

BA

**stichting
mathematisch
centrum**



BA

AFDELING MATHEMATISCHE BESLISKUNDE

BN 12/72

JUNI

M. REM
EEN PROGRAMMA VOOR PRODUKTIE-PLANNING

2e boerhaavestraat 49 amsterdam

BIBLIOTHEEK MATHEMATISCH CENTRUM
AMSTERDAM

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Voorwoord

Er bestaan allerlei 'machine loading problems'. Dit rapport beschrijft een oplossingsmethode voor een klasse daarvan. Te weten de produktieplannings over een aantal perioden, waarbij verschillende produkten op dezelfde machine vervaardigd moeten worden. Hierbij dient, bij bekende produktietijd en -kosten de som van voorraad- en omschakelkosten geminimaliseerd te worden.

De gevolgde methode werd gesuggereerd door Prof.Dr. G. de Leve. Het onderzoek werd uitgevoerd in het kader van een besliskunde-scriptie. De methode garandeert dat binnen een eindige tijd de beste oplossing wordt gevonden. Deze 'eindige tijd' kan echter snel met de grootte van het probleem toenemen.

Het rapport bevat tevens een ALGOL 60-programma voor het machine loading problem met de specificaties zoals die in hoofdstuk 0 genoemd worden.

De schrijver wil zijn dank betuigen voor de waardevolle suggesties die hij mocht ontvangen van Prof.Dr. G. de Leve en Drs.B. Dorhout.

<u>Inhoud</u>	blz.
0. Inleiding	3
1. Mathematische formulering	6
2. Branch and bound	8
3. Update	12
4. Trommel en buffers	14
5. Enkele opmerkingen	17
6. Programmatekst	18
7. Een voorbeeld	30
8. Literatuur	36

0. Inleiding.

Het probleem waar de rest van dit rapport aan gewijd is, is het volgende:

Er zijn een aantal verschillende produkten die op dezelfde machine moeten worden vervaardigd.

De vraag naar deze produkten is voor enkele weken bekend.

Het kost tijd en geld om om te schakelen van het ene naar het andere produkt (schoonmaaktijd en -kosten).

Het kost ook geld als produkten in een eerdere week worden gemaakt, dan de week waarin ze moeten worden afgeleverd (voorraadkosten).

(Opm.: Men kan desgewenst ook nalevering toestaan. De kosten voor nageleverde produkten worden dan met een boete verhoogd.)

Iedere week wordt als een afzonderlijke eenheid beschouwd; men kan zich voorstellen dat aan het eind van de week altijd schoongemaakt wordt.

Een produktie kan dus niet van de ene in de andere week doorlopen, maar wel kan een produktie over verschillende weken verdeeld worden.

Voor de produkten bestaat een ordening. Deze ordening bepaalt voor iedere week de volgorde waarin de produkten moeten worden gedraaid.

De schoonmaaktijd en -kosten zijn afhankelijk van de twee produkten waartussen omgeschakeld wordt.

De bedoeling is de produktie te vinden, waarbij de som van schoonmaak- en voorraadkosten minimaal is.

De strategie volgens welke we dit probleem zullen oplossen, is in grote lijnen de volgende:

We zoeken eerst de beste oplossing zonder de schoonmaaktijd en -kosten er in op te nemen. Deze berekening is (zoals in hoofdstuk 1 zal worden aangetoond) equivalent met het oplossen van het standaard-transport-probleem. Voor het oplossen van dit probleem bestaat een ALGOL 60-procedure 'real stepping stone', die geschreven is door Drs.B. Dorhout. Deze oplossing zal i.h.a. te gunstig zijn, en wel om de volgende twee redenen: de schoonmaakkosten zijn nog niet opgenomen, en door het toevoegen van schoonmaaktijden, kan het zijn dat de in een week geplande produkties niet meer in de overgebleven tijd passen. Hierdoor moeten produkties naar andere weken verschoven worden, hetgeen weer een toename van de voorraadkosten ten gevolge heeft.

Deze eerste berekening verschaft ons echter wel een ondergrens voor de kosten.

Het kan zijn dat we na deze eerste berekening al klaar zijn. Bijvoorbeeld in het geval dat er voor iedere week maar één produktie gepland is; we hoeven dan namelijk nooit tijdens een week van produktie te wisselen. Zijn echter alle schoonmaaktijd en -kosten nog niet opgenomen, dan gaan we voor ieder van de geplande produkties, waarvoor nog niet schoongemaakt is, bepalen hoeveel het tenminste gaat kosten als we hem wel, en als we hem niet uitvoeren.

Als we besluiten een produktie uit de planning metterdaad te gaan maken, (nog zonder te letten op de grootte) dan zullen daar i.h.a. kosten mee gemoeid zijn; als we anderzijds een geplande produktie niet gaan maken, dan zal er een nieuwe (i.h.a. duurdere) planning moeten komen met die produktie er niet in. We maken voor ieder van de geplande produkties een onderschatting van de toename van de kosten als we deze produktie wel en als we hem niet maken. Vervolgens bepalen we die produktie waarbij het verschil tussen deze twee schattingen het grootst is. (Opm.: Omdat voor al die produkties geldt dat ze òf wel, òf niet gemaakt moeten worden, kunnen we terloops een onderschatting geven van de nog te maken kosten.)

De berekening splitst zich nu in twee takken ('branch'), terwijl we in iedere tak een onderschatting ('bound') voor de kosten geven. In de ene tak nemen we het besluit die produktie wel te maken, en in de andere tak het besluit om hem niet te maken. In beide gevallen zoeken we weer met de methode voor het oplossen van het transportprobleem de optimale produktie-planning in de nieuwe omstandigheden. (In de nieuwe omstandigheden wil dan zeggen: met minder beschikbare tijd omdat er schoonmaaktijd afgegaan is, of met een bepaalde produktie verboden, omdat dat zo besloten is.)

We zetten onze 'branch and bound'-arbeid steeds in die tak voort waar de onderschatting van de kosten (som van de gemaakte kosten en de onderschatting van de nog te maken kosten) het laagst is. Dit doen we totdat de tak met de laagste kosten, een oplossing is waarvoor alle

schoonmaaktijd en -kosten ingecalculeerd zijn. Dan hebben we de optimale oplossing gevonden, en zijn we klaar.

De juiste wijze waarop dit proces verloopt, wordt in de hoofdstukken 2 en 3 besproken.

Een uitbreiding van dit probleem zou zijn het geval dat we de beschikking hebben over een aantal (verschillende) machines. Het kan dan voorkomen dat produkten op sommige machines wel en op andere weer niet gemaakt kunnen worden, en op de ene machine weer sneller dan op de andere. Dit heeft tot gevolg dat er in de vergelijkingen van het transportprobleem coëfficiënten verschijnen, zodat we steeds het gegeneraliseerde i.p.v. het standaard-transportprobleem moeten oplossen. De rest van het programma, en met name de strategie zoals die in de rest van dit rapport besproken wordt, hoeft echter niet gewijzigd te worden.

Ook voor het gegeneraliseerde transportprobleem is een ALGOL-procedure geschreven, en wel door Drs.W. Hoffmann.

1. Mathematische formulering.

M = aantal produkten

n = aantal weken

We maken onderscheid tussen produkten uit verschillende weken, door het i^e produkt ($1 \leq i \leq M$) dat in de j^e week ($1 \leq j \leq n$) geleverd moet worden produkt nummer $(j - 1) * M + i$ te noemen.

Omdat er maar één machine is, kunnen we de vraag en de produktie uitdrukken in het aantal tijdseenheden dat het kost om die vraag of die produktie te maken.

a_i = vraag naar produkt i ($1 \leq i \leq M * n$)

b_j = produktietijd in week j ($1 \leq j \leq n$)

x_{ij} = de hoeveelheid die van produkt i in week j gemaakt wordt.

We voeren nog een $(M * n + 1)^e$ produkt in met vraag

$$a_{M*n+1} = \sum_{j=1}^n b_j - \sum_{i=1}^{M*n} a_i ,$$

zodat

$$\sum_{j=1}^n b_j = \sum_{i=1}^{M*n+1} a_i .$$

We noemen $M * n + 1$ voortaan m .

c_{ij} = voorraadkosten voor produkt i als het in week j gemaakt wordt, berekend volgens de volgende algoritme:

$c_{ij} :=$ if $i = m$ v week van produktie = week van levering then 0 else
if week van produktie > week van levering then ∞ else
 (week van levering - week van produktie) * v_i ;

waarin: v_i = voorraadkosten per week voor produkt i .

Als we afzien van schoonmaaktijd en -kosten wordt het probleem:

$$\text{minimaliseer} \quad \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij}$$

$$\text{onder:} \quad \sum_{j=1}^n x_{ij} = a_i \quad (1 \leq i \leq m)$$

$$\sum_{i=1}^m x_{ij} = b_j \quad (1 \leq j \leq n)$$

$$x_{ij} \geq 0$$

Dit is de standaardversie van het transportprobleem. De oplossing ervan geeft ons een absoluut minimum voor de kosten, dat als uitgangspunt zal dienen wanneer we al "branch and boundend" de schoonmaaktijd en -kosten gaan opnemen.

2. Branch and bound.

De oplossing van het in hoofdstuk 1 genoemde transportprobleem m.b.v. de procedure 'real stepping stone', geeft ons een eerste produktieplanning met de daarbij horende kosten, en daarmee de wortel van de binaire boom.

Een aanroep van 'update' ('update' wordt verderop besproken) zorgt er voor dat de wortel gemakkelijk kan worden uitgebreid. Hierna wordt de hele "status-quo" naar de trommel gestuurd ("geswapt") op de wijze die in hoofdstuk 4 besproken wordt.

Laten we aannemen dat we al een stuk van de boom hebben; het uitbreiden van de boom gaat dan als volgt.

In het array 'drcost' staan de kosten die horen bij de uiteinden van de takken van de boom, voorzover deze niet ∞ zijn.

Met ieder element van 'drcost' correspondeert een stuk trommelruimte waar alle belangrijke gegevens staan voor het geval we de boom bij die tak gaan uitbreiden.

Nadat we de tak met de laagste kosten gevonden hebben, wordt - indien deze tak zich niet reeds in het kerngeheugen bevindt - door een "swap-in" de statusquo hersteld van het moment waarop we deze tak aanbrachten.

We kunnen nu zien met welke produktie uit de door 'real stepping stone' voorgestelde optimale produktieplanning we deze tak het beste kunnen uitbreiden. Dat zal met die produktie zijn waarbij het verschil in kosten tussen wel en niet produceren het grootst is. Dit is door 'update' al bepaald.

Er zijn nu twee beslissingen te nemen: de produktie wel maken, en de produktie niet maken. Dit gebeurt in de procedures 'production' en 'no production'. We nemen eerst de ongunstige beslissing.

Als we besluiten een produktie wel te maken moeten we de benodigde schoonmaaktijd reserveren en de schoonmaakkosten bij de reeds gemaakte schoonmaakkosten optellen.

Indien we in week i t tijdseenheden schoonmaaktijd moeten opnemen, voeren we de volgende wijziging uit:

if $b_i \geq t \wedge a_m \geq t$ then begin $b_i := b_i - t$; $a_m := a_m - t$ end else
begin schoonmaakkosten := ∞ ; kap deze tak end;

Als we besluiten de produktie niet te maken zetten we de toepasselijke c_{ij} op ∞ .

In beide gevallen blijft het probleem het standaardtransportprobleem, zodat we na deze beslissing weer een optimale produktieplanning kunnen vinden m.b.v. de procedure 'real stepping stone'.

Vervolgens gaan we een onderschatting maken van de verdere kosten die we in deze tak nog zullen moeten maken, alsmede een lijst van gunstige beslissingen voor de volgende keer dat we deze tak uitbreiden. Dit gebeurt in de procedure 'update'.

Hierna wordt de hele status-quo (indien de kosten niet reeds ∞ zijn geworden) geswapt, terwijl in 'drcost' de kosten worden genoteerd. De kosten bestaan uit drie gedeelten:

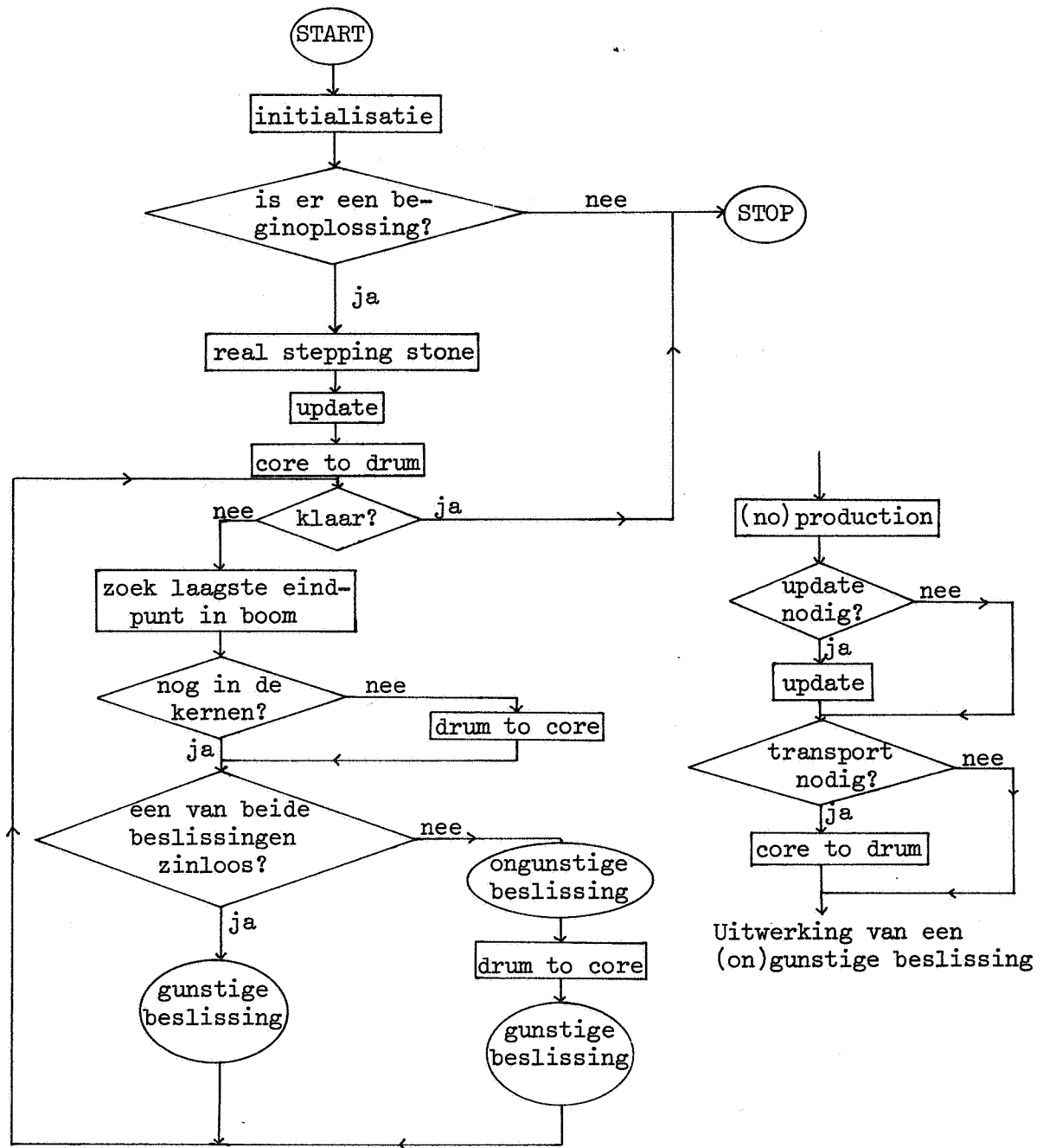
- a) voorraadkosten (real stepping stone)
 - b) schoonmaakkosten (production)
 - c) onderschatting verdere kosten (update)
- (tussen haakjes de procedure die ze berekent)

We nemen vervolgens de gunstige beslissing en voeren daarmee een zelfde proces uit.

Het mag natuurlijk niet voorkomen dat tijdens het branch-and-bound proces de kosten afnemen. We moeten er daarom op letten dat het toevoegen van een produktie de benodigde schoonmaaktijd en -kosten niet doet afnemen. Daartoe dienen deze te voldoen aan de driehoeksongelijkheid:

$$c(P_1, P_3) \leq c(P_1, P_2) + c(P_2, P_3) \quad ,$$

waarbij $c(P_i, P_j)$ de schoonmaaktijd of -kosten zijn bij omschakeling van



Flow chart van het programma

produkt i naar produkt j . Tijdens het inlezen van de schoonmaakgegevens wordt er op getest of ze inderdaad aan deze ongelijkheid voldoen.

Zowel in de procedure 'production' als 'no production' wordt voordat 'real stepping stone' de optimale produktie-planning bepaalt, gekeken of er wel een oplossing met eindige kosten is. Hiertoe wordt voor de c_{ij} 's die op ∞ staan een 1 en voor alle andere een 0 ingevuld. Alleen als 'real stepping stone' nu een oplossing met kosten 0 vindt, is er een eindige oplossing. De oorspronkelijke c_{ij} 's worden in dat geval weer teruggeschreven, en 'real stepping stone' wordt nogmaals aange-roepen, nu met de zojuist gevonden nuloplossing als beginoplossing.

Het is in 'production' niet altijd nodig opnieuw 'real stepping stone' aan te roepen. Dit is niet nodig als de schoonmaaktijd 0 is, of als de schoonmaaktijd van de "slack" (met m^e produkt, dat extra is toegevoegd) af kan. In dat geval is 'update' (behoudens twee uitzonderingen) ook niet nodig. We zullen dan, als we de tak nog eens gaan uitbreiden, als beslissing de volgende uit het - door 'update' vastgestelde - rijtje van gunstige beslissingen nemen.

De ene uitzondering is dat het rijtje van gunstige beslissingen op is. De andere is dat we precies die beslissing hebben genomen waarop de onderschatting van de verdere kosten gebaseerd is, zodat er weer een nieuwe onderschatting moet komen.

3. Update.

We hebben in ons branch-and-bound-proces twee tijdrovende bezigheden: enerzijds is dat het vaststellen wat de volgende beslissing wordt (niet te verwarren met: welke tak we gaan uitbreiden, wat gewoon een kwestie is van in 'drcost' kijken), anderzijds is dat het nemen van de beslissing zelf. De laatstgenoemde bezigheid bestaat uit twee delen: wél produceren, niet produceren, waarvoor we de in hoofdstuk 2 besproken procedures 'production' en 'no production' hebben.

Des te tijdrovender het nemen van een beslissing is, des te nauwgezetter moeten we onderzoeken welke beslissing we gaan nemen. Dit onderzoeken gebeurt in 'update'.

Van de door 'real stepping stone' gevonden optimale produktieplanning wordt voor iedere produktie daarvan, waarvoor de schoonmaakkosten nog niet verrekend zijn, vastgesteld hoeveel het tenminste gaat kosten, als we de produktie niet, en als we de produktie wel maken.

De optimale produktieplanning bestaat uit $m + n - 1$ produkties, waarvan sommige mogelijk 0 zijn. Als we deze in de c_{ij} -matrix verbinden door alle mogelijke horizontale en vertikale lijnen hiertussen te trekken, vinden we een verbonden graph zonder rondgangen.

Tevens levert 'real stepping stone' ons voor ieder element dat niet in de oplossing is opgenomen hoeveel het kost om deze (ter waarde 1) in de oplossing op te nemen. Dit zijn de z.g. "duale waarden W_{ij} ". We zoeken de laagste W_{ij} . Na opname van dit element in de oplossing ontstaat er een rondgang. De procedure 'next' bepaalt recursief deze rondgang. Als we het toegevoegde punt het 0^e hoekpunt noemen, geeft W_{ij} aan hoeveel het ons kost om de oneven hoekpunten van de rondgang voor één minder in de planning op te nemen.

We zoeken nu de op een na laagste duale waarde, en voeren daarmee hetzelfde proces uit.

We gaan zo net zo lang door tot we van iedere non-slack-produktie > 0 uit de planning bepaald hebben hoeveel het kost om er eentje minder van te maken. We noemen dit de off_i .

De laagste off_i van de produkties uit week j noemen we $offmin_j$ ($1 \leq j \leq n$).

Vervolgens bepalen we voor iedere non-slack-produktie >0 , waarvan de schoonmaakkosten nog niet ingecalculleerd zijn, hoeveel het tenminste gaat kosten om de produktie niet (no_i), en hoeveel om hem wel te maken (yes_i), op de volgende wijze:

$no_k = off_k * x_{ij}$, waarin x_{ij} de grootte van de betreffende produktie is.

$yes_k = \text{if } b_j \geq t \wedge a_m \geq t \text{ then}$

$(\text{if } t \leq slack_j \text{ then } c \text{ else } c + (t - slack_j) * offmin_j$

$\text{else } \infty$.

$1 \leq j \leq n$

$1 \leq i \leq m - 1$

c = toename van de schoonmaakkosten bij opnemen van produktie x_{ij}

t = toename van de schoonmaaktijd bij opnemen van produktie x_{ij}

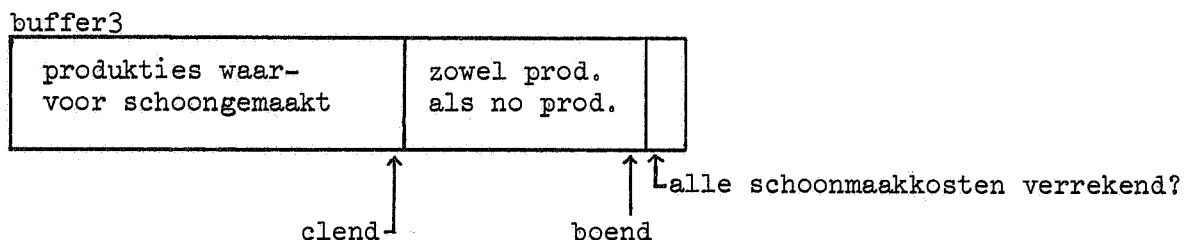
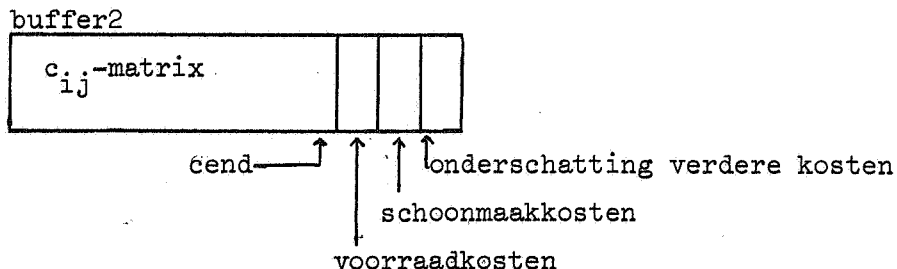
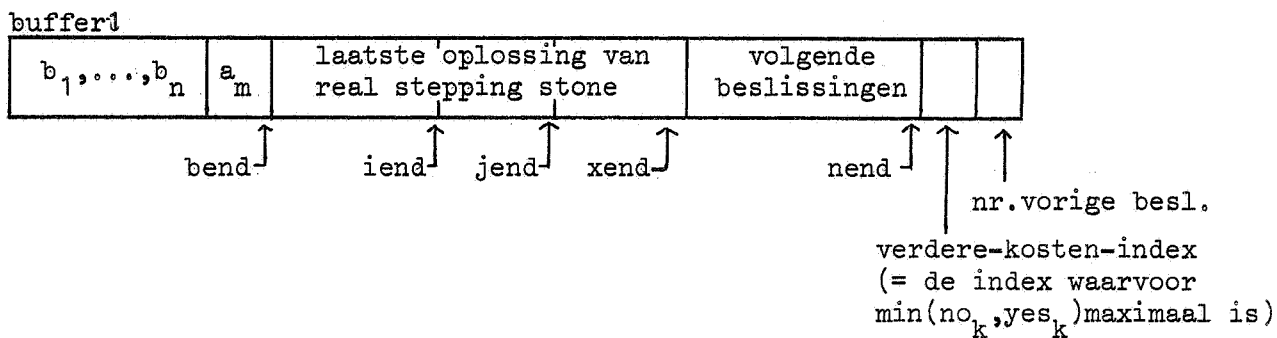
Vervolgens bepalen we $\max_{1 \leq k \leq m+n-1} (\min(no_k, yes_k))$; dit is een onderschat-

ting van de kosten die we in deze tak nog zullen maken. Daarna wordt de rij van te nemen beslissingen opgesteld. Dit gebeurt naar aflopende waarde van $|no_k - yes_k|$. Als een van beide (no of yes) ∞ is, wordt aangetekend dat alleen de andere beslissing behoeft te worden genomen. Als beide ∞ zijn wordt de tak afgekap.

4. Trommel en buffers.

Alle gegevens die belangrijk zijn ter herstelling van de status-quo staan in drie buffers: buffer1 (integer), buffer2 (real) en buffer3 (boolean).

Deze buffers bevatten de volgende gegevens:



bend, iend e.d. zijn vaste pointers in de buffers.

Deze buffers worden (afgezien van de initialisatie) ingevuld/veranderd door de volgende procedures:

```

1,...,bend - 1      : production
bend                : core to drum
bend + 1,...,xend   : real stepping stone
xend + 1,...,nend    : update
nend + 1            : update
nend + 2            : (no) production

```

```

1,...,cend      : no production
cend + 1        : real stepping stone
cend + 2        : production
cend + 3        : update

1,...,clend     : production
clend + 1,...,boend : update
boend + 1       : check branch (zie verderop)

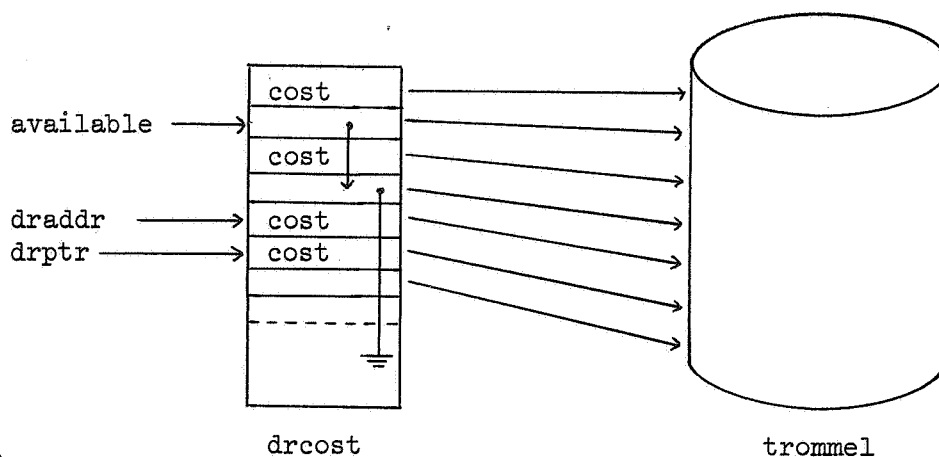
```

Bij een swap-out bepaalt de procedure 'core to drum' welke trommelruimte hiervoor gebruikt moet worden. Dit gebeurt op de volgende wijze.

Er is een stapel ("availstack") waar de vrijgekomen trommelruimte "opgelegd" wordt. Aanvankelijk is deze stapel leeg. Als er trommelruimte nodig is wordt deze van de top van de availstack gepakt (LIFO, Last In First Out); alleen als deze leeg is, wordt er nieuwe trommelruimte gebruikt, onder gelijktijdige ophoging van 'drptr', die aangeeft tot hoever de trommel in gebruik is.

'available' wijst naar de top van de stapel (zie tekening). Het 'aarde-teken' $\perp$ duidt de bodem van de stapel aan.

In 'drcost' zijn de getallen ≥ 0 kosten, en de getallen < 0 pointers van de availstack. 'draddr' wijst naar de tak met de laagste kosten, bij het inswappen wijst 'draddr' dus aan welke trommelbuffers moeten worden ingeswapt.



Voor iedere swap naar de trommel, controleert 'check branch' of dit misschien een tak betreft waarvoor alle schoonmaakkosten verrekend zijn. Als dit inderdaad het geval is, wordt dit aangetekend. Als we een tak willen uitbreiden die al "schoon" is, zijn we klaar. We zijn ook klaar (maar dan zonder dat we een oplossing hebben gevonden) als de kosten van alle eindpunten van takken ∞ zijn.

Er is bij het programmeren op gelet zo mogelijk door te rekenen tijdens de trommeltransporten.

5. Enkele opmerkingen.

Er resten nu nog twee vragen die we onszelf moeten stellen.

1. Kan het voorkomen dat een al gedane beslissing nogmaals wordt genomen?

Nee.

Een beslissing bestaat uit twee gedeelten: niet produceren (de "no-tak") en wel produceren (de "yes-tak"). Beide takken zijn de wortel van een (eventueel verder lege) deelboom.

Stel dat we in de deelboom van de no-tak zitten; dan is het betreffende element van de c_{ij} -matrix op ∞ gezet. De corresponderende produktie kan dan hoogstens met de grootte 0 in een produktie-planning opgenomen worden, maar dat betekent dat die produktie niet in het rijtje van te nemen beslissingen wordt opgenomen. We hebben immers in hoofdstuk 3 gezien dat bij het samenstellen van dit rijtje alleen non-slack-produkties > 0 , waarvan de schoonmaakkosten nog niet ingecalculeerd zijn, in aanmerking komen.

Als we anderzijds in de yes-tak van de beslissing zitten, dan zijn de schoonmaakkosten voor die produktie wel opgenomen, zodat ook dan deze produktie niet in het genoemde rijtje kan worden opgenomen.

2. Stel dat er voor een bepaalde produktie schoonmaaktijd en -kosten worden opgenomen, maar dat deze produktie er in de loop van het branch-and-bound-proces weer 'uitdraait', of met de grootte 0 in de uiteindelijke oplossing voorkomt. Kan dat?

M.a.w.: Bestaat de mogelijkheid dat er in de eindoplossing ten onrechte schoongemaakt wordt?

Nee.

Als er wordt schoongemaakt bevindt de eindoplossing zich blijkbaar in de deelboom van de yes-tak van die produktie waarvoor wordt schoongemaakt. In de deelboom van de no-tak bevindt zich dan een oplossing die verder identiek is aan de (vermeende) eindoplossing, maar goedkoper is omdat er voor die produktie geen schoonmaaktijd en -kosten opgenomen zijn. De eindoplossing was dus geen eindoplossing.

6. Programmatekst.

070572-201 A 7106Y.3 MARTINREM

A 7106Y.3, MARTIN REM

```

1  BEGIN COMMENT A SOLUTION TO A MACHINE LOADING PROBLEM, M. REM 270771.
2  INPUT:
3  NUMBER OF PRODUCTS (>1),
4  NUMBER OF WEEKS INVOLVED (>1),
5  AVAILABLE SPACE ON THE DRUM (USUALLY 81920),
6  SMALL INFINITE (E.G. 10 000 OR >600) (REAL),
7  AVAILABLE PRODUCTION TIME EACH WEEK (EXCL. STANDARD END-OF-WEEK-CLEANING-TIME) (INTEGER),
8  DEMAND OF EACH PRODUCT (EXPRESSED IN PRODUCTION TIME) IN THE FIRST WEEK, SECOND WEEK, ETC. (INTEGER),
9  OVERTIME DELIVERY FINE (IF OVERTIME DELIVERY IS NOT ALLOWED, THEN #1) (REAL),
10 STORAGE COST OF EACH PRODUCT PER WEEK (REAL),
11 CLEANING DATA: UPPER-TRIANGULAR MATRIX OF THE CHANGE-OVER COSTS (REAL);
12 UPPER-TRIANGULAR MATRIX OF THE CHANGE-OVER TIMES (INTEGER),
13 NOTE: THE CLEANING DATA MUST SATISFY THE TRIANGLE INEQUALITY;
14
15 INTEGER MM,M,N,UB,CEND,BEND,IEND,VEND,XEND,CLEND,BOEND,DRMAX,BUFLEN;
16 MM:= READ; N:= MM * N + 1; UB:= M * N - 1; CEND:= M * N; BEND:= N + 1; IEND:= BEND + UB;
17 JEND:= IEND + UB; XEND:= JEND + UB; VEND:= XEND + UB; CLEND:= MM * N;
18 BOEND:= CLEND + UB; BUFLEN:= VEND + 2 * CEND + BOEND + 1; DRMAX:= READ + BUFLEN - 1;
19
20 BEGIN INTEGER I,J,K,NEXTPTR,NUMBER,DRPTR,DRADDR,DECISION,AVAILABLE,NR1,NR2,INTINF;
21 REAL COST,SMINF,INF,FINE,TO,T1;
22 BOOLEAN UPDATE NECESSARY,TRANSPORT NECESSARY,READY,AUX;
23 INTEGER ARRAY BUFFER1(I:END + 2),CT(I:MM * (MM - 1) / 2),A(I:MM);
24 REAL ARRAY BUFFER2(I:CEND + 3),CG(I:MM * (MM - 1) / 2),U(I:MM),V(I:MM),DRCOST(I:DRMAX),SAVE(I:CEND);
25 BOOLEAN ARRAY BUFFER3(I:BOEND + 1);
26
27 PROCEDURE CHECK BRANCH;
28 BEGIN INTEGER I,END;
29 BOOLEAN READY;
30 READY:= TRUE; END:= LUB;
31 FOR I:= 1 STEP 1 UNTIL END DO LE - CLEAN(I) THEN
32 BEGIN READY:= FALSE; I:= END-ENQ;
33 AUX:= BUFFER3(BOEND + 1):= READY
34 END CHECK BRANCH;
35
36 INTEGER PROCEDURE LUB;
37 BEGIN INTEGER I;
38 LUB:= UB;
39 FOR I:= UB STEP -1 UNTIL 1 DO LE BUFFER1(BEND + 1) + M THEN
40 BEGIN LUB;
41
42 INTEGER PROCEDURE CLTIME(PROD1,PROD2); VALUE PROD1,PROD2; INTEGER PROD1,PROD2;
43 CLTIME:= CT((PROD1 - 1) * (MM - PROD1 / 2) + PROD2 - PROD1);
44
45 REAL PROCEDURE CLCOST(PROD1,PROD2); VALUE PROD1,PROD2; INTEGER PROD1,PROD2;
46 CLCOST:= CC((PROD1 - 1) * (MM - PROD1 / 2) + PROD2 - PROD1);
47
48 BOOLEAN PROCEDURE CLEAN(PROD); VALUE PROD; INTEGER PROD;
49 CLEAN:= BUFFER3((BUFFER1(IEND + PROD) - 1) * MM + BUFFER1(BEND + PROD)) =
50 (BUFFER1(BEND + PROD) - 1) + MM * MM; BUFFER1(JEND + PROD) = 0;
51
52 INTEGER PROCEDURE DUMMY(WEEK,OUT); VALUE WEEK; INTEGER WEEK,OUT;
53 BEGIN INTEGER I,END;
54 DUMMY:= 0; OUT:= INTINF; END:= LUB + 1;
55 FOR I:= UB STEP -1 UNTIL END DO LE BUFFER1(IEND + 1) = WEEK THEN
56 BEGIN DUMMY:= BUFFER1(JEND + 1); OUT:= I;

```

078572-201 A 7106Y.3 MARTINREM

```

57  I:= END
58  END ELSE
59  IF BUFFER1(END + 1) < WEEK THEN I:= END
60  END DUMMY;
61
62  REAL PROCEDURE REAL STEPPING STONE(ST); VALUE ST; INTEGER ST;
63  BEGIN COMMENT THIS IS THE PROCEDURE BDO 301069, ADJUSTED TO THIS PROBLEM;
64  INTEGER E,F,G,H,I,J,K,L,M,N; REAL P,Q,R,S,EPS; BOOLEAN W;
65  INTEGER ARRAY CHAIN(1:M + N), NEXTIX,NEXTUX(1:UB), FIRSTX(1:N);
66  REAL ARRAY C(1:UB);
67
68  PROCEDURE HOR(G,R1); INTEGER G; REAL R1;
69  BEGIN INTEGER H; REAL KU;
70  H:= G;
71  FOR H:= NEXTUX(H) WHILE H # G DO
72  BEGIN KU:= BUFFER1(END + H); IF KU < 0 THEN V(-KU):= C(H) - R1 ELSE
73  BEGIN KU:= V(KU):= C(H) - R1; VERT(H,KU) END
74  END
75  END HOR;
76
77  PROCEDURE VERT(H,KU); INTEGER H; REAL KU;
78  BEGIN INTEGER G; REAL R1;
79  G:= H;
80  FOR G:= NEXTIX(G) WHILE G # H DO
81  BEGIN R1:= BUFFER1(END + G); IF R1 < 0 THEN U(-R1):= C(G) - KU ELSE
82  BEGIN R1:= U(R1):= C(G) - KU; HOR(G,R1) END
83  END
84  END VERT;
85
86  REAL PROCEDURE NEW EPS;
87  BEGIN INTEGER G; REAL A,B;
88  A:= 0;
89  FOR G:= 1 STEP 1 UNTIL US DO
90  BEGIN B:= ABS(C(G)) - U(ABS(BUFFER1(END + G))) - V(ABS(BUFFER1(END + G))));
91  IF A < B THEN A:= B
92  END;
93  NEW EPS:= 2 * A
94
95  END NEW EPS;
96
97  PROCEDURE SORT2VEC(VEC1,VEC2,LOW,UPP); VALUE LOW,UPP;
98  INTEGER LOW,UPP; INTEGER ARRAY VEC1,VEC2;
99  BEGIN INTEGER P,Q,X,Y,Z,V,W;
100  A1:
101  IF UPP-LOW > 1 THEN
102  BEGIN P:=(LOW+UPP)/2; Y:=VEC1(P); V:=VEC2(P);
103  VEC1(P):=VEC2(LOW); VEC2(P):=VEC2(LOW); Q:=UPP;
104  FOR P:=LOW+1 STEP 1 UNTIL Q DO
105  BEGIN X:=VEC1(P);
106  IF X > Y THEN
107  BEGIN FOR Q:= Q STEP -1 UNTIL P DO
108  BEGIN Z:=VEC1(Q);
109  IF Z < Y THEN
110  BEGIN VEC1(P):=Z; VEC1(Q):=X;
111  W:=VEC2(P); VEC2(P):=VEC2(Q);
112  VEC2(Q):=W; Q:=Q-1; GOIQ A3
113  END Z < Y
114  END FOR Q;
115  W:=P-1; GOIQ A2
116  END X > Y;
117  A3:
118  END FOR P;
119
120  END FOR P;

```

```

117      A2:      VEC1[LOW]:=VEC1[Q]; VEC1[Q]:=Y; VEC2[LOW]:=VEC2[Q]; VEC2[Q]:=V;
118      LE Q-LOW > UPP-0 THEN
119      BEGIN SORT2VEC(VEC1,VEC2,Q+1,UPP); UPP:=Q-1 END ELSE
120      BEGIN SORT2VEC(VEC1,VEC2,LOW,Q-1); LOW:=Q+1 END;
121      GOIQ A1
122      END UPP-LOW > 1
123      ELSE LE UPP-LOW=1 THEN
124      BEGIN X:=VEC1[LOW]; Z:=VEC1[UPP];
125      LE X > Z THEN
126      BEGIN VEC1[LOW]:=Z; VEC1[UPP]:=X; V:=VEC2[LOW];
127      VEC2[LOW]:=VEC2[UPP]; VEC2[UPP]:=V
128      END
129      END UPP-LOW=1
130      END SORT2VEC;
131
132      PROCEDURE SCANINFEAS;
133      BEGIN
134      INTEGER G;
135      FOR G:= 1 STEP 1 UNTIL ST DO LE C[G] = 1 THEN
136      BEGIN
137      LE C[ST] = 1 THEN
138      BEGIN ST:= ST - 1; LE ST < G THEN GOIQ A2; GOIQ A1 END;
139      BUFFER1[JEND + G]:= BUFFER1[JEND + ST]; BUFFER1[BEND + G]:= BUFFER1[BEND + ST];
140      BUFFER1[IEND + G]:= BUFFER1[IEND + ST]; ST:= ST - 1;
141      A1:
142      END
143      END SCANINFEAS;
144
145      NR1:= NR1 + 1;
146      FOR I:= 1 STEP 1 UNTIL M DO CHAIN[N + I]:= A[I];
147      FOR J:= 1 STEP 1 UNTIL N DO CHAIN[J]:= BUFFER1[J];
148      LE ST = 0 THEN GOIQ B1;
149      W:= FALSE; FOR G:= 1 STEP 1 UNTIL ST DO C[G]:= 0;
150
151      FOR I:= 1 STEP 1 UNTIL M DO BEGIN NEXTIX[I]:= 0; U[I]:= 0 END;
152      FOR J:= 1 STEP 1 UNTIL N DO BEGIN NEXTJX[J]:= 0; V[J]:= 0 END;
153      BEGIN I:= BUFFER1[BEND + G]; U[I]:= U[I] + 1; J:= BUFFER1[IEND + G]; V[J]:= V[J] + 1 END;
154
155      E:= 0;
156      W:= FALSE;
157      FOR G:= 1 STEP 1 UNTIL ST DO LE C[G] = 0 THEN
158      BEGIN
159      I:= BUFFER1[BEND + G]; J:= BUFFER1[IEND + G]; IO:= U[I]; JO:= V[J];
160      LE IO > 1 & JO > 1 THEN GOIQ A22; W:= TRUE;
161      LE IO > 1 & JO > 1 THEN GOIQ A21; C[G]:= 1;
162      LE IO > 1 THEN U[I]:= IO - 1; LE JO > 1 THEN V[J]:= JO - 1; GOIQ A22;
163      C[G]:= 2; H:= CHAIN[N + I]; K:= CHAIN[J];
164      LE H < K & H < IO = 1 THEN
165      BEGIN BUFFER1[JEND + G]:= H; CHAIN[N + I]:= 0; CHAIN[J]:= K - H;
166      V[J]:= JO - 1; NEXTIX[I]:= 1
167      END ELSE
168      BEGIN BUFFER1[JEND + G]:= K; CHAIN[J]:= 0; CHAIN[N + I]:= H - K;
169      U[I]:= IO - 1; NEXTJX[J]:= 1
170      END;
171      E:= E + 1; LE E = UB THEN BEGIN SCANINFEAS; GOIQ B6 END;
172
173      A22:
174      END;
175      LE W THEN GOIQ A1;
176
177      A2:      K:= INTINF; GO:= 0;
178      FOR G:= 1 STEP 1 UNTIL ST DO LE C[G] = 0 THEN

```

```

177 BEGIN H:= BUFFER1[END + G]; LE H < K THEN BEGIN K:= H; G0:= G END END;
178 LE G0 = 0 THEN BEGIN SCANINFEAS; GOIQ B1 END;
179 C[G0]:= 1; I:= BUFFER1[END + G0]; J:= BUFFER1[END + G0]; U[I]:= U[I] - 1; V[J]:= V[J] - 1;
180 LE U[I] = 1 THEN GOIQ A1; LE V[J] = 1 THEN GOIQ A1; GOIQ A2;
181
182 B1: LE ST ≥ 1 THEN G:= ST + 1 ELSE
183 BEGIN FOR I:= 1 STEP 1 UNTIL M DO NEXTIX(I):= 0;
184 FOR J:= 1 STEP 1 UNTIL N DO NEXTJX(J):= 0; G:= 1
185 END;
186 I:= 0;
187
188 B2: I:= 1 + 1; LE NEXTIX(I) = 1 THEN GOIQ B2; F:= CHAIN(N + 1);
189 P:= G:= INF; J0:= J1:= 0;
190 FOR J:= 1 STEP 1 UNTIL N DO LE NEXTJX(J) + 1 THEN
191 R:= BUFFER2[(I - 1) * N + J]; LE ST = -1 THEN R:= R - V[J]; LE Q ≤ R THEN GOIQ B11;
192 LE P ≤ R THEN BEGIN J1:= J; Q:= R; GOIQ B11 END;
193 J1:= J0; J0:= J; Q:= P; P:= R;
194 B11:
195 END;
196
197 B4: C[G]:= LE ST = -1 THEN (P + V[J0]) ELSE P; H:= CHAIN(J0); LE P < H THEN GOIQ B5;
198 BUFFER1[END + G]:= 1; BUFFER1[END + G]:= J0; BUFFER1[END + G]:= H;
199 F:= F - H; CHAIN(J0):= 0; NEXTJX(J0):= 1;
200 LE G = UB THEN GOIQ LE ST ≥ 1 THEN B6 ELSE B7; G:= G + 1;
201 LE J0 = J1 THEN GOIQ B3; J0:= J1; P:= Q; GOIQ B4;
202 BUFFER1[END + G]:= 1; BUFFER1[END + G]:= J0; BUFFER1[END + G]:= F; CHAIN(J0):= H - F;
203 G:= G + 1; LE G ≤ UB THEN GOIQ B2; LE ST < 1 THEN GOIQ B7;
204
205 B6: FOR G:= 1 STEP 1 UNTIL ST DO BEGIN NEXTJX(G):= BUFFER1[END + G]; NEXTIX(G):= G END;
206 SORT2VEC(NEXTJX, NEXTIX, 1, ST);
207 FOR H:= 1 STEP 1 UNTIL ST DO
208 BEGIN G:= NEXTIX(H); I:= BUFFER1[END + G]; J:= BUFFER1[END + G]; C[G]:= BUFFER2[(I - 1) * N + J] END;
209
210 B7: FOR I:= 1 STEP 1 UNTIL M DO CHAIN(I):= 0; FOR J:= 1 STEP 1 UNTIL N DO FIRSTX(J):= 0;
211 FOR G:= 1 STEP 1 UNTIL UB DO
212 BEGIN I:= BUFFER1[END + G]; E:= CHAIN(I);
213 LE E = 0 THEN CHAIN(I):= NEXTJX(G):= G ELSE
214 BEGIN NEXTJX(G):= NEXTJX(E); CHAIN(I):= NEXTJX(E):= G END;
215 J:= BUFFER1[END + G]; E:= FIRSTX(J);
216 LE E = 0 THEN FIRSTX(J):= NEXTIX(G):= G ELSE
217 BEGIN NEXTIX(G):= NEXTIX(E); FIRSTX(J):= NEXTIX(E):= G END
218 END;
219
220 FOR G:= 1 STEP 1 UNTIL UB DO
221 BEGIN LE NEXTIX(G) = G THEN BUFFER1[END + G]:= -BUFFER1[END + G] ELSE
222 LE NEXTJX(G) = G THEN BUFFER1[END + G]:= -BUFFER1[END + G]
223 END;
224
225 I0:= M; CHAIN(I):= 0;
226 I:= BUFFER1[END + 1]; J:= BUFFER1[END + 1]; U[ABS(I)]:= 0; R:= C[I]; V[ABS(J)]:= P;
227 LE I > 0 THEN HOR(1,0); LE J > 0 THEN VERT(1,P); F:= I:= 10;
228 LE I = M THEN BEGIN I:= 0; EPS:= NEW EPS END; I:= I + 1; J0:= 0; Q:= U[I] - EPS;
229 FOR J:= 1 STEP 1 UNTIL N DO
230 BEGIN R:= BUFFER2[(I - 1) * N + J]; P:= R - V[J]; LE P < Q THEN BEGIN J0:= J; Q:= P; S:= R END END;
231 LE J0 = 0 THEN GOIQ LE I = 10 THEN F1 ELSE C2;
232
233 I0:= 1; K:= 2; G:= FIRSTX(J0);
234 I:= BUFFER1[END + G]; LE ABS(I) = 10 THEN BEGIN I:= 10; I0:= P; EPS:= NEW EPS; GOIQ C2 END;
235 LE I < 0 THEN BEGIN G:= NEXTIX(G); GOIQ D1 END; CHAIN(I2):= G;
236 G:= NEXTJX(G); LE BUFFER1[END + G] < 0 THEN GOIQ D2;
237 LE G = CHAIN(K) THEN BEGIN K:= K - 1; GOIQ D3 END; K:= K + 1; CHAIN(K):= G;

```

```

237 D3: G:= NEXTIX[G]; I:= BUFFER1[BEND + G];
238 IE ABS(I) = 10 THEN
239 BEGIN IF K = 1 THEN BEGIN I:= 10; I0:= F; EPS:= NEW EPS; GOIQ C2 END;
240 K:= K + 1; CHAIN[K]:= G; GOIQ E1
241 END;
242 IE I < 0 THEN GOIQ D3;
243 IE G = CHAIN[K] THEN K:= K + 1 ELSE BEGIN K:= K + 1; CHAIN[K]:= G END; GOIQ D2;
244
245 E1: Q:= -INF; F:= INTINF;
246 FOR H:= 2 STEP 2 UNTIL K DO
247 BEGIN G:= CHAIN[H]; E:= BUFFER1[JEND + G]; IE E > F THEN GOIQ E11;
248 IE E < F THEN BEGIN F:= E; G0:= H; GOIQ E11 END;
249 P:= C[G]; IE Q < P THEN BEGIN Q:= P; G0:= H END;
250
251 E11:
252 END;
253
254 FOR H:= 2 STEP 2 UNTIL K DO BEGIN G:= CHAIN[H]; BUFFER1[JEND + G]:= BUFFER1[JEND + G] - F END;
255 FOR H:= 3 STEP 2 UNTIL K DO BEGIN G:= CHAIN[H]; BUFFER1[JEND + G]:= BUFFER1[JEND + G] + F END;
256 E:= G0; G0:= CHAIN[G0]; I1:= ABS(BUFFER1[BEND + G0]); J1:= ABS(BUFFER1[JEND + G0]);
257 BUFFER1[JEND + G0]:= F; C[G0]:= S;
258 IE I0 # 11 THEN
259 BEGIN H:= NEXTJX[G0]; F:= G:= CHAIN[E + 1];
260 FOR G:= NEXTJX[G] WHILE G # G0 DO F:= G;
261 NEXTJX[F]:= H; IE F = H THEN BUFFER1[BEND + F]:= -11; BUFFER1[BEND + G0]:= 10;
262 F:= CHAIN[K]; G:= NEXTJX[G0]:= NEXTJX[F]; NEXTJX[F]:= G0;
263 IE F = G THEN BUFFER1[BEND + F]:= 10
264 END;
265
266 IE J0 # J1 THEN
267 BEGIN H:= NEXTIX[G0]; F:= G:= FIRSTX[J1]:= CHAIN[E - 1];
268 FOR G:= NEXTIX[G] WHILE G # G0 DO F:= G;
269 NEXTIX[F]:= H; IE F = H THEN BUFFER1[JEND + F]:= -J1; BUFFER1[JEND + G0]:= J0;
270 F:= CHAIN[G]; G:= NEXTIX[G0]:= NEXTIX[F]; NEXTIX[F]:= G0;
271 IE F = G THEN BUFFER1[JEND + F]:= J0
272 END;
273 GOIQ C1;
274
275 F1: REAL STEPPING STONE:= VECVEC(I,UB,JEND,C,BUFFER1);
276 K:= 0;
277
278 F2: FOR I:= 1 STEP 1 UNTIL M DO
279 BEGIN FOR H:= 1 STEP 1 UNTIL UB DO IE ABS(BUFFER1[BEND + H]) = 1 THEN GOIQ F21;
280 IE BUFFER1[BEND + H] < 0 THEN BEGIN K:= K + 1; CHAIN[K]:= H; GOIQ F22 END; F:= K;
281 FOR G:= H; NEXTJX[G] WHILE G # H DO
282 BEGIN K:= K + 1; CHAIN[K]:= G; NEXTIX[K]:= ABS(BUFFER1[JEND + G]) END;
283 SORT2VEC(NEXTIX,CHAIN,F + 1,K);
284
285 F22:
286 END;
287
288 F3: FOR G:= 1 STEP 1 UNTIL UB DO NEXTIX[G]:= 1;
289 FOR K:= 1 STEP 1 UNTIL UB DO IE NEXTIX[K] = 1 THEN
290 BEGIN G:= K; F:= BUFFER1[JEND + G]; I0:= BUFFER1[BEND + G]; J0:= BUFFER1[JEND + G];
291 FOR H:= CHAIN[G] WHILE H # K DO
292 BEGIN BUFFER1[JEND + G]:= BUFFER1[JEND + H]; BUFFER1[BEND + G]:= ABS(BUFFER1[BEND + H]);
293 BUFFER1[JEND + G]:= ABS(BUFFER1[JEND + H]); NEXTIX[G]:= 0; G:= H
294 END;
295 BUFFER1[JEND + G]:= F; BUFFER1[BEND + G]:= I0; BUFFER1[BEND + G]:= J0; NEXTIX[G]:= 0
296 END;
297 FOR I:= 1 STEP 1 UNTIL UB + UB DO IE BUFFER1[BEND + I] < 0 THEN BUFFER1[BEND + I]:= - BUFFER1[BEND + I]
298 END REAL STEPPING STONE;

```

```

297 PROCEDURE NO PRODUCTION;
298 BEGIN
299   BUFFER[ABS(DECISION)] := SMINF; BUFFER[NEND + 2] := -NUMBER;
300   FOR I := 1 STEP 1 UNTIL CEND DO
301     BEGIN SAVE(I) := BUFFER(I);
302     BUFFER(I) := BUFFER(2(I));
303     IF BUFFER(2(I)) = SMINF THEN 1 ELSE 0
304   END;
305   IF REAL STEPPING STONE(UB) = 0 THEN
306     BEGIN FOR I := 1 STEP 1 UNTIL CEND DO BUFFER(2(I)) := SAVE(I);
307     BUFFER(2(CEND + 1)) := REAL STEPPING STONE(UB);
308     UPDATE NECESSARY := TRANSPORT NECESSARY := TRUE
309   END ELSE
310     BEGIN BUFFER(2(CEND + 1)) := SMINF;
311     UPDATE NECESSARY := TRANSPORT NECESSARY := FALSE
312   END
313 END NO PRODUCTION;
314
315 PROCEDURE PRODUCTION;
316 BEGIN
317   INTEGER I, J, WEEK, PROD, TIME, DEC, LOW, UPP;
318   REAL COST;
319   PROD := 1 - (1 - 1) * MM * MM; J := (WEEK - 1) * MM;
320   DEC := ABS(DECISION); I := (DEC - 1) * N + 1; WEEK := DEC - N * (I - 1);
321   BUFFER[NEND + 2] := NUMBER;
322   IF BUFFER(3(J + PROD)) THEN BEGIN COST := 0; TIME := 0 END ELSE
323     BEGIN BUFFER(3(J + PROD)) := TRUE; LOW := 0; UPP := MM + 1;
324     FOR I := PROD - 1 STEP -1 UNTIL 1 DO IF BUFFER(3(J + I)) THEN
325       BEGIN LOW := I; I := 1 END;
326     FOR I := PROD + 1 STEP 1 UNTIL MM DO IF BUFFER(3(J + I)) THEN
327       BEGIN UPP := I; I := MM END;
328     IF LOW = 0 THEN
329       BEGIN LE UPP := MM + 1 THEN BEGIN COST := 0; TIME := 0 END ELSE
330         BEGIN COST := CLCOST(PROD, UPP); TIME := CLTIME(PROD, UPP) END
331       END ELSE
332         LE UPP := MM + 1 THEN BEGIN COST := CLCOST(LOW, PROD); TIME := CLTIME(LOW, PROD) END ELSE
333           BEGIN COST := CLCOST(LOW, PROD) + CLCOST(PROD, UPP) - CLCOST(LOW, UPP);
334           TIME := CLTIME(LOW, PROD) + CLTIME(PROD, UPP) - CLTIME(LOW, UPP)
335         END
336       END;
337       LE TIME = 0 THEN
338         BEGIN BUFFER(1(XEND + NEXTPTR)) := 0; TRANSPORT NECESSARY := TRUE;
339         UPDATE NECESSARY := BUFFER(1(XEND + NEXTPTR + 1)) = INTINF * BUFFER(1(NEND + 1)) = DEC;
340         BUFFER(2(CEND + 2)) := BUFFER(2(CEND + 2)) + COST
341       END ELSE
342         IF BUFFER(1(WEEK) < TIME * A(M) < TIME THEN
343           BEGIN BUFFER(2(CEND + 2)) := SMINF; UPDATE NECESSARY := TRANSPORT NECESSARY := FALSE END ELSE
344             BEGIN BUFFER(1(WEEK)) := BUFFER(1(WEEK)) - TIME;
345             A(M) := A(M) - TIME;
346             BUFFER(2(CEND + 2)) := BUFFER(2(CEND + 2)) + COST;
347             IF DUMMY(WEEK, I) > TIME THEN
348               BEGIN BUFFER(1(JEND + 1)) := BUFFER(1(JEND + 1)) - TIME;
349               BUFFER(1(XEND + NEXTPTR)) := 0; TRANSPORT NECESSARY := TRUE;
350               UPDATE NECESSARY := BUFFER(1(XEND + NEXTPTR + 1)) = INTINF * BUFFER(1(NEND + 1)) = DEC
351             END ELSE
352               BEGIN LE I < INTINF THEN BUFFER(1(JEND + 1)) := 0;
353               FOR I := 1 STEP 1 UNTIL CEND DO
354                 BEGIN SAVE(I) := BUFFER(2(I));
355                 BUFFER(2(I)) := LE BUFFER(2(I)) = SMINF THEN 1 ELSE 0
356               END;
357               LE REAL STEPPING STONE(UB) = 0 THEN

```

```

357 BEGIN FOR I:= 1 STEP 1 UNTIL CEND DO BUFFER2(I):= SAVE(I);
358 BUFFER2(CEND + 1):= REAL STEPPING STONE(UB);
359 UPDATE NECESSARY:= TRANSPORT NECESSARY:= TRUE
360 END ELSE
361 BEGIN BUFFER2(CEND + 1):= SMINF;
362 UPDATE NECESSARY:= TRANSPORT NECESSARY:= FALSE
363 END
364 END
365 END PRODUCTION;
366
367 PROCEDURE UPDATE;
368 BEGIN INTEGER I,J,K,SMALL1,SMALL2,NEXTX,CNTR,PATHPTR,END,TIME,PROD,LOW,UPP;
369 REAL W,SMALLW,MAX,COST;
370 INTEGER ARRAY PATH,DECAD(1:UB1);
371 REAL ARRAY OFF,DEC,YES,NO(1:UB),OFFMIN(1:IN);
372
373 PROCEDURE NEXT(TYPE); VALUE TYPE; INTEGER TYPE;
374 BEGIN COMMENT IF TYPE = 0 THEN FIRST TIME,
375 IF TYPE = 1 THEN HORIZONTAL SEARCH,
376 IF TYPE = 2 THEN VERTICAL SEARCH;
377
378 INIEGES I, LAST, LAST1, LASTJ;
379 PATHPTR:= PATHPTR + 1;
380 LE TYPE = 0 THEN
381 BEGIN FOR I:= 1 STEP 1 UNTIL UB DO LE BUFFER1(BEND + I) = SMALL1 THEN
382 BEGIN PATH(PATHPTR):= I; NEXT(2) END
383 ELSE LE BUFFER1(BEND + I) = SMALLJ THEN
384 BEGIN PATH(PATHPTR):= I; NEXT(1) END
385 END ELSE
386 BEGIN LAST:= PATH(PATHPTR - 1); LAST1:= BUFFER1(BEND + LAST); LASTJ:= BUFFER1(BEND + LAST);
387 FOR I:= 1 STEP 1 UNTIL LAST - 1, LAST + 1 STEP 1 UNTIL UB DO LE TYPE = 1 THEN
388 BEGIN LE BUFFER1(BEND + I) = LAST1 THEN
389 BEGIN PATH(PATHPTR):= I;
390 LE BUFFER1(BEND + I) = SMALLJ THEN GO10 OUT ELSE NEXT(2)
391 END
392 END ELSE LE BUFFER1(BEND + I) = LASTJ THEN
393 BEGIN PATH(PATHPTR):= I;
394 LE BUFFER1(BEND + I) = SMALL1 THEN GO10 OUT ELSE NEXT(1)
395 END
396 END;
397 PATHPTR:= PATHPTR - 1
398 END NEXT;
399
400 BOOLEAN PROCEDURE AGAIN;
401 BEGIN INTEGER I;
402 AGAIN:= FALSE;
403 FOR I:= 1 STEP 1 UNTIL END DO LE OFF(I) = -1 THEN
404 BEGIN AGAIN:= TRUE; I:= END END
405 END AGAIN;
406
407 NR2:= NR2 + 1; END:= LUB;
408 FOR I:= 1 STEP 1 UNTIL END DO OFF(I):= LE BUFFER1(JEND + I) = 0 THEN SMINF ELSE -1;
409
410 COMMENT FILL OFF;
411 FOR SMALLW:= SMINF WHILE AGAIN DO
412 BEGIN K:= 1; NEXTX:= (BUFFER1(BEND + 1) - 1) * N + BUFFER1(BEND + 1);
413 FOR I:= 1 STEP 1 UNTIL W DO
414 FOR J:= 1 STEP 1 UNTIL N DO
415 BEGIN CNTR:= (I - 1) * N + J;
416 LE CNTR # NEXTX + BUFFER2(CNTR) > 0 THEN

```

```

417 BEGIN W:= - U(I) - V(J) + BUFFER2(CNTR);
418 IF W < 0 THEN W:= 0 ELSE IF W < SMALLW THEN
419 BEGIN SMALLW:= W; SMALLI:= I; SMALLJ:= J END
420 END ELSE IF CNTR = NEXTX THEN
421 BEGIN K:= K + 1; NEXTX:= LE K ≤ UB THEN
422 N * (BUFFER1(BEND + K) - 1) + BUFFER1(LEND + K) ELSE INTINF
423 END
424 END;
425 IF SMALLW > SMINF / 2 THEN
426 BEGIN FOR I:= 1 STEP 1 UNTIL END DO LE OFF(I) = -1 THEN OFF(I):= SMINF
427 END ELSE
428 BEGIN BUFFER2((SMALLI - 1) * N + SMALLJ):= - BUFFER2((SMALLI - 1) * N + SMALLJ) - 1;
429 PATHPTR:= 0; NEXT(0);
430 FOR I:= 1 STEP 2 UNTIL PATHPTR DO
431 BEGIN J:= PATH(I); LE J ≤ END ^ OFF(J) = -1 THEN OFF(J):= SMALLW
432 END
433 END
434 END;
435 FOR I:= 1 STEP 1 UNTIL CEND DO LE BUFFER2(I) < 0 THEN BUFFER2(I):= - BUFFER2(I) - 1;
436 FOR I:= 1 STEP 1 UNTIL N DO OFFMIN(I):= SMINF;
437 FOR I:= 1 STEP 1 UNTIL END DO
438 BEGIN J:= BUFFER1(LEND + I);
439 IF OFF(I) < OFFMIN(J) THEN OFFMIN(J):= OFF(I)
440 END;
441 END;
442 COMMENT FILL YES AND NO;
443 FOR CNTR:= 1 STEP 1 UNTIL END DO LE - CLEAN(CNTR) THEN
444 BEGIN J:= BUFFER1(LEND + CNTR);
445 LOW:= 0; UPP:= MM + 1; I:= BUFFER1(BEND + CNTR);
446 PROD:= I - (I - 1) * MM * MM; K:= (J - 1) * MM;
447 FOR I:= PROD - 1 STEP -1 UNTIL 1 DO LE BUFFER3(K + 1) THEN
448 BEGIN LOW:= I; I:= 1 END;
449 FOR I:= PROD + 1 STEP 1 UNTIL MM DO LE BUFFER3(K + 1) THEN
450 BEGIN UPP:= I; I:= 'M' END;
451 LE LOW = 0 THEN
452 BEGIN LE UPP = MM + 1 THEN BEGIN COST:= 0; TIME:= 0 END ELSE
453 BEGIN COST:= CLCOST(PROD,UPP); TIME:= CLTIME(PROD,UPP) END
454 END ELSE
455 LE UPP = MM + 1 THEN
456 BEGIN COST:= CLCOST(LOW,PROD); TIME:= CLTIME(LOW,PROD) END ELSE
457 BEGIN COST:= CLCOST(LOW,PROD) + CLCOST(PROD,UPP) - CLCOST(LOW,UPP);
458 TIME:= CLTIME(LOW,PROD) + CLTIME(PROD,UPP) - CLTIME(LOW,UPP)
459 END;
460 NO(CNTR):= OFF(CNTR) + BUFFER1(JEND + CNTR);
461 YES(CNTR):= LE TIME ≤ BUFFER1(J) * TIME $ A(M) THEN
462 (LE TIME $ DUMMY(J,K) THEN COST ELSE COST + (TIME - DUMMY(J,K)) * OFFMIN(J))
463 ELSE SMINF
464 END ELSE YES(CNTR):= -1;
465 COMMENT UNDER ESTIMATE FURTHER COST;
466 MAX:= 0; J:= -1;
467 FOR I:= 1 STEP 1 UNTIL END DO LE YES(I) ≠ -1 THEN
468 BEGIN W:= LE NO(I) 2 YES(I) THEN YES(I) ELSE NO(I);
469 LE W > SMINF / 2 THEN
470 BEGIN BUFFER2(CEND + 31):= SMINF; GO TO OUTINF END;
471 LE W > MAX THEN BEGIN MAX:= W; J:= 1 END
472 END;
473 LE J = -1 THEN FOR I:= 1 STEP 1 UNTIL END DO LE YES(I) ≠ -1 THEN
474 BEGIN J:= I; I:= END END;
475 BUFFER2(CEND + 3):= MAX;
476

```

```

477 BUFFER1(NEND + 1) := (BUFFER1(BEND + J) - 1) * N + BUFFER1(IEND + J);
478
479 COMMENT FIND NEXT DECISION;
480 FOR I := 1 STEP 1 UNTIL END DO LE YES(I) # -1 THEN
481 BEGIN DEC(I) := ABS(ND(I) - YES(I)); DECAD(I) := (BUFFER1(BEND + I) - 1) * N + BUFFER1(IEND + I);
482 END ELSE DEC(I) := 0;
483 CNTR := 1;
484 FOR I := 1 STEP 1 UNTIL END DO LE DEC(I) # 0 THEN
485 BEGIN MAX := DEC(I); J := I;
486 FOR K := 1 + 1 STEP 1 UNTIL END DO LE DEC(K) > MAX THEN BEGIN MAX := DEC(K); J := K END;
487 LE MAX = 0 THEN I := END ELSE
488 BEGIN K := DECAD(J); DEC(J) := DEC(I); DECAD(J) := DECAD(I);
489 LE NO(J) > YES(J) THEN
490 BEGIN LE NO(J) > SMINF / 2 THEN
491 BEGIN BUFFER3(CLEND + CNTR) := FALSE; BUFFER1(XEND + CNTR) := K END
492 ELSE BEGIN BUFFER3(CLEND + CNTR) := TRUE; BUFFER1(XEND + CNTR) := - K END
493 END ELSE
494 LE YES(J) > SMINF / 2 THEN
495 BEGIN BUFFER3(CLEND + CNTR) := FALSE; BUFFER1(XEND + CNTR) := - K END
496 ELSE BEGIN BUFFER3(CLEND + CNTR) := TRUE; BUFFER1(XEND + CNTR) := K END;
497 CNTR := CNTR + 1
498 END
499 END;
500 IF CNTR = 1 THEN FOR I := 1 STEP 1 UNTIL END DO LE YES(I) # -1 THEN
501 BEGIN BUFFER1(XEND + CNTR) := (BUFFER1(BEND + I) - 1) * N + BUFFER1(IEND + I);
502 CNTR := CNTR + 1; I := END
503 END;
504 BUFFER1(XEND + CNTR) := INTINF;
505 OUTINF;
506 END UPDATE;
507
508 PROCEDURE OUT(TEXT); SIBING TEXT;
509 BEGIN INTEGER I, DEC;
510 REAL CLOCK;
511 CLOCK := TIME; DEC := ABS(DECISION);
512 NLCR := (DEC - 1) / N + 1;
513 ABSFIXT(4, 0, NUMBER); PRINTTEXT(TEXT); ABSFIXT(4, 0, I);
514 ABSFIXT(4, 0, DEC - N * (I - 1)); ABSFIXT(4, 2, CLOCK - T1); PRINTTEXT(4, SEC);
515 ABSFIXT(5, 2, CLOCK - T0); PRINTTEXT(4, SEC);
516 LE COST < SMINF / 2 THEN ABSFIXT(9, 2, COST) ELSE PRINTTEXT(4
517 FIXT(5, 0, J); T1 := TIME;
518 LE AUX THEN PRINTTEXT(4
519 (CLEAN);
520 END OUT;
521
522 PROCEDURE CORE TO DRUM;
523 BEGIN INTEGER DRSPACE;
524 LE AVAILABLE = DRMAX + 1 THEN
525 BEGIN DRPTR := DRPTR + 1;
526 LE DRPTR > DRMAX THEN
527 BEGIN NLCR; PRINTTEXT(4, DRPTR); NLCR; GOTO EXIT
528 END ELSE DRSPACE := DRPTR
529 END ELSE BEGIN DRSPACE := AVAILABLE; AVAILABLE := DRCONST(AVAILABLE) - 1 END;
530 BUFFER1(BEND) := A[M];
531 TO DRUM(BUFFER1, DRSPACE * BUFLN);
532 TO DRUM(BUFFER2, DRSPACE * BUFLN + NEND + 2);
533 TO DRUM(BUFFER3, DRSPACE * BUFLN + NEND + 2 * CEND + 8);
534 DRCONST(DRSPACE) := COST
535 END CORE TO DRUM;
536

```

```

070672-201      A 7106Y.3 MARTINREM
537 PROCEDURE DRUM TO CORE(FREE); VALUE FREE; BOOLEAN FREE;
538 BEGIN FROM DRUM(BUFFER1,DRADDR * BUFLN);
539 FROM DRUM(BUFFER2,DRADDR * BUFLN + 2);
540 FROM DRUM(BUFFER3,DRADDR * BUFLN + 2 * CEND + 8);
541 IF FREE THEN BEGIN DRCDST(DRADDR):= AVAILABLE - 1; AVAILABLE:= DRADDR END
542 END DRUM TO CORE;
543
544 COMMENT INITIALIZE;
545 INF:= 627; INTINF:= 67 108 663; DRPTR:= -1;
546 FOR I:= 1 STEP 1 UNTIL CEND DO BUFFER3(I):= FALSE;
547 NR1:= NR2:= DRADDR:= NUMBER:= 0; BUFFER2(CEND + 2):= 0; AVAILABLE:= DRMAX + 1;
548 TO:= TIME; READY:= FALSE; BUFFER1(NEND + 2):= 0;
549
550 COMMENT READ DATA;
551 SMINF:= READ;
552 I:= J:= 0;
553 FOR K:= 1 STEP 1 UNTIL N DO
554 BEGIN BUFFER1(K):= READ; I:= I + BUFFER1(K) END;
555 FOR K:= 1 STEP 1 UNTIL M - 1 DO
556 BEGIN A(K):= READ; J:= J + A(K) END;
557 IF I > J THEN A(M):= I - J ELSE
558 BEGIN NLCR; PRINTTEXT('DEMAND EXCEEDS AVAILABLE PRODUCTION TIME'); NLCR; GOIQ EXIT END;
559 FINE:= READ; IF FINE = - 1 THEN FINE:= SMINF;
560 FOR I:= 0 STEP 1 UNTIL MM - 1 DO
561 BEGIN COST:= READ;
562 FOR K:= 1 STEP 1 UNTIL N - 1 DO IF A(K * MM + I + 1) = 0 THEN
563 BEGIN FOR J:= 1 STEP 1 UNTIL K DO BUFFER2((M - 1) * K + I * N + J):= SMINF
564 END ELSE
565 FOR J:= 1 STEP 1 UNTIL K DO BUFFER2((M - 1) * K + I * N + J):= COST * (K - J + 1)
566 END;
567 FOR K:= 0 STEP 1 UNTIL N - 1 DO
568 BEGIN BUFFER2(CEND + K):= 0;
569 FOR I:= 0 STEP 1 UNTIL MM - 1 DO
570 BEGIN NEXTPTR:= K * M + I * N + 1;
571 IF A(K * MM + I + 1) # 0 THEN
572 BEGIN BUFFER2(NEXTPTR):= 0;
573 FOR J:= 1 STEP 1 UNTIL N - K - 1 DO BUFFER2(NEXTPTR + J):= FINE
574 END ELSE
575 FOR J:= 0 STEP 1 UNTIL N - K - 1 DO
576 BEGIN BUFFER2(NEXTPTR + J):= SMINF
577 END
578 END;
579 J:= MM * (MM - 1) / 2;
580 FOR I:= 1 STEP 1 UNTIL J DO CC(I):= READ;
581 FOR I:= 1 STEP 1 UNTIL J DO CT(I):= READ;
582 FOR I:= 1 STEP 1 UNTIL MM - 2 DO
583 FOR J:= I + 2 STEP 1 UNTIL MM DO
584 FOR K:= I + 1 STEP 1 UNTIL J - 1 DO
585 IF CCOST(I,J) > CCOST(I,K) + CCOST(K,J) THEN
586 BEGIN NLCR; PRINTTEXT('CLEANING DATA DO NOT SATISFY THE TRIANGLE INEQUALITY');
587 ABSFIXT(4,0,(I - 1) * (MM - 1 / 2) + J - 1);
588 ABSFIXT(4,0,(I - 1) * (MM - 1 / 2) + K - 1);
589 ABSFIXT(4,0,(K - 1) * (MM - K / 2) + J - K); NLCR; GOIQ EXIT
590 END;
591 PRINTTEXT('THE SOLUTION OF THIS MACHINE LOADING PROBLEM (');
592 ABSFIXT(LE MM < 10 THEN 1 ELSE LE MM < 100 THEN 2 ELSE 3,0,MM);
593 PRINTTEXT('PRODUCTS AND'); ABSFIXT(LE N < 10 THEN 1 ELSE 2,0,N); PRINTTEXT('WEEKS ');
594 NLCR; NLCR;
595 FOR I:= 1 STEP 1 UNTIL CEND DO

```

A 7106Y.3 MARTINREM

070072-201

```

597 BEGIN SAVE(1):= BUFFER2(1);
598 BUFFER2(1):= LE BUFFER2(1) = SMINF THEN 1 ELSE 0
599 END;
600 LE REAL STEPPING STONE(0) = 0 THEN
601 BEGIN FOR I:= 1 STEP 1 UNTIL CEND DO BUFFER2(1):= SAVE(1);
602 BUFFER2(CEND + 1):= REAL STEPPING STONE(UB)
603 END ELSE
604 BEGIN PRINTTEXT(4NO SOLUTION); NLCR; GOIQ EXIT END;
605 UPDATE;
606 COST:= BUFFER2(CEND + 1);
607 PRINTTEXT(4 INITIAL COST); ABSFIX(10,2,COST); NLCR; NLCR;
608 PRINTTEXT(4 DECISION|PROD.|WEEK|COMPUTING TIME|TOTAL TIME|L.B. OF COST|PREV. DEC.);
609 NLCR; T1:= TIME;
610 CHECK BRANCH;
611 CORE TO DRUM;
612
613 COMMENT MAIN CYCLE;
614 FOR NUMBER:= NUMBER + 1 WHILE - READY DO
615 BEGIN
616   TRANSPORT NECESSARY:= FALSE;
617   FOR I:= 0 STEP 1 UNTIL DROT DO LE DRCOST(1) > 0 ^ DRCOST(1) < COST THEN
618     BEGIN COST:= DRCOST(1); DRADDR:= 1; TRANSPORT NECESSARY:= TRUE END;
619     LE COST > SMINF / 2 THEN BEGIN NLCR; PRINTTEXT(4NO SOLUTION); NLCR; GOIQ EXIT END;
620     LE TRANSPORT NECESSARY THEN BEGIN DRUM TO CORE(4FALSE); A(MI:= BUFFER1(BEND) END;
621     LE BUFFER3(BEND + 1) THEN READY:= TRUE ELSE
622     BEGIN FOR I:= 1 STEP 1 UNTIL UB DO LE BUFFER1(XEND + 1) > 0 THEN
623       BEGIN DECISION:= BUFFER1(XEND + 1) NEXTPTR:= 1; I:= UB END;
624       LE BUFFER3(CEND + NEXTPTR) THEN
625         BEGIN J:= BUFFER1(NEND + 2);
626         LE DECISION > U THEN PRODUCTION ELSE NO PRODUCTION;
627         LE UPDATE NECESSARY THEN UPDATE;
628         COST:= BUFFER2(CEND + 1) + BUFFER2(CEND + 2) + BUFFER2(CEND + 3);
629         LE TRANSPORT NECESSARY ^ COST < SMINF / 2 THEN
630           BEGIN CHECK BRANCH; CORE TO DRUM END ELSE AUX:= FALSE;
631           LE DECISION > U THEN OUT(4YES) ELSE OUT(4NO);
632           J:= BUFFER1(NEND + 2);
633           LE DECISION > U THEN NO PRODUCTION ELSE PRODUCTION;
634           LE UPDATE NECESSARY THEN UPDATE;
635           COST:= BUFFER2(CEND + 1) + BUFFER2(CEND + 2) + BUFFER2(CEND + 3);
636           LE TRANSPORT NECESSARY ^ COST < SMINF / 2 THEN
637             BEGIN CHECK BRANCH; CORE TO DRUM END ELSE AUX:= FALSE;
638             LE DECISION > U THEN OUT(4NO) ELSE OUT(4YES); NLCR
639           END ELSE
640             BEGIN DRCOST(DRADDR):= - AVAILABLE - 1; AVAILABLE:= DRADDR;
641             J:= BUFFER1(NEND + 2);
642             LE DECISION > U THEN PRODUCTION ELSE NO PRODUCTION;
643             LE UPDATE NECESSARY THEN UPDATE;
644             COST:= BUFFER2(CEND + 1) + BUFFER2(CEND + 2) + BUFFER2(CEND + 3);
645             LE TRANSPORT NECESSARY ^ COST < SMINF / 2 THEN
646               BEGIN CHECK BRANCH; CORE TO DRUM END ELSE AUX:= FALSE;
647               LE DECISION > U THEN OUT(4YES) ELSE OUT(4NO); NLCR
648             END
649           END
650         END;
651
652 NEW PAGE; PRINTTEXT(4FINAL SOLUTION, WITH A COST OF: ?);
653 ABSFIX(10,2,BUFFER2(CEND + 1) + BUFFER2(CEND + 2)); NLCR; NLCR;
654 FOR I:= 1 STEP 1 UNTIL N DO
655   BEGIN PRINTTEXT(4WEEK); ABSFIX(12,0,I); PRINTTEXT(4 (IDLE TIME);
656   ABSFIX(4,0,DUMMAY(1,J)); PMSYM(99);

```

```

070672-201      A 7106Y.3 MARTINREM
657      NLCR; NLCR; PRINTTEXT($ AMOUNT PROD WEEK$); NLCR;
658      NEXTPTR:= LUB;
659      EQB J:= 1 STEP 1 UNTIL NEXTPTR DO LE BUFFER1(JEND + J) = 1 ^ BUFFER1(JEND + J) > 0 THEN
660      BEGIN K:= BUFFER1(BEND + J);
661      ABSFIXT(10,0,BUFFER1(JEND + J)); ABSFIXT(5,0,K - (K - 1) ^ MM * MM);
662      ABSFIXT(3,0,(K - 1) ^ MM + 1); NLCR
663      END;
664      NLCR
665      END;
666
667      EXIT: PRINTTEXT($NUMBER OF CALLS OF 'REAL STEPPING STONE';); ABSFIXT(6,0,NR1); NLCR;
668      PRINTTEXT($NUMBER OF CALLS OF 'UPDATE';); ABSFIXT(6,0,NR2); NLCR;
669      PRINTTEXT($USED SPACE ON THE DRUM;); ABSFIXT(6,0,(DRPTR + 1) * BUFLen); NLCR;
670      PRINTTEXT($TIME SINCE INITIALIZATION;); ABSFIXT(6,2,TIME - T0); PRINTTEXT($SEC$);
671      END INNER BLOCK
672      END
673

```

7. Een voorbeeld.

Als voorbeeld beschouwen we een probleem met negen produkten en drie weken, waarbij nalevering niet toegestaan is.

Het volgende is een copie van de getallenband. De betekenis van de getallen wordt verklaard in de regels 2 t/m 11 van het ALGOL-programma in hoofdstuk 6.

'getallen machine loading problem'

```

num of prod      9
num of weeks     3
drspace          81920
sminf            5000
prodtime         160, 160, 160
demand           12, 1, 1, 11, 0, 1, 1, 0, 13
                  62, 6, 3, 1, 25, 13, 3, 18, 34
                  3, 2, 22, 17, 3, 60, 12, 5, 46
fine             -1
stcost           5.8, 1.2, 0.3, 1.8, 0.8, 7.0, 2.6, 0.4, 3.5

clcost           4.1, 6.1, 6.4, 7.2, 7.9, 16.0, 9.8, 18.3
                  5.9, 6.3, 7.1, 7.7, 15.7, 9.7, 17.8
                  5.8, 6.2, 7.5, 15.7, 9.6, 16.4
                  13.8, 14.0, 14.3, 15.1, 15.3
                  5.9, 14.2, 8.3, 15.3
                  14.0, 8.1, 14.8
                  3.9, 14.5
                  14.4

cltime           2, 3, 3, 4, 4, 8, 5, 6
                  3, 3, 4, 4, 8, 5, 6
                  3, 3, 4, 8, 5, 5
                  7, 7, 7, 8, 8
                  3, 7, 4, 4
                  7, 4, 4
                  2, 3
                  2

```

Na 55 beslissingen (rekentijd ongeveer 11 minuten) wordt een oplossing gevonden waarvoor de schoonmaaktijd en -kosten ingecalculeerd zijn. Aan- gezien er op dat moment geen eindpunten met lagere kosten zijn, moet de gevonden oplossing ook de beste zijn.

Deze oplossing bestaat uit een pad van 28 beslissingen, t.w. 10 maal "nee" en 18 maal "ja".

De boom bestaat aan het eind van de berekening uit 111 knooppunten (in- clusief de wortel), waarvan er (uiteraard) 56 eindpunt zijn. Van deze

56 hebben er weer 11 als kosten ∞ , zodat er zich op de trommel 45 kandidaten voor uitbreiding bevinden.

Nu volgen nog vier bladzijden output. De eerste drie daarvan geven een overzicht van de genomen beslissingen, en daarmee van het verloop van de berekening.

We zien v.l.n.r.: nummer en aard van de beslissing, produkt, produktieweek, rekentijd, totale rekentijd, kosten (incl.onderschatting) en de vorige beslissing (+ = "ja", - = "nee"). Het produkt wordt genoteerd door een nummer als geformuleerd in hoofdstuk 1.

De vierde pagina van de output geeft de optimale oplossing, gerangschikt per produktieweek. Binnen iedere week zijn de produkties gerangschikt naar week van aflevering. Per afleveringsweek hebben de produkties dezelfde ordening als op de getallenband. De produkties hoeven dus niet in de volgorde te staan waarin ze uitgevoerd moeten worden.

220971- 83 D 7156X.3 MARTINREM

THE SOLUTION OF THIS MACHINE LOADING PROBLEM * (9 PRODUCTS AND 3 WEEKS)

INITIAL COST: 7.70

DECISION | PRCD. | WEEK | COMPUTING TIME | TOTAL TIME | L.B. OF COST | PREV DEC.

1 YES	9	1	.26 SEC	10.05 SEC	7.70	+
2 YES	1	1	6.88 SEC	16.95 SEC	38.20	+
3 YES	4	1	.10 SEC	17.07 SEC	41.60	+
4 YES	2	1	.10 SEC	17.20 SEC	45.60	+
5 YES	3	1	.10 SEC	17.32 SEC	51.00	+
6 YES	6	1	.11 SEC	17.45 SEC	64.50	+
7 YES	7	1	6.75 SEC	24.22 SEC	66.80	+
8 NO	24	3	9.64 SEC	33.89 SEC	560.40	+
8 YES	24	3	.21 SEC	34.13 SEC	66.80	+
9 NO	14	2	8.28 SEC	42.43 SEC	436.10	+
9 YES	10	2	.24 SEC	42.69 SEC	66.80	+
10 NO	27	3	7.95 SEC	50.67 SEC	274.10	+
10 YES	27	3	7.97 SEC	58.66 SEC	102.10	+
11 NO	18	2	8.01 SEC	66.69 SEC	211.00	+
11 YES	18	2	7.99 SEC	74.70 SEC	119.30	+
12 NO	15	2	8.06 SEC	82.79 SEC	204.90	+
12 YES	15	2	9.13 SEC	91.94 SEC	124.50	+
13 NO	25	3	7.57 SEC	99.54 SEC	155.60	+
13 YES	25	3	7.96 SEC	107.53 SEC	141.80	+
14 NO	22	3	7.67 SEC	115.23 SEC	172.10	+
14 YES	22	3	8.38 SEC	123.63 SEC	153.20	+
15 YES	13	2	7.93 SEC	131.58 SEC	168.10	+
15 NO	13	2	7.90 SEC	139.51 SEC	154.60	+
16 NO	19	3	7.65 SEC	147.19 SEC	169.80	-1
16 YES	19	3	7.93 SEC	155.15 SEC	163.60	-1
17 NO	22	3	7.62 SEC	162.80 SEC	198.60	-1
17 YES	22	3	7.45 SEC	170.28 SEC	173.80	-1
18 YES	26	2	7.78 SEC	178.08 SEC	172.10	+
18 NO	26	2	7.58 SEC	185.68 SEC	163.00	+
19 YES	16	2	7.36 SEC	193.06 SEC	179.50	-1
19 NO	16	2	7.36 SEC	200.44 SEC	169.10	-1
20 NO	19	3	7.75 SEC	208.21 SEC	184.00	+
20 YES	19	3	8.24 SEC	216.47 SEC	176.30	+
21 YES	17	2	7.37 SEC	223.86 SEC	177.60	-1

220971- 83

D 7156X.3 MARTINREM

21 NO	17	2	7.38 SEC	231.26 SEC	169.70	-10
22 YES	17	1	.10 SEC	231.38 SEC	173.50	-2
23 YES	26	2	7.60 SEC	239.01 SEC	178.30	-16
23 NO	26	2	7.81 SEC	246.84 SEC	169.80	-16
24 YES	16	2	7.34 SEC	254.20 SEC	185.70	-23
24 NO	16	2	7.29 SEC	261.52 SEC	175.30	-23
25 YES	13	2	7.43 SEC	268.98 SEC	183.20	-14
25 NO	13	2	7.34 SEC	276.35 SEC	173.10	-14
26 NO	14	2	7.81 SEC	284.18 SEC	183.00	+18
26 YES	14	2	7.67 SEC	291.87 SEC	178.50	+18
27 YES	16	2	7.27 SEC	299.16 SEC	191.60	-25
27 NO	16	2	7.19 SEC	306.38 SEC	178.50	-25
28 NO	14	2	7.46 SEC	313.86 SEC	189.20	+22
28 YES	14	2	7.13 SEC	321.01 SEC	178.40	+22
29 YES	13	2	7.43 SEC	328.47 SEC	191.10	+17
29 NO	13	2	7.32 SEC	335.81 SEC	174.80	+17
30 NO	19	3	7.08 SEC	342.91 SEC	192.80	-29
30 YES	19	3	7.31 SEC	350.24 SEC	183.00	-29
31 YES	17	2	7.30 SEC	357.56 SEC	183.80	-24
31 NO	17	2	7.54 SEC	365.12 SEC	175.90	-24
32 YES	17	1	.10 SEC	365.24 SEC	179.70	-31
33 YES	16	2	7.98 SEC	373.25 SEC	193.20	+20
33 NO	16	2	7.56 SEC	380.83 SEC	183.10	+20
34 NO	14	2	7.38 SEC	388.23 SEC	189.20	+21
34 YES	14	2	7.29 SEC	395.54 SEC	182.70	+21
35 YES	20	3	7.69 SEC	403.25 SEC	187.00	+23
35 NO	20	3	7.45 SEC	410.73 SEC	179.90	+23
36 YES	12	2	7.07 SEC	417.82 SEC	184.00	+20
36 NO	12	2	6.96 SEC	424.81 SEC	179.00	+20
37 NO	19	3	7.10 SEC	431.93 SEC	196.50	-27
37 YES	19	3	7.16 SEC	439.12 SEC	188.80	-27
38 YES	20	3	7.53 SEC	446.67 SEC	184.90	+26
38 NO	20	3	7.70 SEC	454.39 SEC	179.30	+26
39 YES	20	3	7.11 SEC	461.52 SEC	186.00	-36
39 NO	20	3	6.96 SEC	468.51 SEC	179.80	-36
40 YES	23	3	7.76 SEC	476.30 SEC	189.20	-38
40 NO	23	3	7.84 SEC	484.16 SEC	179.30	-38
41 YES	23	1	.11 SEC	484.30 SEC	185.00	-4
41 NO	23	1	7.04 SEC	491.36 SEC	179.30	-4

220971- 83 D 7156X,3 MARTINREM

42 YES	16	2	6.82 SEC	498.20 SEC	191.50	-41
42 NO	16	2	6.85 SEC	505.08 SEC	184.30	-41
43 YES	23	2	7.53 SEC	512.64 SEC	184.50	+12
43 NO	23	2	7.52 SEC	520.19 SEC	179.50	+12
44 YES	23	3	7.25 SEC	527.46 SEC	191.80	-43
44 NO	23	3	7.55 SEC	535.03 SEC	180.00	-43
45 NO	14	2	7.09 SEC	542.15 SEC	195.40	+32
45 YES	14	2	7.21 SEC	549.38 SEC	184.60	+32
46 YES	23	3	6.84 SEC	556.25 SEC	189.80	-39
46 NO	23	3	6.96 SEC	563.24 SEC	180.40	-39
47 YES	23	3	7.47 SEC	570.74 SEC	188.00	-35
47 NO	23	3	7.31 SEC	578.07 SEC	180.90	-35
48 YES	23	1	6.01 SEC	584.10 SEC	182.40	-44
49 YES	21	2	6.72 SEC	590.84 SEC	186.00	-46
49 NO	21	2	6.52 SEC	597.38 SEC	181.00	-46
50 YES	23	1	.22 SEC	597.63 SEC	186.60	-47
50 NO	23	1	6.70 SEC	604.35 SEC	181.90	-47
51 YES	26	3	6.24 SEC	610.61 SEC	185.60	-49
51 NO	26	3	6.16 SEC	616.80 SEC	181.40	-49
52 YES	21	3	6.10 SEC	622.93 SEC	189.90	-51
52 NO	21	3	5.94 SEC	628.90 SEC	182.60	-51
53 YES	23	2	6.56 SEC	635.49 SEC	188.50	-51
54 YES	14	2	7.26 SEC	642.78 SEC	191.20	+43
54 NO	14	2	7.41 SEC	650.22 SEC	194.90	+43
55 NO	11	2	5.82 SEC	656.06 SEC	188.20	-52
55 YES	11	2	4.80 SEC	660.88 SEC	182.60	-52

(CLEAN)

220971- 83 D 7156X,3 MARTINREM

FINAL SOLUTION, WITH A COST OF: 182.60

WEEK 1 (IDLE TIME 42)

AMOUNT	PROD	WEEK
12	1	1
1	2	1
1	3	1
11	4	1
1	6	1
1	7	1
13	9	1
3	3	2
1	4	2
3	7	2
18	8	2
22	3	3
5	8	3

WEEK 2 (IDLE TIME 2)

AMOUNT	PROD	WEEK
62	1	2
6	2	2
25	5	2
13	6	2
34	9	2
2	2	3
3	5	3

WEEK 3 (IDLE TIME 2)

AMOUNT	PROD	WEEK
3	1	3
17	4	3
60	6	3
12	7	3
46	9	3

NUMBER OF CALLS OF 'REAL STEPPING STONE':	170
NUMBER OF CALLS OF 'UPDATE':	89
USED SPACE ON THE DRUM:	13635
TIME SINCE INITIALIZATION:	661.28 SEC

8. Literatuur.

J.D.C. Little, K.G. Murty, D.W. Sweeney and K. Caroline: An Algorithm For The Travelling Salesman Problem, Operations Res. 11 (1967), 972-989.

(Dit artikel geeft een duidelijke verhandeling over branch-and-bound-technieken).