMAX = 1) or upper bound (if MINMAX = 2) on the optimal solution value (to evaluate Z when BACK $\geq$ 0 on input).

All the parameters are integer. On return of MTG all the input parameters are unchanged, with the following two exceptions. BACK gives the number of backtrackings performed; P(I, J) is set to 0 for all pairs (I, J) such that W(I, J) > C(I).

```
INTEGER P(10,100), W(10,100), C(10), XSTAR(100), Z, BACK
INTEGER DD(10), UD(10), Q(10), PAKL(10), IP(10), IR(10)
```

```
INTEGER IL(10), IF(10), WOBBL(10), KQ(10), FLREP(10)
INTEGER XS(100), BS(100), B(100), KA(100), XXS(100)
INTEGER IOBBL(100), JOBBL(100), BEST(100), XJJUB(100)
REAL DS(100)
INTEGER PS(101), WS(101)
INTEGER E(150), CC(150), CS(150)
INTEGER TYPE(150), US(150), UBL(150)
INTEGER A(10,100), X(10,100)
INTEGER PAK(10,100), KAP(10,100), MIND(10,100)
INTEGER D(150,10), VS(150,10)
INTEGER V(150,10), LB(150,10), UB(150,10)
INTEGER Y
INTEGER DMYR1(10), DMYR2(10), DMYR3(10)
INTEGER DMYR4(10), DMYR5(10)
INTEGER DMYC1(100), DMYC2(100), DMYC3(100)
INTEGER DMYC4(100), DMYC5(100), DMYC6(100)
INTEGER DMYC7(100), DMYC8(100), DMYC9(100)
INTEGER DMYC10(100), DMYC11(100), DMYC12(100)
INTEGER DMYC13(100)
INTEGER DMYCC1(101), DMYCC2(101)
REAL DMYCR1(100)
COMMON /PACK/ MASK1(30), ITWO(30), MASK, Y(150,100)
```

A.7.2 Code MTHG

```
SUBROUTINE MTHG  (N, M, P, W, C, MINMAX,
                  Z, XSTAR, JCK)
```

This subroutine heuristically solves the generalized assignment problem

$$\begin{aligned} \text{opt } Z = \ & P(1,1)\,X(1,1) + \ldots + P(1,N)\,X(1,N) \ + \\ & \qquad \ldots \qquad + \\ & P(M,1)\,X(M,1) + \ldots + P(M,N)\,X(M,N) \end{aligned}$$

(where opt = min if MINMAX = 1, opt = max if MINMAX = 2)

$$\begin{aligned} \text{subject to } \ & W(I,1)\,X(I,1) + \ldots + W(I,N)\,X(I,N) \le C(I) \\ & \qquad\qquad \text{for } I = 1,\ldots, M, \\ & X(1,J) + \ldots + X(M,J) = 1 \quad \text{for } J = 1,\ldots, N, \\ & X(I,J) = 0 \text{ or } 1 \text{ for } I = 1,\ldots, M, \ J = 1,\ldots, N. \end{aligned}$$

The program implements the polynomial-time algorithms described in Section 7.4.

The input problem must satisfy the conditions

(1) $2 \leq M \leq$ JDIMR;

(2) $2 \leq N \leq$ JDIMC (JDIMR and JDIMC are defined by the first two executable statements);

(3) P(I, J), W(I, J) and C(I) positive integers;

(4) W(I, J) $\leq$ C(I) for at least one I, for J = 1, ..., N;

(5) C(I) $\geq$ min (W(I, J)) for I = 1, ..., M.

MTHG calls 6 procedures: CHMTHG, FEAS, GHA, GHBCD, GHX and TRIN.

Communication to the program is achieved solely through the parameter list of MTHG.
No machine-dependent constant is used.

MTHG needs

6 arrays (C, DMYR1, DMYR2, DMYR3, DMYR4 and DMYR5) of length at least JDIMR;
7 arrays (XSTAR, BEST, DMYC1, DMYC2, DMYC3, DMYC4 and DMYCR1) of length at least JDIMC;
3 arrays (P, W and A) of length at least JDMR $\times$ JDIMC.

The arrays are currently dimensioned to allow problems for which

$M \leq 50$,
$N \leq 500$

(so, in the calling program, arrays P and W must be dimensioned at (50,500)). Changing such limits necessitates changing the dimension of all the arrays in subroutine MTHG, as well as the first two executable statements.

Meaning of the input parameters:

N = number of items;

M = number of knapsacks;

P(I, J) = profit of item J if assigned to knapsack I (I = 1, ..., M; J = 1, ..., N);

W(I, J) = weight of item J if assigned to knapsack I (I = 1, ..., M; J = 1, ..., N);

C(I) = capacity of knapsack I (I = 1, ..., M);

MINMAX = 1 if the objective function must be minimized,
= 2 if the objective function must be maximized;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if Z > 0,
= 0 if no feasible solution is found,
= error in the input data (when JCK = 1) if Z < 0:
condition −Z is violated;

XSTAR(J) = knapsack where item J is inserted in the solution found.

All the parameters are integer. On return of MTHG all the input parameters are unchanged, but P(I, J) is set to 0 for all pairs (I, J) such that W(I, J) > C(I).

```
INTEGER P(50,500), W(50,500), C(50), XSTAR(500), Z
INTEGER BEST(500)
INTEGER A(50,500)
INTEGER DMYR1(50), DMYR2(50), DMYR3(50)
INTEGER DMYR4(50), DMYR5(50)
INTEGER DMYC1(500), DMYC2(500), DMYC3(500)
INTEGER DMYC4(500)
REAL DMYCR1(500)
```

A.8 BIN-PACKING PROBLEM

A.8.1 Code MTP

```
SUBROUTINE MTP (N, W, C, Z, XSTAR,
                JDIM, BACK, JCK, LB,
                WR, XSTARR, DUM, RES, REL, X, R, WA,
                WB, KFIX, FIXIT, XRED, LS, LSB, XHEU)
```

This subroutine solves the bin packing problem

$$\text{minimize } Z = Y(1) + \ldots + Y(N)$$

$$\text{subject to } \quad W(1)\,X(I,1) + \ldots + W(N)\,X(I,N) \le C\,Y(I) \quad \text{for } I = 1,\ldots,N,$$

$$X(1,J) + \ldots + X(M,J) = 1 \qquad \text{for } J = 1,\ldots,N,$$

$$Y(I) = 0 \text{ or } 1 \qquad \text{for } I = 1,\ldots,N,$$

$$X(I,J) = 0 \text{ or } 1 \quad \text{for } I = 1,\ldots,N, \quad J = 1,\ldots,N$$

(i.e., minimize the number of bins of capacity C needed to allocate N items of size W(1), . . . , W(N)).

The program implements the branch-and-bound algorithm described in Section 8.5.

The input problem must satisfy the conditions

(1) $2 \leq N \leq JDIM$;

(2) W(J) and C positive integers;

(3) $W(J) \leq C$ for $J = 1, \ldots, N$;

(4) $W(J) \geq W(J + 1)$ for $J = 1, \ldots, N - 1$.

In the output solution (see below) the Z lowest indexed bins are used.

MTP calls 14 procedures: CHMTP, ENUMER, FFDLS, FIXRED, HBFDS, INSERT, LCL2, L2, L3, MWFDS, RESTOR, SEARCH, SORTI2 and UPDATE.

Communication to the program is achieved solely through the parameter list of MTP.
No machine-dependent constant is used.

MTP needs

17 arrays (W, XSTAR, WR, XSTARR, DUM, RES, REL, X, R, WA, WB, KFIX, FIXIT, XRED, LS, LSB and XHEU) of length at least JDIM.

Meaning of the input parameters:

N = number of items;

W(J) = weight of item J;

C = capacity of the bins;

JDIM = dimension of the 17 arrays;

BACK = −1 if exact solution is required,
= maximum number of backtrackings to be performed, if heuristic solution is required;

JCK = 1 if check on the input data is desired,
= 0 otherwise.

Meaning of the output parameters:

Z = value of the solution found if $Z > 0$,
= error in the input data (when JCK = 1) if $Z < 0$: condition $-Z$ is violated;

XSTAR(J) = bin where item J is inserted in the solution found;

LB = lower bound on the optimal solution value (to evaluate Z when BACK ≥ 0 on input).

All the arrays except W and XSTAR are dummy.

All the parameters are integer. On return of MTP all the input parameters are unchanged except BACK, which gives the number of backtrackings performed.

```
INTEGER W(JDIM), XSTAR(JDIM), C, Z, BACK
INTEGER WR(JDIM), XSTARR(JDIM), DUM(JDIM)
INTEGER RES(JDIM), REL(JDIM), X(JDIM), R(JDIM)
INTEGER WA(JDIM), WB(JDIM), KFIX(JDIM)
INTEGER FIXIT(JDIM), XRED(JDIM), LS(JDIM)
INTEGER LSD(JDIM), XHEU(JDIM)
```

Glossary

$O(f(n))$	order of $f(n)$
$\|S\|$	cardinality of set S
$r(A)$	worst-case performance ratio of algorithm A
$\varepsilon(A)$	worst-case relative error of algorithm A
$\rho(B)$	worst-case performance ratio of bound B
$\lfloor a \rfloor$	largest integer not greater than a
$\lceil a \rceil$	smallest integer not less than a
$z(P)$	optimal solution value of problem P
$C(P)$	continuous relaxation of problem P
$L(P, \lambda)$	Lagrangian relaxation of problem P through multiplier λ
$S(P, \pi)$	surrogate relaxation of problem P through multiplier π
$i \pmod j$	$i - \lfloor i/j \rfloor j$ $(i, j$ positive integers)
$\arg\max \{s_1, \ldots, s_n\}$	index k such that $s_k \geq s_i$ for $i = 1, \ldots, n$
$\max \{s_1, \ldots, s_n\}$	$s_{\arg\max \{s_1, \ldots, s_n\}}$
$\arg\max_2 \{s_1, \ldots, s_n\}$	$\arg\max (\{s_1, \ldots, s_n\} / \{s_{\arg\max \{s_1, \ldots, s_n\}}\})$
$\max_2 \{s_1, \ldots, s_n\}$	$s_{\arg\max_2 \{s_1, \ldots, s_n\}}$

arg min, min, arg $\min_2$, $\min_2$ are immediate extensions of the above

Bibliography

A.V. Aho, J.E. Hopcroft, J.D. Ullman (1983). *Data Structures and Algorithms,* Addison-Wesley, Reading, MA.

J.H. Ahrens, G. Finke (1975). Merging and sorting applied to the 0-1 knapsack problem. *Operations Research* **23**, 1099–1109.

L. Aittoniemi (1982). Computational comparison of knapsack algorithms, Presented at XIth International Symposium on Mathematical Programming, Bonn, August 23–27.

L. Aittoniemi, K. Oehlandt (1985). A note on the Martello–Toth algorithm for one-dimensional knapsack problems. *European Journal of Operational Research* **20**, 117.

R.D. Armstrong, D.S. Kung, P. Sinha, A.A. Zoltners (1983). A computational study of a multiple-choice knapsack algorithm. *ACM Transactions on Mathematical Software* **9**, 184–198.

G. d'Atri (1979). Analyse probabiliste du problème du sac-à-dos. *Thèse*, Université de Paris VI.

G. d'Atri, C. Puech (1982). Probabilistic analysis of the subset-sum problem. *Discrete Applied Mathematics* **4**, 329–334.

D. Avis (1980). Theorem 4. In V. Chvátal. Hard knapsack problems, *Operations Research* **28**, 1410–1411.

L.G. Babat (1975). Linear functions on the N-dimensional unit cube. *Doklady Akademiia Nauk SSSR* **222**, 761–762.

A. Bachem, M. Grötschel (1982). New aspects of polyhedral theory. In B. Korte (ed.), *Modern Applied Mathematics, Optimization and Operations Research*, North Holland, Amsterdam, 51–106.

B.S. Baker, E.G. Coffman Jr. (1981). A tight asymptotic bound for next–fit–decreasing bin packing. *SIAM Journal on Algebraic and Discrete Methods* **2**, 147–152.

E. Balas (1967). Discrete programming by the filter method. *Operations Research* **15**, 915–957.

E. Balas (1975). Facets of the knapsack polytope. *Mathematical Programming* **8**, 146–164.

E. Balas, R. Jeroslow (1972). Canonical cuts on the unit hypercube. *SIAM Journal of Applied Mathematics* **23**, 61–69.

E. Balas, R. Nauss, E. Zemel (1987). Comment on 'some computational results on real 0-1 knapsack problems'. *Operations Research Letters* **6**, 139.

E. Balas, E. Zemel (1978). Facets of the knapsack polytope from minimal covers. *SIAM Journal of Applied Mathematics* **34**, 119–148.

E. Balas, E. Zemel (1980). An algorithm for large zero-one knapsack problems. *Operations Research* **28**, 1130–1154.

R.S. Barr, G.T. Ross (1975). A linked list data structure for a binary knapsack algorithm. Research Report CCS 232, Centre for Cybernetic Studies, University of Texas.

R. Bellman (1954). Some applications of the theory of dynamic programming—a review. *Operations Research* **2**, 275–288.

R. Bellman (1957). *Dynamic Programming*, Princeton University Press, Princeton, NJ.

R. Bellman, S.E. Dreyfus (1962). *Applied Dynamic Programming*, Princeton University Press, Princeton, NJ.

R.L. Bulfin, R.G. Parker, C.M. Shetty (1979). Computational results with a branch and bound algorithm for the general knapsack problem. *Naval Research Logistics Quarterly* **26**, 41–46.

A.V. Cabot (1970). An enumeration algorithm for knapsack problems. *Operations Research* **18**, 306–311.

G. Carpaneto, S. Martello, P. Toth (1988). Algorithms and codes for the assignment problem. In B. Simeone, P. Toth, G. Gallo, F. Maffioli, S. Pallottino (eds), *Fortran Codes for Network Optimization, Annals of Operations Research* **13**, 193–223.

L. Chalmet, L. Gelders (1977). Lagrange relaxation for a generalized assignment-type problem. In M. Roubens (ed.), *Advances in Operations Research*, North-Holland, Amsterdam, 103–109.

S.K. Chang, A. Gill (1970a). Algorithmic solution of the change-making problem. *Journal of ACM* **17**, 113–122.

S.K. Chang, A. Gill (1970b). Algorithm 397. An integer programming problem. *Communications of ACM* **13**, 620–621.

L. Chang, J.F. Korsh (1976). Canonical coin-changing and greedy solutions. *Journal of ACM* **23**, 418–422.

N. Christofides, A. Mingozzi, P. Toth (1979). Loading problems. In N. Christofides, A. Mingozzi, P. Toth, C. Sandi (eds), *Combinatorial Optimization*, Wiley, Chichester, 339–369.

V. Chvátal (1980). Hard knapsack problems. *Operations Research* **28**, 402–411.

E.G. Coffman Jr., M.R. Garey, D.S. Johnson (1984). Approximation algorithms for bin-packing—an updated survey. In G. Ausiello, M. Lucertini, P. Serafini (eds), *Algorithm Design for Computer System Design*, Springer, Vienna, 49–106.

J. Cord (1964). A method for allocating funds to investment projects when returns are subject to uncertainty. *Management Science* **10**, 335–341.

H. Crowder, E.L. Johnson, M.W. Padberg (1983). Solving large-scale zero-one linear programming problems. *Operations Research* **31**, 803–834.

G.B. Dantzig (1957). Discrete variable extremum problems. *Operations Research* **5**, 266–277.

A. De Maio, C. Roveda (1971). An all zero-one algorithm for a certain class of transportation problems. *Operations Research* **19**, 1406–1418.

R.S. Dembo, P.L. Hammer (1980). A reduction algorithm for knapsack problems. *Methods of Operations Research* **36**, 49–60.

B.L. Dietrich, L.F. Escudero (1989a). More coefficient reduction for knapsack-like constraints in 0-1 programs with variable upper bounds. IBM T.J. Watson Research Center. RC-14389, Yorktown Heights (NY).

B.L. Dietrich, L.F. Escudero (1989b). New procedures for preprocessing 0-1 models with knapsack-like constraints and conjunctive and/or disjunctive variable upper bounds. IBM T.J. Watson Research Center. RC-14572, Yorktown Heights (NY).

K. Dudzinski, S. Walukiewicz (1984a). Upper bounds for the 0-1 knapsack problem. Report MPD–10–49/84, Systems Research Institute, Warsaw.

K. Dudzinski, S. Walukiewicz (1984b). A fast algorithm for the linear multiple-choice knapsack problem. *Operations Research Letters* **3**, 205–209.

K. Dudzinski, S. Walukiewicz (1987). Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research* **28**, 3–21.

M.E. Dyer (1984). An $O(n)$ algorithm for the multiple-choice knapsack linear program. *Mathematical Programming* **29**, 57–63.

M.E. Dyer, N. Kayal, J. Walker (1984). A branch and bound algorithm for solving the multiple-choice knapsack problem. *Journal of Computational and Applied Mathematics* **11**, 231–249.

S. Eilon, N. Christofides (1971). The loading problem. *Management Science* **17**, 259–267.

B. Faaland (1973). Solution of the value-independent knapsack problem by partitioning. *Operations Research* **21**, 332–337.

D. Fayard, G. Plateau (1975). Resolution of the 0-1 knapsack problem: comparison of methods. *Mathematical Programming* **8**, 272–307.

D. Fayard, G. Plateau (1982). An algorithm for the solution of the 0-1 knapsack problem. *Computing* **28**, 269–287.
M. Fischetti (1986). Worst-case analysis of an approximation scheme for the subset-sum problem. *Operations Research Letters* **5**, 283–284.
M. Fischetti (1989). A new linear storage, polynomial time approximation scheme for the subset-sum problem. *Discrete Applied Mathematics* (to appear).
M. Fischetti, S. Martello (1988). A hybrid algorithm for finding the kth smallest of n elements in $O(n)$ time. In B. Simeone, P. Toth, G. Gallo, F. Maffioli, S. Pallottino (eds), *Fortran Codes for Network Optimization, Annals of Operations Research* **13**, 401–419.
M. Fischetti, P. Toth (1988). A new dominance procedure for combinatorial optimization problems. *Operations Research Letters* **7**, 181–187.
M.L. Fisher (1980). Worst-case analysis of heuristic algorithms. *Management Science* **26**, 1–17.
M.L. Fisher (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science* **27**, 1–18.
M.L. Fisher, R. Jaikumar, L.N. Van Wassenhove (1986). A multiplier adjustment method for the generalized assignment problem. *Management Science* **32**, 1095–1103.
J.C. Fisk, M.S. Hung (1979). A heuristic routine for solving large loading problems. *Naval Research Logistics Quarterly* **26**, 643–650.
A.M. Frieze (1986). On the Lagarias-Odlyzko algorithm for the subset sum problem. *SIAM Journal on Computing* **15**, 536–539.
M.R. Garey, D.S. Johnson (1975). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing* **4**, 397–411.
M.R. Garey, D.S. Johnson (1978). "Strong" NP-completeness results: motivation, examples and implications. *Journal of ACM* **25**, 499–508.
M.R. Garey, D.S. Johnson (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.
R.S. Garfinkel, G.L. Nemhauser (1972). *Integer Programming*, John Wiley and Sons, New York.
G.V. Gens, E.V. Levner (1978). Approximation algorithms for scheduling problems. *Izvestija Akademii Nauk SSSR, Engineering Cybernetics* **6**, 38–43.
G.V. Gens, E.V. Levner (1979). Computational complexity of approximation algorithms for combinatorial problems. In J. Bečvař (ed.), *Mathematical Foundations of Computer Science 1979*, Lecture Notes in Computer Science 74, Springer, Berlin, 292–300.
G.V. Gens, E.V. Levner (1980). Fast approximation algorithms for knapsack type problems. In K. Iracki, K. Malinowski, S. Walukiewicz (eds), *Optimization Techniques, Part 2*, Lecture Notes in Control and Information Sciences 23, Springer, Berlin, 185–194.
A. Geoffrion (1969). An improved implicit enumeration approach for integer programming. *Operations Research* **17**, 437–454.
P.C. Gilmore, R.E. Gomory (1961). A linear programming approach to the cutting stock problem I. *Operations Research* **9**, 849–858.
P.C. Gilmore, R.E. Gomory (1963). A linear programming approach to the cutting stock problem II. *Operations Research* **11**, 863–888.
P.C. Gilmore, R.E. Gomory (1965). Multi-stage cutting stock problems of two and more dimensions. *Operations Research* **13**, 94–120.
P.C. Gilmore, R.E. Gomory (1966). The theory and computation of knapsack functions. *Operations Research* **14**, 1045–1074.
F. Glover (1965). A multiphase dual algorithm for the zero-one integer programming problem. *Operations Research* **13**, 879–919.
F. Glover, D. Klingman (1979). A $o(n \log n)$ algorithm for LP knapsacks with GUB constraints. *Mathematical Programming* **17**, 345–361.
A.V. Goldberg, A. Marchetti-Spaccamela (1984). On finding the exact solution to a zero-one knapsack problem. *Proc. 16th Annual ACM Symposium Theory of Computing*, 359–368.
E.S. Gottlieb, M.R. Rao (1988). Facets of the knapsack polytope derived from disjoint and

overlapping index configurations. *Operations Research Letters* **7**, 95–100.

E.S. Gottlieb, M.R. Rao (1989a). The generalized assignment problem: valid inequalities and facets. *Mathematical Programming* (to appear).

E.S. Gottlieb, M.R. Rao (1989b). (1,k)-configuration facets for the generalized assignment problem. *Mathematical Programming* (to appear).

H. Greenberg (1985). An algorithm for the periodic solutions in the knapsack problem. *Journal of Mathematical Analysis and Applications* **111**, 327–331.

H. Greenberg (1986). On equivalent knapsack problems. *Discrete Applied Mathematics* **14**, 263–268.

H. Greenberg, I. Feldman (1980). A better-step-off algorithm for the knapsack problem. *Discrete Applied Mathematics* **2**, 21–25.

H. Greenberg, R.L. Hegerich (1970). A branch search algorithm for the knapsack problem. *Management Science* **16**, 327–332.

M.M. Guignard, S. Kim (1987). Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming* **39**, 215–228.

M.M. Guignard, K. Spielberg (1972). Mixed-integer algorithms for the (0,1) knapsack problem. *IBM Journal of Research and Development* **16**, 424–430.

P.L. Hammer, E.L. Johnson, U.N. Peled (1975). Facets of regular 0-1 polytopes. *Mathematical Programming* **8**, 179–206.

D. Hartvigsen, E. Zemel (1987). On the complexity of lifted inequalities for the knapsack problem. Report 740, Department of Managerial Economics and Decision Sciences, Northwestern University, Evanston, Illinois.

D.S. Hirschberg, C.K. Wong (1976). A polynomial-time algorithm for the knapsack problem with two variables. *Journal of ACM* **23**, 147–154.

E. Horowitz, S. Sahni (1974). Computing partitions with applications to the knapsack problem. *Journal of ACM* **21**, 277–292.

T.C. Hu (1969). *Integer Programming and Network Flows*, Addison-Wesley, New York.

T.C. Hu, M.L. Lenard (1976). Optimality of a heuristic solution for a class of knapsack problems. *Operations Research* **24**, 193–196.

P.D. Hudson (1977). Improving the branch and bound algorithms for the knapsack problem. Queen's University Research Report, Belfast.

M.S. Hung, J.R. Brown (1978). An algorithm for a class of loading problems. *Naval Research Logistics Quarterly* **25**, 289–297.

M.S. Hung, J.C. Fisk (1978). An algorithm for 0-1 multiple knapsack problems. *Naval Research Logistics Quarterly* **24**, 571–579.

O.H. Ibarra, C.E. Kim (1975). Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of ACM* **22**, 463–468.

G.P. Ingargiola, J.F. Korsh (1973). A reduction algorithm for zero-one single knapsack problems. *Management Science* **20**, 460–463.

G.P. Ingargiola, J.F. Korsh (1975). An algorithm for the solution of 0-1 loading problems. *Operations Research* **23**, 1110–1119.

G.P. Ingargiola, J.F. Korsh (1977). A general algorithm for one-dimensional knapsack problems. *Operations Research* **25**, 752–759.

D.S. Johnson (1973). Near-optimal bin packing algorithms. Technical Report MAC TR-109, Project MAC, Massachusetts Institute of Technology, Cambridge, MA.

D.S. Johnson (1974). Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences* **9**, 256–278.

D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, R.L. Graham (1974). Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing* **3**, 299–325.

S.C. Johnson, B.W. Kernighan (1972). Remarks on algorithm 397. *Communications of ACM* **15**, 469.

K. Jörnsten, M. Näsberg (1986). A new Lagrangian relaxation approach to the generalized assignment problem. *European Journal of Operational Research* **27**, 313–323.

R. Kannan (1980). A polynomial algorithm for the two-variables integer programming problem. *Journal of ACM* **27**, 118–122.

S. Kaplan (1966). Solution of the Lorie-Savage and similar integer programming problems by the generalized Lagrange multiplier method. *Operations Research* **14**, 1130–1136.

R.M. Karp (1972). Reducibility among combinatorial problems. In R.E. Miller, J.W. Thatcher (eds), *Complexity of Computer Computations*, Plenum Press, New York, 85–103.

R.M. Karp, J.K. Lenstra, C.J.H. McDiarmid, A.H.G. Rinnooy Kan (1985). Probabilistic analysis. In M. O'hEigeartaigh, J.K. Lenstra, A.H.G. Rinnooy Kan (eds), *Combinatorial Optimization: Annotated Bibliographies*, Wiley, Chichester, 52–88.

T.D. Klastorin (1979). An effective subgradient algorithm for the generalized assignment problem. *Computers and Operations Research* **6**, 155–164.

D.E. Knuth (1973). *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, MA.

P.J. Kolesar (1967). A branch and bound algorithm for the knapsack problem. *Management Science* **13**, 723–735.

N.W. Kuhn (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* **2**, 83–97.

J.C. Lagarias, A.M. Odlyzko (1983). Solving low-density subset sum problems. *Proc. 24th Annual IEEE Symposium Foundations of Computer Science*, 1–10.

B.J. Lageweg, J.K. Lenstra (1972). Algoritmend voor knapzack problemen. Report BN 14/72, Stichting Mathematisch Centrum, Amsterdam.

M. Laurière (1978). An algorithm for the 0-1 knapsack problem. *Mathematical Programming* **14**, 1–10.

E.L. Lawler (1976). *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart & Winston, New York.

E.L. Lawler (1979). Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research* **4**, 339–356.

E.V. Levner, G.V. Gens (1978). *Discrete Optimization Problems and Approximation Algorithms*. Moscow, CEMI (Russian).

G.S. Lueker (1975). Two NP-complete problems in nonnegative integer programming. Report No. 178, Computer Science Laboratory, Princeton University, Princeton, NJ.

G.S. Lueker (1982). On the average difference between the solutions to linear and integer knapsack problems. In R.L. Disney, T.J. Ott (eds), *Applied Probability—Computer Science: the Interface, Vol. I*, Birkhauser, Basel, 489–504.

N. Maculan (1983). Relaxation Lagrangienne: le problème du knapsack 0-1. INFOR *(Canadian Journal of Operational Research and Information Processing)* **21**, 315–327.

M.J. Magazine, J.L. Nemhauser, L.E. Trotter Jr. (1975). When the greedy solution solves a class of knapsack problems. *Operations Research* **23**, 207–217.

M.J. Magazine, O. Oguz (1981). A fully polynomial approximate algorithm for the 0-1 knapsack problem. *European Journal of Operational Research* **8**, 270–273.

A. Marchetti-Spaccamela, C. Vercellis (1987). Efficient on-line algorithms for the knapsack problem. In T. Ottman (ed.), *Automata, Languages and Programming*, Lecture Notes in Computer Science 267, Springer, Berlin, 445–456.

S. Martello, P. Toth (1977a). An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research* **1**, 169–175.

S. Martello, P. Toth (1977b). Computational experiences with large-size unidimensional knapsack problems. Presented at the TIMS/ORSA Joint National Meeting, San Francisco.

S. Martello, P. Toth (1977c). Solution of the bounded and unbounded change-making problem. Presented at the TIMS/ORSA Joint National Meeting, San Francisco.

S. Martello, P. Toth (1977d). Branch and bound algorithms for the solution of the general unidimensional knapsack problem. In M. Roubens (ed.), *Advances in Operations Research*, North-Holland, Amsterdam, 295–301.

S. Martello, P. Toth (1978). Algorithm for the solution of the 0-1 single knapsack problem. *Computing* **21**, 81–86.
S. Martello, P. Toth (1979). The 0-1 knapsack problem. In N. Christofides, A. Mingozzi, P. Toth, C. Sandi (eds), *Combinatorial Optimization*, Wiley, Chichester, 237–279.
S. Martello, P. Toth (1980a). Solution of the zero-one multiple knapsack problem. *European Journal of Operational Research* **4**, 276–283.
S. Martello, P. Toth (1980b). Optimal and canonical solutions of the change-making problem. *European Journal of Operational Research* **4**, 322–329.
S. Martello, P. Toth (1980c). A note on the Ingargiola–Korsh algorithm for one-dimensional knapsack problems. *Operations Research* **28**, 1226–1227.
S. Martello, P. Toth (1981a). A bound and bound algorithm for the zero-one multiple knapsack problem. *Discrete Applied Mathematics* **3**, 275–288.
S. Martello, P. Toth (1981b). Heuristic algorithms for the multiple knapsack problem. *Computing* **27**, 93–112.
S. Martello, P. Toth (1981c). An algorithm for the generalized assignment problem. In J.P. Brans (ed.), *Operational Research '81*, North-Holland, Amsterdam, 589–603.
S. Martello, P. Toth (1984a). A mixture of dynamic programming and branch-and-bound for the subset-sum problem. *Management Science* **30**, 765–771.
S. Martello, P. Toth (1984b). Worst-case analysis of greedy algorithms for the subset-sum problem. *Mathematical Programming* **28**, 198–205.
S. Martello, P. Toth (1985a). Approximation schemes for the subset-sum problem: survey and experimental analysis. *European Journal of Operational Research* **22**, 56–69.
S. Martello, P. Toth (1985b). Algorithm 632. A program for the 0-1 multiple knapsack problem. *ACM Transactions on Mathematical Software* **11**, 135–140.
S. Martello, P. Toth (1987). Algorithms for knapsack problems. In S. Martello, G. Laporte, M. Minoux, C. Ribeiro (eds), *Surveys in Combinatorial Optimization, Annals of Discrete Mathematics* 31, North-Holland, Amsterdam, 213–257.
S. Martello, P. Toth (1988). A new algorithm for the 0-1 knapsack problem. *Management Science* **34**, 633–644.
S. Martello, P. Toth (1989). An exact algorithm for the bin packing problem. Presented at EURO X, Beograd.
S. Martello, P. Toth (1990a). An exact algorithm for large unbounded knapsack problems. *Operations Research Letters* (to appear).
S. Martello, P. Toth (1990b). Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics* (to appear).
J.B. Mazzola (1989). Generalized assignment with nonlinear capacity interaction. *Management Science* **35**, 923–941.
M. Meanti, A.H.G. Rinnooy Kan, L. Stougie, C. Vercellis (1989). A probabilistic analysis of the multiknapsack value function. *Mathematical Programming* (to appear).
H. Müller-Merbach (1978). An improved upper bound for the zero-one knapsack problem: a note on the paper by Martello and Toth. *European Journal of Operational Research* **2**, 212–213.
R.A. Murphy (1986). Some computational results on real 0-1 knapsack problems. *Operations Research Letters* **5**, 67–71.
R.M. Nauss (1976). An efficient algorithm for the 0-1 knapsack problem. *Management Science* **23**, 27–31.
R.M. Nauss (1978). The 0-1 knapsack problem with multiple choice constraints. *European Journal of Operational Research* **2**, 125–131.
A. Neebe, D. Dannenbring (1977). Algorithms for a specialized segregated storage problem. Technical Report 77–5, University of North Carolina.
G.L. Nemhauser, L.E. Trotter (1974). Properties of vertex packing and independence system polyhedra. *Mathematical Programming* **6**, 48–61.

G.L. Nemhauser, Z. Ullmann (1969). Discrete dynamic programming and capital allocation. *Management Science* **15**, 494–505.

G.L. Nemhauser, L.A. Wolsey (1988). *Integer and Combinatorial Optimization*, Wiley, Chichester.

M.W. Padberg (1975). A note on zero-one programming. *Operations Research* **23**, 833–837.

M.W. Padberg (1979). Covering, packing and knapsack problems. *Annals of Discrete Mathematics* **4**, 265–287.

M.W. Padberg (1980). (1,k)-configurations and facets for packing problems. *Mathematical Programming* **18**, 94–99.

C.H. Papadimitriou, K. Steiglitz (1982). *Combinatorial Optimization*, Prentice-Hall, Englewood Cliffs, NJ.

G. Plateau, M. Elkihel (1985). A hybrid algorithm for the 0-1 knapsack problem. *Methods of Operations Research* **49**, 277–293.

W.R. Pulleyblank (1983). Polyhedral combinatorics. In A. Bachem, M. Grötschel, B. Korte (eds), *Mathematical Programming: the State of the Art–Bonn 1982*, Springer, Berlin, 312–345.

A.H.G. Rinnooy Kan (1987). Probabilistic analysis of algorithms. In S. Martello, G. Laporte, M. Minoux, C. Ribeiro (eds), *Surveys in Combinatorial Optimization, Annals of Discrete Mathematics* **31**, North-Holland, Amsterdam, 365–384.

G.T. Ross, R.M. Soland (1975). A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming* **8**, 91–103.

B.F. Ryder, A.D. Hall (1981). The PFORT verifier. Computer Science Report 2, Bell Laboratories.

S. Sahni (1975). Approximate algorithms for the 0-1 knapsack problem. *Journal of ACM* **22**, 115–124.

S. Sahni, T. Gonzalez (1976). P-complete approximation problems. *Journal of ACM* **23**, 555–565.

H.M. Salkin (1975). *Integer Programming*, Addison-Wesley, New York.

H.M. Salkin, C.A. de Kluyver (1975). The knapsack problem: a survey. *Naval Research Logistics Quarterly* **22**, 127–144.

A. Schrijver (1986). *Theory of Linear and Integer Programming*, Wiley, Chichester.

P. Sinha, A.A. Zoltners (1979). The multiple-choice knapsack problem. *Operations Research* **27**, 503–515.

V. Srinivasan, G.L. Thompson (1973). An algorithm for assigning uses to sources in a special class of transportation problems. *Operations Research* **21**, 284–295.

U. Suhl (1978). An algorithm and efficient data structures for the binary knapsack problem. *European Journal of Operational Research* **2**, 420–428.

M.M. Syslo, N. Deo, J.S. Kowalik (1983). *Discrete Optimization Algorithms with Pascal Programs*, Prentice-Hall, Englewood Cliffs, NJ.

K. Szkatula, M. Libura (1987). On probabilistic properties of greedy-like algorithms for the binary knapsack problem. Report 154, Instytut Badan Systemowych, Polska Akademia Nauk, Warsaw.

H.A. Taha (1975). *Integer Programming*, Academic Press, New York.

B.N. Tien, T.C. Hu (1977). Error bounds and the applicability of the greedy solution to the coin-changing problem. *Operations Research* **25**, 404–418.

G. Tinhofer, H. Schreck (1986). The bounded subset sum problem is almost everywhere randomly decidable in $O(n)$. *Information Processing Letters* **23**, 11–17.

M. Todd (1980). Theorem 3. In V. Chvátal. Hard knapsack problems, *Operations Research* **28**, 1408–1409.

P. Toth (1976). A new reduction algorithm for 0-1 knapsack problems. Presented at the ORSA/TIMS Joint National Meeting, Miami.

P. Toth (1980). Dynamic programming algorithms for the zero-one knapsack problem. *Computing* **25**, 29–45.

G.P. Veliev, K.Sh. Mamedov (1981). A method of solving the knapsack problem. *USSR Computational Mathematics and Mathematical Physics* **21**, 75–81.

A. Verebriusova (1904). On the number of solutions of indefinite equations of the first degree with many variables. *Mathematicheskii Sbornik* **24**, 662–688.

P.R.C. Villela, C.T. Bornstein (1983). An improved bound for the 0-1 knapsack problem. Report ES31-83, COPPE-Federal University of Rio de Janeiro.

H.M. Weingartner (1963). *Mathematical Programming and the Analysis of Capital Budgeting Problems*, Prentice-Hall, Englewood Cliffs, NJ.

H.M. Weingartner (1968). Capital budgeting and interrelated projects: survey and synthesis. *Management Science* **12**, 485–516.

H.M. Weingartner, D.N. Ness (1967). Methods for the solution of the multi-dimensional 0-1 knapsack problem. *Operations Research* **15**, 83–103.

L.A. Wolsey (1975). Faces of linear inequalities in 0-1 variables. *Mathematical Programming* **8**, 165–178.

J.W. Wright (1975). The change-making problem. *Journal of ACM* **22**, 125–128.

E. Zemel (1978). Lifting the facets of zero-one polytopes. *Mathematical Programming* **15**, 268–277.

E. Zemel (1980). The linear multiple choice knapsack problem. *Operations Research* **28**, 1412–1423.

E. Zemel (1984). An $O(n)$ algorithm for the linear multiple choice knapsack problem and related problems. *Information Processing Letters* **18**, 123–128.

E. Zemel (1988). Easily computable facets of the knapsack polytope. Report 713, Department of Managerial Economics and Decision Sciences, Northwestern University, Evanston, Illinois.

A.A. Zoltners (1978). A direct descent binary knapsack algorithm. *Journal of ACM* **25**, 304–311.

Author index

Note: listing in references section is indicated by bold page numbers.

Subject index

Note: abbreviations used in the text and in this index:

BCMP	= Bounded Change-Making Problem
BKP	= Bounded Knapsack Problem
BPP	= Bin-Packing Problem
CMP	= Change-Making Problem
GAP	= Generalized Assignment Problem
KP	= 0-1 Knapsack Problem
MCKP	= Multiple-Choice Knapsack Problem
MKP	= 0-1 Multiple Knapsack Problem
SSP	= Subset-Sum Problem
UEMKP	= Unbounded Equality Constrained Min-Knapsack Problem
UKP	= Unbounded Knapsack Problem