Multi-dimensional map-folding

W. F. Lunnon

Science Research Council, Atlas Computer Laboratory, Chilton, Didcot, Berkshire and Department of Computer Science, University of Manchester

Lunnon (1968) counted foldings of a one-dimensional map (otherwise known (Touchard, 1950; Koehler, 1968) as a strip of stamps); here the problem is generalised to many dimensions, with especial reference to $p \times q$ (two-dimensional) and $2 \times 2 \times \ldots \times 2$ maps. A computer program and its results are described.

(Received November 1969)

1. Sections in two and many dimensions

A '$p \times q$-map' is the rectangle in the Cartesian plane $0 \leqslant x \leqslant p$, $0 \leqslant y \leqslant q$ (**Fig. 1.**) Its 'creases' are the lines

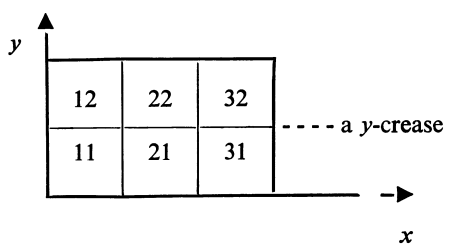

Fig. 1

x = integer and y = integer ('y-creases' and 'x-creases') and its 'leaves' the lattice squares. The map is 'folded' by rotating parts of it about its creases to superpose its leaves: it may be bent and stretched, but not torn. A '$p \times q$-folding' is complete when all leaves are superposed on the leaf 11 nearest the origin; this leaf is fixed throughout to ensure that the 'front cover' (marked with a dot) always faces the same way.

A folding is completely described by the order of its leaves. Alternatively we can section it by planes perpendicular respectively to the x- and to the y-creases: **Fig. 2** shows one of the 60 possible 3×2-foldings with its sections.

The x-section consists of p one-dimensional q-maps folded together, and the y-section of q p-maps. Provided only that we know which is leaf 11 we can deduce which are the others from the interconnections in Fig. 2: so a folding is completely described by its sections and the position of the first leaf. Furthermore, any pair of sections which looks as if it should define a folding actually does so:

Theorem 1:
A 'pile' (which is just an ordering) of the leaves of a map is a folding iff all its sections are (one-dimensional, multiple) foldings.

Proof:
Firstly, a pile is a folding iff no crease between any adjacent pair of leaves crosses any other crease. Obviously if the creases cross we can't have a folding, for the map mustn't be torn. It's not so clear that any non-crease-crossing pile can be unfolded again to a flat map: this is accomplished by shrivelling or concertina-ing it, one unit at a time along one axis at a time, until it is reduced to a single point at the origin; then expanding it again flat.

Secondly, a pile is non-crease-crossing iff all its sections are, that is iff they are proper one-dimensional foldings: for if—say—an x-crease crosses any other crease, that must also be an x-crease and so there will be a crossing in the x-section.

Combining these two observations completes the proof.

We now extend these ideas to many dimensions. An account of rotation and perpendicularity in higher space is to be found in Coxeter (1963).

A '$p_1 \times \ldots \times p_d$-map' is the region $0 \leqslant x_i \leqslant p_i$ for $i = 1, \ldots, d$ of Cartesian d-space. The hyperplanes x_j = integer are its 'x_j-creases', and the unit lattice hypercubes its 'leaves'. We write n for the number of leaves, i.e.

$$n = \prod_{i=1}^{d} p_i$$

A 'folding' is any deformation of the map causing all n leaves to be superposed on the first leaf (that nearest the origin), which is fixed to preserve the orientation. These

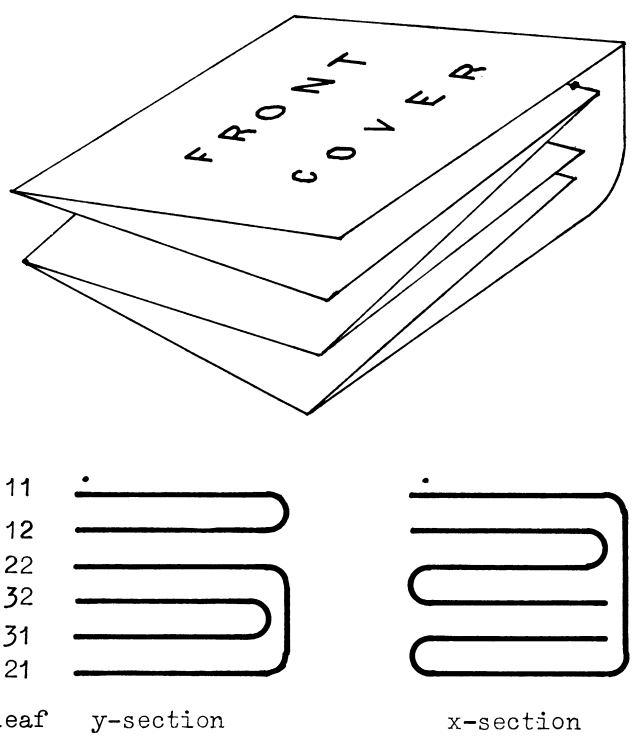

Fig. 2

deformations may be visualised as sequences of rotations (in $d + 1$ dimensions) of parts of the map about its $(d - 1$ dimensional) creases, together with any distortions necessary to interleaving.

The 'x_j-section' of a folding is its intersection in $(d + 1)$-space with the plane (perpendicular to the x_j-creases)

$$x_i = \tfrac{1}{2} \text{ for } i = 1, \ldots, j-1, j+1, \ldots, d.$$

Each leaf meets this plane in a line, and the whole section comprises a (one-dimensional) folding of n/p_j disconnected, interleaved p_j-maps. The set of d x_j-sections for $j = 1, \ldots, d$, together with the position of the first leaf, completely describes the folding; and by Theorem 1 any pile of leaves whose sections are foldings is itself a folding.

2. Reducing the counting problem

We wish to count the total $G = G(p_1, \ldots, p_d)$ of possible $p_1 \times \ldots \times p_d$-foldings: this can be reduced to counting a subset in several ways which we shall describe. Trivially, the order of the p_i is immaterial, so we can assume $p_1 \geqslant p_2 \geqslant \ldots \leqslant p_d$, Any $p_i = 1$ may be ignored. If any $p_i = 0$, $n = 0$ and it seems reasonable to convene $G = 1$ for the null folding (contradicting Lunnon (1968)).

In Lunnon (1968) we showed that $G(p)$ is divisible by $2p$. The same construction works in many dimensions:

Theorem 2:
G is divisible by n. If some $p_i > 2$, G is divisible by $2n$.

Briefly, to get n-divisibility we 'rotate' the folding: that is, drag its topmost leaf down through the others to the bottom while preserving the connections to other leaves. Each 'normal' folding (i.e. whose first leaf is uppermost, e.g. Fig. 2) now corresponds to $n - 1$ non-normal others by rotation, so G/n foldings are normal. For 2-divisibility, suppose $p_1 > 2$ and the folding is normal. The first x_1-crease must turn downwards since the first leaf is at the top, but the second may turn up or down. If it turns up, we can make it turn down by rotating the first leaf to the bottom and turning the whole folding upside-down (making it normal again) and turning the map upside-down inside its folding (facing the front cover up again). So $G/2n$ normal foldings have their second crease turned downwards.

Theorem 3:
G is divisible by 2^d if all $p_i > 1$.

For any given folding we can 'reverse' it along the x_i-axis: e.g. if $i = 1$ the old first leaf $11 \ldots 1$ becomes the new leaf $p_1 1 \ldots 1$. The first leaf can be reversed to any of the 2^d corners, so 2^d divides G.

Now if all the p_i are odd, Theorems 2 and 3 combine to give G divisibility by $2^d n$ (since n is odd). This fact is computationally useless, since we have no means of enumerating a subset of size $G/2^d n$. Our results suggest that even when the p_i are not restrained to oddness it is still true that

Conjecture:
G is divisible by $2^k n$ where k of the p_i are > 2.

Can we prove this by extending the second-crease-down idea of Theorem 2 to all directions? E.g. for $p \times q$-foldings with $p, q > 2$ we want equality between $G/4n$ and the total of normal foldings both of whose second creases turn down. By actual counting we find that

for 3 × 3 these are	38 and	38
4 × 3	324	325
5 × 3	3,354	3,362
4 × 4	4,697	4,584

which seems to dispose of that idea. It would be useful to discover a subset which does total $G/2^k n$.

If some of the p_i are equal G can be further reduced by swopping axes. For example in two dimensions, if $p = q$, to every folding there corresponds another whose x-section is the first one's y-section and vice-versa. Normality is preserved, so in the special case $p_1 = p_2 = \ldots = p_d = p$ we can combine swopping with Theorem 2 to give

Theorem 4:
$G(p^d)$ is divisible by $d!p^d$; the extension to other patterns of equality between the p_i is obvious (see below).

It is certainly not possible to combine Theorem 3 with this as well: the conjecture that $G(p_d)$ is divisible by $p^d.2^d.d!$ fails for 4^2. In spite of this, we give in **Table 1** the factorisation of G in which the first factor is $n.2^k.\Pi e_i!$, where e_i is the number of times each distinct p_i is repeated and k of the p_i exceed 2.

[Notice that our 'axis-swops' and 'reversals' are just the symmetries (rotations plus reflections) of the d-hypercube with centre the origin. If the $(d + 1)$st dimension is bent round to form a 'torus', our 'rotations' are rotations of d-space along this torus.]

G is tabulated in Table 1. Most of our results are for the $p \times q$ and p^d cases, these being more intuitively appealing. We had hoped that the multidimensional problem might reduce theoretically to the one-dimensional as it does computationally: but such is not the case. We shall therefore discuss only

3. 2^d—maps

These are the simplest multi-dimensional maps, with $p_i = 2$ for $i = 1, \ldots, d$. Here at least it is easy to find an answer to our problem. By analogy with 2×2-foldings we see that there are $d!$ ways to choose the order in which the d creases are folded, and for each choice and each crease there are two ways to fold it: up or down. This gives altogether $2^d d!$ foldings (the minimum by Theorem 4), of which for $d = 3$ a typical one is shown in **Fig. 3.** Its creases were folded in the order x, y, z and all downwards. There are 48 like it.

Bearing this in mind we viewed our program with gentle reproach on production of the answer $G(2^3) = 96$. The error proved difficult to locate, having taken up unsuspected residence in the above reasoning. While rotation of any 2×2 folding yields another of the same basic shape, rotation of Fig. 3 yields the new shape (**Fig. 4**); and all 2^3-foldings are axis swops and reversals of one of these two shapes ($d!2^d = 48$ from each shape), rather than of Fig. 3 alone.

[A 'shape' is a class of foldings which are swops and reversals of each other: that is, their sections all look alike.]

Perhaps then any 2^d-folding is either an 'ordinary' one (as in Fig. 3) or some rotation of it? Of the 12 2^4-shapes, only 4 are of this form: the rest are the rotations of the shape got by folding Fig. 4 in two (instead of three).

[A 'roto-shape' is a class of shapes which are rotations of each other.]

Well then, are all 2^d-foldings rotations of some 2^{d-1}-folding folded in two? That is, is the general 2^d-roto-shape a doubled 2^{d-1}-shape? There are 9 roto-shapes in 5 dimensions. **Fig. 5** shows the single one which is not a doubling of any 2^4-shape. It speaks for itself.

Not only is the natural guess wrong, but no simple construction accounts for objects like Fig. 5. Having failed to explicitly evaluate even $G(2^d)$, we leave the reader to contemplate its first few values (Table 1). Notice that the

Table 1

$p.q.\ldots$	$G(p,q,\ldots)$	$p.q.\ldots$	$G(p,q,\ldots)$
1	1 = 1·1	3·2	60 = 12·5
2	2 = 2·1	4·2	320 = 16·20
3	6 = 6·1	5·2	1980 = 20·99
4	16 = 8·2	6·2	10512 = 24·438
5	50 = 10·5	7·2	60788 = 28·2171
6	144 = 12·12	8·2	320896 = 32·10028
7	462 = 14·33	9·2	1787904 = 36·49664
8	1392 = 16·87	10·2	9381840 = 40·234546
9	4356 = 18·252	11·2	51081844 = 44·1160951
10	14060 = 20·703	4·3	15552 = 48·324
11	46310 = 22·2105	5·3	201240 = 60·3354
12	146376 = 24·6099	6·3	2016432 = 72·28006
		7·3	21582624 = 84·256936
2^2	8 = 8·1	5·4	6139920 = 80·76749
2^3	96 = 48·2		
2^4	4608 = 384·12	$3\cdot2^2$	2448 = 48·51
2^5	798720 = 3840·208	$4\cdot2^2$	30720 = 64·480
2^6	361267200 = 46080·7840	$5\cdot2^2$	394320 = 80·4929
3^2	1368 = 72·19	$3^2\cdot2$	227952 = 144·1583
3^3	85109616 = 1296·65671		
4^2	300608 = 128·2348½		
5^2	186086600 = 200·930433		

first factor (see Section 2) $n.2^k.\Pi e_i!$ reduces to $2^d d!$ in this instance.

4. The counting program

From the programming point of view we have more or less reduced the problem to the one-dimensional case described in Lunnon (1968). We have merely to fold d n-maps simultaneously, the i-th of which is disconnected into p_i pieces. As an example, suppose we are constructing 3 × 3 maps and have already produced some legitimate arrangement of the first 8 leaves (**Fig. 6**). To insert the ninth leaf 33, we first look at the x-section and the leaf 23 to which 33 is joined by an x-crease. Since this crease mustn't cross any other, leaf 33 is constrained to the gaps above leaf 11 or below 11, 31, 12, 13 or 23. Similarly, the y-section constrains it to the gap below leaf 22, 32, 31 or 13. The gaps common to both sets lie below leaves 31 and 13, so these are the only gaps for 33.

If the new leaf has no neighbour already present in a given section then that section imposes no constraints. If it is not the final leaf, for each gap in turn we insert it there and repeat the process on the next leaf.

If $d > 1$ we may well reach a situation where there are no gaps for the next leaf, obliging us to backtrack without having discovered any new foldings. It may be possible to

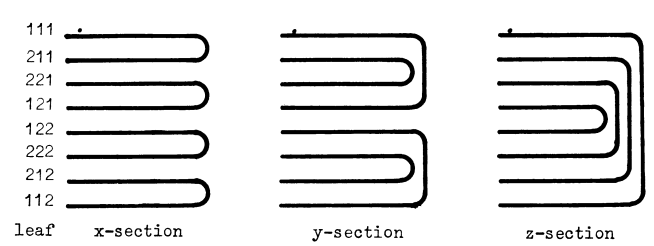

Fig. 3

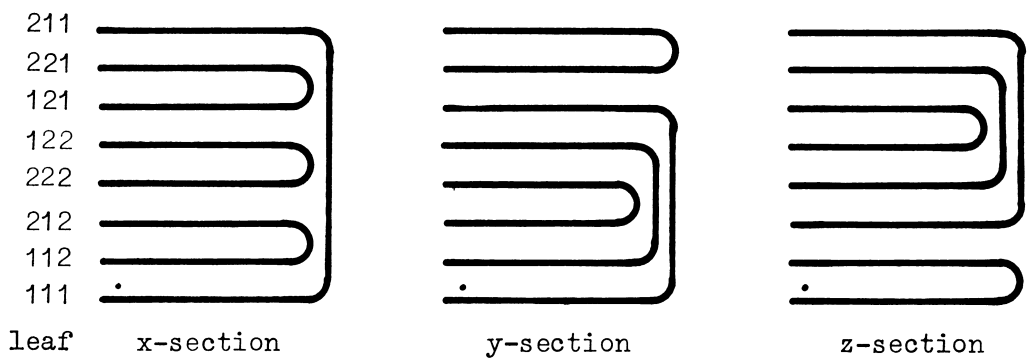

Fig. 4

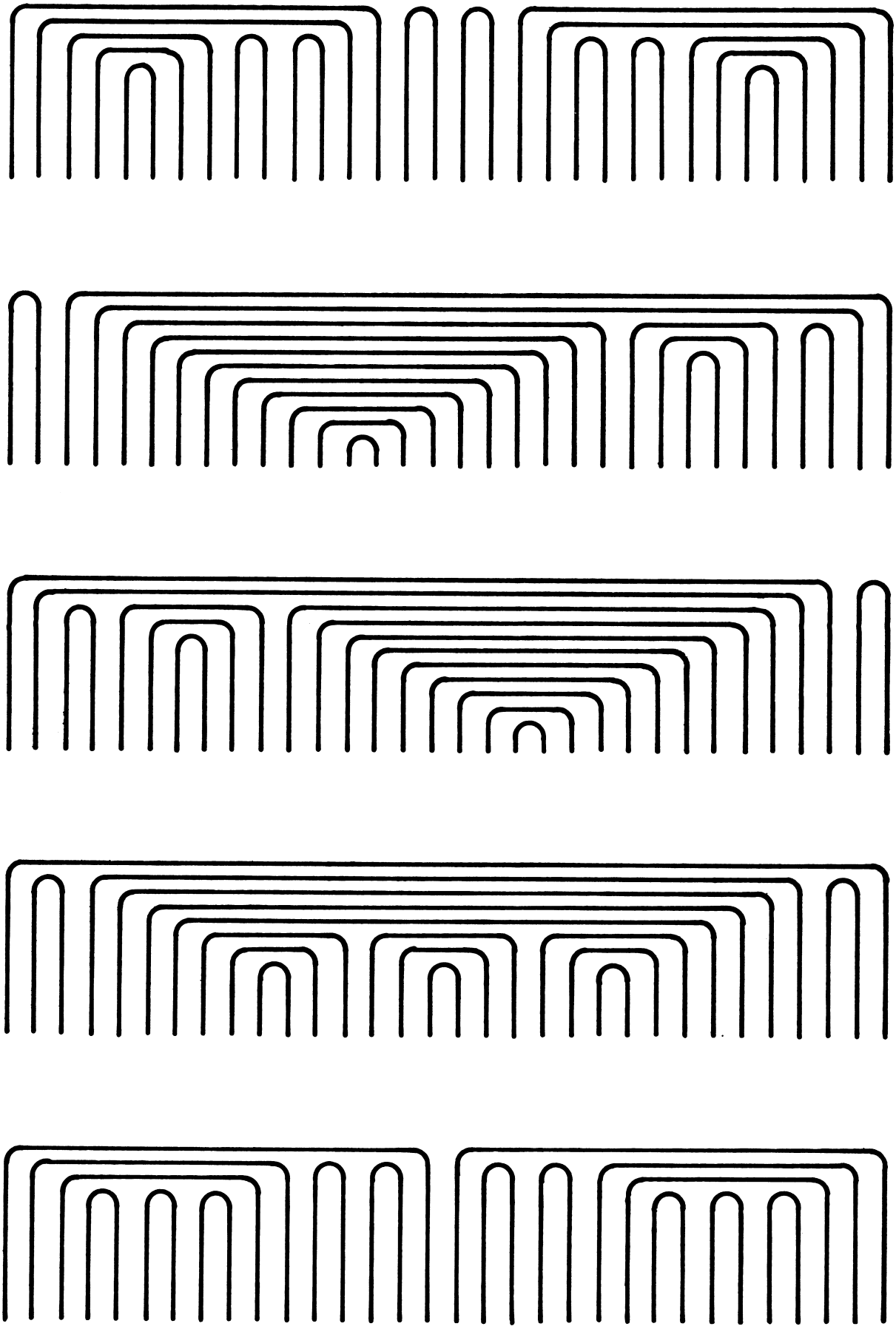

Fig. 5

reduce this 'blind-alley' effect by varying the order in which leaves are inserted. The order we have used is alphabetical.

Another reasonable order would be diagonal: that is all leaves on the hyperplane $\sum q_i = k$ are inserted before those on $\sum q_i = k + 1$. We have not tried this, nor have we counted the proportion of blind alleys: we doubt if re-ordering could improve the time by more than a factor of 2 or so, if at all. The object is to choose leaves which are difficult to fit in as early as possible.

We now move on to the details of implementation. The current (partial) folding is most conveniently stored as a vector B such that: if m is the number of any leaf in alphabetical order, then $B[m]$ is the number of the leaf directly below it. [If the coordinates of a leaf are $q_1 \ldots q_d$ then its alphabetical number is

$$m = \prod_{i=1}^{i=d} q_i P_{i-1} \text{ where } P_i = \prod_{j=1}^{j=i} p_j$$

See the bracketed numbers in Fig. 6.] Let $D(i, l, m)$ be the number of the leaf connected to leaf m in the x_i-section on whichever side is appropriate to the insertion of the next leaf l; this is a simple function of the coordinates. The fourth paragraph of the program sets up D in an array, using P_i and the coordinates C_{im} of leaf m. Then the sequence (paragraph 7)

for $m := D(i, l, l), D(i, l, B[m])$ **while** $m \neq n$ **do** . . . ;

causes m to run through all gaps to which the x_i-section constrains leaf l. If l happens to have no neighbours in the i-th section, that section imposes no constraints and is ignored ('*add*' counts these). If this happens for all i—only possible if $l = 1$ with the present numbering system—any gap is possible.

Having found these gaps for each i we must detect the common ones. Each gap found in any section is entered in the '*gap*' list and simultaneously its '*count*' (initially zero) is incremented. The gap is common iff its count is eventually d; the remainder are expunged from the gap list by the next loop (paragraph 8).

Finally (paragraph 9) the new leaf l is inserted in each gap in turn, and the process repeated for $l + 1$. This recursion is programmed explicitly (involving the stack '*gapter*' for the pointer g to the current gap for l, and the '*above*' vector $A[m]$ with a view to machine coding.

The program is presented as a routine to perform a given 'job' on each folding. For example, this sequence puts the total $G(p)$ into '*sum*', where the integer array 'p' has been loaded with $p_1, \ldots, p_d$:

procedure *count* (*A*, *B*); **integer array** *A*, *B*;
  sum := *sum* + 1;
integer *sum*; *sum* := 0; *foldings* (*p*, *count*);

To merely count foldings we obviously need not insert the final leaf $l = n$. Furthermore the results of Section 2 enable us to enumarete only a known fraction of the total. To get only normal foldings we drop the fictional leaf 0 of the program (invented to make paragraph 7 work for outside gaps) and start off instead with $B[1] := 1$ (i.e. the first leaf is below itself.) By manipulating B and D and paragraph 7 we can simultaneously force the 'second crease' of Theorem 2 down; or (but not and), along axes whose p_i are equal, we can force the second leaves in a specific order to eliminate axis-swops.

The roto-shapes of Section 3 require some straightforward but messy manipulations involving axis-reversals. These are of no interest.

5. Program and results

```
procedure foldings (p, job); integer array p; procedure job;
  begin comment perform job (A, B) on each folding of a
    p[1] x . . . x p[d] map, where A and B are the above
    and below vectors. p[d + 1] < 0 terminates p;

  integer d, n, j, i, m, l, g, gg, dd;
  n := 1; i := d := 0; for i := i + 1 while p[i] ⩾ 0 do
    begin d := i; n := n*p[i] end
  comment d dimensions and n leaves;

  begin integer array A, B, count, gapter [0:n], gap [0:n*n];
  comment B[m] is the leaf below leaf m in the current
    folding, A[m] the leaf above. count[m] is the no. of
    sections in which there is a gap for the new leaf l below
    leaf m, gap[gapter[l − 1] + j] is the j-th (possible or
    actual) gap for leaf l, and later gap [gapter[l]] is the
    gap where leaf l is currently inserted;

  integer array P [0:d], C [0:d, 0:n], D [0:d, 0:n, 0:n];
  P[0] := 1; for i := 1 step 1 until d do P[i] :=
    P[i − 1]*p[i];
  for i := 1 step 1 until d do for m := 1 step 1 until n do
      C[i, m] := (m − 1)÷P[i − 1] −
        (m − 1)÷P[i]*p[i] + 1;
  for i := 1 step 1 until d do for l := 1 step 1 until n do
    for m := 0 step 1 until l do
      D[i, l, m] := if m = 0 then 0 else
        if C[i, l] − C[i, m] = (C[i, l] − C[i, m])÷2*2
          then
```

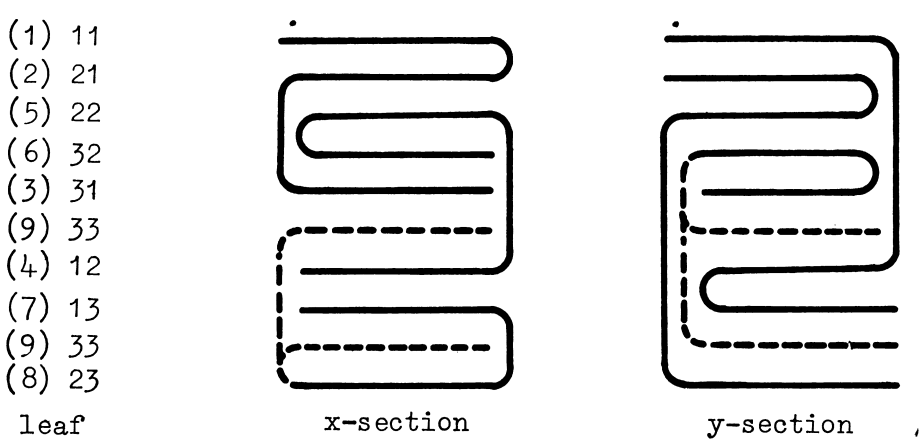

Fig. 6

```
            (if C[i, m] = 1 then m else m − P[i − 1])
              else
            (if C[i, m] = p[i] > m + P[i − 1] > 1 then
              m else m + P[i − 1])
comment P[i] = p[1] x . . . x p[i], C[i, m] = i-th co-
  ordinate of leaf m,
    D[i, l, m] = leaf connected to m in section i when
      inserting l;

for m := 0 step 1 until n do count[m] := 0;
A[0] := B[0] := g := l := 0; goto entry;
comment kick off with null folding;

down: add := 0; gg := g := gapter[l − 1];
      comment dd is the no. of sections in which l is
                unconstrained,
              gg the no. of possible and g the no. of
                actual gaps for l, + gapter[l − 1];

      comment find the possible gaps for leaf l in each
                section, then discard those not common
                to all. All possible if dd = d;
      for i := 1 step 1 until d do if D[i, l, l] = l then
        dd := dd + 1 else
          for m := D[i, l, l], D[i, l, B[m]] while
            m ≠ l do
              begin gap[gg] := m; if count[m] = 0
                then gg := gg + 1;
              count[m] := count[m] + 1 end;

      if dd = d then for m := 0 step 1 until l − 1 do
              begin gap[gg] := m; gg := gg + 1 end;
      for j := g step 1 until gg − 1 do
              begin gap[g] := gap[j]; if
                count[gap[j]] = d − dd then
                  g := g + 1;
                count[gap[j]] := 0 end;

      comment for each gap insert leaf l, call self recursively,
                remove leaf l;
along: if g = gapter[l − 1] then goto up; g := g − 1;
      A[l] := gap[g]; B[l] := B[A[l]];
        B[A[l]] := A[B[l]] := l;
entry: gapter[l] := g; l := l + 1; if l ⩽ m then goto
        down else job(A, B);

up:   l := l − 1; B[A[l]] := B[l]; A[B[l]] := A[l];
      if l > 0 then goto along;
      end; end of foldings;
```


References

COXETER, H. S. M. (1963). *Regular Polytopes*, p. 124. London: Macmillan.
KOEHLER, J. E. (1968). Folding a Strip of Stamps, *J. Comb. Theory*, Vol. 5, pp. 135-152.
LUNNON, W. F. (1968). A Map-folding Problem, *Math. Comp.*, Vol. 22, pp. 193-199.
TOUCHARD, J. (1950). Contributions a l'étude du problème des timbres poste, *Canad. J. Math.*, Vol. 2, pp. 385-398.