Manual for MkInvar and GrEnum

Chris Knight, Corey Beck and Sherwin J. Singer

May 19, 2010

Contents

1 INTRODUCTION: H-bond disorder and H-bond topology in water ices

The hydrogen bond pattern of defect-free water ice is governed by what are called the Bernal-Fowler ice rules:[2] Each water must donate a hydrogen bond (H-bond) to two neighboring waters, and accept an H-bond from two other neighbors. These rules are followed in water ices up to pressures in the $15GPa$ range, by which point protons begin to tunnel across H-bonds.

The phases of ice adjacent to the liquid phase – ice Ih, III, V, VI, and VII – shown as shaded in Fig. 1, all exhibit H-bond disorder. A very large number of H-bond arrangements consistent with the ice rules is possible, and they are explored in all of these phases. With the exception of ice VI, a phase transformation has been observed for these phases which transform into an H-bond ordered counterpart at low temperatures. Ice Ih transforms to ice XI, ice III to ice IX (not shown in Fig. 1 because ice IX is metastable with respect to ice II), and ice VII to ice VIII. The observation of ice XIII, an ordered form of ice V, has just recently been reported.[3–5]

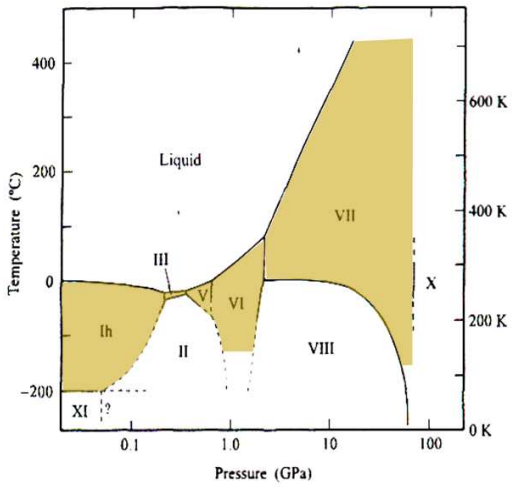

Figure 1: Phase diagram of ice, modified from a figure in Petrenko and Whitworth.[1]

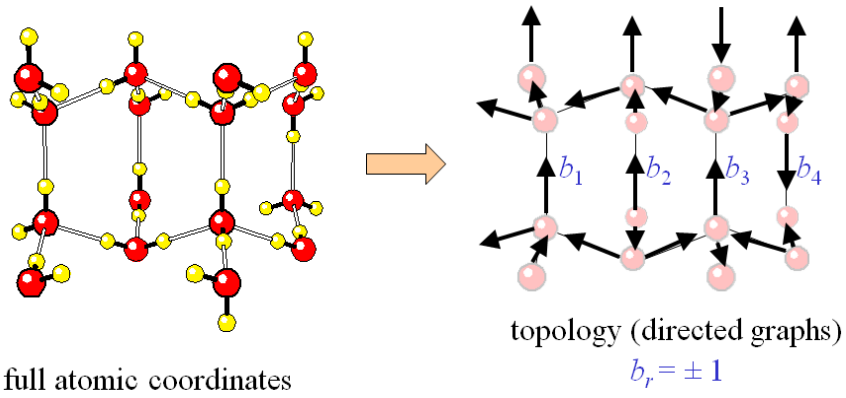

Figure 2: Abstraction from detailed crystal configuration to H-bond topology.

When we speak of an H-bond configuration, we are abstracting from the detailed crystal geometry, i.e. x, y, z coordinates for all atoms of the crystal, to the *topology* or connectivity of the H-bond pattern (Fig. 2). Each H-bond is directional, so the H-bond connectivity is captured by a directed graph in which vertices, representing oxygen atoms in ice, are connected by a single, directed bond. Physically, each directed graph containing only two outgoing and two incoming bonds at each vertex represents a deep local minimum of the potential surface of defect-free ice. Of course, defects which break the ice rules are exceedingly important and interesting. They can also be

represented and usefully described by directed graphs.

The H-bond configuration can be described by bond variables b_r that take the values ± 1 depending whether H-bond r points with or against some previously defined direction. Obviously there are 2^N bond configurations before enforcing the ice rules. After the ice rules are taken into account, Pauling estimated that for ice Ih (and it turns out many other ices) the number of configurations falls to roughly $\left(\frac{3}{2}\right)^N$,[6] very close to the exact result.[7]

For theoretically studying the ordered and disordered ice phases, it is most useful to have an enumeration of the symmetry-distinct H-bond configurations of relatively small unit cells. These are unit cells small enough either for electronic structure calculations, or (not as small) suitable for statistical simulations. This is one of the tasks accomplished by the programs *MkInvar* and *GrEnum*. Nominally, finding symmetry-distinct H-bond configurations for N water molecules is an $O(N^2)$ process since it involves comparison of pairs of configurations and eliminating one member of the pair if a space group symmetry operation brings it into coincidence with the other member. We have shown that this process can be reduced to an $O(N \ln N)$ calculation using invariant polynomials in the bond variables.[8] We use the name *graph invariants* for these polynomials.[8,9] Since graph invariants are useful in the enumeration problems, and have other uses as well, we pause to explain them more fully.

Graph invariants are linear combinations of bond variables

$$\sum_r d_r b_r \,, \tag{1}$$

(first-order invariants), bilinear combinations of bond variables

$$\sum_{rs} d_{rs} b_r b_s \,, \tag{2}$$

(second-order invariants), trilinear combinations . . ., and so on.[8,9] The graph invariant polynomials are unchanged under a permutation of the bond variables according to a space group operation g_α, $b_r \to g_\alpha(b_r)$. All graph invariants must evaluate to the same value for two configurations that are symmetry-related. By evaluating a series of graph invariants for a series of H-bond configurations, an $O(N)$ process, we obtain a symmetry "fingerprint" of each configuration. Let's say there are P invariants evaluated for each H-bond configuration. The configurations are sorted into groups according to the value of the first invariants. Then those groups are subdivided into smaller groups according to the second invariant. The process goes on until the configurations in each group falls below a threshold or all P sorting invariants are exhausted. At this point, the expensive $O(N^2)$ symmetry testing is applied to the small groups of configurations that have the same fingerprint. Those with different fingerprints cannot possibly be symmetry-related. Assuming some reasonable effectiveness in breaking down the original list into smaller groups, this makes elimination of symmetry-related configurations an $O(N \ln N)$ process.[8]

Graph invariants have other applications besides making enumeration more efficient. If scalar physical properties, most notably but not exclusively the energy, of each of the H-bond isomers

can be linked with H-bond topology, then graph invariants provide a useful hierarchy of basis functions for describing this linkage. For example, energy as a function of H-bond topology can be expressed as follows.

$$E(b_1, b_2, \ldots) = E_0 + \sum_r a_r I_r(b_1, b_2, \ldots) + \sum_{rs} a_{rs} I_{rs}(b_1, b_2, \ldots) + \ldots \tag{3}$$

First-order I_r and second-order I_{rs} invariant functions are shown explicitly in Eq. (3). Essentially, Eq. (3) is a spin-lattice Hamiltonian for H-bond fluctuations in ice. In practice the expansion can be continued to higher-order invariants, but in practice we have found that stopping at second-order gives a good account of H-bond energetics. In most cases, the first-order invariants are identically zero, leading to a very compact expression for the energy. The few coefficients appearing in Eq. (3) can be established from electronic structure calculations on small unit cells. We then rely on a theorem which states that invariants for small cells are automatically invariants for larger unit cells that are multiples of the small cells. This provides a bootstrap procedure by which the coefficients in Eq. (3) can be extracted from electronic structure calculations on small unit cells and then statistical simulations can be performed on cells large enough to approximate the bulk limit. Our group has used graph invariants to link the energy of H-bond arrangements in various phases of ice to the H-bond topology. With this linkage in hand, we were able to construct a statistical mechanical model of H-bond fluctuations and predict the properties of the ice Ih-XI,[10,11] VII-VIII,[10,11] III-IX,[12] and V-XIII[13] phase transitions.

The *MkInvar* program is a Fortran program which takes as input information on the coordinates and symmetry of the system and generates as output coordinates of atoms in large simulations cells and graph invariants. Also generated is output for a second Fortran program, *GrEnum*, which enumerates all symmetry-distinct H-bond configurations allowed by the ice rules. This manual discusses the background information, use of the codes, and a detailed description of the algorithms used.

2 Definition of cell vectors

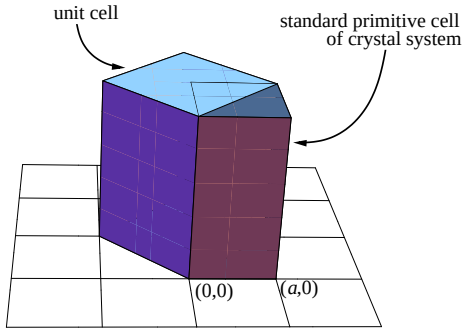

Figure 3: Primitive and larger unit cell of a tetragonal system.

We will use several sets of vectors that may be called "lattice" or "basis" vectors, and it is best to clearly define them at the outset. First, there are the standard basis vectors associated with the 7 crystal systems. These are not input because they are standard for each of the crystal systems. For example, it is standard to take $(a,0,0),(0,a,0)$ and $(0,0,c)$ as the basis vectors for a tetragonal system and $(a,0,0),(\frac{a}{2},\frac{\sqrt{3}a}{2},0)$ and $(0,0,c)$ for a hexagonal system. This set of vectors uses the traditional standard lattice parameters, $a,b,c,\alpha,\beta,\gamma$, or fewer with symmetry.

The *MkInvar* code generates graph invariants for a small unit cell, hereafter known simply as "the unit cell", and larger cells obtained by replicating that small cell an integer number of times along each of the cell vectors. This may or may not coincide with the standard primitive cell of the crystal system. For example, it may be convenient to take the unit cell for calculations on a tetragonal system as one rotated 45° from the standard cell as shown in Fig. 3. We might make this choice if we want to examine H-bond arrangements in a cell modestly larger than the standard unit cell without, say, doubling the tetragonal a lattice constant.

The vectors that define "the unit cell" in the figure are linear combinations of the standard basis vectors $[\mathbf{l}_1 = (a,0,0), \mathbf{l}_2 = (0,a,0), \mathbf{l}_3 = (0,0,c)]$ of the tetragonal system,

$$\mathbf{L}_k = \sum_{j=1}^{3} c_{kj}\mathbf{l}_j \,, \tag{4}$$

where the coefficients c_{kj} are summarized below in matrix form.

$$\mathbf{c} = \begin{pmatrix} 1 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

The c_{kj} coefficients are always integers or rational fractions. The rows of the matrix in Eq. (5) will be known as the cell vectors, $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$.

The user has the freedom to make the actual cell under consideration the unit cell replicated an integral numbers of times in each direction defined by cell vectors $m_1\mathbf{L}_1, m_2\mathbf{L}_2, m_3\mathbf{L}_3$. This is useful when an orderly sequence of cells of increasing size is needed. When the integers m_1, m_2, m_3 are sufficiently large, the cell is large enough to be considered a simulation cell. We will always refer to the larger cell defined by m_1, m_2, m_3 as the "simulation cell" to distinguish it from the basic unit cell, defined by the c_{jk} coefficients in Eqs. (4-5), even when the larger cell is not really large enough to approximate the thermodynamic limit.

3 *MkInvar*

3.1 Compilation

The *MkInvar* program is written in Fortran 77 and thus any of the commonly available Fortran compilers should be sufficient. To compile the code, one simply goes to the source directory and types "make" on the command line. This should generate an executable named "MkInvar."

3.2 Running MkInvar

The following command is used to run the program.

```
MkInvar
```

In total, there are a minimum of three input files necessary to run the program. The first input file that the code reads is a file named "MkInvar.inp" which contains all commands necessary to run a calculation. The second file contains information regarding the space group to be used in the calculation. The third file contains either the vertices or H-bond definitions for the system. Output of the code is written to a number of different files. We discuss the contents and format of each of these files in turn.

3.3 Input Files

At the very least, there are three input files necessary to run the *MkInvar* program: "MkInvar.inp", a space group file, and "VertDef.txt" or "HBondsDef.txt". With these input files, it is possible to generate an H-bond structure for an arbitrary simulation cell as well as construct all possible first- and second-order graph invariants. For large simulation cells, an additional input file, "BondPairsDef.txt", is used to construct only those invariants that the user chooses. These are typically invariants also found in small unit cells.

3.3.1 MkInvar.inp

This input file contains information regarding the dimensions and symmetry of the system, parameters to control the generation of the H-bond lattice, and which graph invariants will be generated. Below is a example of an input file for calculations on the unit cell of ice VII.

```
 1 1 1                        (Simulation cell dimensions)
      1 1   0 0   0 0         (Matrix of cell vectors as row vectors)
      0 0   1 1   0 0
      0 0   0 0   1 1
224 1                         (Space Group Index)
../../SpaceGroups             (Path to space group database)
3.337 3.337 3.337  90 90 90 (Lattice Parameters: a,b,c,alpha,beta,gamma)
0.1                           (Minimum allowed OO distance)
2.50  3.13                    (Min/Max allowed OO distance for H-Bonds)
1                             (0: Generate H-bonds / 1: Read H-bonds)
1                             (0: No Invar / 1: All / 2: Some)
1                             (0: User specified covering cell / 1: Auto.)
```

All text following the numbers, contained in parentheses, is ignored by the program. The first line contains three non-negative integers, the m_j's in section 2, page 7, which defines the dimensions of the simulation cell in terms of the number of unit cells along each lattice vector $\mathbf{L}_j$.

```
 1 1 1                        (Cell dimensions)
```

Here, we are performing calculations on a simulation cell measuring $1 \times 1 \times 1$ unit cells on each side, i.e. the unit cell. The next three lines, which each contain six integers, define the components of the cell vectors for the simulation, the **c** matrix. For the case of a primitive unit cell, the matrix of cell vectors would simply be the identity matrix.

```
      1 1   0 0   0 0         (Matrix of cell vectors as row vectors)
      0 0   1 1   0 0
      0 0   0 0   1 1
```

Each component of a cell vector is given as a pair of integers, a numerator and denominator. These components are then stored internally as the ratio of the two integers. Whenever the numerator is zero, the denominator is stored as a real number. For example, there are two ways define the same cell vector.

```
      1 3   1 1   3 2              (Cell vector using integers)

      0  0.333   0  1   0  1.5     (Cell vector using reals)
```

The next two lines define the space group to be used in calculations and the path to the directory containing the necessary space group file.

```
224 1                         (Space Group Index)
../../SpaceGroups             (Path to space group database)
```

In this example, the space group is #224 (Pn-3m). For this space group, there are two possible origin choices. The number one which follows the space group index indicates we are using the first origin choice. We say more about the possible convention choices below when we discuss the space group files. The file containing the required symmetry elements is contained in the "SpaceGroups" directory which is found two directories above the current working directory. The format of the space group file will be described below. The next line of the input file contains the six lattice parameters which define the unit cell lattice vectors for the crystal system.

```
3.337 3.337 3.337  90 90 90 (Lattice Parameters: a,b,c,alpha,beta,gamma)
```

The first three numbers, read as reals, indicate the length of each lattice vector in units of Ångström. Actually, the units used are arbitrary as long as the units of all input parameters are consistent. All output files will be written in terms of this unit of length. The last three numbers, also reals, are the angles between lattice vectors. These numbers must be in units of degrees.

When there are fewer than 6 lattice parameters, as for the example of a cubic lattice system given above, all 6 parameters must still be input. The program exits if the input parameters are not compatible with lattice symmetry, e.g. $a \neq b$ for a cubic system. The next line contains a distance criterion for the minimum allowed OO distance. When generating the oxygen positions, if the distance between two oxygen atoms falls below the allowed distance, then the code will exit with a message stating so.

```
0.1                          (Minimum allowed OO distance)
```

The user has the option of providing a list of H-bonded vertices in the unit cell, or else allowing the program to determine the H-bonded vertices based on a distance criterion. If the latter is selected, the next line contains two parameters that control the generation of the H-bond structure by providing an allowed range of H-bonded oxygen-oxygen distances.

```
2.50  3.13                   (Min/Max allowed OO distance for H-Bonds)
```

Even if the user furnishes a list of H-bonded vertices, this input is still required because the program will check that input for consistency. The units used for these two distances must be consistent with the units of length used in the lattice parameters. When identifying possible H-bonds, only those with distances within the allowed values will be kept.

The next line controls how the unit cell H-bond structure is generated.

```
1                            (0: Generate H-bonds / 1: Read H-bonds)
```

The structure can be generated in one of two ways.

0. Use positions of oxygen vertices in asymmetric unit, employing the distance criteria input on the previous line.

1. Use H-bond definitions for entire unit cell.

In the first case, the input file "VertDef.txt" is required while in the second case, the file "HBonds-Def.txt" is required. The format and use of these files will be discussed in more detail below.

Once the H-bonded structure has been generated, one is then ready to calculate graph invariants. The options available for calculating graph invariants follow.

0. Do not calculate invariants.

1. Calculate all possible first- and second-order invariants.

2. Only calculate those graph invariants for which the generating bond pairs have been listed in the input file "BondPairsDef.txt."

The user specifies the desired option on the following line.

```
1                              (0: No Invar / 1: All / 2: Some)
```

The last line of this input file is specific to the case for when a covering cell is to be used. For those cases when a covering cell is necessary and the user wants to specify the dimensions and shape of the covering cell (as opposed to having the *MkInvar* code do this), this option can be given the value of zero.

0. User specified covering cell. The simulation cell dimensions and cell vectors read at the start of the input file define the covering cell. The cell vectors for the system of interest are given immediately after this line of input. See examples below in the tutorial.

1. Automatic generation of covering cell. If not necessary, then no action will be performed.

A covering cell is not necessary for this system and so we choose a value of 1.

```
1                              (0: User specified covering cell / 1: Auto.)
```

Those cases for when a covering cell is required is when the simulation cell does not have the same symmetry as the crystal lattice for the space group. A covering cell would be used if one wanted to do calculations on a simulation cell measuring $2 \times 2 \times 1$ unit cells along each side. Another example, as discussed earlier in section 2, is the case of constructing a tetragonal cell from a cubic unit cell. We will discuss the cases for when a covering cell is necessary in the tutorial below, in sections 4.5 and 4.6.

3.3.2 Space Group File

The space group file contains the symmetry elements which generate the group. In the "SpaceGroups" directory, one can find files for all 230 space groups named "###.n" where ### is the corresponding space group index. The "n" in the space group file name indicates which convention for the space group is used. We are indebted to Xianlong Wang for placing space group information in a *Mathematica*[14] notebook,[15] a format that was convenient for us to generate the files given in the "SpaceGroups" directory. Dr. Wang obtained the space group information from the Bilbao Crystallographic Server.[16–18]

The suffix on space group file names is included to allow for more than one space group convention, whether it be the choice of origin or unique axis. For the space group files located in the "SpaceGroups" directory, we have taken the following convention for file naming. For space groups with multiple origin choices, files named "###.1" correspond to the first origin choice while files named "###.2" are for the second origin choice. For those space groups with conventions that differ in their choice of unique axis, files named "###.1" and "###.2" correspond to choosing the *b* or *c* axis as the unique axis respectively. For space groups which can have rhombohedral or hexagonal axes, the files are named "###.1" and "###.2" respectively. For those space groups with only a single convention, the files are named "###.1." New space group definitions are implemented by simply creating a new space group file and placing that file in the directory specified as input in the "MkInvar.inp" file. The space group index and convention choice can currently take any integer value between 0 and 999. Below is the space group file for #224 $Pn\bar{3}m$ for the first origin in standard tables.

```
224 (Pn-3m)
6 48
cub
 1 0 0 0 1 0 0 0 1
 0 1 0 1 0 1
 -1 0 0 0 -1 0 0 0 1
 0 1 0 1 0 1
 -1 0 0 0 1 0 0 0 -1
 0 1 0 1 0 1
 0 0 1 1 0 0 0 1 0
 0 1 0 1 0 1
 0 1 0 1 0 0 0 0 -1
 1 2 1 2 1 2
 -1 0 0 0 -1 0 0 0 -1
 1 2 1 2 1 2
```

The first line contains the space group index and name. Only the index is read by the program and it must match that given in the input file. The second line contains two integers, the number

of generators and the order of the space group, or more accurately, the number of coset representatives. For group 224, there are 6 symmetry operations that are used to generate the full group and these will be read below. When the full group has been generated by the program, it should contain 48 symmetry elements. If the number of symmetry elements generated in the full group does not match 48, the program aborts with a warning message. The third line contains a three letter abbreviation for the lattice type of this space group:

- Cubic: cub
- Tetragonal: tet
- Orthorhombic: ort
- Trigonal: trg
- Hexagonal: hex
- Monoclinic: mon
- Triclinic: tri

This lattice type must agree with the lattice parameters supplied in the input file. The rest of the space group file contains the symmetry operations used to generate the group. Symmetry elements in space groups, affine transformations, are described by a 3×3 rotation matrix and a 3-dimensional translation vector. To describe point groups used in cluster calculations, the translation vectors would simply be zero vectors. Each symmetry operation is defined by two lines of integers.

```
1 0 0 0 1 0 0 0 1
0 1 0 1 0 1
```

The first line, a list of nine integers($0, \pm 1$) defines the 3×3 rotation matrix row-by-row. The second line, a list of six integers, contains the translation vector. Each component of this vector is given as the ratio of two integers. Unlike the "MkInvar.inp" input file, both numbers in each pair must be integers and the second number can not be zero.

$$\textit{line \#1} = g_{11} \quad g_{12} \quad g_{13} \quad g_{21} \quad g_{22} \quad g_{23} \quad g_{31} \quad g_{32} \quad g_{33} \tag{6}$$

$$\textit{line \#2} \quad g_{14a} \quad g_{14b} \quad g_{24a} \quad g_{24b} \quad g_{34a} \quad g_{34b} \tag{7}$$

$$\mathbf{g}_j = \left\{ \begin{pmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{pmatrix}, \begin{pmatrix} g_{14a}/g_{14b} \\ g_{24a}/g_{24b} \\ g_{34a}/g_{34b} \end{pmatrix} \right\} \tag{8}$$

Applying the 6 generators of group 224 given above on an arbitrary point in space yields the following:

$$\mathbf{g}_1\{x,y,z\} = \{x,y,z\} \quad (9)$$
$$\mathbf{g}_2\{x,y,z\} = \{-x,-y,z\} \quad (10)$$
$$\mathbf{g}_3\{x,y,z\} = \{-x,y,-z\} \quad (11)$$
$$\mathbf{g}_4\{x,y,z\} = \{z,x,y\} \quad (12)$$
$$\mathbf{g}_5\{x,y,z\} = \{y+1/2, x+1/2, -z+1/2\} \quad (13)$$
$$\mathbf{g}_6\{x,y,z\} = \{-x+1/2, -y+1/2, -z+1/2\}. \quad (14)$$

3.3.3 VertDef.txt

The text file "VertDef.txt" contains a list of coordinates for the oxygen atoms (vertices) in the system. At the very least, this list must contain coordinates for atoms in the asymmetric unit for *MkInvar* to generate the full structure. To generate the full structure, *MkInvar* applies all symmetry operations of the full group and identifies all unique vertices (to within some tolerance). This distance tolerance is the value of 1.0 assigned to the minimum allowed H-bond distance in the "MkInvar.inp" file above. This list can contain as many vertices as the user likes. All duplicate vertices generated from application of the symmetry group are removed. Below is an example of "VertDef.txt" for the ice III system, the primitive unit cell of which contains twelve water molecules.

```
2   # of vertices in asymmetric unit
2  5   1  5   1 3
1 10   1 10   0 1
```

The first line contains the number of oxygen atoms to be read from the file. The remaining text on the line is ignored. The asymmetric unit of the ice III unit cell only contains two oxygen atoms. The coordinates for each vertex are given as fractions of the unit cell lattice vectors, the $\mathbf{L}_k$ in section 2. The coordinates for each oxygen atom are given as a list of six integers. Each pair of integers corresponds to the numerator and denominator of an x, y, or z component. For example, these two oxygen atoms have the following fractional coordinates respectively: $(0.400, 0.200, 0.333)$ and $(0.100, 0.100, 0.00)$.

3.3.4 HBondsDef.txt

The text file "HBondsDef.txt" contains a list coordinates for all H-bonds contained in the unit cell. If any H-bonds are missing from this list that are related by symmetry, then the code will write to the "MkInvar.log" file any newly generated H-bonds and exit with error. The user can then inspect this list and add or remove H-bonds as found appropriate. The code will exit whenever some oxygens are not four coordinated. H-bonds are defined by coordinates for the two oxygen atoms

involved in hydrogen bonding. It is not yet possible to supply a list of H-bonds in the asymmetric unit and have *MkInvar* generate the full set of H-bonds. The ice III unit cell contains twelve water molecules and thus there are twenty-four H-bonds. Below is portion of "HBondsDef.txt" used for this system.

```
24   # of bonds in unit cell
1 10   1 10   0 1      1  5   2  5  -1  3
1 10   1 10   0 1      2  5   1  5   1  3
2  5   3  5   1 4      1 10   7 10  -1 12
2  5   3  5   1 4      3 10   9 10   7 12
3  5   2  5   3 4      7 10   1 10  13 12
3  5   2  5   3 4      9 10   3 10   5 12
9 10   9 10   1 2      3  5   4  5   5  6
...
```

The first line contains the number of H-bonds to be read from the file. The remaining text on the line is ignored. On each of the following lines, the coordinates for each of the two vertices of an H-bond are listed. In total, there are twelve integers on each line that define the H-bonds. Each pair of integers corresponds to the numerator and denominator of an x, y, or z component of a vertex. Eventually, output configurations will be generated that enumerate the symmetry-distinct H-bond arrangements that satisfy the ice rules. In the output, the convention is taken that H-bonds in a configuration pointing from the first vertex on a line in "HBondsDef.txt" towards the second vertex will be assigned a value of "1", while H-bonds pointing from the second entry to the first entry will be assigned "-1".

3.3.5 BondPairsDef.txt

As specified on page 11, calculation of none, some, or all graph invariants can be requested. When some are requested, the graph invariants to be calculated are specified in this file. It is conveniently constructed by modifying the output file "BondPairs.txt" generated when *MkInvar* calculates graph invariants. It would be common practice to generate this file from a small cell calculation and then use it as input to generate small cell invariants for a large simulation cell. This input file can be used on any simulation cell size.

Recall that graph invariants are invariant polynomials in the bond variables b_r, that is, variables that take the values ± 1 to describe the direction of H-bonds. Space group operations map bond variables into other bond variables, sometimes with a change of sign. Graph invariant polynomials are unchanged by such a mapping.[8,9] Linear, quadratic, cubic, ... polynomials are term first, second, third ... order graph invariants. They are generated by applying the group theoretical projection operator for the totally symmetric representation to bond variables $\hat{G}(b_r)$, products of two bond variables $\hat{G}(b_r b_s)$, products of three bond variables $\hat{G}(b_r b_s b_t)$, ... When dealing with impurities, it is useful to assign a different "color" to H-bonds that involve an impurity.[19,20]

An example "BondPairsDef.txt" file is given below from the ice III system.

```
 4     # of invariants
0 : Type
1  10       1  10       0   1            1   5       2   5      -1   3
0 : Type
1  10       7  10      11  12           -1   5       3   5       7   6
0 : Type
1  10       7  10      11  12            1  10      11  10       1   1
2 : Type
1  10       1  10       0   1            1   5       2   5      -1   3
1  10       1  10       0   1            1   5       2   5      -1   3
```

In this case, we are choosing to construct the three first-order graph invariants plus a single second-order graph invariant. The first line indicates the number of graph invariants we would like constructed. For each invariant, there is one line of descriptive information indicating the type of invariant to be generated. The convention taken by the code is the following:

0. $\hat{G}(b_r)$ (first-order invariant)
1. $\hat{G}(b_r b_s)$ (second-order invariant)
2. $\hat{G}(b_r c_s)$ (second-order invariant, two bond "colors")

Invariants beyond second order are not yet implemented. For first- and second-order invariants, respectively, there will be one and two additional lines for each invariant. These lines give the vertices for the generating H-bonds. Again, each H-bond is given as a pair of vertices and each component of the vertices is given as the ratio of two integers.

3.4 Output Files

Certain types of output files are generated for each cell size required by a particular calculation. For other output files, only a single version is generated. Output files for the unit cell are always generated and distinguished from the others by the suffix "_Unit" in the file name. Using the notation in section 2, this corresponds to the case when the m_j's all equal 1 independent of the form of the matrix of cell vectors, **c**. For larger cells where at least one of the m_j's is larger than one, the corresponding output files have the suffix "_Sim". (The mnemonic is that cells with larger m_j values, used to describe the thermodynamic limit, are "simulation" cells.) When a large covering cell is necessary, as described below, files associated with this cell have the suffix "_Cov". For each of the output files, we will indicate whether or not corresponding files are generated for each cell size used in the calculation.

3.4.1 MkInvar.log

Output showing progress of the code and debugging info are written to the file named "MkInvar.log." The contents of this file will be discussed in detail below.

3.4.2 Group_Unit.txt

This file contains all elements of the full symmetry group generated from the symmetry elements contained in the space group file. Below is an example output file generated for the ice III system using space group #92($P4_12_12$).

```
          8 # of symmetry elements in group
 1  0  0     0  1  0     0  0  1       0   1     0   1     0   1
-1  0  0     0 -1  0     0  0  1       0   1     0   1     1   2
 0 -1  0     1  0  0     0  0  1       1   2     1   2     1   4
-1  0  0     0  1  0     0  0 -1       1   2     1   2     1   4
 0  1  0    -1  0  0     0  0  1       1   2     1   2     3   4
 1  0  0     0 -1  0     0  0 -1       1   2     1   2     3   4
 0 -1  0    -1  0  0     0  0 -1       0   1     0   1     1   2
 0  1  0     1  0  0     0  0 -1       0   1     0   1     0   1
```

The first line contains the number of symmetry elements in the group. Each line after that contains a symmetry operator where the elements of the rotation matrix and translation vector are given. The first nine integers make up the row vectors of the rotation matrix. The last six integers make up the components of the translation vector.

$$\begin{aligned} &\textit{each line} && (15)\\ &= \quad g_{11}\ \ g_{12}\ \ g_{13}\ \ g_{21}\ \ g_{22}\ \ g_{23}\ \ g_{31}\ \ g_{32}\ \ g_{33}\quad g_{14a}\ \ g_{14b}\ \ g_{24a}\ \ g_{24b}\ \ g_{34a}\ \ g_{34b} \end{aligned}$$

$$\mathbf{g}_j = \left\{ \begin{pmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{pmatrix}, \begin{pmatrix} g_{14a}/g_{14b} \\ g_{24a}/g_{24b} \\ g_{34a}/g_{34b} \end{pmatrix} \right\} \qquad (16)$$

For larger cells, the symmetry group is written to the file "Group_Sim.txt". When a covering cell is used, the symmetry group for the covering cell also written to "Group_Sim.txt".

3.4.3 Structure_Unit.txt

This file contains lattice vectors, $\mathbf{L}_j$'s, and fractional coordinates for the vertices and H-bonds of the unit cell. Below is an example from running calculations on the ice III unit cell.

```
6.666000    0.000000    0.000000   Primitive Unit Cell Lattice Vectors
0.000000    6.666000    0.000000
0.000000    0.000000    6.936000
6.666000    0.000000    0.000000   Lattice Vectors
0.000000    6.666000    0.000000
0.000000    0.000000    6.936000
12   # of vertices
1  10     1  10     0   1
2   5     3   5     1   4
3   5     2   5     3   4
9  10     9  10     1   2
1  10     7  10    11  12
1   5     2   5     2   3
...
24   # of HBonds
1  10     1  10     0   1         1   5     2   5    -1   3      1  6
1  10     1  10     0   1         2   5     1   5     1   3      1  8
2   5     3   5     1   4         1  10     7  10    -1  12      2  5
2   5     3   5     1   4         3  10     9  10     7  12      2  7
...
```

The first three lines contain the lattice vectors for the primitive unit cell in the same units used in the input file. It is these vectors combined with the fractional coordinates that generate the physical coordinates to be found in the output files discussed next. The next three lines contain the lattice vectors for this cell. These vectors are used to apply periodic boundary conditions. For this small cell, both sets of lattice vectors are the same. They will differ for cells larger than the unit cell and for the case of non-primitive lattice vectors. The next line states the number of vertices in the unit cell and is followed by fractional coordinates for each vertex. After the vertices, the number of H-bonds found is given. This is then followed by the fractional coordinates for each H-bond. For each H-bond, the last two integers on each line are the indices corresponding to each vertex. For example, the first H-bond is defined as pointing from vertex #1 to vertex #6. H-bond definitions follow the convention that the first vertex is always contained within the cell boundaries while the second vertex may lie outside. For example, the second vertex of H-bond #1 is related to vertex #6 by a lattice translation along the c−axis. Similar files for the simulation and covering cells, "Structure_Sim.txt" and "Structure_Cov.txt" respectively, will be written if warranted. For those cases when a covering cell calculation is performed, triggered by the last option in the "MkInvar.inp" file, a "StructureC.txt" file is generated containing vertices and H-bonds for the cell of interest(not the covering cell).

3.4.4 Vert_Unit.xyz

This file contains the *xyz* coordinates of the vertices suitable to be read as input by a visualization program. The coordinates are generated using the lattice vectors and fractional coordinates found in the output file "Structure_Unit.txt." Vertices are labeled with "O" as they correspond to the oxygen atoms in ice. Also, dummy atoms, "X," are placed at the corners of the unit cell.

```
20
Unit Cell: Oxy
  O      0.666600      0.666600      0.000000
  O      2.666400      3.999600      1.734000
  O      3.999600      2.666400      5.202000
  O      5.999400      5.999400      3.468000
  O      0.666600      4.666200      6.358000
  O      1.333200      2.666400      4.624000
  O      1.999800      5.999400      4.046000
  O      2.666400      1.333200      2.312000
  O      3.999600      5.332800      5.780000
  O      4.666200      0.666600      0.578000
  O      5.332800      3.999600      1.156000
  O      5.999400      1.999800      2.890000
  X      0.000000      0.000000      0.000000
  X      0.000000      0.000000      6.936000
  X      0.000000      6.666000      0.000000
  X      0.000000      6.666000      6.936000
  X      6.666000      0.000000      0.000000
  X      6.666000      0.000000      6.936000
  X      6.666000      6.666000      0.000000
  X      6.666000      6.666000      6.936000
```

The first line contains the number of vertices plus eight(number of corners). The second line is a title giving a short description. All lines that follow contain an atom type and the *xyz* coordinates in the same units as given in the input file. Similar files are also generated for the simulation cell, "Vert_Sim.xyz," and covering cell, "Vert_Cov.xyz," if warranted. When a covering cell calculation is requested, a file name "VertC.xyz" is generated.

3.4.5 Bond_Unit.xyz

This file contains the *xyz* coordinates of the vertices and H-bonds suitable to be read as input by a visualization program. The coordinates are generated using the lattice vectors and fractional coordinates in the output file "Structure_Unit.txt." Vertices are labeled with "O" as they correspond

to the oxygen atoms in ice. Using the H-bond definitions, hydrogen atoms, "H," are placed at the midpoints of the H-bonds. Also, dummy atoms, "X," are placed at the corners of the cell.

```
 44
 Unit Cell: Oxy + Hyd at midpoints
   O      0.666600      0.666600      0.000000
...
   O      5.999400      1.999800      2.890000
   H      0.999900      1.666500     -1.156000
   H      1.666500      0.999900      1.156000
   H      1.666500      4.332900      0.578000
   H      2.333100      4.999500      2.890000
   H      4.332900      1.666500      6.358000
...
   H      3.999600      3.999600      5.491000
   H      5.999400      0.666600      0.289000
   H      3.999600      3.999600      1.445000
   H      5.999400      0.666600      3.179000
   X      0.000000      0.000000      0.000000
   X      0.000000      0.000000      6.936000
...
```

The first line contains the number of vertices and H-bonds plus eight(number of corners). The second line is a title giving a short description. All lines that follow contain an atom type and the *xyz* coordinates in the same units as given in the input file. A similar file is also generated for the simulation cell, "Bond_Sim.xyz," and covering cell, "Bond_Cov.xyz," if warranted. When a covering cell calculation is requested, a file name "BondC.xyz" is generated.

3.4.6 Bond_Unit_Wrap.xyz

This is the same as "Bond_Unit.xyz" except that hydrogen atoms which lie outside of the cell boundary are translated back into the cell. A similar file is also generated for the simulation cell, "Bond_Sim_Wrap.xyz," and, if need be, covering cell, "Bond_Cov_Wrap.xyz,". When a covering cell calculation is requested, a file name "BondC_Wrap.xyz" is generated.

3.4.7 GraphInvar.txt

This file contains all requested graph invariants constructed for the cell with lattice vectors $m_j\mathbf{L}_j$. The first line contains the total number of graph invariants contained in the file. For each graph invariant, there are two lines of descriptive information followed by a line for each term of the invariant. For example, here is the initial portion of the "GraphInvar.txt" file generated for the ice III unit cell.

```
       120                    (# of Invariants)
0                             (0: b[j],   1:b[j]b[k],   2:b[j]c[k])
   1     0     8      8      (Bond Pair, # of Terms & Normalization)
   1     1     0
   1     2     0
   1     3     0
   1     4     0
   1     5     0
   1     6     0
   1     7     0
   1     8     0
0                             (0: b[j],   1:b[j]b[k],   2:b[j]c[k])
   9     0     8      8      (Bond Pair, # of Terms & Normalization)
   1     9     0
   1    10     0
   1    11     0
   1    12     0
   1    13     0
   1    14     0
   1    15     0
   1    16     0
...
```

We can see that 120 graph invariants were generated for this unit cell. This number includes all first- and second-order graph invariants:$\hat{G}(b_r)$, $\hat{G}(b_r b_s)$ and $\hat{G}(b_r c_s)$. For each graph invariant, the first line contains an integer indicating the graph-invariant type (see page 16). The second line includes a total of four integers. The first two numbers are the indices of the generating bond pairs. These are then followed by the number of terms contained in the invariant and the normalization constant. In the case of first-order invariants, where a single bond generates the invariant, the index for the second bond is taken to be zero. For every term in the invariant, there is a line which contains the coefficient and bond indices. Again, by default, the second bond index is set to zero for first-order graph invariants. The two first-order graph invariants shown above take the following form:

$$\hat{G}(b_1) = \frac{1}{8}(b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + b_8) \tag{17}$$

$$\hat{G}(b_9) = \frac{1}{8}(b_9 + b_{10} + b_{11} + b_{12} + b_{13} + b_{14} + b_{15} + b_{16}). \tag{18}$$

From the same file, here are examples of second-order graph invariants

```
...
2                                   0: b[j],   1:b[j]b[k],   2:b[j]c[k]
  17      24       8       8      Bond Pair, # of Terms & Normalization
   1      17      24
   1      18      23
   1      19      22
   1      20      21
   1      21      20
   1      22      19
   1      23      18
   1      24      17
...
1                                   0: b[j],   1:b[j]b[k],   2:b[j]c[k]
   1       3       8       8      Bond Pair, # of Terms & Normalization
   1       1       3
   1       1       6
   1       2       4
   1       2       5
   1       3       8
   1       4       7
   1       5       7
   1       6       8
```

and their corresponding mathematical form:

$$\hat{G}(b_{17}c_{24}) = \frac{1}{8}(b_{17}c_{24} + b_{18}c_{23} + b_{19}c_{22} + b_{20}c_{21} + b_{21}c_{20} + b_{22}c_{19} + b_{23}c_{18} + b_{24}c_{17}) \quad (19)$$

$$\hat{G}(b_1b_3) = \frac{1}{8}(b_1b_3 + b_1b_6 + b_2b_4 + b_2b_5 + b_3b_8 + b_4b_7 + b_5b_7 + b_6b_8) \quad (20)$$

When a covering cell calculation is requested, a file name "GraphInvarC.txt" is also generated which contains the graph invariants for the cell of interest.

3.4.8 GenBondPair.txt

For every graph invariant listed in the file "GraphInvar.txt," this file contains the H-bond definitions for all bond pairs which generate each invariant. Actually, there is a variable "nBondPairsMax" in the "param.inc" file which controls how many bond pairs for each invariant are stored. For each invariant, the first bond pair given is also written to a separate file, "BondPairs.txt," which can be used as input for additional calculations, particularly when one seeks to take invariants from a smaller unit cell and generate the corresponding invariant for a much larger "simulation" cell.

The top portion of this file generated for the ice III unit cell, which begins with first order invariants, is given below.

```
 Total # of invariants generated:      120

 # of 1st order invariants:      3
 ========================================


Invariant:  1,  Type:  0
Generating Bonds:
1 :  1      1  10     1  10     0   1       1   5     2   5    -1   3
2 :  2      1  10     1  10     0   1       2   5     1   5     1   3
3 :  3      2   5     3   5     1   4       1  10     7  10    -1  12
4 :  4      2   5     3   5     1   4       3  10     9  10     7  12
5 :  5      3   5     2   5     3   4       7  10     1  10    13  12
6 :  6      3   5     2   5     3   4       9  10     3  10     5  12
7 :  7      9  10     9  10     1   2       3   5     4   5     5   6
8 :  8      9  10     9  10     1   2       4   5     3   5     1   6

 Invariant:  2,  Type:  0
 Generating Bonds:
1 :  9      1  10     7  10    11  12      -1   5     3   5     7   6
2 : 10      1   5     2   5     2   3       1  10     7  10    11  12
3 : 11      3  10     9  10     7  12       2   5     6   5     1   3
4 : 12      2   5     1   5     1   3       7  10     1  10     1  12
5 : 13      3   5     4   5     5   6       3  10     9  10     7  12
6 : 14      7  10     1  10     1  12       3   5    -1   5    -1   6
7 : 15      4   5     3   5     1   6       9  10     3  10     5  12
8 : 16      9  10     3  10     5  12       6   5     2   5     2   3
...
```

We can see that a total of 120 graph invariants were generated for this system. Three of those invariants are first-order. For each of the first two invariants, there happen to be eight generating bonds, specified on each line following the text "Generating Bonds:". The first integer on those lines is an index counting the number of generating bonds for each invariant. After the colon on each line, there is an index for the generating bond, after that the corresponding H-bond definition. From the data given above, application of the group theoretical projection operator to bonds 1-8 generate invariant 1, while application of the projection operator to bonds 9-16 generate invariant 2.

The format for second order graph invariants is similar as seen in the example below.

```
...
Invariant:  8,  Type:  2
Minimum Distance Generating Bond Pairs:
1 :  2   2.728    1 10     1 10     0  1       2  5     1  5     1  3
     3            2  5     3  5     1  4       1 10     7 10    -1 12
2 :  3   2.728    2  5     3  5     1  4       1 10     7 10    -1 12
     2            1 10     1 10     0  1       2  5     1  5     1  3
3 :  4   2.728    2  5     3  5     1  4       3 10     9 10     7 12
     8            9 10     9 10     1  2       4  5     3  5     1  6
4 :  6   2.728    3  5     2  5     3  4       9 10     3 10     5 12
     7            9 10     9 10     1  2       3  5     4  5     5  6
5 :  7   2.728    9 10     9 10     1  2       3  5     4  5     5  6
     6            3  5     2  5     3  4       9 10     3 10     5 12
6 :  8   2.728    9 10     9 10     1  2       4  5     3  5     1  6
     4            2  5     3  5     1  4       3 10     9 10     7 12
...
```

For each generating product of two bond variables, there are now two lines of descriptive information, one for each H-bond of a product which the group theoretical projection operator converts into a second order invariant. We only show output for the first six generating bond pairs above. "Type : 2" indicates that is a second-order defect-type invariant (page 11). The first generating bond product is b_2c_3. For second-order invariants, the program calculates the distance between the two H-bonds. Here distance between H-bonds is defined as the shortest of the four possible distances that can be formed between one of the vertices from one H-bond and one from the other H-bond. If the two H-bonds of the product generating the invariant share a common vertex, the distance is zero. When a covering cell calculation is requested, a file name "GenBondPairC.txt" is also generated.

Most readers may safely skip this paragraph, which describes a fine point that may only concern true graph invariant aficionados: Of course, all bond pairs generating a particular invariant should be symmetry related and be characterized by the same distance. Recall that, by convention, we assign H-bonds to unit cells according to the location of their first vertex. Sometimes according to this convention the members of a generating bond pair lie in different unit cells. However, the program always tabulates the representatives of those bonds that lie in the same unit cell. When it comes to calculating distances between generating bond pairs, those pairs with members that straddle two unit cells will have an *apparently* longer distance between them. Such pairs are simply deleted from the list given in "GenBondPair.txt." Only those invariants generated by bond pairs separated by the minimum apparent separation distance are written to the file. Hence, not all possible generating pairs are listed in "GenBondPair.txt." Instead, only those pairs that lie close to each other in a unit cell and are most conveniently visualized are listed. In this example, the minimum apparent distance between these two bonds is 2.728Å. As mentioned above, the units of length is determined from the input file.

3.4.9 BondPairs.txt

This file contains a list of bonds used to generate each of the graph invariants. It can be considered an abridged version of "GenBondPair.txt" in which only the first instance of generating bond(s) is listed for each invariant. This file can be renamed "BondPairsDef.txt" and read as input by *MkInvar* for a new calculation. (See discussion in section 3.3.5.) Typically, this file would be generated for a small cell and then used as input to generate small-cell invariants for a large simulation cell. Below is an portion of the "BondPairs.txt" file generated for the ice III unit cell.

```
120     # of invariants
 0 : Type
           1  10    1  10    0   1          1   5     2   5    -1   3
 0 : Type
           1  10    7  10   11  12         -1   5     3   5     7   6
 0 : Type
           1  10    7  10   11  12          1  10    11  10     1   1
 2 : Type
           1  10    1  10    0   1          1   5     2   5    -1   3
           1  10    1  10    0   1          1   5     2   5    -1   3
 2 : Type
           1  10    1  10    0   1          1   5     2   5    -1   3
           1  10    1  10    0   1          2   5     1   5     1   3
...
```

The first line contains the number of graph invariants for which generating bonds are given. For each invariant, there is one line of descriptive information indicating the type of invariant. Then follows one (two) lines giving the vertices of the generating bond(s). A "BondPairsC.txt" file should be generated when a covering cell calculation is requested.

3.4.10 Output for Enumeration

The following is a list of files generated as output from *MkInvar* that are used as input for the enumeration code, *GrEnum*. The format of these files and use of this code are discussed in section 5.4.

- Gv
- Bonds
- BondsC(Generated when a covering cell is used.)
- Gbonds
- Invar

4 *MkInvar* Tutorial

This tutorial will walk a user through a typical calculation, using the *MkInvar* code for the ice VII system. The tasks to be accomplished are:

1. Build the structure of the unit cell and calculate all graph invariants.
2. Repeat the procedure for two larger cells containing 16 and 32 water molecules respectively.
3. Using graph invariants from the 16-water cell, we generate invariants for a large simulation cell containing 1024 water molecules.

We will then consider a couple of other cases, including a disscussion on covering cell calculations, in an attempt to explore all possibilities that can currently be handled with the *MkInvar* program.

4.1 Unit cell of ice VII

We begin by constructing the three input files to perform calculations on the ice VII system.

- The unit cell of ice VII is cubic with space group #224($Pn\bar{3}m$) and contains two water molecules. All symmetry elements of this space group can be generated from six symmetry operations. The full group contains a total of 48 symmetry elements which will be generated by *MkInvar*. We use the first origin choice and construct the space group file "224.1" like so. This file can be found in the "SpaceGroups" directory.

```
224 (Pn-3m)
6 48
cub
 1 0 0 0 1 0 0 0 1
 0 1 0 1 0 1
 -1 0 0 0 -1 0 0 0 1
 0 1 0 1 0 1
 -1 0 0 0 1 0 0 0 -1
 0 1 0 1 0 1
 0 0 1 1 0 0 0 1 0
 0 1 0 1 0 1
 0 1 0 1 0 0 0 0 -1
 1 2 1 2 1 2
 -1 0 0 0 -1 0 0 0 -1
 1 2 1 2 1 2
```

- We name the main input file "MkInvar.inp" just as we have been doing all along. We will first perform calculations on the primitive unit cell and so we set all of the m_j's equal to

one. The matrix of cell vectors, **c** is equal to the identity matrix. We will (arbitrarily) place the space group file in the directory just above where we are currently working. The lattice parameters for this cubic system are in units of Ångström and taken from the literature.[21] We will be asking *MkInvar* to generate H-bonds so that we only need to give a list of vertices for the asymmetric unit, supplied in the "VertDef.txt" input file. We will generate all graph invariants possible for this small unit cell. A covering cell will not be necessary for this calculation.

```
 1 1 1                        (Simulation cell dimensions)
      1 1   0 0   0 0         (Matrix of cell vectors as row vectors)
      0 0   1 1   0 0
      0 0   0 0   1 1
224 1                         (Space Group Index)
../../SpaceGroups             (Path to space group database)
3.337 3.337 3.337 90 90 90    (Lattice Parameters: a,b,c,alpha,beta,gamma)
0.1                           (Minimum allowed OO distance)
2.50 3.13                     (Min/Max allowed OO distance for H-Bonds)
0                             (0: Generate H-bonds / 1: Read H-bonds)
1                             (0: No Invar / 1: All / 2: Some)
1                             (0: User specified covering cell / 1: Auto.)
```

- The final input file is "VertDef.txt" which contains the vertices of the asymmetric unit for the ice VII unit cell. In this case, there is only one oxygen which is located at the origin.

```
1   # of vertices in asymmetric unit
0 1 0 1 0 1
```

`MkInvar`

Upon successful completion of a calculation six files will be generated as output:

1. Group_Unit.txt
2. Structure_Unit.txt
3. Vert_Unit.xyz
4. Bond_Unit.xyz
5. Bond_Unit_Wrap.xyz
6. MkInvar.log: Note that this file is only written to if verbose output is turned on. Any error messages generated during the calculation are also written to this file.

The first two files contain information on the symmetry group and structure generated by the code before it exits. The next three files contain the coordinates for vertices and H-bonds to visualize with another program. The last file is a log file which contains information about errors the program encountered while running.

With the two input files in our current directory and the location of the space group file specified, we are ready to run the code. *An extra line is needed at the end of each of the input files in order for the program to run successfully.* With the executable in the same directory,simply type "MkInvar" to run the code. With the current input files, the *MkInvar* code will exit without successfully calculating the structure and invariants. This is an *expected* result because we have not supplied enough information to properly construct the structure of the ice VII unit cell, as we will soon discuss.

Viewing the file "Bond_Unit_Wrap.xyz", shown below in Figure 4 (page 37), and reading the end of the "MkInvar.log" file should make it obvious why the program terminated, but we'll get to that in a little bit. The last file, if generated, contains all output generated by the *MkInvar* program. If we look through the file "MkInvar.log," we can follow what the *MkInvar* code is doing at each stage.

1. Read input files and initialize.
2. Generate full symmetry group.
3. Generate all possible vertices.
4. Use vertices to generate all possible H-bonds.
5. Recognize problem with structure, write coordinates to .xyz files, and terminate.
6. Analyze structure and create "HBondsDef.txt".
7. Run *MkInvar* a second time.
8. Generate graph invariants.

When the program terminates after step 5, we will have to create a new input file using output generated by the code. We will then have to run a second calculation to successfully generate graph invariants. We'll discuss each of these steps in detail below.

4.1.1 Read input files and initialize

In the "MkInvar.log" file, we will see the following output, most of which is simply restating what was given in the input files.

```
+++ Entering Subroutine ( ReadInput ) ***
 Simulation Cell:  1 1 1
  --This is a unit cell

 Matrix of Cell Vectors:
  1.00  0.00  0.00
  0.00  1.00  0.00
  0.00  0.00  1.00

 Primitive Cell?  T

 Inverse of Matrix of Cell Vectors:
  1.00  0.00  0.00
  0.00  1.00  0.00
  0.00  0.00  1.00
```

The simulation cell dimensions were read which in this case corresponds to the unit cell. The matrix of cell vectors was read where it was determined that this is a primitive cell, i.e. the matrix **c** from section 2 is the identity matrix. The inverse of the matrix of cell vectors was then calculated.

Next, information about the space group is reported.

```
 SpaceGroup Index:               224
 SpaceGroup Convention:            1

+++ Entering Subroutine ( Get_SpaceGroup_FileName ) ***

 Path to SpaceGroup database: ../../SpaceGroups
 Space Group read from file:  224.1
 Full path to file:           ../../SpaceGroups/224.1

--- Exiting  Subroutine ( Get_SpaceGroup_FileName ) ***

 Reading space group file...
                           Successful

 Unique denominators of Trans. Vectors:  1 2
 Lowest Common Denominator of Trans. Vectors:  2
```

The space group index was read from the input file and this will be used to compare against the index given in the space group file, the location of which should be the directory above the current directory. The contents of the space group file were then successfully read. Next, the lowest common denominator(LCD) for components of the translation vectors of the symmetry

operations was determined. Use of this quantity will allow *MkInvar* to use integer arithmetic for the construction of the symmetry group, structure, and graph invariants. Next, the lattice parameters are read and the symmetry is compared to that given in the space group file, in this case cubic.

```
+++ Entering Subroutine ( Check_LatticeParam ) ***

 Lattice Constants
 Length of vector 1(a):                 3.337000
 Length of vector 2(b):                 3.337000
 Length of vector 3(c):                 3.337000
 Angle between vectors 2 & 3(Alpha):   90.000000
 Angle between vectors 1 & 3(Beta):    90.000000
 Angle between vectors 1 & 2(Gamma):   90.000000

 a == b?                        T
 a == c?                        T
 b == c?                        T
 a == b == c?                   T
 Alpha == 90?                   T
 Beta  == 90?                   T
 Gamma == 90?                   T
 Gamma == 60?                   F
 Gamma == 120?                  F
 Alpha == Beta == 90?           T
 Alpha == Gamma == 90?          T
 Alpha == Beta == Gamma?        T
 Alpha == Beta == Gamma == 90?  T

 Cell Type Found:               cub
 ORTHOGONAL(ORTHOG)?            T

--- Exiting  Subroutine ( Check_LatticeParam ) ***
```

The lattice constants were read from "MkInvar.inp" and the lattice type is determined to be cubic which must agree with the lattice type contained in the space group file. As a note, the test of whether two real numbers are equivalent involves testing if the two numbers differ by some tolerance. The value for this tolerance is initialized in the "param.inc" file at compile time. The current value for "TOL_ZERO" is 1×10^{-10}. In this crystal system, the lattice vectors are orthogonal and this permits the use of some optimized routines.

```
+++ Entering Subroutine ( Generate_LatticeVector ) ***

 Angles(Radians):
 Alpha:           1.570796
 Beta:            1.570796
 Gamma:           1.570796

 Cos(alpha):      0.000000

 Sin(beta):       1.000000
 Cos(beta):       0.000000

 Sin(gamma):      1.000000
 Cos(gamma):      0.000000
 Tan(gamma):  0.222734E+11

--- Exiting  Subroutine ( Generate_LatticeVector ) ***

 Matrix of Unit Cell Lattice Vectors:
    3.33700    0.00000    0.00000
    0.00000    3.33700    0.00000
    0.00000    0.00000    3.33700

 Inverse of Unit Cell Lattice Vectors:
    0.29967    0.00000    0.00000
    0.00000    0.29967    0.00000
    0.00000    0.00000    0.29967
```

Using the lattice parameters, lattice vectors, $\mathbf{L}_j$'s in section 2, for the unit cell are constructed. These lattice vectors will used with the fractional coordinates to generate atomic coordinates for the .xyz files.

```
 Minimum allowed OO distance:            0.100000
 Minimum allowed H-bonded OO distance:   2.500000
 Maximum allowed H-bonded OO distance:   3.130000
```

This portion is simply repeating what was present in the input file. Since the lattice parameters were given in Ångström, these distances must also be in units of Ångström. If when generating the structure, the program finds that the minimum image distance between any two oxygen atoms less than 0.1Å, then the program will terminate with a warning message. We only search for H-bonds where the oxygen atoms are separated by 2.50 – 3.13Å.

```
 Method to determine H-bonds(jobHBONDS):        0
 0: Read vertices from VertDef.txt.
 1: Read H-bonds from HBondsDef.txt.

 Method to generate invariants(jobGENINVAR):    1
 0: Do not generate invariants
 1: Generate all invariants.
 2: Read bond pairs from BondPairDef.txt.
```

We have asked for the program to generate H-bond definitions and so it will be expecting to find the file "VertDef.txt." We have also asked that all graph invariants for this cell be calculated.

```
+++ Entering Subroutine ( Read_Asym ) ***

 LCD for vertices, bonds, symmetry operations:  2

 # of vertices in asymmetric unit:              1
 Vertices(real*8)
      0.00000  0.00000  0.00000

--- Exiting  Subroutine ( Read_Asym ) ***
```

The file "VertDef.txt" was successfully read. The LCD from the vertex coordinates is compared with that from the symmetry elements. The LCD for all vertices, bonds, and symmetry elements will be used to scale all coordinates so integer arithmetic can be used. The fractional coordinates of the vertices in the asymmetric unit are written to output as real numbers. The user should confirm these coordinates are correct.

```
 Generators of symmetry group:
 # of generators:             6
  1  0  0     0  1  0     0  0  1        0   1     0   1     0   1
 -1  0  0     0 -1  0     0  0  1        0   1     0   1     0   1
 -1  0  0     0  1  0     0  0 -1        0   1     0   1     0   1
  0  0  1     1  0  0     0  1  0        0   1     0   1     0   1
  0  1  0     1  0  0     0  0 -1        1   2     1   2     1   2
 -1  0  0     0 -1  0     0  0 -1        1   2     1   2     1   2

--- Exiting  Subroutine ( ReadInput ) ***
```

After the generators are scaled, as a test, we undo the scaling and write them to output to see if we get back the original symmetry elements. The program has just finished reading all of the input files and will proceed with generating the full symmetry group.

4.1.2 Generate Group

```
+++ Entering Subroutine ( GenGroup_Unit ) ***
 Current Size of Group:            6
 Current Size of Group:           20
 Current Size of Group:           48
 Current Size of Group:           48
 Group has stopped growing.

 # of symmetry elements expected:          48
 # of symmetry elements found:             48

     Group written to Group_Unit.txt.

--- Exiting  Subroutine ( GenGroup_Unit ) ***
```

Repeated multiplication of the symmetry elements is performed until the total number of elements in the group is a constant. We can see that by the third iteration, the size of the group has converged to 48 which agrees with the number read from the space group file. The symmetry elements for the group were successfully written to output in the file "Group_Unit.txt."

4.1.3 Generate all possible vertices

```
+++ Entering Subroutine ( Generate_Vertices_Asym ) ***
 Asym Vert:  1 # of vertices:  2

 # of unique vertices in unit cell:  2

--- Exiting  Subroutine ( Generate_Vertices_Asym ) ***
```

For every vertex given in "VertDef.txt," symmetry operations of the full group are applied and all unique vertices generated are saved. In this system, application of the symmetry group generated two unique vertices. This is what is expected since the unit cell of ice VII contains only two water molecules.

4.1.4 Use vertices to generate all possible H-bonds

After generation of all vertices in the unit cell, *MkInvar* uses the distance criterion from the input file "MkInvar.inp" to identify all possible H-bonds. Again, there are two unique vertices in this system, one at the origin and one at the center of the cube.

```
+++ Entering Subroutine ( Generate_HBonds_Asym ) ***

 Min H-Bond Distance Tolerance:   2.5
 Max H-Bond Distance Tolerance:   3.13

 # of oxygens:  2
 Index     x          y          z        Coord.
 ------------------------------------------------
   1    0.00000    0.00000    0.00000        8
   2    0.50000    0.50000    0.50000        8

 # of H-Bonds:  8
             Vertex 1                Vertex 2           Oxygen
 Index   x      y      z         x      y      z        Indices
 ------------------------------------------------------------------
   1   0.00   0.00   0.00     -0.50  -0.50  -0.50       1     2
   2   0.00   0.00   0.00     -0.50  -0.50   0.50       1     2
   3   0.00   0.00   0.00     -0.50   0.50  -0.50       1     2
   4   0.00   0.00   0.00     -0.50   0.50   0.50       1     2
   5   0.00   0.00   0.00      0.50  -0.50  -0.50       1     2
   6   0.00   0.00   0.00      0.50  -0.50   0.50       1     2
   7   0.00   0.00   0.00      0.50   0.50  -0.50       1     2
   8   0.00   0.00   0.00      0.50   0.50   0.50       1     2

--- Exiting  Subroutine ( Generate_HBonds_Asym ) ***
 TERMINATE:  T
```

Using the distance criteria, eight possible H-bonds were identified. For each H-bond, the fractional coordinates of the oxygen atoms as well as the corresponding vertex indices are given. In our first attempt to run the code, because we used a distance criterion to identify H-bonds for the ice VII system, the H-bond coordination for each of the oxygens came out to be eight, more H-bonds than allowed. If any of the vertices are not four coordinated, the program terminates before attempting to calculate graph invariants. This is not always the case but in ice VII non-H-bonded oxygens are just as close as H-bonded vertices. We purposely chose an example where this problem would occur, so we could illustrate how the output generated to this point can be used to recover.

4.1.5 Recognize problem and terminate

Before terminating, the structure and atomic coordinates of the system in its present state is written to file.

```
 Writing Vert_Unit.xyz
 Writing Bond_Unit.xyz
 Writing Bond_Unit_Wrap.xyz

+++ Entering Subroutine ( Write_Structure ) ***

 Writing structure to file: Structure_Unit.txt

--- Exiting  Subroutine ( Write_Structure ) ***

 The program will be terminated because some
 oxygens are not four coordinated.  Using the
 file Structure_Unit.txt and the various .xyz
 files, one should be able to construct a file
 named HBondsDef.txt in which all oxygens are
 four coordinated.
```

The file "Structure_Unit.txt" as well as .xyz files for visualization are written as output. We can see the warning message that not all oxygens were four coordinated. At this point, we see a warning message that the program will soon be terminated.

```
+++ Entering Subroutine ( Orbit_Vertices ) ***

 Vertex:     Symmetry Related Vertices
      1:   1   2
      2:   1   2

--- Exiting  Subroutine ( Orbit_Vertices ) ***

+++ Entering Subroutine ( Orbit_HBonds ) ***

 Bond  :     Symmetry Related Bonds
      1:   1   4   6   7
      2:   2   3   5   8
      3:   2   3   5   8
      4:   1   4   6   7
      5:   2   3   5   8
      6:   1   4   6   7
      7:   1   4   6   7
      8:   2   3   5   8
 # of New H-bonds:  0


--- Exiting  Subroutine ( Orbit_HBonds ) ***

 The program will be terminated because vertices &
 bonds did not form complete orbits or oxygen atoms
 were not four coordinated.
```

Vertices and H-bonds were grouped together in orbits. An orbit is a group of objects all related by symmetry. We can see that all vertices in this system are related to one another by symmetry. From the eight H-bonds found, we see that there are two orbits. The first orbit contains H-bonds 1, 4, 6, and 7 while the second orbit contains H-bonds 2, 3, 5, and 8. This information along with the coordinate files generated should give us enough information to identify those H-bonds which do not belong.

4.1.6 Analyze structure and create "HBondsDef.txt"

By analyzing the structures generated by the code at the point, shown in Figure 4, we can identify those H-bonds which do not belong. In this case, we need to remove H-bonds 1, 4, 6, and 7. We can use the H-bond definitions for H-bonds 2, 3, 5, and 8 in "Structure_Unit.txt" to create a "HBondsDef.txt" input file.

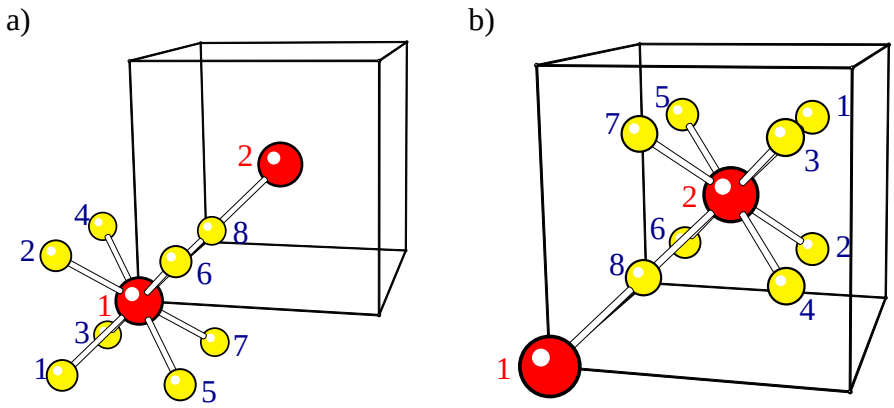

Figure 4: The structure of the ice VII unit cell generated from *MkInvar*. a) and b) are the structures contained in the files "Bond_Unit.xyz" and "Bond_Unit_Wrap.xyz" respectively. The red and blue numbers indicate the indices of the oxygen (red spheres) and hydrogen (yellow spheres) atoms, respectively. H-bonds 1, 4, 6, and 7, a complete orbit, will be removed to generate the correct structure for the ice VII unit cell.

```
4    # of HBonds
      0   1      0   1      0   1       -1   2     -1   2      1   2       1       2
      0   1      0   1      0   1       -1   2      1   2     -1   2       1       2
      0   1      0   1      0   1        1   2     -1   2     -1   2       1       2
      0   1      0   1      0   1        1   2      1   2      1   2       1       2
```

We can then use this input file to run the *MkInvar* program a second time. It should be pointed out that this step and the next are not necessary when the distance criterion is sufficient to correctly generate the H-bond structure.

4.1.7 Run *MkInvar* a second time

Now, we make the following change to line 10 of the input file "MkInvar.inp" so that we now use the H-bond definitions in "HBondsDef.txt" to generate the structure.

```
1                              0: Generate H-bonds / 1: Read H-bonds
```

We use the same command as before to run the program a second time.

```
MkInvar
```

This time, we will find that the code successfully completes all tasks. In addition to the previously written output files, which were just overwritten, the following additional files are generated:

1. GraphInvar.txt

2. GenBondPair.txt
3. BondPairs.txt

as well as input for the enumeration code, *GrEnum*:

1. Bonds
2. Gbonds
3. Gv
4. Invar

If we look through the new "MkInvar.log" file, we will see differences when the code begins to generate the structure.

```
+++ Entering Subroutine ( Generate_Vertices_HBonds ) ***
 Using H-Bonds to identify unique vertices

 Physical Oxygen Coordinates
   1     0.00     0.00     0.00
   2     1.67     1.67     1.67
```

Using the H-bond definitions, the *MkInvar* program has identified two unique vertices, consistent with the previous calculation. These are the physical coordinates in the same units as the lattice parameters. The vertices and H-bond definitions are given a second time in terms of fractional coordinates.

```
 Min H-Bond Distance Tolerance:   2.5
 Max H-Bond Distance Tolerance:   3.13

 # of oxygens:  2
 Index     x        y        z     Coord.
 ----------------------------------------
   1     0.00     0.00     0.00      4
   2     0.50     0.50     0.50      4

 # of H-Bonds Found:  4
              Vertex 1                  Vertex 2           Oxygen
 Index    x       y       z          x       y       z      Indices
 ------------------------------------------------------------------
   1    0.00    0.00    0.00      -0.50   -0.50    0.50     1   2
   2    0.00    0.00    0.00      -0.50    0.50   -0.50     1   2
   3    0.00    0.00    0.00       0.50   -0.50   -0.50     1   2
   4    0.00    0.00    0.00       0.50    0.50    0.50     1   2

--- Exiting  Subroutine ( Generate_Vertices_HBonds ) ***
 TERMINATE:  F
 Writing Vert_Unit.xyz
 Writing Bond_Unit.xyz
 Writing Bond_Unit_Wrap.xyz

+++ Entering Subroutine ( Write_Structure ) ***

 Writing structure to file: Structure_Unit.txt

--- Exiting  Subroutine ( Write_Structure ) ***
```

We can see that the two vertices are now four coordinated as expected. At this point, application of the symmetry elements to vertices and H-bonds in the cell should not generate any new vertices or H-bonds. The orbits for vertices and H-bonds are identified where we can confirm that no new vertices or H-bonds are generated by symmetry elements of the space group. Now that the structure has been successfully generated, symmetry properties of the H-bonds can be determined.

```
+++ Entering Subroutine ( Orbit_Vertices ) ***

 Vertex:     Symmetry Related Vertices
      1:   1   2
      2:   1   2

--- Exiting  Subroutine ( Orbit_Vertices ) ***

+++ Entering Subroutine ( Orbit_HBonds ) ***

 Bond  :     Symmetry Related Bonds
      1:   1   2   3   4
      2:   1   2   3   4
      3:   1   2   3   4
      4:   1   2   3   4
 # of New H-bonds:  0

--- Exiting  Subroutine ( Orbit_HBonds ) ***
```

4.1.8 Generate graph invariants

At this point, we have all of the information necessary to calculate graph invariants. In the input file, "MkInvar.inp", we specified that we wanted all possible invariants to be generated.

```
+++ Entering Subroutine ( GenInvariant ) ***
 Copying bonds from unit cell

+++ Entering Subroutine ( GenInvar_ALL ) ***

+++ Entering Subroutine ( Invar_Gen_ALL_B ) ***

 Generating 1st order invariants: b[j]
   # of invariants:  0

--- Exiting  Subroutine ( Invar_Gen_ALL_B ) ***
```

The first set of invariants that are generated are the first-order invariants, $\hat{G}(b_r)$. For this system, we find that all first-order invariants are identically zero. Next, second-order invariants are generated.

```
+++ Entering Subroutine ( Invar_Gen_ALL_BC ) ***

 Generating 2nd order invariants: b[j]c[k]
   # of invariants:  2

--- Exiting  Subroutine ( Invar_Gen_ALL_BC ) ***
```

The code first generates second-order graph invariants of the type $\hat{G}(b_r c_s)$, for which there are two in this small unit cell. Next, invariants of the type $\hat{G}(b_r b_s)$ are constructed.

```
+++ Entering Subroutine ( Invar_Gen_ALL_BB_from_BC ) ***

 Generating 2nd order invariants: b[j]b[k]

 Range of indices for b[j]c[k] invariants:  1   2
   # of invariants:  2

--- Exiting  Subroutine ( Invar_Gen_ALL_BB_from_BC ) ***

--- Exiting  Subroutine ( GenInvar_ALL ) ***

 Graph Invariants written to GraphInvar.txt.

 Writing generating bond pair info to BondPairs.txt.

--- Exiting  Subroutine ( GenInvariant ) ***
```

Again, due to the small size and high symmetry of the system, there are only two second-order graph invariants. All first- and second-order invariants have now been generated and written to file, "GraphInvar.txt". Generating bond pairs for these invariants are also written to file. The file "GenBondPair.txt" contains all generating bond pairs for each invariant. For second-order invariants, only generating bond pairs with the shortest distance between them are reported for each invariant. The first generating bond pair for each invariant is also reported in a second file, "BondPairs.txt". This file can immediately be used for additional calculations on other cell sizes.

4.1.9 Write Input for GrEnum

```
+++ Entering Subroutine ( Write_Enumeration_Input ) ***
 Writing file Bonds...
 Writing file Gv...
 Writing file Gbonds...
 Writing file Invar...

--- Exiting  Subroutine ( Write_Enumeration_Input ) ***
```

The last portion of the output file says that the *MkInvar* code generated the input files necessary to run the *GrEnum* code on this system.

4.2 $2 \times 2 \times 2$ simulation cell of ice VII

In the previous section, we went through a step-by-step discussion of how to use the *MkInvar* program to correctly generate the structure and graph invariants for the ice VII unit cell. In this section, we will use some of those output files to perform calculations on a slightly larger cell size. We will generate the structure and graph invariants for a 16-water cell measuring $2 \times 2 \times 2$ unit cells along each side. To do this calculation, we only need to make one small change to the "MkInvar.inp" file regarding the simulation cell dimensions.

```
 2 2 2                              (Simulation cell dimensions)
```

That is the only change we need to make! We will use the same "HBondsDef.txt" that we used in the previous calculation to correctly generate the ice VII structure. We use the same command as before

```
MkInvar
```

and will find that the calculation successfully finishes relatively quickly. Looking through the "MkInvar.log" file, we see essentially the same thing that we saw in the previous calculation with a few additions.

```
+++ Entering Subroutine ( GenGroup_Unit ) ***
 Current Size of Group:            6
 Current Size of Group:           20
 Current Size of Group:           48
 Current Size of Group:           48
 Group has stopped growing.

 # of symmetry elements expected:          48
 # of symmetry elements found:             48

     Group written to Group_Unit.txt.

--- Exiting  Subroutine ( GenGroup_Unit ) ***

+++ Entering Subroutine ( GenGroup_Sim ) ***
 Cell Dimensions:  2 2 2

 Expected size of group in simulation cell:  384
  -- Not accurate for unit cells that do not
     possess the full symmetry of the lattice.
```

Exactly as in the previous calculation, the full symmetry group is generated for the unit cell defined by the c_{kj} coefficients, as in Eqs. (4-5). Then, using the number of symmetry elements in the group and the simulation cell dimensions (specified by m_1, m_2, m_3), an estimate for the number of coset representatives in the group for the simulation cell is given. When the unit cell does not have the full symmetry of the lattice (that is, some of the coset representatives can move a vertex out of the unit cell), special complications arise. As noted in the output, this estimate may not be accurate. When the **c** matrix [Eq. (5)] is the identity matrix or a multiple of the identity matrix, the estimate of the number of coset representatives for the simulation cell is accurate.

```
 Matrix of Lattice Vectors for Simulation Cell:
  2.00  0.00  0.00
  0.00  2.00  0.00
  0.00  0.00  2.00

 Inverse of Matrix of Lattice Vectors:
  0.50  0.00  0.00
  0.00  0.50  0.00
  0.00  0.00  0.50

 Size of Group:  384
 Group written to Group_Sim.txt

--- Exiting  Subroutine ( GenGroup_Sim ) ***
```

The matrix of cell vectors for the simulation cell is calculated. Operating the translation group from the simulation cell on the full symmetry group from the unit cell generates the symmetry group for the simulation cell. The unit cell calculation is repeated and then using that information, the structure and symmetry group are constructed for the simulation cell. The *MkInvar* program finds 384 symmetry elements in the group which is eight times larger than the size of the group in the unit cell. This is expected since the simulation cell is eight times larger.

```
+++ Entering Subroutine ( Generate_Vertices_HBonds_Sim ) ***
 # of vertices in Simulation Cell:  16
 # of bonds in Simulation Cell:     32
  Constructing BondListSim...

--- Exiting  Subroutine ( Generate_Vertices_HBonds_Sim ) ***
```

Using the translation group from the simulation cell and H-bonds from the unit cell, the structure for the simulation cell can be generated. Again, we find that the number of vertices and H-bonds is eight times larger than in the unit cell. With the structure correctly generated, the code proceeds to calculate all possible graph invariants as instructed in the "MkInvar.inp" file. Again, first-order invariants are identically zero in this system and now there are eight invariants for each of the second-order invariant types. In addition to those files generated from the previous calculation, new output files are written containing information specific to the symmetry group and structure of the simulation cell.

1. Group_Sim.txt
2. Structure_Sim.txt
3. Vert_Sim.xyz

4. Bond_Sim.xyz

5. Bond_Sim_Wrap.xyz

As a side note, we could have performed this same calculation on the $2 \times 2 \times 2$ cell using a slightly different input file. Instead of changing the simulation cell dimensions, we could have changed the matrix of cell vectors like so.

```
1 1 1                       (Simulation cell dimensions)
     2 1   0 0   0 0        (Matrix of cell vectors as row vectors)
     0 0   2 1   0 0
     0 0   0 0   2 1
```

The code will generate exactly the same output as before, but without taking advantage of the translational symmetry. The *MkInvar* program will treat this system as if it were a unit cell ignoring the fact that this cell consists of $2 \times 2 \times 2$ unit cells. It is advantageous to use the translational symmetry of the system and only change the simulation cell dimensions in this case.

4.3 $5 \times 5 \times 5$ simulation cell of ice VII

In this example, we will again use information from calculations on smaller cells to generate the structure and invariants for a larger cell. This time, instead of calculating all possible invariants, we'll only generate those invariants that were present in the $2 \times 2 \times 2$ cell. Again, only a couple of changes to "MkInvar.inp" are necessary.

```
5 5 5                       (Simulation cell dimensions)
```

We specify that the simulation cell consists of $5 \times 5 \times 5$ unit cells along each side. We also select that we only want some of the graph invariants to be generated, not all of them.

```
2                           (0: No Invar / 1: All / 2: Some)
```

To only generate invariants from the $2 \times 2 \times 2$ cell, we just use the "BondPairs.txt" file that we created in the previous section. We rename this file as "BondPairsDef.txt." When only calculating some invariants, this is a necessary input file. Executing the code just like before generates output, similar to what we saw in the previous section, for this large simulation cell. Remember that any files created by the *MkInvar* program will overwrite any files that already exist. When the *MkInvar* program is ready to calculate invariants, we see the following in the "MkInvar.log" file.

```
+++ Entering Subroutine ( GenInvar_BP ) ***

 Generating Bond Pairs:
 # of 1st order invariants, b[j]:      0
 # of 2nd order invariants, b[k]b[k]:  8
 # of 2nd order invariants, b[k]c[k]:  8

 Invariants Generated:
 # of 1st order invariants, b[j]:      0
 # of 2nd order invariants, b[k]b[k]:  8
 # of 2nd order invariants, b[k]c[k]:  8

--- Exiting  Subroutine ( GenInvar_BP ) ***
```

There were a total of sixteen invariants for which the generating bond pairs were read from the file "BondPairsDef.txt". The simulation cell invariants, using these generating bond pairs, were constructed and written to the "GraphInvar.txt" output file.

4.4 Note on the Use of Covering Cells

Before moving on to the next couple of examples in this tutorial, we will first say a few things about the use of covering cells. For those cases where the simulation cell of interest does not have the full symmetry of the crystal lattice for the spacegroup, a covering cell must be used. The covering cell, which does have the full symmetry of the crystal lattice, consists of the simulation cell and its images, i.e. exact copies of the simulation cell related by translations. If an H-bond in the simulation cell takes a particular orientation, then that same H-bond in all of the images will be found in the same orientation. These constraints between the simulation cell and its images will be used to construct graph invariants and enumerate symmetry-distinct H-bond configurations.

When a covering cell is used, additional output files will be created. The letter 'C' will be appended to end of the filenames so as not to overwrite the corresponding files for the covering cell. One example is the "GraphInvar.txt" file. When a covering cell is requested, this file will contain graph invariants for the covering cell. The graph invariants for the smaller simulation cell will be found in the file "GraphInvarC.txt" indicating that a covering cell was used to generate this file.

4.5 $2 \times 2 \times 1$ simulation cell of ice VII

Structures and graph invariants can be generated for simulation cells that do not have the same symmetry as the unit cell. For these calculations, a covering cell, $2 \times 2 \times 2$, is first constructed which does have the same symmetry as the unit cell. Then, using the symmetry group and structure from this covering cell, graph invariants for the $2 \times 2 \times 1$ cell are calculated. An example of an input file for this calculation follows. (The "HBondsDef.txt" file created previously needs to be used)

```
 2 2 1                          (Simulation cell dimensions)
      1 1   0 0   0 0           (Matrix of cell vectors as row vectors)
      0 0   1 1   0 0
      0 0   0 0   1 1
224 1                           (Space Group Index)
../../SpaceGroups               (Path to space group database)
3.337 3.337 3.337 90 90 90      (Lattice Parameters: a,b,c,alpha,beta,gamma)
0.1                             (Minimum allowed OO distance)
2.50 3.13                       (Min/Max allowed OO distance for H-Bonds)
1                               (0: Generate H-bonds / 1: Read H-bonds)
1                               (0: No Invar / 1: All / 2: Some)
1                               (0: User specified covering cell / 1: Auto.)
```

For those cases where we only alter the simulation cell dimensions and not the matrix of cell vectors, the *MkInvar* code will automatically detect that a covering cell is necessary. A covering cell will be used anytime the simulation cell dimensions are not all identical. The code will automatically determine the size of the covering cell. We do not need to specify "0" to use a covering cell

for this case. Examining the output file "MkInvar.log," we notice some comments regarding the use of a covering cell.

```
+++ Entering Subroutine ( ReadInput ) ***
 Simulation Cell:             2 2 1

 A covering cell used for this calculation.
 Covering Cell Dimensions:    2 2 2
```

The *MkInvar* code automatically determined that the dimensions of the covering cell were $2\times2\times2$ unit cells along each side.

```
+++ Entering Subroutine ( GenGroup_Sim ) ***
 Cell Dimensions:  2 2 2
```

The covering cell is the cell for which the symmetry group is generated. The vertices and H-bonds for the $2 \times 2 \times 1$ system, the simulation cell, are generated and cartesian coordinates are written to file. We see that since the simulation cell is four times larger than the unit cell, there are four times as many vertices and H-bonds than present in the unit cell.

```
+++ Entering Subroutine ( Generate_Vertices_HBonds_Sim ) ***
 # of vertices in Simulation Cell:  8
 # of bonds in Simulation Cell:     16
  Constructing BondListSim...

--- Exiting  Subroutine ( Generate_Vertices_HBonds_Sim ) ***
 Writing Vert_Sim.xyz
 Writing Bond_Sim.xyz
 Writing Bond_Sim_Wrap.xyz

+++ Entering Subroutine ( Write_Structure ) ***

 Writing structure to file: Structure_Sim.txt

--- Exiting  Subroutine ( Write_Structure ) ***
```

The same is also done for the covering cell and we see that there are eight times as many vertices and H-bonds than present in the unit cell.

```
+++ Entering Subroutine ( Generate_Vertices_HBonds_Cov ) ***
 # of vertices in Covering Cell:  16
 # of bonds in Covering Cell:     32
  Constructing BondListCov...

--- Exiting  Subroutine ( Generate_Vertices_HBonds_Cov ) ***
 Writing Vert_Cov.xyz
 Writing Bond_Cov.xyz
 Writing Bond_Cov_Wrap.xyz

+++ Entering Subroutine ( Write_Structure ) ***

 Writing structure to file: Structure_Cov.txt

--- Exiting  Subroutine ( Write_Structure ) ***
```

If we were to inspect the next portion of the output file, we would find that the graph invariants were generated for the covering cell. Just as in the previous section, we would find that all first-order invariants are identically zero and there would be eight second-order invariants for each type. The *MkInvar* code then generates output for the *GrEnum* program.

```
+++ Entering Subroutine ( Write_Enumeration_Input ) ***
 Writing file Bonds...
 Writing file Gv...
 Writing file Gbonds...
 Writing file Invar...
 Writing file BondsC...

--- Exiting  Subroutine ( Write_Enumeration_Input ) ***
```

We see an additional output file was written. The "BondsC" file was generated because a covering cell was used. The contents of this file are discussed in more detail below, in section 5.4. As discussed below, the use of this file currently requires running the enumeration code *GrEnumC*. The final portion of the output file shows the generation of graph invariants for the simulation cell.

```
+++ Entering Subroutine ( GenInvariantCover ) ***

+++ Entering Subroutine ( GenInvar_Sim_from_Cover ) ***

 # of 1st order invariants, b[j]:      0
 # of 2nd order invariants, b[k]b[k]:  8
 # of 2nd order invariants, b[k]c[k]:  8

 # of invariants in simulation cell:   16
 These invariants may have algebraic linear dependencies.

--- Exiting  Subroutine ( GenInvar_Sim_from_Cover ) ***

 Graph Invariants written to GraphInvarC.txt.

 Writing generating bond pair info to BondPairsC.txt.

--- Exiting  Subroutine ( GenInvariantCover ) ***
```

Applying the H-bond constraints, discussed in the previous section, to graph invariants in the covering cell generates graph invariants for the simulation cell. Since all first-order invariants were identically zero in the covering cell, there are no first-order invariants in the simulation cell. We also find eight unique second-order invariants for each type in the simulation cell. As the note states, unlike the covering cell, the graph invariants for the simulation cell are not necessarily linearly independent.

Notice that the graph invariants and their generating bond pairs were written to the files "GraphInvarC.txt" and "BondPairsC.txt" respectively, as discussed in the previous section.

Instead of letting the *MkInvar* code automatically choose the covering cell for this calculation, we could have specified the covering cell of our choosing. The following input file would have accomplished the same task. Although these two input files generate different sets of output files, the same information for the $2 \times 2 \times 1$ cell is generated.

```
 2 2 2                        (Simulation cell dimensions)
      1 1   0 0   0 0         (Matrix of cell vectors as row vectors)
      0 0   1 1   0 0
      0 0   0 0   1 1
224 1                         (Space Group Index)
../../SpaceGroups             (Path to space group database)
3.337 3.337 3.337 90 90 90    (Lattice Parameters: a,b,c,alpha,beta,gamma)
0.1                           (Minimum allowed OO distance)
2.50 3.13                     (Min/Max allowed OO distance for H-Bonds)
0                             (0: Generate H-bonds / 1: Read H-bonds)
1                             (0: No Invar / 1: All / 2: Some)
0                             (0: User specified covering cell / 1: Auto.)
      2 1   0 0   0 0         (Matrix of cell vectors for covering cell )
      0 0   2 1   0 0
      0 0   0 0   1 1
```

4.6 $2\sqrt{2} \times 2\sqrt{2} \times 2$ simulation cell of ice VII

As described briefly above, calculations on non-primitive cells is also possible. In this example, calculations on an ice VII cell measuring $2\sqrt{2} \times 2\sqrt{2} \times 2$ unit cells on each side would be possible using the following input file:

```
 4 4 4                          (Simulation cell dimensions)
      1 1   0 0   0 0           (Matrix of lattice vectors as row vectors)
      0 0   1 1   0 0
      0 0   0 0   1 1
224 1                           (Space Group Index)
../../SpaceGroups               (Path to space group database)
3.337 3.337 3.337 90 90 90      (Lattice Parameters: a,b,c,alpha,beta,gamma)
0.1                             (Minimum allowed OO distance)
2.50 3.13                       (Min/Max allowed OO distance for H-Bonds)
0                               (0: Generate H-bonds / 1: Read H-bonds)
1                               (0: No Invar / 1: All / 2: Some)
0                               (0: User specified covering cell / 1: Auto.)
      2 1  -2 1   0 0           (Matrix of cell vectors for covering cell )
      2 1   2 1   0 0
      0 0   0 0   1 1
```

We are using a $4 \times 4 \times 4$ covering cell to do calculation on this tetragonal system.

An input file that doesn't use a covering cell follows.

```
 2 2 2                          (Simulation cell dimensions)
      1 1  -1 1   0 0           (Matrix of lattice vectors as row vectors)
      1 1   1 1   0 0
      0 0   0 0   1 1
224 1                           (Space Group Index)
../../SpaceGroups               (Path to space group database)
3.337 3.337 3.337 90 90 90      (Lattice Parameters: a,b,c,alpha,beta,gamma)
0.1                             (Minimum allowed OO distance)
2.50 3.13                       (Min/Max allowed OO distance for H-Bonds)
0                               (0: Generate H-bonds / 1: Read H-bonds)
1                               (0: No Invar / 1: All / 2: Some)
1                               (0: User specified covering cell / 1: Auto.)
```

4.7 Common Error Messages

- There should be at least one new line at the end of each input file. Error messages, like the one that follows, can be remedied by simply placing a new line at the end of the file, in this

case, the “HBondsDef.txt” file.

```
At line 661 of file ReadWrite.f (unit = 1, file = 'HBondsDef.txt')
Fortran runtime error: End of file
```

5 *GrEnum*

5.1 Compilation

The *GrEnum* program is written in a Fortran 77/90 mix. We have found that commonly available Fortran 90 compilers are sufficient. Also necessary is a set of math libraries, such as LAPACK and BLAS, to call the DPOSVX subroutine to solve a system of linear equations. To compile the code, one simply goes to the source directory and types "make" on the command line making sure that the correct path to the math libraries is used. This should generate an executable named "GrEnum."

5.2 Running *GrEnum*

The following command is used to run the program.

```
GrEnum
```

In total, there is a minimum of five input files necessary to run the program. Most of these input files contain information on the H-bonds, symmetry operations, and graph invariants and are generated as output from the *MkInvar* program. We discuss the contents and format of each of these files in turn. All output related to the progress of the *GrEnum* program is written to the "Log" file.

At any point during the calculation, the program will stop if it detects the presence of a text file named "EXIT" in the working directory. This will cause the program to exit in a controlled fashion after finishing the current symmetry comparison. Note that no temporary files will not be removed by the program in this case.

5.3 GrEnumC

The *GrEnumC* code is a modified version of the *GrEnum* code to handle covering cell calculations. It is compiled in the same directory as the *GrEnum* code by using the following command:

```
make GrEnumC
```

This code runs in exactly the same way as the *GrEnum* code except that it requires one extra input file.

5.4 Input Files

With the exception of "GrEnum.inp", all of these input files are generated as output from the *MkInvar* code. All of these files essentially contain lists of integers used to define the mapping of vertices and H-bonds when acted on by symmetry operations.

5.4.1 GrEnum.inp

This input file initializes a number of parameters and controls the volume of output written to the output file "Log."

```
     6              nInvForDeal
   300              InvariantThreshold0
   300              InvariantThreshold
 20000              maxInvariantSampleSize
 .true.             UseInvariants
.false.             VERBOSE
.false.             DEBUG
 .true.             Reorder H-bonds
```

Only the first entry on each line is read by the *GrEnum* code while all text that follows is ignored. The first three lines contain integers while the last four contain logical statements. A description for each line of the input file follows.

1. The maximum number of graph invariants used to sort H-bond configurations.
2. The minimum number of configurations required to implement the invariant procedure in order to eliinate symmetry-equivalent configurations.
3. The maximum number of H-bond configurations allowed in a temporary "XX###" file before symmetry comparison. Note that symmetry comparison on files containing more configurations than this threshold is possible if **nInvForDeal** invariants is not sufficient to meet the threshold requirement.
4. The maximum number of H-bond configurations used to sample graph invariants.
5. This controls whether graph invariants are used to sort configurations.
6. This increases the amount of output written to the "Log" file. When this option is turned on, verbose output regarding the input files and initialization is written to file. Information regarding the sampling of invariants is also written.
7. This further increases the level of output written to the "Log" file. This option controls whether detailed output related to the dealing of H-bond configurations is written to output.
8. This is an option to rearrange the order in which H-bonds are added during the enumeration. The current algorithm adds H-bonds to maximize the number of vertices that are four-coordinated at each step. The order in which H-bonds are added during the enumeration has a significant impact on the efficiency of the calculation.

5.4.2 Bonds

For each of the H-bonds in the cell, the indices of the vertices are given. The convention is taken that H-bonds point from the first oxygen towards the second oxygen. This orientation is assigned the bond variable +1. The bond variable for the opposite orientation is assigned the value −1. The following is an example from the ice III unit cell.

```
 1 2
 1 3
 4 5
 4 6
 7 8
 7 9
 10 11
 10 12
 5 12
...
```

This is exactly the same information given in the file "Structure_Unit.txt" for the H-bond definitions. In this example, the first H-bond is taken to point from vertex 1 towards vertex 2.

5.4.3 Gv

This file contains the indices for the vertices permuted by the symmetry operations. Each line corresponds to a symmetry operation and each column is associated with a vertex. The following is an example from the ice III unit cell.

```
  1    2    3    4    5    6    7    8    9   10   11   12
 10   12   11    7    9    8    4    6    5    1    3    2
  4    5    6   10   12   11    1    3    2    7    8    9
  4    6    5    1    3    2   10   12   11    7    9    8
  7    9    8    1    2    3   10   11   12    4    6    5
  7    8    9   10   11   12    1    2    3    4    5    6
 10   11   12    4    6    5    7    9    8    1    2    3
  1    3    2    7    8    9    4    5    6   10   12   11
```

In this example, there are eight symmetry elements (more precisely, coset representative) in the space group for the ice III unit cell which contains twelve water molecules. By inspection, we can see that the first symmetry element corresponds to the identity operation.

5.4.4 Gbonds

This file is similar to the file "Gv" except it contains the indices of H-bonds generated when acted upon by symmetry operations.

5.4.5 Invar

This file contains the coefficients and bond indices for second-order graph invariants of the type $\hat{G}(b_r b_s)$. The first line contains two integers, the number of invariants and the total number of terms that will be read from this file. In the example, from the ice III unit cell, there are forty-five second-order graph invariants and 300 terms in total.

```
45 300
8
1 1 1
2 2 1
3 3 1
4 4 1
5 5 1
6 6 1
7 7 1
8 8 1
4
1 2 1
3 4 1
5 6 1
7 8 1
...
```

For each invariant, the first line indicates the number of terms. For every term, there is a line which gives the bond indices and coefficient. In this example, the first invariant contains eight terms while the second invariant has four. For each term in the invariant, the first two integers are the bond indices and the third integer is the coefficient. In mathematical form, these two invariants are written as:

$$\hat{G}(b_1 b_1) = b_1 b_1 + b_2 b_2 + b_3 b_3 + b_4 b_4 + b_5 b_5 + b_6 b_6 + b_7 b_7 + b_8 b_8 \quad (21)$$

$$\hat{G}(b_1 b_2) = b_1 b_2 + b_3 b_4 + b_5 b_6 + b_7 b_8 \quad (22)$$

The normalization of invariants is not necessary for the enumeration algorithm.

5.4.6 RESTART

This file contains a single integer specifying which of the restart files will be read to restart an enumeration. At the beginning of a calculation, the *GrEnum* code checks for the presence of this file. If the file is found, the integer is read, and the H-bond configurations from the corresponding "XFile.##.rst" file is read. The calculation then proceeds as normal. For example, if we wanted to restart a calculation starting on the 50th H-bond, the "RESTART" file would simply contain the number "50" followed by a blank line.

5.4.7 BondsC

This is the one additional input file required for the enumeration of a system when a covering cell is used. This file contains the bond constraints that were discussed in

section 4.6. The following is a portion of the "BondsC" file.

```
32 # of Groups
 8 # of bonds in Group  1
 1
 5
 9
 13
 161
 165
 169
 173
 8 # of bonds in Group  2
 2
 6
 10
 14
 162
 166
 170
 174
 ...
```

For each H-bond in the cell of interest, there is a group of H-bonds constrained to be similarly oriented. In the cell of interest, there are 32 H-bonds and thus 32 groups of H-bond constraints. For this system, the covering cell contains eight replicas of the cell of interest. For each group, the first line indicates the number of H-bond constraints. This is then followed by the indices for H-bonds constrained to be similarly oriented.

5.5 Output Files

5.5.1 Log

This file contains all output generated as the enumeration calculation proceeds. The amount of output is controlled by the **VERBOSE** and **DEBUG** variables initialized in the "GrEnum.inp" file.

5.5.2 XFile

This output file contains the bond variables for all symmetry-distinct H-bond configurations. This file is only generated after successful completion of the program. For every configuration, there is a line of bond variables, one for each H-bond. Below is an example for the unit cell of ice Ic containing eight water molecules. There are only four symmetry-distinct configurations.

```
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  1 -1  1 -1 -1  1 -1  1
 1  1  1  1  1  1  1 -1 -1  1  1 -1  1  1 -1  1
 1  1  1  1  1 -1  1 -1 -1  1 -1  1  1  1  1  1
```

5.5.3 XFile.##.rst

After the successful completion of adding an H-bond and performing the symmetry comparison, the current version of "XFile" is copied to "XFile.##.rst" to assist in restarting the enumeration. For example, before the eighth H-bond is added, "XFile" is copied to "XFile.7.rst." This file contains all configurations enumerated up to and including the seventh H-bond.

5.5.4 Temporary Files

During the course of the enumeration calculation, there are a large number of temporary files generated. All of these temporary files are written as binary and are not human-readable. They are all removed automatically by the *GrEnum* program after successful completion. Two of those files are named "XFile" and "XFile1." The other files, potentially thousands of them, all have names starting with "XX" followed by a sequence of numbers. In the event that the *GrEnum* program is terminated before a successful calculation, a short script named "job_RemoveXX" is available in the /tools directory to assist in removing all of these temporary files.

6 *GrEnum* Tutorial

In this section, we will walk through a typical use of the *GrEnum* code as applied to the ice III system.

6.1 Running the code

We begin by constructing the four input files necessary to perform the enumeration calculation on an arbitrary ice system: "Bonds", "Gv", "Gbonds", and "Invar." We can easily generate these files by using the *MkInvar* program described above. They are standard output files for the unit cell of a system.

6.2 MkInvar input files

We begin by constructing the three input files needed by "MkInvar" for ice III.

- Create a file named "MkInvar.inp" with the following information;

```
 1 1 1                        (Simulation cell dimensions)
      1 1   0 0   0 0         (Matrix of cell vectors as row vectors)
      0 0   1 1   0 0
      0 0   0 0   1 1
92 1                          (Space Group Index)
../../SpaceGroups             (Path to space group database)
6.666 6.666 6.936 90 90 90    (Lattice Parameters: a,b,c,alpha,beta,gamma)
0.1                           (Minimum allowed OO distance)
2.50 3.13                     (Min/Max allowed OO distance for H-Bonds)
1                             (0: Generate H-bonds / 1: Read H-bonds)
1                             (0: No Invar / 1: All / 2: Some)
1                             (0: User specified covering cell / 1: Auto.)
```

- Create a file named "VertDef.txt" with the following information;

```
2   # of vertices in assymmetric unit
2 5 1 5 1 3
1 10 1 10 0 1
```

- Create a file named "HBondsDef.txt" with the following information;

```
24   # of bonds in unit cell
1 10 1 10 0 1 1 5 2 5 -1 3
1 10 1 10 0 1 2 5 1 5 1 3
2 5 3 5 1 4 1 10 7 10 -1 12
2 5 3 5 1 4 3 10 9 10 7 12
3 5 2 5 3 4 7 10 1 10 13 12
3 5 2 5 3 4 9 10 3 10 5 12
9 10 9 10 1 2 3 5 4 5 5 6
9 10 9 10 1 2 4 5 3 5 1 6
1 10 7 10 11 12 -1 5 3 5 7 6
1 5 2 5 2 3 1 10 7 10 11 12
3 10 9 10 7 12 2 5 6 5 1 3
2 5 1 5 1 3 7 10 1 10 1 12
3 5 4 5 5 6 3 10 9 10 7 12
7 10 1 10 1 12 3 5 -1 5 -1 6
4 5 3 5 1 6 9 10 3 10 5 12
9 10 3 10 5 12 6 5 2 5 2 3
1 10 7 10 11 12 1 10 11 10 1 1
1 5 2 5 2 3 3 5 2 5 3 4
3 10 9 10 7 12 -1 10 9 10 1 2
2 5 1 5 1 3 2 5 3 5 1 4
3 5 4 5 5 6 3 5 2 5 3 4
7 10 1 10 1 12 11 10 1 10 0 1
4 5 3 5 1 6 2 5 3 5 1 4
9 10 3 10 5 12 9 10 -1 10 1 2
```

Running the *MkInvar* executable with these three files will generate the necessary input for "GrEnum". Place "Bonds" , "Gv" , "Gbonds" and "Invar" in a working directory. It is recommended that this directory should be some type of local scratch space. The algorithm that *GrEnum* uses to make enumeration an $O(N \ln N)$ process can potentially involve the reading and writing of thousands of scratch files. These files are created and removed by the program, provided the program exits successfully. Assuming the *GrEnum* executable is in the same directory with the VEBOSE option turned on, we type the following command to run the program.

```
./GrEnum
```

Checking the contents of the directory, one should be able to see the presence of the "Log", "XFile", "XFile1", and several "XFile.##.rst" files as well as a number of scratch files with names starting with "XX". It is best not to alter any of these temporary files while the program is running.

In the current version of the code, here are typical timings seen for enumerating configurations in phases of ice.

- For systems containing 10−20 water molecules, typical running times can range from seconds to several minutes.
- The enumeration of systems containing >30 water molecules can last from a few hours to a couple days.

These timings are typical for the linux workstations in our group with Intel or AMD processors. Of course, the exact timings depend on available hardware and values for the adjustable parameters within the code, which are discussed below in section 6.5. Progress is being made to develop an internal optimization strategy that determines the best values for the adjustable parameters "on the fly."

6.3 The Log file

In this section, we will go through and explain the output found in the "Log" file that one will see in a typical use of the *GrEnum* program. We will be examining the enumeration of the ice III unit cell which contains 12 water molecules. We will find that there are 102 symmetry-distinct H-bond configurations in this system.

6.3.1 Read input and initialize

The initial portion of the "Log" file lists the type of integers being used in the current version of the code.

```
Log File


Short integer is kind  2.
Invariant values have kind  4.
```

These values are probably not of interest to a typical user, but as the upper limits of system size that can be handled by the code increases, these values may need to be changed. The first thing that the *GrEnum* program does is read each of the input files. We can see that the first input file read was the "Bonds" file.

```
          24  bonds read,           12 vertices identified.
(  1,  2)(  1,  3)(  4,  5)(  4,  6)(  7,  8)(  7,  9)( 10, 11)
( 10, 12)(  5, 12)(  2,  5)(  6,  3)(  3,  8)( 11,  6)(  8, 11)
( 12,  9)(  9,  2)(  5,  1)(  2,  7)(  6, 10)(  3,  4)( 11,  7)
(  8,  1)( 12,  4)(  9, 10)(
```

We can see that the first H-bond is between vertices 1 and 2. The next input file that is read is the "Gv" file. In this file, the permutation of the vertices by the symmetry operations of the full group are given.

```
           8  group elements read.
  1   2   3   4   5   6   7   8   9 10 11 12
 10  12  11   7   9   8   4   6   5   1   3   2
  4   5   6  10  12  11   1   3   2   7   8   9
  4   6   5   1   3   2  10  12  11   7   9   8
  7   9   8   1   2   3  10  11  12   4   6   5
  7   8   9  10  11  12   1   2   3   4   5   6
 10  11  12   4   6   5   7   9   8   1   2   3
  1   3   2   7   8   9   4   5   6  10  12  11
```

In this system, there are only eight symmetry operations in the space group for the ice III unit cell. Next, we see the model that will be used to determine allowed H-bond configurations.

```
  Constraints on H-bonds
            max        max        min
vertex   in-bonds  out-bonds out-bonds
    1           2          2          2
    2           2          2          2
    3           2          2          2
    4           2          2          2
    5           2          2          2
    6           2          2          2
    7           2          2          2
    8           2          2          2
    9           2          2          2
   10           2          2          2
   11           2          2          2
   12           2          2          2
```

In this model for ice, the "ice rules" are strictly enforced. This means that in every H-bond configuration, all water molecules will donate and accept two H-bonds. Any configurations that violate these rules are rejected by program. When using the *GrEnum* program to enumerate clusters of water molecules, a different model will be used. (We plan to clean up our code for finite water clusters and make it public as well.)

The next input file, "Invar", contains the graph invariants.

```
   45 invariants read
 inv      bonds    coef
   1      1   1      1
   1      2   2      1
   1      3   3      1
   1      4   4      1
   1      5   5      1
   1      6   6      1
   1      7   7      1
   1      8   8      1
   2      1   2      1
   2      3   4      1
...
```

For the ice III unit cell, there are 45 unique second-order invariants of the type $\hat{G}(b_r b_s)$. This is the only type of graph invariant that is currently used to enumerate H-bond configurations. The *MkInvar* program only includes this type of invariant in the "Invar" file. The four numbers on each line indicate the invariant index, H-bond indices, and the coefficient for each term.

Next, the program defines the convention that will be taken for defining the bond variables.

```
  Bond look-up table
vertices bond (sign)         vertices  bond (sign)
   2   1    1 (-1)             1   2     1 ( 1)
   3   1    2 (-1)             1   3     2 ( 1)
   4   3   20 (-1)             3   4    20 ( 1)
   5   1   17 ( 1)             1   5    17 (-1)
   5   2   10 (-1)             2   5    10 ( 1)
...
```

Each line contains the vertices, bond index, and value of the bond variable for both orientations of the H-bond. The H-bond definitions read from the file "Bonds" will be taken as the orientations which define bond variables equal to +1. This orientation of H-bonds will be referred to as the canonical orientation of H-bonds. Those H-bonds in an orientation opposite to this definition are assigned bond variable equal to −1. Next, the *GrEnum* program identifies those pairs of oxygen atoms that should not form H-bonds with one another.

```
  These vertices should not be connected
vertices bond (sign)         vertices  bond (sign)
   1   1    0 ( 0)             1   1     0 ( 0)
   2   2    0 ( 0)             2   2     0 ( 0)
   3   2    0 ( 0)             2   3     0 ( 0)
...
```

The last input file read is "Gbonds" which contains the permutations on the H-bonds due to the symmetry operations of the space group.

```
  Permutation group on bonds
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1

  8  7  6  5  4  3  2  1 16 15 14 13 12 11 10  9 24 23 22 21 20 19 18 17
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
...
```

For each symmetry operation, there are two lines of output. The first line indicates the index of the H-bond after application of the symmetry operation. The second line indications the orientation of the H-bond relative to the canonical orientation. From inspection, we can see that the first symmetry element corresponds to the identity operation.

```
CPU time:            0.0089980 setup
```

After the *GrEnum* program has finished reading input files and initializing, it reports the amount of time that has passed since the program started. The timings for various operations during the calculation will be reported and explained below.

6.4 Enumeration

In this section, we will explain the output found in the "Log" file during the enumeration of H-bonds configurations. Similar output will be seen for every H-bond in the system. We start by looking at the output generated after the first H-bond was added to the system.

```
----------     1   ----------
1  configurations from previous bond
Before checking for distinct configs      2 configs enumerated for bond 1
CPU time:            0.0000000 add bonds at bond   1
```

To start the enumeration process, we insert the the first H-bond in each of the two orientations into all previous configurations. In this case, there is only a single configuration which has no H-bonds. This generates two different H-bonds configurations. At this point, any H-bond configurations that violated the model, in this case the "ice rules," would have been eliminated. Since both H-bond configurations are do not violate the model, they will both be included in the next step. This stage of the enumeration calculation is so quick, we can ignore the output from the timing routines.

```
CPU time:            0.0000000 symmetry search w/o invariants at bond   1
Bond   1:        2 ->       2 by elimination of symmetry-related configs.
total CPU time:           0.0000000 bond   1
```

At this stage, H-bond configurations will be sorted into groups using graph invariants only if there are enough configurations present. It would be common to only use the sorting algorithm when there are more than 1000 H-bond configurations, when the $O(N^2)$ symmetry comparison starts to become costly. The parameter that controls when configurations are sorted is discussed in more detail below in section 6.5 At this early stage of the calculation, there are not enough configurations present to take advantage of the sorting algorithm. Using the symmetry comparison algorithm on these two configurations, we find that they are not related by symmetry, thus at this stage they are symmetry-distinct. Having generated all symmetry-distinct configurations possible after adding the first H-bond, we now add the second H-bond.

```
----------     2   ----------
2  configurations from previous bond
Before checking for distinct configs      4 configs enumerated for bond 2
CPU time:            0.0000000 add bonds at bond   2
CPU time:            0.0010000 symmetry search w/o invariants at bond   2
Bond   2:        4 ->       3 by elimination of symmetry-related configs.
total CPU time:            0.0010000 bond   2
```

Initially, there were two H-bond configurations from the previous step of the calculation. We add the second H-bond to each of these configurations in each of the two possible orientation. After checking to see if any configurations violate the imposed model, we find that there are now four H-bond configurations. This number is still too small to use the sorting algorithm and so all configurations will be used in the symmetry comparison. After eliminating symmetry equivalent configuration, we see that there are now three symmetry-distinct H-bond configurations after adding two H-bonds. We continue on by adding the third H-bond.

```
----------     3   ----------
3  configurations from previous bond
Before checking for distinct configs     6  configs enumerated for bond 3
CPU time:            0.0010000 add bonds at bond   3
CPU time:            0.0000000 symmetry search w/o invariants at bond   3
Bond   3:        6 ->       6 by elimination of symmetry-related configs.
total CPU time:            0.0010000 bond   3
```

After adding the third bond and eliminating symmetry-equivalent configurations, we see that there are six symmetry-distinct H-bond configurations. This process of adding H-bonds, removing configurations that violate the model, and then removing configurations that are related by symmetry will continue until all H-bonds have been added to the system.

We now skip ahead to the point where 16th H-bond is being added to the system.

```
----------    16   ----------
897  configurations from previous bond
Before checking for distinct configs  1059 configs enumerated for bond 16
CPU time:            0.0079990 add bonds at bond  16
```

We see that after adding the 16th H-bond and removing those configurations which violate the "ice rules," 1059 H-bond configurations remain. Due to the large number of H-bond configurations, the sorting algorithm is now used. H-bond configurations will be sorted into groups depending on what values the graph invariants take. Configurations with similar values for invariants will be grouped together. Of all the graph invariants given as input, only the "best" ones will be used for the sorting algorithm. A graph invariant is useful for speeding up the identification of symmetry-distinct configurations if it breaks the H-bond configurations into several groups of roughly equal size, each having a different value of that invariant. If a graph invariant evaluates to the same value for all H-bond configurations, then that invariant would place configurations in the same group (not very efficient!). If a large number of H-bond configurations, let's say one million, separate into one group of 999,998 and two groups of 1, then that invariant is nearly as useless. Hence, the program selects invariants that break the set of H-bond configurations into as many roughly equal-sized groups as possible.

The next portion of the output follows the calculation of the best invariants for the configurations generated thus far.

```
 -- Invariant characteristics --
Sample of     1059 out of total     1059 configs.    tol =  0.10E-03
inv. norm     actv? #vals values/population
   1             no   1        8
                           1059
...
```

For each of the 1059 H-bond configurations, the first invariant is evaluated. The number of unique invariant values and the number of H-bond configurations that generated each value is recorded. For the first invariant, the value of the graph invariant was the same for all 1059 configurations. There is one unique value and it is eight. This invariant is not considered to be good and will not be included in the group used to sort configurations.

Moving on to the second graph invariant, we see that there are five unique values that this invariant takes.

```
...
   2  0.21E+05  yes   5        8       4       0      -4      -8
[  5.6119]                   130     210     411     210      98
...
```

There are 130 configurations with an invariant value of 8, there are 210 configurations with an invariant value of 4, and so on. Using these statistics a score is calculated, the number in square

brackets, and this will be used later to decide which are the best invariants to use for sorting. After all invariants have been evaluated for all configurations sampled, the graph invariants are then sorted by the scored they received and ones with the highest scores are used for sorting.

```
  Best active invariants
10        9         8       0      -8       6       4       2      -2      -4      -6
[  9.7460]        11     269      11      42     122     214     218     127      45
8         9         8       0      -8       6       4      -2      -4       2      -6
[  9.7365]         9     279       7      43     152     182      93     264      30
...
CPU time:             0.0199970 invariant sample at bond  16
```

The number of invariants used for sorting is an adjustable parameter discussed in section 6.5. In this case, the 10th graph invariant had the highest score yielding nine unique values when evaluated for the sampling of H-bond configurations. For each of the nine unique values, the value of the invariant and number of configurations with that value are listed.

The *GrEnum* program will now loop through each of the best invariants sorting configurations according to their value for the invariants. Only configurations with identical values for all invariants will be found in the same groupings.

```
          11  configs written to XX001
         269  configs written to XX002
          11  configs written to XX003
          42  configs written to XX004
         122  configs written to XX005
         214  configs written to XX006
         218  configs written to XX007
         127  configs written to XX008
          45  configs written to XX009
      1059 configs read at sorting level   1
   1059 configs written at sorting level   1
```

Using the first invariant, the configurations were sorted into nine different groups. These groups are stored in temporary scratch files names "XX###". As a consistency check, the number of configurations sorted into files is checked against the number of configurations written. If there were more than 1000 configurations in the first file, "XX001", those configurations would then be sorted using the next invariant into temporary files, "XX001###". This procedure is repeated until all of the best invariants have been used or groups no longer contain more than 1000 configurations.

```
 skipped XX001
 skipped XX002
 skipped XX003
 skipped XX004
 skipped XX005
 skipped XX006
 skipped XX007
 skipped XX008
 skipped XX009

     1059 configs sorted into     9 files
           file      configs      file name
           1          11   XX001
           2         269   XX002
           3          11   XX003
           4          42   XX004
           5         122   XX005
           6         214   XX006
           7         218   XX007
           8         127   XX008
           9          45   XX009
```

In this case, since no files contain more than 1000 configurations, the sorting algorithm is stopped here and the symmetry comparison is done only for configurations within the same file.

```
CPU time:            0.0299950 deal configs at bond  16
CPU time:            0.0219980 symmetry search at bond  16
Bond  16:     1059 ->     898 by elimination of symmetry-related configs.
total CPU time:            0.0789890 bond  16
```

After the symmetry comparison, we see that there are now 898 symmetry-distinct configurations after adding the 16th H-bond. This procedure of adding bonds will continue until the very last H-bond has been added.

```
----------    24   ----------
102  configurations from previous bond
Before checking for distinct configs  102 configs enumerated for bond  24
CPU time:            0.0000000 add bonds at bond  24
CPU time:            0.0019990 symmetry search w/o invariants at bond  24
Bond  24:      102 ->     102 by elimination of symmetry-related configs.
total CPU time:            0.0019990 bond  24
```

We see that there were 102 symmetry-distinct configurations after adding the 23rd H-bond. Adding the 24th and final H-bond did not generate any additional structures. Symmetry comparison after adding the 24th H-bond yields a total of 102 symmetry-distinct H-bond configurations for the ice III unit cell.

6.5 Adjustable parameters for efficient enumeration

For the present state of the code, there are two adjustable parameters that can have a impact on the efficiency of the enumeration calculation. The values of these parameters are currently determined at compile time.

- InvariantThreshold: The number of configurations present before sorting algorithm is used. In the example above, this was set at 1000.
- nInvForDeal: The maximum number of invariants used to sort configurations. In the example above, this was set to 6.

One can adjust these parameters to find an optimal set for a particular problem and computing environment. Increasing the number of files, by decreasing "InvariantThreshold" and increasing "nInvForDeal", involved in the sorting process reduces the time spent during the symmetry comparison. However, one can run into a disk access bottleneck when reading and writing too many files.

7 Additional Small Programs

7.1 HBondConfigXYZ

This small program reads vertices, H-bonds, and bond variables as input and generates cartesian coordinates for H-bond configurations. This program, found in the /Tools directory, is written in Fortran 70. Using the GNU Fortran compiler, the program can be compiled with the following command:

```
g77 HBondConfigXYZ.f -I HBondConfigXYZ.inc -o HBondConfigXYZ
```

This program reads two input files, one generated by each of the *MkInvar* and *GrEnum* codes. Information about vertices and H-bonds is supplied using one of the "Structure.txt" files generated as output from the *MkInvar* code. The bond variables for the H-bond configurations are contained in the file "XFile" generated by the *GrEnum* code, the ASCII version generated after a successful completion. The following command will generate an output file containing the cartesian coordinates of oxygen and hydrogen atoms for all H-bond configurations given in the file "XFile."

```
HBondConfigXYZ < Structure.txt
```

The program assumes that the file "XFile" is located in the same directory as the command was executed.

An example of a portion of the output file, "HBondConfigXYZ.xyz," is shown below.

```
 48
 Config:  1
  O         0.000000       0.000000       0.000000
  O         1.668500       1.668500       1.668500
  O         0.000000       0.000000       3.337000
  O         1.668500       1.668500       5.005500
...
  H         0.556167       0.556167       0.556167
  H         1.112333       2.224667       2.224667
  H         2.224667       1.112333       2.224667
  H         2.780833       2.780833       0.556167
...
  H        -0.556167      -0.556167       3.893167
 48
 Config:  2
  O         0.000000       0.000000       0.000000
```

In this example, there are 16 oxygen atoms and 32 hydrogen atoms making a total of 48 atoms. For each H-bond configuration, there will be 48+2=50 lines of text. The first two lines for each H-bond configuration contain the total number of atoms and descriptive information respectively. These two lines are then followed by a line for each of the atoms. Each line contains four entries: atomic type, x, y, and z coordinates. Again, the units of length are whatever were used for the lattice parameters in the *MkInvar* code.

References

[1] Victor F. Petrenko and Robert W. Whitworth. *Physics of Ice*. Oxford, 1999.

[2] J. D. Bernal and R. H. Fowler. A theory of water and ionic solution, with particular reference to hydrogen and hydroxyl ions. *J. Chem. Phys.*, 1(8):515, 1933.

[3] Christoph G. Salzmann, Paolo G. Radaelli, Andreas Hallbrucker, Erwin Mayer, and John L. Finney. The preparation and structures of hydrogen ordered phases of ice. *Science*, 311(5768):1758, 2006.

[4] Christoph G. Salzmann, Andreas Hallbrucker, John L. Finney, and Erwin Mayer. Raman spectroscopic features of hydrogen-ordering in ice XII. *Chem. Phys. Lett.*, 429(4-6):469–473, 2006.

[5] C. G. Salzmann, P. G. Radaelli, A. Hullbrucker, E. Mayer, and J. L. Finney. New hydrogen ordered phases of ice. In Werner F. Kuhs, editor, *Physics and Chemistry of Ice*, Symposium on the Physics and Chemistry of Ice, page 521, Cambridge, 2007. Royal Society of Chemistry.

[6] Linus Pauling. The structure and entropy of ice and of other crystals with some randomness of atomic arrangement. *J. Am. Chem. Soc.*, 57(12):2680–2684, 1935.

[7] J. F. Nagle. Lattice statistics of hydrogen bonded crystals. I. The residual entropy of ice. *J. Math. Phys.*, 7(8):1484–1491, 1966.

[8] Jer-Lai Kuo, James V. Coe, Sherwin J. Singer, Yehuda B. Band, and Lars Ojamäe. On the use of graph invariants for efficiently generating hydrogen bond topologies and predicting physical properties of water clusters and ice. *J. Chem. Phys.*, 114(6):2527, 2001.

[9] Jer-Lai Kuo and Sherwin J. Singer. Graph invariants for periodic systems: towards predicting physical properties from the hydrogen bond topology of ice. *Phys. Rev.*, E67(1):016114, 2003.

[10] Sherwin J. Singer, Jer-Lai Kuo, Tomas K. Hirsch, Chris Knight, Lars Ojamäe, and Michael L. Klein. Hydrogen bond topology and the ice VII/VIII and Ih/XI proton ordering phase transitions. *Phys. Rev. Lett.*, 94(13):135701, 2005.

[11] Chris Knight, Sherwin J. Singer, Jer-Lai Kuo, Tomas K. Hirsch, Lars P. Ojamäe, and Michael L. Klein. Prediction of the Ih/XI and VII/VIII proton ordering phase transitions. *Phys. Rev.*, E73(5):056113, 2006.

[12] Chris Knight and Sherwin J. Singer. A reexamination of the ice III/IX hydrogen bond ordering phase transition. *J. Chem. Phys.*, 125(6):64506, 2006.

[13] Chris Knight and Sherwin J. Singer. Hydrogen bond ordering in ice V and the transition to ice XIII. *J. Chem. Phys.*, 129(16):164513, 2008.

[14] Wolfram Research, Inc. *Mathematica*. (Champaign, Illinois).

[15] Xianlong Wang. Symmetry operations of crystallographic point groups and space groups. Wolfram Library Archive (library.wolfram.com/infocenter/MathSource/6347), 2006.

[16] Mois Ilia Aroyo, Juan Manuel Perez-Mato, Cesar Capillas, Eli Kroumova, Svetoslav Ivantchev, Gotzon Madariaga, Asen Kirov, and Hans Wondratschek. Bilbao crystallographic server: I. databases and crystallographic computing programs. *Z. Kristallogr.*, 221(1):15–27, 2006.

[17] Mois I. Aroyo, Asen Kirov, Cesar Capillas, J. M. Perez-Mato, and Hans Wondratschek. Bilbao crystallographic server. ii. representations of crystallographic point groups and space groups. *Acta Crystallogr., Sect. A: Found. Crystallogr.*, 62(2):115–128, 2006.

[18] Bilbao crystallographic server. www.cryst.ehu.es.

[19] Stephen D. Belair, Joseph S. Francisco, and Sherwin J. Singer. Hydrogen bonding in cubic $(H_2O)_8$ and $OH \cdot (H_2O)_7$ clusters. *Phys. Rev.*, A71(1):013204, 2005.

[20] C. Knight and S. J. Singer. Theoretical study of a hydroxide ion within the ice-Ih lattice. In Werner F. Kuhs, editor, *Physics and Chemistry of Ice*, Symposium on the Physics and Chemistry of Ice, page 339, Cambridge, April 2007. The Royal Society of Chemistry.

[21] W. F. Kuhs, J. L. Finney, C. Vettier, and D. V. Bliss. Structure and hydrogen ordering in ices VI, VII and VIII by neutron powder diffraction. *J. Chem. Phys.*, 81(8):3612, 1984.