a journal of the Society for Music Theory

Volume 4, Number 3, May 1998
Copyright © 1998 Society for Music Theory

White Note Fantasy

Stephen Soderberg

KEYWORDS: (Part I:) John Clough, Gerald Myerson, interval string, WARP function, white note system, interval matrix, interval vector, covariance; (Part II:) Eytan Agmon, John Clough, Jack Douthett, David Lewin, maximally even set, microtonal music, hyperdiatonic system, Riemann system, efficient linear transformation

ABSTRACT: In Part I "topological" transformations are utilized to generate pre-compositional systems called "white note systems." Applying the procedures developed there, Part II combines some of the work done by others on three fundamental aspects of diatonic systems: underlying scale structure, harmonic structure, and basic voice leading. This synthesis allows the recognition of select hyperdiatonic systems which, while lacking some of the simple and direct character of the usual ("historical") diatonic system, possess a richness and complexity which have yet to be fully exploited.

PART I.

A General Theory of White Note Systems.

Contents:

The following study focuses on white note systems, certain well-defined sets of "tonal" relationships arising from any m-note subset of any n-note set.[(1)] This (still informal) definition may seem somewhat surprising at first since we are not accustomed to think of tonal "stuff" as a random selection of notes. But, as we shall see in Part I, much of the "system" in tonal system comes less from our selection of notes than from the ways we choose to arrange our selection. In the present study we will generate white note systems using "topological" transformations[2], the most basic of which is the WARP function. WARP maps any interval string (to be defined) from one referential chromatic space to another via a scale string. In the section labelled "Vector Transformations" we will define a transformation matrix which concomitantly defines interval vectors. Vectors (as carriers of the most basic significant information between two "chords"), and the way they transform, prove useful throughout this study, but discussion of their ultimate analytical importance, or their possible future compositional importance, lies beyond our scope.

---

0. Symbols Used

<=	is less than or equal to
>=	is greater than or equal to
SUM(A)	sum of elements in A
#(A)	number of elements in A (cardinality of A)
[[x/y]]	largest integral value of x/y
.&.	set theoretic union
.^.	set theoretic intersection
U(A)	interval vector of A (para. [3.9])
V(A)	interval-class vector of A (para. [3.30])
(=)	correspondence (para. [4.4])
A//B	A covers B (para. [4.16])

---

1. Interval String Theory.

[1.1] REFERENTIAL SPACES. We begin by positing a referential musical space Cm with octave equivalence; m is called the size of Cm. A primitive object (point) in Cm can be thought of as a pitch class (pc) or integer mod m or as a point on a circle whose circumference is m units. Thus, for any primitive object p in Cm, an appropriate integer can be assigned to p such that 0 <= p <= m-1. Cm is sometimes called a chromatic universe. The usual chromatic C12 is the referential space underlying the usual diatonic.

[1.2] DISTANCES. The interval between any two pitch classes p1 and p2 in Cm is int(p1,p2) = (p2-p1) mod m. The interval class formed by p1 and p2 is defined as ic(p1,p2) = int(p1,p2) if int(p1,p2) <= [[m/2]], otherwise ic(p1,p2) = m-int(p1,p2).

[1.3] On a circle, int(p1,p2) measures the *clockwise* circumferential distance between p1 and p2, whereas ic(p1,p2) measures the *shortest* distance between p1 and p2, whether clockwise or counterclockwise. Thus in C7 int(1,5) = 4, but since [[m/2]] = [[7/2]] = 3 and int(1,5) > [[m/2]], ic(1,5) = 7-int(1,5) = 3.

[1.4] If S = {a,b, . . . } and T = {g,h, . . . } are two sets of pcs in Cm, classic set theory defines the set-theoretic intersection of S and T as the set containing all those elements which are common to S and T, i.e., all elements s-in-S and t-in-T such that s = t or, what is the same in the present context, such that int(s,t) = 0 and ic(s,t) = 0. But in a music-theoretic context it is often important to form the set of all pairs (s,t) in SXT such that int(s,t) is an integer other than 0. Thus if S = {1,3,4,7} and T = {4,8,11,15} in C16, U = {(3,4),(7,8)} in SXT is the set of pairs of pcs satisfying int(s,t) = 1.

[1.5] In general, for s-in-S and t-in-T, intj(S,T) will represent the set of all pairs in SXT such that int(s,t) = j; and #intj(S,T) will indicate the cardinality of the set intj(S,T), or equivalently, the "multiplicity" of j in SXT. Thus in the example just cited we can write int1(S,T) = {(3,4),(7,8)} and #int1(S,T) = 2. Obviously, int0(S,T) is identical to the intersection S.^.T; so both expressions can be used to define the set of "common tones" between S and T. icj(S,T) and #icj(S,T) are similarly defined.[3]

[1.6] An important special case of [1.5] appears when T = S, that is, when all pairs of pcs (s,t) belong to SXS. intj(S,S) and #intj(S,S) are straightforward, and we will write intj(S) and #intj(S) to indicate the discrete pairs of pcs *in* S separated by intj and the cardinality of that set of pairs, respectively. But icj(S,S) is not the same as icj(S) since #icj(S) is always half the value of #icj(S,S). Every instance of icj is counted twice (once "up" and once "down") in icj(S,S), but the pairs of icj-related pitch classes *in* S are (traditionally) unordered and so (s,t) and (t,s) are counted as one. So it is important to remember that (while intj(S,S) = intj(S)) icj(S,S) and icj(S) are different, but closely related, sets. E.g., if S = {0,1,3,9,10} in C12, then int2(S,S) = int2(S) = {(1,3),(10,0)}; int10(S,S) = int10(S) = {(3,1),(0,10)}; ic2(S,S) = {(1,3),(3,1),(10,0),(0,10)}; but ic2(S) = {(1,3),(10,0)}.[4]

[1.7] INTERVAL STRINGS. An interval string in Cm is an ordered n-tuple of integers s = <i1,i2, . . . , in> whose elements sum to m. Thus <111215421> is an interval string in C18; <345>, <4332>, and <21212121> are interval strings in C12.[5]

[1.8] A substring of the interval string s is any ordered k-tuple r = /j1,j2, . . . , jk/ where j1, . . . , jk are k consecutive elements of s. If s = <111215421>, /5421/, /111/, /5/, /21/ are substrings of s. /42111/ is also a substring since s is circular. /254/ is not a substring since it consists of non-consecutive intervals of s, and /512/ is not a substring of s since it presents consecutive intervals but in retrograde.

[1.9] Given any string s = <i1,i2, . . . , in> in Cm, we can form the string inversion of s, s' = <-i1,-i2, . . . , -in> mod m = <in, . . . , i2,i1>. In other words, we are identifying a simple retrograde of a string (the reverse ordering of a circularly ordered n-tuple) with an "inversion" (mod m) of that string. This identification will be clarified below in [1.14].

[1.10] STRUCTURES. Given a pitch class p and a string s = <i1,i2, . . . , in>, any structure (complex object) in Cm can be fully described by naming the ordered pair (p,s) which represents the set {pk : pk is the sum of p and the first k elements of s (k = 0,1, . . . , n-1) mod m}. For any structure in Cm, the shape and orientation of that structure are given by s, and the location of that structure is given by p. Contextually, (p,s) might be referred to as a pc-set, chord, scale, sonority, or polygon-inscribed-in-a-circle (the latter being a particularly useful metaphor in what follows). The pitch class p might be referred to as an initial, base, or root of (p,s). We will write ps in place of (p,s) where the context is clear.

[1.11] The notation (p,s) will be recognized as a generalization of the (p,sign) notation introduced by David Lewin.[6] If p is any pitch class in C12, "+" indicates "major," and "-" indicates "minor," then for example, (C♯,+) indicates a C♯ major triad and (F,-) an F minor triad (in any inversion). Replacing "sign" with "interval string," assigning C♯ = 1, and stipulating the interval string x = <435>, we may also write (C♯,+) = (1,x) = 1x = {1,1+4,1+4+3} = {1,5,8} = any major (diatonic) triad whose root is C♯.

[1.12] Extending this nomenclature to virtually any pc-set in any referential space is then quite simple. Using the previous examples in [1.7], if we define s = <111215421> then the structure (klang, set, chord, scale) named 3s is the (unordered) pc-set {3,4,5,6,8,9,14,0,2} in C18. If we set t = <345>, u = <4332>, and v = <21212121>, then 7t = {7,10,2} (a G minor triad), 2u = {2,6,9,0} (a D major dominant seventh chord), and 5v = {5,7,8,10,11,1,2,4} (an octatonic scale on F).

[1.13] While any two circular permutations of a given string express the same shape, care must be taken when naming a pc-set. s = <3142> and t = <4231> are circular permutations of one another and therefore describe the same shape in C10, however for any p in C10, (p,s) is not the same set as (p,t), but (p,s) and (p+4,t) *are* equal.

[1.14] We can now clarify the idea of equating "inversion" and "retrograde" (with respect to string relationships) introduced in [1.9]. If s = <1,1,2,4> in C8, its inversion (by the definition in [1.9]) is s' = <-1,-1,-2,-4> mod 8 = <7,7,6,4>. This is not a "proper" string as defined in [1.7] where the elements of a string are required to sum to m. We can find a "proper" string representation of s' in C8 as follows. If we identify s' with the root 0, forming the structure 0s' = {0,7,6,4}, and rearrange 0s' in ascending order as, say, 0s" = {0,4,6,7}, we see that s" = <4,2,1,1> and thus s" is a retrograde of s.

[1.15] The traditional concept of "set class" can be expressed with string notation in the following way. If s is a string in Cm and s' is the retrograde of s, the set class S is the set of all structures (p,s).&.(p,s') for all values of p in Cm. By analogy we also define the concept "string class" as the set of all strings s.&.s' for all circular permutations of s.

[1.16] If S = ps and T = qt are two sets (structures) in Cm, where p,q are pcs and s,t are strings, certain operations on sets can be written in terms of operations on the strings alone. For instance, #icj(S) = #icj(s), where #intj(s) counts the number of instances of icj in the string s (e.g., if s = <1123>, #ic2(s) = 2). But other operations cannot be so transferred, e.g., #icj(S,T) is meaningful but #icj(s,t) is undefined.

2. Generating White Note Systems.

[2.1] “TOPOLOGICAL” TRANSFORMATIONS BETWEEN REFERENTIAL SPACES. Given a k-element set A in the referential space Cm and a second referential space Cn, the “topological” transformation (TT)

R: A-in-Cm --> B-in-Cn

is the mapping, according to some rule R, of each of the k elements of A-in-Cm to a corresponding element in Cn which, taken together, form the k-element set B-in-Cn.

[2.2] A TT can be represented by imagining Cm and Cn to be circles drawn on a plane surface with a peg at each of the m and n equally spaced points on their respective circumferences and imagining A as a rubber band stretched between k of Cm’s pegs to form a polygon. A TT describes how, according to some rule, the rubber band is removed from Cm’s pegs and placed on k of Cn’s pegs.

[2.3] If n = m, the transformations are *within* the space Cm. Here, two well known TT’s of any set are transposition (rotating A in Cm) and simple inversion (flipping A around some axis, still in Cm). In both of these cases the TT retains a structure’s basic shape, i.e., these mappings are conformal.

[2.4] If n > m or n < m, however, the most notable thing about the TT of the rubber band is not so much which pegs it vacates in Cm and which pegs it then occupies in Cn, but how its k sides are warped (stretched or shrunk),

R: A-string --> B-string.

Thus a transformation which takes A from a smaller space to a larger space, or vice versa, often changes the characteristic shape of A, i.e., these are often non-conformal mappings. We will now define a non-conformal TT with n > m called a WARP function.

[2.5] WARP FUNCTION — DEFINITION. If q = <q1,q2, . . . , qk> is a string in Cm and r = <r1,r2, . . . , rm> is a string in Cn such that (k = #(q)) <= (m = SUM(q) = #(r)) <= (n = SUM(r)), then the r-WARP of q is defined as the string in Cn,

s = WARP(q,r) = <q1r,q2r, . . . , qkr>,

where q1r = the sum of the first q1 elements of r, q2r = the sum of the next q2 elements of r, . . . , and qkr = the sum of the final qk elements of r. To accord with the terminology first introduced by John Clough and Gerald Myerson[7] now in general use, any three interval strings in the relationship described by the above definition of WARP will be refered to as a generic string (q), scale string (r), and specific string (s).

[2.6] The WARP function isn’t as complex as it might appear at first glance. All that is required is any pair of strings q and r such that the (arithmetic) sum of the elements in q is equal to the cardinality of r. q can then be WARPed into a new string s by collecting r’s elements into q-counted substrings and arithmetically summing the elements in each substring. Thus q is “scaled” by or through r.

[2.7] If q is the string <2243> it can be WARPed by any string containing 2+2+4+3 = 11 elements. If we choose r = <11211311111> then the r-WARP of q is WARP(q,r) = s = <(1+1),(2+1),(1+3+1+1),(1+1+1)> = <2363>, and q-in-C11 has WARPed into s-in-C14. If x = <31134112222> then WARP(q,x) = <4486> = y, and q-in-C11 has WARPed into y-in-C22.

[2.8] The example in [2.7] demonstrates that the same generic string can WARP into different specific strings by varying the scale string. The following example shows that different generic strings can WARP into the same specific string. Let a = <214>, b = <1231222>, c = <321>, d = <111217>. Then WARP(a,b) = WARP(c,d) = <337> = s, and both a-in-C7 and b-in-C6 have WARPed into s-in-C13.

[2.9] WARP is associative with respect to composition:

WARP(WARP(x,y),z) = WARP(x,WARP(y,z));

therefore WARP(w,x,y, . . . , z) is unambiguous given appropriately defined strings w,x,y, . . . , z.

[2.10] We will define the order of a (composite) WARP as the number of strings in the composition. Thus WARP(x,y) is a second-order WARP and WARP(w,x,y,z) is a fourth-order WARP. The special case of a first-order WARP will be defined below in [2.13]. Note that, no matter what the order of the composite WARP, the cardinality of the generic string is the cardinality of the specific string, and the size of the space in which the final scale string is embedded is the size of the space in which the specific string is embedded.

[2.11] For an example of a third-order WARP, let a = <112>, b = <2131>, c = <1122142>. Then

```
WARP(WARP(a,b),c)
        = WARP(WARP(<112>,<2131>),<1122142>)
        = WARP(<214>,<1122142>)
        = <229>
= WARP(a,WARP(b,c))
        = WARP(<112>,WARP(<2131>,<1122142>)
        = WARP(<112>,<2272>)
        = <229>
= WARP(a,b,c).
```

[2.12] Identity for WARP is defined as an appropriately sized string of unit intervals <1, . . . , 1>. Let I identify a generic (left identity) string and let J identify a scale (right identity) string such that #(I) = SUM(I) = m and #(J) = SUM(J) = n (with m <= n). For any string x such that #(x) = m and SUM(x) = n,

WARP(I,x) = WARP(x,J) = WARP(I,x,J) = x,

even though I does not equal J unless m = n. Note that I is Cm's interval string and J is Cn's interval string, i.e., for points e-in-Cm and f-in-Cn, Cm = (e,I) and Cn = (f,J).

[2.13] In noting the order of a composite WARP, we will ignore any (normally suppressed) identity strings. Thus if I and J are identities, WARP(I,a,b,J,c,d) = WARP(a,b,c,d) is a fourth-order WARP, not a sixth-order WARP. Any string can therefore be viewed as a first-order WARP, i.e., x = WARP(I,x) or WARP(x,J), depending on our perspective.

[2.14] WARPs are generally non-commutative, i.e., WARP(a,b) does not equal WARP(b,a) unless a = b = I = J. Furthermore, if a does not equal b, then at least one or the other of WARP(a,b) and WARP(b,a) is undefined since SUM(a) = #(b) and SUM(b) = #(a) cannot both be true.

[2.15] We may now extend the WARP function by collecting circular permutations of the generic and scale strings. This will result in the set of all WARP-related strings within a given scale string. But WARPSET is more than just a mere list since the generic and scale strings' structures are reflected in the relationships between the individual WARPs. This will become especially apparent when both generic and scale strings are symmetric.

[2.16] WARPSET FUNCTION — DEFINITION. If q and r are strings as in [2.5] (#(q) = k and #(r) = m),

```
S = WARPSET(q,r)
        = {WARP(q(u),r(v)) : u = 0, . . . , k-1; v = 0, . . . , m-1},
```

where q(u) and r(v) are the u-th and v-th circular permutation of the generic string q and the scale string r respectively. WARPSET thus generates a set S of specific strings [8] by "rotating" the generic and scale strings. Since varying both u and v results in duplications (as circular permutations), we will set u = 0 and only consider

```
S = WARPSET(q,r)
        = {WARP(q(0),r(v)) : v = 0, . . . , m-1}.
```

We will refer to this fixing of one generic string as the WARPSET convention.

[2.17] For an example of WARPSET, let q = q(0) = <124> and let r(0) = <1121322> be the zero-th permutation of r. This will yield the following set of WARPs S = {WARP(q(0),r(v)}:

v	r(v)	S
0	1121322	138
1	1213221	138
2	2132211	246
3	1322112	156
4	3221121	345
5	2211213	237
6	2112132	228

[2.18] As demonstrated by the example in [2.17] where WARP(q(0),r(0)) = WARP(q(0),r(1)), different permutations of the scale string will often result in the same ordered set of sums, so all of the specific strings in the WARPSET S are not necessarily unique. Nevertheless, since identical strings in S will combine with different pcs to form distinct structures in CHORDSET below in [2.22], WARPSET's cardinality will be defined as #(S) = #(r) = m. [9]

[2.19] If q' and r' are the inversions of the generic string q and the scale string r, respectively, then

WARPSET(q',r') = MIRROR(WARPSET(q,r));

that is, every string in WARPSET(q',r') is some inversion of a string in WARPSET(q,r). [10]

[2.20] When both the generic and scale strings are symmetric (i.e., when there is a circular permutation which, when read in reverse, reproduces the string), some of the most interesting relationships appear in the form of a symmetric patterning of the set of resultant specific strings. A detailed categorization and discussion of these internal symmetries is beyond the scope of the present study, so we will simply note here that symmetries among the set of specific strings resulting from a WARPSET are, at least in part a result of the degrees (and kinds) of symmetry possessed by the generic and scale strings. As an example, let q = q(0) = <1133> and let r(0) = <12311321> be the zero-th permutation of r. Both of these strings have one degree of inversional symmetry, but the order of the scale strings produced by the rotation of the scale string shows a curious sort of "skew-symmetric" pattern:

v	r(v)	S	Name
0	12311321	1256	a
1	23113211	2354	b
2	31132112	3164	c
3	11321123	1166	d
4	13211231	1346	c'
5	32112311	3245	b'
6	21123113	2165	a'

7	11231132	1166	d

Specific strings in S with the same letter name belong to the same string class in C14 (here the prime mark indicates the inversion of some rotation of the indicated string). So this WARPSET, whose cardinality is 8, has generated four distinct string classes, three of which are asymmetric.[11]

[2.21] While WARPSET can tell us a great deal about the relationsips between WARP-generated strings, it still doesn't fix the precise positions of these strings with respect to one another. This is necessary to adequately describe and examine "progressions" (chord pairs) within a given WARP-system. The general description "from string x to string y" conveys little information since, although the "shapes" or "character" of the termini may have been stipulated, their locations have not. The difference is similar to that between "from a major triad to a minor triad" and "from a C-major triad to a C♯-minor triad." By fixing chord positions, CHORDSET makes it possible to measure distances (i.e., to calculate and compare spanning vectors) as well as to identify points of incidence (common tones) and "voice- leading" subsets between WARP-related sets.

[2.22] CHORDSET FUNCTION — DEFINITION. Let q(0) be a generic string in Cm and r(v) be the v-th circular permutation of the scale string r = <r1,r2, . . . , rk> in Cn (0 <= v <= k-1); and let s(v) = WARP(q(0),r(v)). The chord structure (pv,s(v)) is a subset of the scale structure (0,r(0)) = {p0,p1,p2, . . . , pk-1} where p0=0; p1=r1; p2=r1+r2; . . . ; pk-1=r1+r2+ . . . +(rk-1).

CHORDSET(q,r) = {(pv,s(v)) : v=0, . . . , k-1}

is then the set of k *distinct* subsets of (0,r(0)) generated by WARPSET(q,r).

[2.23] CHORDSET concretizes a scale structure and procedes to assign one, and only one, specific interval string to each of its (pc) elements. Since the WARPSET convention holds the generic string constant, the result is a set of distinct WARP-related chords, each one of which is "built" on a "scale degree."

[2.24] Continuing the WARPSET example in [2.20], let q = q(0) = <1133> and let r(0) = <12311321> be the zero-th permutation of r. Given the referential scale (0,r(0)) = {0,1,3,6,7,8,11,13}, CHORDSET yields the following WARP-related structures (pv,s(v)) corresponding to the set of specific strings s(v):

v	**r(v)**	**s(v)**	**pv**	**(pv,s(v))**	**Name**
0	12311321	1256	0	{0,1,3,8}	0a
1	23113211	2354	1	{1,3,6,11}	1b
2	31132112	3164	3	{3,6,7,13}	3c
3	11321123	1166	6	{6,7,8,0}	6d
4	13211231	1346	7	{7,8,11,1}	7c'
5	32112311	3245	8	{8,11,13,3}	8b'
6	21123113	2165	11	{11,13,0,6}	11a'
7	11231132	1166	13	{13,0,1,7}	13d

[2.25] We may now formally define a white note system (or WARP-generated system) as the ordered triple (A,B,C), where A is a set of WARPSET-generated (specific) strings, B is a scale used as a basis for CHORDSET, and C is the corresponding set of chords generated from A and B via CHORDSET. So in the example just given, A = {s(v)}, B = {pv}, C = {(pv,s(v))} for all values of v.

3. Vector Transformations.

[3.1] Thus far we have developed a family of WARP functions which methodically generate white note systems. We are now in a position to study transformations between those systems that are somewhat akin to coordinate transformations—at least this will be seen as the underlying metaphor. After introducing the interval multiplicity matrix and relating it to WARP, we will apply it to a brief study of interval and interval-class vector transformations.[12]

[3.2] INTERVAL MULTIPLICITY MATRICES. Given any string r in Cs (s = SUM(r)), let r;x,y be a non-reduced set of substrings of r where x is the number of elements in the substring and y is the sum of the elements in the substring, with 1 <= x <= #(r) and 1 <= y <= SUM(r).[13] We then define the int-multiplicity matrix (or simply int-matrix)

MINT(r) = [exy]

for WARP(I,r) = r, where entry exy = #(r;x,y).

[3.3] For example, if r = <124113>, then r;1,4 = {/4/}, r;2,2 = {/11/}, r;2,4 = {/13/,/31/}, r;3,6 = {/312/,/411/}, r;4,6 = {/1131/}, r;1,1 = {/1/,/1/,/1/}, r;2,8 = {}, etc. MINT(r) then counts the number of elements in each set of substrings and displays them thus:

x \ y	1	2	3	4	5	6	7	8	9	10	11	12	U(I)
1	3	1	1	1	0	0	0	0	0	0	0	0	6
2	0	1	1	2	1	1	0	0	0	0	0	0	6
3	0	0	0	0	2	2	2	0	0	0	0	0	6
4	0	0	0	0	0	1	1	2	1	1	0	0	6
5	0	0	0	0	0	0	0	1	1	1	3	0	6
6	0	0	0	0	0	0	0	0	0	0	0	6	6
U(r)	3	2	2	3	3	4	3	3	2	2	3	6	

The right column labelled U(I) and the bottom row labelled U(r) are not properly part of the matrix. Their presence here will be explained shortly. The left column (x) and the top row (y) are the exy indices and will normally be omitted unless necessary for clarity—but they will often be referred to.

[3.4] NB: We will assume that if x = #(r) and y = SUM(r) then exy = #(r); so in the example #(r;6,12) = 6, not 1. In effect, this matrix element counts the circular permutations of r.

[3.5] We will need to distinguish below (e.g. in [3.21]) between *all* the elements of a row (or column) and only the non-zero elements of that row (or column). Thus the ordered tuple of non-zero elements of row x (read consecutively from left to right) will be referenced as ROWx; but if we are refering to *all* the elements in row x we will write rowx. Similarly, the non-zero elements of column y (read from top to bottom) will be referenced as COLy; and coly will refer to *all* of column y's elements. In the example in [3.3], row5 = [0,0,0,0,0,0,0,1,1,1,3,0], ROW5 = [1,1,1,3], col7 = [0,0,2,1,0,0], and COL7 = [2,1].

[3.6] Any int-matrix has its own two-fold inversional symmetry, i.e., ignoring the #(r)-row and the SUM(r)-column, every row (and ROW) has an inversion mod #(r) and every col (and COL) has an inversion mod SUM(r). Thus row5 in the example in [3.3] is the inversion of row1, and col7 is the inversion of col5.

[3.7] The following propositions relate MINT(r) to WARP(I,r).

(1) The sum of (non-zero) entries in ROWx of MINT(r) is the multiplicity of interval x in string I which, by the definition of WARP, must be the same as the number of elements in string r; i.e., for any x,

SUM(ROWx) = #intx(I) = #(r).

(2) The number of entries in rowx of MINT(r) (including zeroes) is the sum of the elements in string r; i.e., for any x,

#(rowx) = SUM(r).

(3) The sum of (non-zero) entries in COLy of MINT(r) is the multiplicity of interval y in string r; i.e., for any y,

SUM(COLy) = #inty(r).

(4) The number of entries in coly of MINT(r) (including zeroes) is the sum of the elements in string I which, by the definition of an identity string, must be the same as the number of elements in I; i.e., for any y,

#(coly) = SUM(I) = #(I).

[3.8] For I = <111111> and r = <124113>, if x = 4, say, then

(1) SUM(ROW4) = #int4(I) = #(r) = 6, and
(2) #(row4) = SUM(r) = 12.

If we set y = 7, then

(3) SUM(COL7) = #int7(r) = 3, and
(4) #(col7) = SUM(I) = #(I) = 6.

[3.9] INTERVAL VECTORS. From (1) and (3) in [3.7] we define two interval-vectors derived from MINT(r) and hence associated with WARP(I,r).

(1) Let g = SUM(I). The interval vector U(I) is the ordered g-tuple [a1,a2, . . . , ag] comprised of the sums of the rows of MINT(r) such that, for any x, vector component ax = #intx(I) = SUM(ROWx).

(2) Let h = SUM(r). The interval vector U(r) is the ordered h-tuple [b1,b2, . . . , bh] comprised of the sums of the columns of MINT(r) such that, for any y, vector component by = #inty(r) = SUM(COLy).

[3.10] Again using r = <124113> and referring to the display of MINT(r) above in [3.3], U(I) = [666666] (displayed in the right column) and U(r) = [322334332236] (displayed in the bottom row).[14]

[3.11] Now suppose we are given two transposition-related chords S = (p,r) and T = (p+y,r), where p is a pitch class in Cs and 1 <= y <= (s = #(r)). The multiplicity of any interval z connecting, or spanning, S and T is given by

#intz(S,T) = SUM(COLy'),

where y' = y-z.

[3.12] If z = 0, [3.11] reduces to an int-matrix version of the common-tone theorem for transposition from atonal theory; i.e., if T is the y-transpose of S, the number of common tones shared by S and T is #int0(S,T) = SUM(COLy).

[3.13] Keeping r = <124113>, if p = 7 and y = 8, S = (7,r) = {7,8,10,2,3,4} and T = (7+8,r) = (3,r) = {3,4,6,10,11,0}. S and T share three tones (pc3, pc4, pc10), or what is the same, there are three pcs in S that are int0 from three pcs in T. Checking MINT(r), we see that SUM(COL8) is 3, as is, of course, the value of element b8 in U(r).

[3.14] But [3.11] says more than this. If we wish to know the multiplicity of int1 spanning S and T, we can set z = 1 so that y' = y-z = 7. Then we may read #int1(S,T) = SUM(COL7) = 3 from MINT(r) or simply read the value for b7 in U(r). This suggests collecting these multiplicities of spanning intervals into a spanning vector which can be used to describe pairs of structures.

[3.15] Given the sets S and T under the conditions described in [3.11], the vector U(S,T) is the ordered h-tuple [c1,c2, . . . , ch] where h = SUM(r). Any vector entry cz = #intz(S,T) = SUM(COLy').

[3.16] To remain consistent with the notation for U(r) and the most natural display of MINT(r), we will stipulate that 1 <= z <= SUM(r). So if T is the y-transpose of S, the vector comprised of intervals spanning S and T can be read from MINT(r) by going to column y and listing column sums to the left (counter-clockwise) beginning with column y-1. (15)

[3.17] For r = <124113> again, let S = {7,8,10,2,3,4} and T = {3,4,6,10,11,0} as in [3.13]. T is the transposition of S at y = 8, so we refer to column 8 of MINT(r) (or component b8 of U(r)) and read the column sums (vector component values) leftward beginning with coly-1 = col7; so U(S,T) = [343322363223]. This 12-tuple tells us that, between S and T, #int1 = 3, #int2 = 4, . . . , #int11 = 2, and #int12 = #int0 = 3. Also note that for y = 0, U(S,T) = U(S,S) = U(r) as we would expect.

[3.18] One interesting feature of int-matrices is how they reflect certain string patterns. Let us define a clone string of order n as any interval string c consisting only of n consecutive identical substrings. We recognize n as the number of degrees of transpositional (rotational) symmetry (DTS) from atonal set theory. Thus c = <132> in C6 can be considered as a clone string of order 1 (DTS = 1). c can then generate the second order clone string c' = <132132> in C12 (DTS = 2), the third order clone string c" = <132132132> in C18 (DTS = 3), and so on. The clone pattern then appears in the int-matrix of the clone string. For c = <132>,

```
                 /  1 1 1 0 0 0  \
     MINT(c)  =  |  0 0 1 1 1 0  |
                 \  0 0 0 0 0 3  /
```

for c = <132132>,

```
                  /  2 2 2 0 0 0 0 0 0 0 0 0  \
                  |  0 0 2 2 2 0 0 0 0 0 0 0  |
     MINT(c')  =  |  0 0 0 0 0 6 0 0 0 0 0 0  |
                  |  0 0 0 0 0 0 2 2 2 0 0 0  |
                  |  0 0 0 0 0 0 0 0 2 2 2 0  |
                  \  0 0 0 0 0 0 0 0 0 0 0 6  /
```

and, generally, for the n-th order clone string c = <132 . . . 132>, the int-matrix will be of the form

```
/  n   n   n   0   0   0                       \
|  0   0   n   n   n   0                       |
|  0   0   0   0   0  3n                       |
|                          .                   |
|                           .                  |
|                            .                 |
|                            n   n   n   0   0   0   |
|                            0   0   n   n   n   0   |
\                            0   0   0   0   0  3n   /
```

In other words, the int-matrix of the original string is multiplied by n and repeated n times down the diagonal.

[3.19] Although the clone feature will not be dwelled upon in this study, when we come to the section on efficient linear transformations we will see how reducing a clone string can simplify the treatment of seemingly complex structures. For instance, that the string <12> supports efficient linear transformations will be seen to imply that any string <1212 . . . 12> is saturated with these transformations as well.

[3.20] INTERVAL VECTOR TRANSFORMATIONS. For any generic string q, the int-matrix of the scale string r can be interpreted as a vector transformation matrix for WARP(q,r) = s. In other words, as q maps to s via an r-WARP, U(q) maps to U(s) via MINT(r). This is possible because I and q (which can contain no interval other than those in I) both belong to the same referential space, say, Ca. So a generic interval in q (represented by one of the matrix's index values x) maps into r's space Cb to some specific interval in s (represented by one of the index values y corresponding to a non-zero matrix entry). [16]This situation can be summarized in our present symbolism as

x-in-Ca --> {y-in-Cb : exy-in-MINT(r) > 0}.

Thus for r = <124113> we can construct the interval map:

```
x-in-C6 --> y-in-C12
   1        1,2,3,4
   2        2,3,4,5,6
   3        5,6,7
   4        6,7,8,9,10
   5        8,9,10,11
   6        12
```

If the generic int1 is present in q-in-C6, for example, it will map specifically to either int1, int2, int3, or int4 in s-in-C12.

[3.21] When COLy has more than one element we have the situation labelled an "ambiguity" by Jay Rahn[17] (e.g., in MINT(r) for r = <124113>, y = 3 has two possible x-values, i.e., COL3 = [1,1]) so #COL3 = 2. In presenting vector transformations (and especially later when we investigate efficient linear transformations in Part II) it will be important to distinguish between partitioned matrices (ones free of ambiguities[18]) and non-partitioned matrices.

[3.22] If the number of (non-zero) elements in COLy is less than 2 for every value of y, then we will say that MINT(r) is partitioned. This in turn identifies a partition of U(s) in such a way that any component in U(q) equals (maps to) the sum of some discrete combination of components in U(s).

[3.23] An example of a partition is the following based on r = <3322332> whose int-matrix

```
/  0 3 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  \
|  0 0 0 1 4 2 0 0 0 0 0 0 0 0 0 0 0 0  |
|  0 0 0 0 0 0 2 5 0 0 0 0 0 0 0 0 0 0  |
|  0 0 0 0 0 0 0 0 0 5 2 0 0 0 0 0 0 0  |
|  0 0 0 0 0 0 0 0 0 0 0 2 4 1 0 0 0 0  |
|  0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 3 0 0  |
\  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7  /
```

yields the interval map

```
x-in-C7 --> y-in-C18
   1        2,3
   2        4,5,6
   3        7,8
```

```
4         10,11
5         12,13,14
6         15,16
7         18.
```

Since MINT(r) is partitioned in this example, U(r) is also partitioned as [@;ROW1;ROW2;ROW3;@;ROW4;ROW5;ROW6;@;ROW7] = [@;34;142;25;@;52;241;43;@;7], where “@” indicates a “dead” zero—a component from an empty column which can’t take part in any r-based transformation.

[3.24] We can then define a partition-sum vector by summing all the ROWs and dropping the dead zeroes:

U'(r) = [SUM(ROW1),SUM(ROW2), . . .].

Thus in the current example, U'(r) = [3+4,1+4+2,2+5,5+2,2+4+1,4+3,7] = [7777777] = U(I).

[3.25] But more interesting is that, whenever we are using a scale string r that has a partitioned int-matrix, the vector transformation U(I) --> U(r) from the identity U'(r) = U(I) for WARP(I,r) = r, implies the “bundle” of transformations U(q) --> U(s) from U'(s) = U(q) for WARP(q,r) = s.

[3.26] For example, given q = <2131> and r = <3322332> so that WARP(q,r) = s = <6282>, we can calculate separately that U(q) = [2133124] and U(s) = [020001030301000204]. U(s) is partitioned after U(r) as [@;20;001;03;@;30;100;02;@;4] and U'(s) = [2+0,0+0+1,0+3,3+0,1+0+0,0+2,4] = U(q).

[3.27] This rather cumbersome notation will now be simplified considerably by the use of interval class notation. But we will of necessity return to the int-matrix display when we investigate voice-leading generally and efficient linear transformations specifically in Part II.

[3.28] IC-MATRICES. Let us now reduce all values of x and y in r;x,y to their interval class values. Let j,k be integers such that 1 <= j <= [[#(r)/2]] and 1 <= k <= [[SUM(r)/2]]. We may then define the ic-multiplicity matrix (or simply ic-matrix)

MIC(r) = [fjk]

for WARP(I,r) = r, where entry fjk = #(r;j,k) except when #(r) is even and j = #(r)/2, in which case fjk = #(r;j,k)/2.

[3.29] For r = <124113>, MIC(r) can then be displayed:

```
    k| 1  2  3  4  5  6 |
j    |                  | V(I)
----|------------------|-----
1    | 3  1  1  1  0  0 |  6
     |                  |
2    | 0  1  1  2  1  1 |  6
     |                  |
3    | 0  0  0  0  2  1 |  3
----|------------------|-----
     |                  |
V(r)| 3  2  2  3  3  2 |
     |                  |
```

Note that, since both I and r belong to even spaces in the example, element f36 in MIC(r) is half of e36 in MINT(r).

[3.30] IC-VECTORS. The sums of the columns of MIC(r) form the (generalized) Forte ic-vector [19] of the specific string r which we will call V(r). So for r = <124113> we can read V(r)= [322332]. Likewise, the sums of the rows of MIC(r) form the ic-vector of the identity generic string I. Here I = <111111> so V(I) = [663].

[3.31] As above, we can use the ic-matrix to map interval classes through WARPs. Continuing with the example in [3.29], MIC(r) yields the ic-map

```
j-in-C6 --> k-in-C12
   1        1,2,3,4
   2        2,3,4,5,6
   3        5,6
```

[3.32] If MINT(r) is partitioned, then MIC(r) will be considered partitioned as well, and we can refer to ROWj and COLk of MIC(r) and create a partition-sum vector V' without complications. If r = <3322332> (see [3.23]), MIC(r) is partitioned:

```
/  0 3 4 0 0 0 0 0 0  \
|  0 0 0 1 4 2 0 0 0  |
\  0 0 0 0 0 0 2 5 0  /
```

So the ic-map

```
j-in-C7 --> k-in-C18
   1         2,3
   2         4,5,6
   3         7,8
```

can be used to relate V(q) and V(s) in WARP(q,r) = s. If q = <223> and r = <3322332>, then s = <648>. V(s) = [000101010] can be partitioned as [@;00;101;01;@] and it is easily verified that V'(s) = [0+0,1+0+1,0+1] = [021] = V(q). Changing the generic string to q = <313> and keeping r = <3322332>, WARP(q,r) = s = <828>; then V(s) = [010000020] so that, after partitioning V(s) patterned on the partition of V(r), we have V'(s) = [1+0,0+0+0,0+2] = [102] = V(q).

4. Transformation Covariants.

[4.1] GENERIC COVARIANCE (G-COV). The partition-sum vector V' (as well as U') represents a covariance with respect to the generic string; that is, holding the scale string constant, if we vary the generic string, the relation between V(q) and V(s) based on the partition pattern of MIC(r) will vary as well, but in a predictable way (see the example in [3.32]). The partition pattern is *invariant*, making the vector relationships *covariant*. For reference, we will label this phenomenon generic covariance (G-COV).

[4.2] We now procede to describe two other covariance relations. G-COV resulted from holding the scale string constant. We will now see what happens when the generic string is held constant.

[4.3] WARPSET generates a one-to-one correspondence between the circular permutations of the scale string and the specific strings (see [2.17]). CHORDSET adds corresponding scale-steps and chords based on those scale-steps (see [2.24]). Building on that basic set of correspondences, we now show that two fundamental relationships, scale covariance and cover covariance, hold between any WARP- generated (white note) systems sharing the same scale string. One immediately interesting consequence of scale covariance will be the transport of the “common note theorem” for transposition from atonal theory to tonal theory. Cover covariance will be important in exploring and extending Riemann systems in Part II.

[4.4] CORRESPONDENCE. Let X = WARPSET(q,J) in Ck where J = <1, . . . , 1> and #(J) = SUM(q) = k. Obviously X consists of k strings, all of which are the string q. Nevertheless, CHORDSET(q,J) = Q based on the (chromatic) scale E = (0,J) = {0,1,2, . . . , k-1} contains k distinct subsets, Q = {(0,q),(1,q), . . . , (k-1,q)}. As defined in [2.25], (X,E,Q) is a white note system. Now consider Y = WARPSET(q,r) = {s0,s1,s2, . . . , s(k- 1)} in Ck' (k' > k) also consisting of k strings. Based on the scale F = (0,r(0)) = {0,p1,p2, . . . , p(k-1)}, Y generates CHORDSET(q,r) = R = {(0,s0),(p1,s1), . . . , (p(k-1),s(k-1))}. (Y,F,R) is also a white note system. Placing the defining scales E and F in (cyclical) one-to-one correspondence: 0(=)0, 1(=)p1, 2(=)p2, etc., *also* places the member sets of Q and R in one-to-one correspondence: (0,q)(=)(0,s0), (1,q)(=)(p1,s1), etc. The symbol "(=)" will hereafter be used to indicate "correspondence" in the above sense between any parallel lists of elements or strings or sets arranged in some (circular) sequence and associated with a WARPSET or CHORDSET.

[4.5] As an example, consider q = <2131> and r = <3322332>. Each column of the following table lists sequentially the elements of one of the sets just discussed; each row is a list of corresponding elements between sets X, E, Q, Y, F, R. (Each string and structure has been named for reference in examples below.)

X	E	Q	Y	F	R
2131=q	0	{0,2,3,6}=0q	6282=s	0	{0,6,8,16} = 0s
2131=q	1	{1,3,4,0}=1q	5283=t	3	{3,8,10,0} = 3t
2131=q	2	{2,4,5,1}=2q	4383=u	6	{6,10,13,3}= 6u
2131=q	3	{3,5,6,2}=3q	5382=t'	8	{8,13,16,6}= 8t'
2131=q	4	{4,6,0,3}=4q	6282=s	10	{10,16,0,8}=10s
2131=q	5	{5,0,1,4}=5q	5373=v	13	{13,0,3,10}=13v
2131=q	6	{6,1,2,5}=6q	5373=v	16	{16,3,6,13}=16v

We will write (X,E,Q) (=) (Y,F,R) for an arrangement such as the one just written.

[4.6] Given SYS1 = (X,E,Q) in Ck and SYS2 = (Y,F,R) in Ck' as described in [4.4], the theorems that follow are easily derived by correspondence. (NB: We will continue to assume that all scale strings have a partitioned ic-matrix. Hence we are giving a weak form of the theorems here. Strong covariance theorems would not require partitioned scale strings and would unnecessarily complicate the present study.)

[4.7] SCALE COVARIANCE (S-COV). If A,B are chords in Q, then there exist chords C,D in R such that A(=)C, B(=)D, and

#intx(A,B) = SUM(#inty(C,D))

where x is an interval in Ck and y is the set of all possible images of x in Ck' under WARP(q,r) (i.e., the y-indices corresponding to the elements of ROWx in MINT(r)).

[4.8] We are given any two chords A,B in Q, say (from [4.5]), A = 0q = {0,2,3,6} and B = 5q = {5,0,1,4}. Setting x = 2 as an example, we know that #intx(A,B) = 3. The table in [4.5] indicates that C = 0s = {0,6,8,16} and D = 13v = {13,0,3,10} correspond to A and B, respectively. From the interval map associated with r = <3322332> (see [3.23]), int2 in C7 maps to int4, int5, or int6 in C18, which [4.7] tells us to collect and sum. So SUM(#inty(C,D)) = #int4(C,D) + #int5(C,D) + #int6 = 1+2+0 = 3 = #intx(A,B). If SYS2 = (Y,F,R) is rotated so that C' = 8t' = {8,13,16,6} corresponds to A and D' = 3t = {3,8,10,0} corresponds to B, then (despite the rotation) once again SUM(#inty(C',D')) = 1+2+0 = 3.

[4.9] Now suppose that we keep the generic string q = <2131> but use a different (compatible) scale string, say, r' = <3232323> (note that MINT(r') is also partitioned). We then write out the sets Y', F', R' as we did for SYS1 and SYS2 to reveal SYS3 = (Y',F',R'). Again setting x = 2 and selecting A and B as in [4.8], we now find that, in SYS3, C" = {0,5,8,15) (=) A and D" = {13,0,3,10} (=) B. So SUM(#inty(C",D") = 0+2+1 = 3 = #intx(A,B) once again. In general, we note that, since SYS1 (=) SYS2 (=) SYS3, S-COV holds between SYS2 and SYS3 whose (non-chromatic) scales are superficially unrelated. This example is a demonstration of the following generalization and leads into the S-COV corollaries.

[4.10] S-COV holds for any two WARP-generated systems sharing the same generic string.

[4.11] Defining x and y as interval classes in Ck and Ck', respectively, [4.7] can also be stated

#icx(A,B) = SUM(#icy(C,D))

when A(=)C and B(=)D.

[4.12] If we collect the icx's in Ck and the icy's in Ck' and display their multiplicities as ic-vectors, we may also write

V(A,B) = V'(C,D)

when A(=)C and B(=)D.

[4.13] As an example for [4.12], given SYS1 and SYS2 as before, if we select chords 3t = {3,8,10,0} and 13v = {13,0,3,10} in R, then V(3t,13v) = [013030240]. For corresponding chords 1q = {1,3,4,0} and 5q = {5,0,1,4} in Q, [4.12] predicts that V'(3t,13v) = [1+3,0+3+0,2+4] = [436] = V(1q,5q) which the reader may verify by referring back to MIC(r) and the ic-map in [3.32].

[4.14] The special case of [4.7] (for both int- and ic- versions of S-COV) where x = y = 0 allows us to extend the common tone theorem for transposition (CTT) [(20)] from atonal to tonal theory, so #int0(A,B) = #int0(C,D) or #ic0(A,B) = #ic0(C,D) when A(=)C and B(=)D.

[4.15] Working through an example again, let us begin with the two chords A = 1q = {1,3,4,0} and B = 4q = {4,6,0,3} in Q and note that they have 3 elements (tones) in common. This can be related to the ic-vector of q, V(q) = [213]. Since Q = CHORDSET(q,J) is a "chromatic" or truly "atonal" system (there being no privileged string or scale-step since they are all the same), we may apply the basic (atonal) CTT and read V(q) as a common tone multipicity pattern. Thus, since A and B are 3 (chromatic) steps apart, we go to the third component in V(q) = [213] and see that any two chords with the ic- vector [213] which are related by T3 (or the mod 7 equivalent T4) have 3 tones in common. But what is now posited by [4.14] is that when q is used as a generic with any scale string at all, V(q) is retained as the common tone pattern for corresponding chords. Here, since C = 3t = {3,8,10,0} (=) A and D = 10s = {10,16,0,8} (=) B, we know that C and D in R must also have 3 elements in common. [(21)]

[4.16] COVERS. An additional covariance may be observed when the generic string is held constant, but first we need the following definition. If (A,B,C) is a white note system and X is a subset of chords in C such that every element in B belongs to at least one chord in X, then X is said to cover the scale B, written: X//B. [(22)]

[4.17] COVER COVARIANCE (C-COV). Referring again to SYS1 = (X,E,Q) and SYS2 = (Y,F,R) as defined in [4.4], let K be a subset of chords in Q and let L be a subset of chords in R such that K(=)L. Then K covers E if, and only if, L covers F:

K//E <--> L//f

[4.18] In giving an example of a cover we can also illustrate the idea of a minimal cover (the least number of chords that can (completely) cover a scale). [(23)] In Q (SYS1), any two chords related by T2 (or T5) have only one element in common, therefore they must form a (minimal) cover of E. Selecting 5q and 0q we note that this is indeed the case: K = 5q.&.0q = {0,1,2,3,4,5,6} = E. C-COV then says that since 13v(=)5q and 0s(=)0q, L = 14w.&.0s = {0,3,6,9,11,14,17} = F.

[4.19] Obviously any chordset covers its scale (e.g., Q//E and R//F), but this does not necessarily form a minimal cover. And, using the current example, if any *two* tetrachords in either Q or R have more than one tone in common, they cannot cover their respective scales.

* * *

Part I has concentrated almost solely on what we might label an "abstract white note system" theory with the aim of collecting, generalizing and extending certain related developments in the scale and pc-set theory literature within the recent past. The examples we have used have thus been aimed at illustrating the principles immediately involved, and we have purposely avoided any directed "application."

In Part II this approach will change considerably. After briefly translating John Clough and Jack Douthett's "maximally even" sets into string notation, we will begin by framing the "usual diatonic" as a model WARP system. The result will then be generalized to other (hyper)diatonic systems, beginning with a WARP system in quarter-tone space. Utilizing cover covariance, we will then examine David Lewin's "Riemann systems" and generalize these to the hyperdiatonic systems we have developed. Utilizing int-matrices we will also expand Eytan Agmon's work on "efficient linear transformations" as it applies to these systems. Together, the Lewin and Agmon generalizations will show that our hyperdiatonic systems display many of the deep attributes we have come to associate with the historic diatonic.

Finally, based on our findings in both parts, we will briefly present a (hypothetical) fourth-order hyperdiatonic system which displays some unusual (but potentially useful) musical attributes. Thus Part II, and the article, will end with one possible response to the challenge implied by John Clough and Gerald Myerson when they wrote, "We believe that our theory also has considerable potential for application in the composition of microtonal music."

Stephen Soderberg
Library of Congress
Music Division
101 Independence Avenue SE
Washington, DC 20540
ssod@loc.gov

Footnotes

1. "White note system" and "tonal system" may, with caution, be used interchangeably here. The former term has the advantage of dissociating the present study from many unnecessary historical (but not necessarily incorrect) implications loaded into the word "tonal."
Return to text

2. Our use of the word "topological" here and elsewhere should not be confused with its usual meaning in mathematics, but the metaphorical connection between the two meanings should not be dismissed. The intuitive (and somewhat naive) sense often associated with "topology" is "rubber sheet geometry." Here we wish to transfer that intuition to musical systems as "rubber band geometry." For the reader who wishes a clearer distinction, perhaps a formal definition of "musical topology" might read, "the study of the properties of musical structures that remain invariant under certain transformations." This now shifts the emphasis to "certain transformations," which is precisely the object of the present study. At any rate we will periodically remind the reader of our special sense of the term topological by the use of quotation marks or, more simply, by the abbreviation TT.
Return to text

3. #intj(S,T) is identical to David Lewin's interval function IFUNC which he first posited in "Re: Intervallic Relations Between Two Collections of Notes," *Journal of Music Theory* 3.2 (November 1959): 298–301.
Return to text

4. While we are not attempting to "interpret" these basic relationships, to understand the distinctions we are making here and throughout, it may be helpful to review the last section ("Philosophical musings") in David Lewin's classic article "Forte's Interval Vector, My Interval Function, and Regener's Common-Note Function," *Journal of Music Theory* 21:2 (fall 1977) 227ff.
Return to text

5. The idea of characterizing a pc set by listing its contiguous intervals when the set is in "close" position has a history that stretches back at least 80 years. The better known references to this practice are Richard Chrisman, "Describing Structural Aspects of Pitch-Sets Using Successive-Interval Arrays," *Journal of Music Theory* 21:1 (spring 1977) and Eric Regener, "On Allen Forte's Theory of Chords," *Perspectives of New Music* 13:1 (1974). Also, Robert Morris uses "interval succession" in

Composition With Pitch-Classes: A Theory of Compositional Design (New Haven: Yale University Press, 1987). In *The Harmonic Materials of Twentieth-Century Music* (New York: Appleton- Century-Crofts, 1960) Howard Hanson employs "intervallic order" throughout to accompany his version of ic vectors. But the most fascinating use (pointed out to the author by Richard Cohn in private correspondence) can be found in Ernst Lecher Bacon's "Our Musical Idiom" first published in *The Monist* 27:1 (October 1917). Not only does Bacon employ an intervallic string notation, he uses it to define a chord's normal order and gives what appears to be a complete list of set classes mod 12.
Return to text

6. David Lewin, *Generalized Musical Intervals and Transformations* (New Haven: Yale University Press, 1987): 175ff.
Return to text

7. John Clough and Gerald Myerson, "Variety and Multiplicity in Diatonic Systems," *Journal of Music Theory* 29.2 (fall 1985).
Return to text

8. Clough and Myerson (op cit.) refer to this set of specific strings as "species."
Return to text

9. In Clough and Myerson (op cit.) this equality appears as "cardinality equals variety."
Return to text

10. The reader is invited to work a few examples to help clarify this concept. Suggestion 1: let q = <1123>, q' = <3211>, r = r' = <1211213>. Suggestion 2: let q = q' = <1222>, r = <1112132>, r' = <2312111>. Suggestion 3: let q = 1123, q' = <3211>, r = <1112132>, r' = <2312111>. Note especially the effect produced by "symmetric" strings as preparation for the comments in [2.20].
Return to text

11. To give some indication of what might lie ahead in a general study of these patterns, consider a seemingly innocent change in this example's generic string from <1133> to q = <1313>. The generic string now has two degrees of transpositional symmetry and two degrees of inversional symmetry. If a = <1616> and b = some circular permutation of <2534>, the pattern of specific strings is:

```
a-b-b'-a-a-b-b'-a.
```

If we then take rotations of b into account and set b(0) = <2534>, b(1) = <5342>, etc., we then get

```
a-b(0)-b(3)'-a-a-b(2)-b(1)'-a.
```

Furthermore, if we rotate the generic to q = <3131>, we pick up all the other b and b' string rotations, forming a kind of "complementary" WARPSET:

```
a*-a*-b(1)-b(2)'-a*-a*-b(3)-b(0)',
```

(where a* = <6161>). Together, the two final patterns include all possible a and b string forms.
Return to text

12. We assume the reader has some knowledge of "interval vectors" and related concepts which, over the past 30 years have made appearances in a wide variety of forms and contexts. To trace the history of this concept would take us too far afield in the present study. But the most important connections, especially relating "multiplicity" and "common tones" are brought out by David Lewin in "Forte's Interval Vector, My Interval Function, and Regener's Common-Note Function."
Return to text

13. There is a convergence of ideas here that is worth noting. First, John Clough and Jack Douthett ("Maximally Even Sets," *Journal of Music Theory* 35.1–2: 118) define the multiplicity function DFUNC(X,k,I) as "the numbers of intervals of [chromatic length] k and [diatonic length] I in the set X." If the interval string of X is r, then DFUNC is clearly identical to our

"substring counting function" since DFUNC(X,k,I) = #(r;I,k) in all cases. But for the present study we find it useful to allow all values of #(r;I,k) including zeroes so as to arrange these values in matrix form. Second, Robert Morris (op cit., p.40) defines a cyclic interval succession function CINT which is related to our "substring listing function" r;x,y. An example should make this relationship clear. If X = {0,3,5,9} in C12, say, for m=2 CINTm(X) = (5,6,-5,6) = (a,b,c,d). X's interval string is <3243>, so CINTm(X) gives us the y-values and x = m for r;x,y:

```
{/32/} = r;2,5 = r;m,a
{/24/,/33/} = r;2,6 = r;m,b = r;m,d
{/43/} = r;2,-5 = r;2,7 = r;m,c
```

Of course Morris takes CINT in a different direction than ours leading, in one application, to Stravinsky's "rotational arrays" (op cit., p.108). We mention this here since some readers may want to remember this connection for future use. Further investigation and generalizations of CINT (and other "atonal" concepts presented by Morris and others) in connection with "topological transformations" may prove fruitful in a future study of "hyper-atonal" systems embedded in microtonal spaces.
Return to text

14. At first this may seem like a circuitous path to the obvious, but we will find increasingly that the matrix form for displaying interval multiplicities reveals more than the (summary) vector form. In Part II, int-matrices will be indispensible in understanding efficient linear transformation saturation.
Return to text

15. Just as the interval string is a circle, the corresponding interval multiplicity matrix is a torus. So when the "left edge" of the matrix is reached, the reading is continued at the "right edge."
Return to text

16. The set of y's corresponding to non-zero MINT entries for a given x is Clough and Myerson's "spectrum" of x (op cit.).
Return to text

17. Jay Rahn, "Coordination of Interval Sizes in Seven-Tone Collections," *Journal of Music Theory* 35:1–2 (spring–fall 1991).
Return to text

18. Freedom from ambiguity is what Eytan Agmon terms "coherence." See for example, "Coherent Tone Systems: a Study in the Theory of Diatonicism," *Journal of Music Theory* 40.1 (spring 1996).
Return to text

19. Allen Forte, *The Structure of Atonal Music* (New Haven: Yale University Press, 1973).
Return to text

20. A concise statement of this principle can be found in John Rahn, *Basic Atonal Theory* (New York: Schirmer Books, 1980): 107ff.
Return to text

21. Keep in mind that we are working with ic-vectors at this point. q = <2131> is a string in C7, an odd-sized space, so V(q) = [213] accurately reflects the common tone pattern; but if q was from an even-sized space, the last component of V(q) must be doubled to read it as a common-tone pattern.
Return to text

22. There is some resonance here with Richard Cohn's concept of "transpositional combination" whereby pc sets are constructed by combining two or more transpositionally related subsets ("Transpositional Combination in Twentieth Century Music," Ph.D. Dissertation, University of Rochester, 1987). In Part II it will be noted that a "Riemann cover" is in effect one possible generalization of Cohn's idea to microtonal spaces.
Return to text

23. G. Mazzola has developed the idea of "partial" covers, what he terms "minimal cadence sets," an interesting special case of the concept of covering a scale. His work is described in Daniel Muzzulini, "Musical Modulation by Symmetries," *Journal of Music Theory* 39.2 (fall 1995).
Return to text

Copyright Statement

Copyright © 1998 by the Society for Music Theory. All rights reserved.

[1] Copyrights for individual items published in *Music Theory Online* (*MTO*) are held by their authors. Items appearing in *MTO* may be saved and stored in electronic or paper form, and may be shared among individuals for purposes of scholarly research or discussion, but may *not* be republished in any form, electronic or print, without prior, written permission from the author(s), and advance notification of the editors of *MTO.*

[2] Any redistributed form of items published in *MTO* must include the following information in a form appropriate to the medium in which the items are to appear:

> This item appeared in *Music Theory Online* in [VOLUME #, ISSUE #] on [DAY/MONTH/YEAR]. It was authored by [FULL NAME, EMAIL ADDRESS], with whose written permission it is reprinted here.

[3] Libraries may archive issues of *MTO* in electronic or paper form for public access so long as each issue is stored in its entirety, and no access fee is charged. Exceptions to these requirements must be approved in writing by the editors of *MTO,* who will act in accordance with the decisions of the Society for Music Theory.

This document and all portions thereof are protected by U.S. and international copyright laws. Material contained herein may be copied and/or distributed for research purposes only.

Prepared by Jon Koriagin and Rebecca Flore, Editorial Assistants