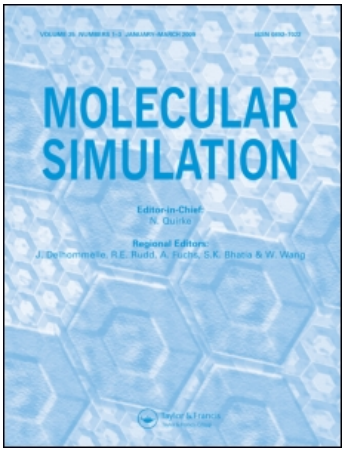


**Molecular Simulation**
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

# Efficient Test Molecule Sampling in Molecular Simulation

G. L. Deitrick[a]; L. E. Scriven[a]; H. T. Davis[a]
[a] Department of Chemical Engineering and Materials Science, University of Minnesota,

**To cite this Article** Deitrick, G. L. , Scriven, L. E. and Davis, H. T.(1992) 'Efficient Test Molecule Sampling in Molecular Simulation', Molecular Simulation, 8: 3, 239 — 247
**To link to this Article: DOI:** 10.1080/08927029208022479
**URL:** http://dx.doi.org/10.1080/08927029208022479

*Molecular Simulation*, 1992, Vol. 8, pp. 239–247
Reprints available directly from the publisher
Photocopying permitted by license only



# EFFICIENT TEST MOLECULE SAMPLING IN MOLECULAR SIMULATION

G.L. DEITRICK, L.E. SCRIVEN and H.T. DAVIS

*Department of Chemical Engineering and Materials Science, University of Minnesota*

(*Received August 1990; accepted May 1991*)

Excluded volume map sampling (EVMS) is a particularly efficient means of performing test molecule sampling to estimate or impose chemical potential in molecular simulations. This paper discusses the motivation and applications of excluded volume map sampling, presents computer code demonstrating its implementation, and gives an example of its application.

KEY WORDS: Excluded volume map sampling, test molecule sampling, chemical potential



## DESCRIPTION AND APPLICATION OF THE METHOD

The chemical potential is a measure of the "escaping tendency" of a molecule from a system. Measuring (or enforcing) chemical potential within molecular simulations, therefore, requires some scheme to sample the ease with which molecules can leave or enter the system. Test molecule sampling is a means of sampling the "ease" with which a molecule can enter a system. The computational cost of most methods of estimating the chemical potential in liquids is overwhelmingly determined by the inefficiency of the test molecule sampling. Analogously, the computational cost of most methods of imposing the chemical potential in liquids, e.g. the grand canonical ensemble Monte Carlo sampling (GCEMC), is overwhelmingly determined by the inefficiency of the molecule insertion/removal sampling. Recently, we introduced an excluded volume map sampling scheme [1] which dramatically reduces the inefficiency of test molecule sampling for chemical potential estimation and promises valuable improvements of GCEMC insertion/removal sampling via Mezi's cavity-biasing scheme [2, 3] but is difficult to code. In this paper we summarize the applications of excluded volume map sampling (EVMS) to estimating and enforcing chemical potential within molecular simulations, present an example of its implementation, and give the code in Pascal. A recent review of simulation methods for the chemical potential has been published by Gubbins [4].

The inherent inefficiency of test molecule sampling for estimating the chemical potential of liquids is best illustrated using Widom's formula [5, 6, 7] for spherical molecules,

$$\alpha \equiv \exp\left[(\mu(N, V, T) - \mu^0(N, V, T))/kT\right] = \langle \exp(-u/kT) \rangle_N^{-1} \qquad (1)$$

where $T$ is the temperature, $N$ the number of molecules, $V$ the volume, $\mu$ the chemical potential and $\mu^0$ the ideal gas chemical potential. The notation $\langle\ \rangle_N$ denotes an $N$-molecule ensemble average. $u$ is the change in the system potential energy upon

insertion of a test molecule into a randomly selected configuration of the $N$-molecule ensemble at a randomly selected position. In terms of the probability density of $u$, $f(u)$, i.e., $f(u)\mathrm{d}u$ is the probability of observing a change in potential energy $u - \mathrm{d}u/2$ and $u + \mathrm{d}u/2$ if a test molecule is inserted randomly into a randomly-selected $N$-molecule ensemble, Equation (1) is

$$\alpha = \left[ \int_{-\infty}^{\infty} f(u) \exp(-u/kT)\mathrm{d}u \right]^{-1} \tag{2}$$

Figure 1 summarizes the estimation of $\alpha$ from Equation (2) for a Lennard-Jones liquid. Only test molecules which produce a very favorable change in potential energy contribute significantly to the integral of Equation (2) or, equivalently, the ensemble average of Equation (1). As liquid density increases it becomes exceedingly unlikely that a randomly selected test molecule position will produce such a favorable change in potential energy. Thus the overwhelmingly majority, approximately 99.9% in Figure 1 (c.f. panels C and D), of test molecule samples will contribute negligibly to the ensemble average.

Two techniques for improving test molecule sampling were suggested by Shing and Gubbins [8] and Guissani *et al.* [9, 10]. Shing and Gubbins [8] generate test molecule positions using a Monte Carlo scheme to move a test molecule through a fixed configuration of real molecules. Naturally the procedure is repeated for many configurations to obtain an ensemble average. In their simulations of polyatomic molecules Guissani *et al.* [9, 10] would assign a zero contribution, i.e., a large value of $u$, to all test molecules whose center of mass fell within a minimum distance from a real molecule. While this strategy avoided unnecessary evaluations of the expensive polyatomic intermolecular potential it offers little advantage for simple fluid simulations because the intermolecular interaction computation is inexpensive compared to the determination of the intermolecular separations.

Excluded volume map sampling [1] maps out the space occupied by the real molecules (i.e., their "excluded volume") of the configuration and samples uniformly the remaining portion of the simulation volume (i.e., the "free volume"). It requires three elements: 1. an appropriate definition of the excluded volume of a real molecule, 2. a correction for sampling only the free volume, and 3. an efficient means of generating the map. An appropriate definition of a molecule's excluded volume depends upon the particular application. For Widom's method an appropriate definition would be the volume which, if a test molecule were inserted into it, would produce a value of $u > 0$. The correction for sampling only the free volume depends not only on the application but also the definition of excluded volume. Figure 1 indicates that, under the above definition of excluded volume, one can assign a zero value to the quantity $\exp(-u/kT)$ for all test molecules which fall within the configuration's excluded volume. The focus of this paper is to present an efficient means of generating the excluded volume map and uniformly sampling the free volume.

Evaluation of Equation (1) is only one application of excluded volume map sampling. Indeed, the introduction of excluded volume map sampling [1] compared several formulas similar to Widom's for estimating $\alpha$ and found Bennett's [11] to be the method of choice. Mezi's cavity-biased GCEMC method requires a scheme to estimate the probability of finding a cavity of a specified radius as well as a means of locating such cavities; the excluded volume map performs both these functions exceptionally well and should be included in GCEMC studies of simple liquids.

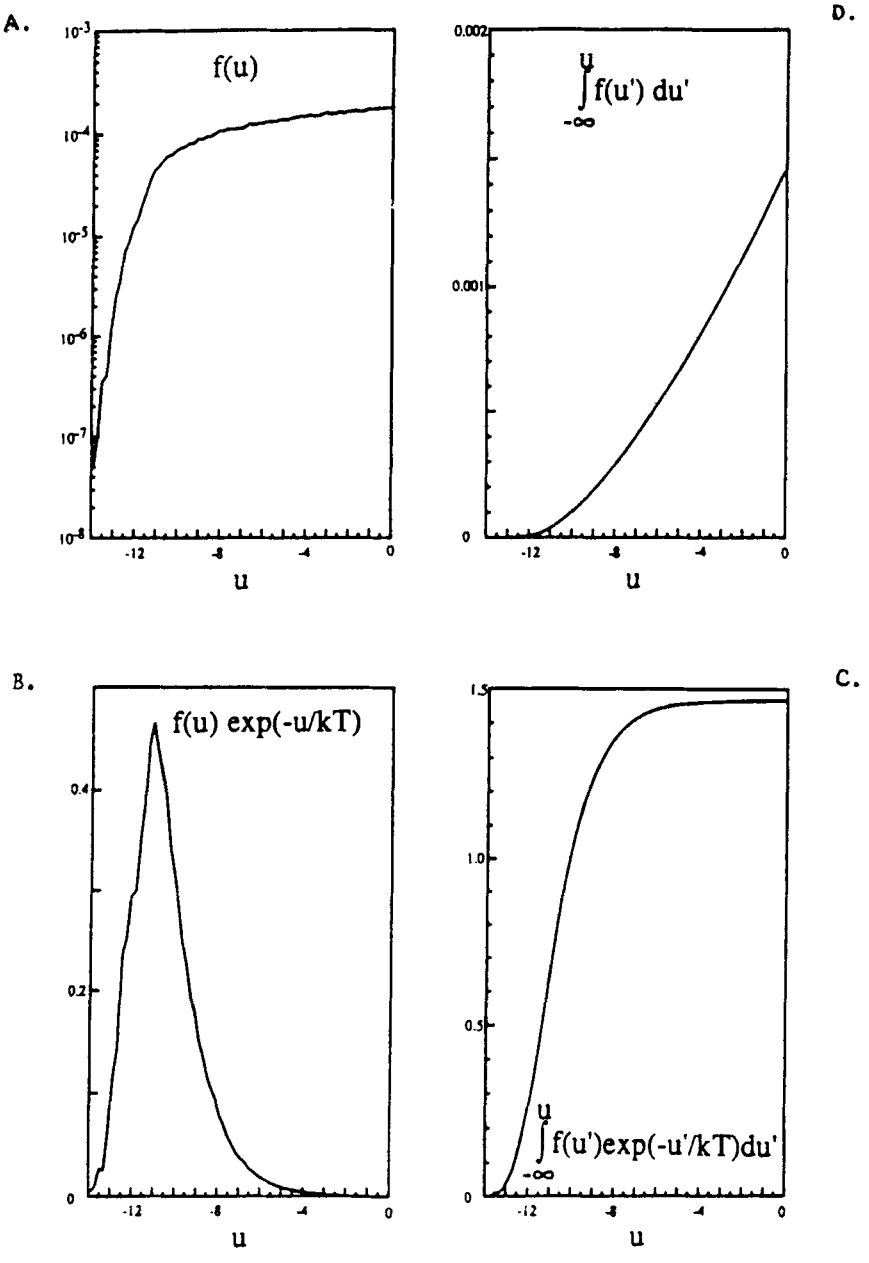


**Figure 1** Calculating the fugacity coefficient $\alpha$ using Widom's method, Equation (2). Panel A: the probability density $f(u)$ of a Lennard-Jones liquid. Panel B: the integrand of Equation (2). Panel C: the integral of Equation (2); $1/\alpha = \int_{-\infty}^{\infty} f(u) \exp(-u/kT)du$. Panel D: The cumulative probability; i.e., the probability that a randomly selected configuration will increase in potential energy by an amount less than $u$ if a test-molecule were randomly inserted into it.

Even with the excluded volume map sampling, test molecules sampling should not be extended to multicomponent simulations of liquids. While test molecule sampling is necessary to estimate the chemical potential of one component, it is much more efficient to estimate the difference in chemical potential between two components and sum the results to estimate the chemical potential of a second component than it is to estimate the chemical potential of the second component directly from test molecule sampling. The difference in chemical potential between two components is simply the change in free energy upon changing one molecule of one component to one molecule of the other; Mezei and Beveridge [12] present a review of methods for calculating free energy differences. Similarly, in multicomponent GCEMC simulations of liquids, it is best to use excluded volume map sampling with Mezei's cavity-based insertion/removal sampling only to impose the chemical potential of one component, and Kofke and Glandt's semigrand sampling [13] to impose the difference in chemical potential between this component and the other components in the simulation. The overriding principle in all of these situations is that dealing with (either estimating or enforcing) free energy differences (e.g., chemical potentials) is *much* cheaper (computationally) between systems which are most similar.

The following code demonstrates the excluded volume map sampling algorithm for simple fluid mixtures. The code, as well as any references to specific variable or procedure names are set in a different font, e.g., x refers to the variable which appears in the code. Within the code identifiers that are in lower case are defined by the program; identifiers in upper case are Pascal reserved words or library functions described, for example, in the Cray Pascal documentation. The first block of definitions are global to all subsequent procedures; in FORTRAN these variables would need to be defined in a COMMON block which appeared in the main program as well as the three subprograms. FORTRAN programmers should also note that anything between { and } is a comment ignored by the compiler, and that the notation `ARRAY [0 . . 3] OF REAL` defines a real array with indices 0, 1, 2, and 3. The only serious pitfall in translating this code to FORTRAN is removing the 0 array index values. The code uses five bit-oriented extensions to standard Pascal: `LSHIFT` and `RSHIFT` return their first argument after shifting it Left or Right (filling in with zeros) the number of bits indicated by their second argument; `BAND` and `BOR` return the bitwise logical AND (OR) of their two arguments; `POP` returns the number of 1 bits in its argument. `I32` is essentially a 32-bit integer; the `INTEGER` type is 64 bits but was avoided, where possible, due to performance concerns.

The simulation volume $\{(x, y, z)$: $|x| \leqslant$ `xmax`, $|y| \leqslant$ `ymax`, $|z| \leqslant$ `zmax`$\}$, is subdivided into `xbox` × `ybox` × `zbox` small boxes of dimensions `deltax` × `deltay` × `deltaz` while keeping `deltax` ≈ `deltay` ≈ `deltax`. `xmax`, `ymax` and `zmax` are initialized from an input data file, and `xbox` is the number of bits in an integer data type (`xbox` = 64 for the Cray-2 computer). `ybox`, `zbox`, `deltax`, `deltay` and `deltaz` are initialized in `PROCEDURE init_map`, which also constructs the bitmap templates `bits_set[ j ]` to have only the center $j$ bits set.

A molecule's excluded volume is defined as those boxes whose centers lie within the excluded volume radius of the center of the box containing the center of the molecule. Thus the mapping essentially rounds all real (and test) molecule positions to the nearest box center. The excluded volume radius is the relative excluded volume radius, `min_sep`, multiplied by the length scale of the interaction of the real and test molecule. If the test molecule was of component 1 and the simple fluid obeyed the

Lennard-Jones intermolecular potential, i.e.,

$$\phi_{\alpha\beta}(r) = 4\varepsilon_{\alpha\beta}\left[\left(\frac{\sigma_{\alpha\beta}}{r}\right)^{12} - \left(\frac{\sigma_{\alpha\beta}}{r}\right)^{6}\right], \tag{3}$$

(where $\alpha\beta$ denotes the interaction of a component $\alpha$ molecule with a component $\beta$ molecule) then the exclued volume radius of a component 1 molecule is $\sigma_{11} \times$ min_sep and that of a component 2 molecule is $\sigma_{12} \times$ min_sep. The value of min_sep is initialized from an input data file as are the values rhat [$\alpha$, $\beta$] ($=\sigma_{\alpha\beta}$ for Lennard-Jones systems), numofcomp (total number of components), numofmol (total number of molecules) and typeofmol [$i$] (component number of molecule $i = 1, \ldots,$ numofmol).

The excluded volume map is constructed by PROCEDURE construct_map defined within PROCEDURE calc_fn (which selects test molecule positions from the map). Molecular positions {(x[$i$], y[$i$], z[$i$]): $i = 1, \ldots,$ numofmol} over the real-valued ranges {($-\eta$max, $\eta$max): $\eta$ = x, y, z} are transformed to box indices {(xindex[$i$], yindex[$i$], zindex[$i$]): $i = 1, \ldots,$ numofmol} over the integer-valued ranges {(0, $\eta$box $-$ 1): $\eta$ = x, y, z}. The excluded volume map convention is that bit $i_x$ of the xbox-bit integer map[$i_y$, $i_z$] is 1 if the box with indices ($i_x$, $i_y$, $i_z$) is part of the excluded volume of the configuration and 0 if the box is not. The map construction itself is short and efficient but not easily described. The $y$ and $z$ dimensions are handled in a straightforward fashion, one index at a time, but the $x$ dimension, including periodic reflections, is managed by bit shifting of the bitmap templates. Vectorization concerns complicate matters further.

After the map is constructed the number of free volume (nonexcluded volume) boxes is determined. Test molecule positions are selected uniformly from the centers of the free volume boxes so that the number of test molecules, num_tmol, is approximately equal the target number of test molecules, target_num_tmol. For chemical potential calculations one would subsequently compute the potential energy of interaction between each test molecule and the real molecules but the code for these calculations has been removed. Each test molecule result for the configuration must be weighted by the factor

$$\frac{\text{free volume}}{\text{total volume} \times \text{num\_tmol}}$$

before it is averaged with samples from other configurations.

```
{=============================================================================}
CONST
        maxmol          =       301;
        maxcomp         =       3;
TYPE
        vector          =       ARRAY [1..maxmol] OF REAL;
        ivector         =       ARRAY [1..maxmol] OF I32;
        matrix          =       ARRAY [0..maxcomp,0..maxcomp] OF REAL;
        shortvector     =       ARRAY [1..maxcomp] OF REAL;
        shortivector    =       ARRAY [1..maxcomp] OF I32;
CONST
        xbox                    =       64;
        half_xbox               =       xbox DIV 2;
        three_half_xbox         =       3  *  half_xbox;
        target_maxtmol          =       9000;
VAR
```

```
        bits_set                :       ARRAY[0..half_xbox] OF INTEGER;
        deltax                  :       REAL;
        deltay                  :       REAL;
        deltaz                  :       REAL;
        map                     :       ARRAY[0..xbox,0..xbox] OF INTEGER;
        max_dx_index            :       I32;
        max_dy_index            :       I32;
        max_dz_index            :       I32;
        min_sep                 :       REAL;
        numofcomp               :       I32;
        numofmol                :       I32;
        rhat                    :       matrix;
        rhat_2                  :       ARRAY[1..maxcomp] OF vector;
        target_numtmol          :       I32;
        twoxmax                 :       REAL;
        twoymax                 :       REAL;
        twozmax                 :       REAL;
        typeofmol               :       ivector;
        xmax                    :       REAL;
        ybox                    :       I32;
        ymax                    :       REAL;
        zbox                    :       I32;
        zmax                    :       REAL;
{=============================================================================}
{=============================================================================}
PROCEDURE init_map;
{.............................................................................}
{SYSTEM:  CRAY2, Unicos Operating System            LANGUAGE:  Cray Pascal 3.2}
{.............................................................................}
VAR
        i               :       I32;
        rhat_max        :       REAL;
{.............................................................................}
BEGIN   {PROCEDURE init_map}
{.............................................................................}
deltax          :=      twoxmax / xbox;
ybox            :=      ROUND( twoymax / deltax );
deltay          :=      twoymax / ybox;
zbox            :=      ROUND( twozmax / deltax );
deltaz          :=      twozmax / zbox;
bits_set[0]     :=      0;
bits_set[1]     :=      1;
bits_set[1]     :=      LSHIFT(bits_set[1],half_xbox);
FOR i:=2 TO half_xbox DO
  bits_set[i] := BOR( bits_set[i-1], BOR( LSHIFT(bits_set[1],(i-1) DIV 2),
                                          RSHIFT(bits_set[1], i DIV 2)    ) );
rhat_max        :=      rhat[1,1];
FOR i:=2 TO numofcomp DO
     IF (rhat[1,i]>rhat_max) THEN rhat_max  :=  rhat[1,i];
max_dx_index    :=      TRUNC(rhat_max * min_sep / deltax);
max_dy_index    :=      TRUNC(rhat_max * min_sep / deltay);
max_dz_index    :=      TRUNC(rhat_max * min_sep / deltaz);
{.............................................................................}
END;    {PROCEDURE init_map}
{=============================================================================}
{-----------------------------------------------------------------------------}
  PROCEDURE construct_map( VAR x,y,z:vector );
  {...........................................................................}
  {SYSTEM:  CRAY2, Unicos Operating System          LANGUAGE:  Cray Pascal 3.2}
  {...........................................................................}
  VAR
        abs_dy_index    :       I32;
        abs_dz_index    :       I32;
        dy_index        :       I32;
        dz_index        :       I32;
        i               :       I32;
        min_sep_2       :       REAL;
        word            :       ARRAY [1..maxmol] OF INTEGER;
        xbits           :       ivector;
        xindex          :       ivector;
        xlength_2       :       vector;
        xword           :       ARRAY [1..maxmol] OF INTEGER;
```

```
      y_plus_dy       :       ivector;
      yindex          :       ivector;
      z_plus_dz       :       ivector;
      zindex          :       ivector;
{.....................................................................}
BEGIN   {Of PROCEDURE construct_map}
{.....................................................................}
FOR i:=1 TO numofmol DO   {Calculate the indices of the boxes containing mols}
  BEGIN
    xindex[i] :=      TRUNC( (x[i]/twoxmax + 0.5) * xbox );
    xindex[i] :=      xindex[i]  MOD  xbox;
    yindex[i] :=      TRUNC( (y[i]/twoymax + 0.5) * ybox );
    yindex[i] :=      yindex[i]  MOD  ybox;
    zindex[i] :=      TRUNC( (z[i]/twozmax + 0.5) * zbox );
    zindex[i] :=      zindex[i]  MOD  zbox;
  END;
min_sep_2     :=      SQR(min_sep);
deltax        :=      twoxmax  /  xbox;
deltay        :=      twoymax  /  ybox;
deltaz        :=      twozmax  /  zbox;
FOR dy_index:=0 TO ybox-1 DO
  FOR dz_index:=0 TO zbox-1 DO map[dy_index,dz_index] := 0;
FOR dy_index:=-max_dy_index TO max_dy_index DO
  BEGIN
    abs_dy_index        :=      ABS(dy_index);
    FOR i:=1 TO numofmol DO
          y_plus_dy[i] := (yindex[i] + dy_index + ybox) MOD ybox;
    FOR dz_index:=-max_dz_index TO max_dz_index DO
      BEGIN
        abs_dz_index    :=      ABS(dz_index);
        FOR i:=1 TO numofmol DO
          BEGIN
            z_plus_dz[i]:= (zindex[i] + dz_index + zbox) MOD zbox;
            xlength_2[i]:=      rhat_2[i][i]*min_sep_2
                              - SQR(dy_index*deltay) - SQR(dz_index*deltaz);
            IF (xlength_2[i]>0.0)
              THEN xbits[i] := TRUNC(SQRT(xlength_2[i])/deltax) * 2  +  1
              ELSE xbits[i] := 0;
          END;
        FOR i:=1 TO numofmol DO
         BEGIN
          xword[i]    :=      bits_set[xbits[i]];
          word[i]:=BOR(LSHIFT(xword[i],(half_xbox+xindex[i]) MOD xbox),
                       RSHIFT(xword[i],(three_half_xbox-xindex[i]) MOD xbox));
         END;
        FOR i:=1 TO numofmol DO
          map[y_plus_dy[i],z_plus_dz[i]]    :=
                                  BOR(map[y_plus_dy[i],z_plus_dz[i]],word[i]);
      END;
  END;
{.....................................................................}
END;  {Of PROCEDURE construct_map}

{-------------------------------------------------------------------------------}
{===============================================================================}
PROCEDURE calc_fn ( VAR x,y,z:vector );
{-------------------------------------------------------------------------------}
{SYSTEM:  CRAY2, Unicos Operating System            LANGUAGE:  Cray Pascal 3.2}
{-------------------------------------------------------------------------------}
CONST
       tmol_vector_dim =      2 * target_maxtmol + 3;
TYPE
       tmol_vector     =      ARRAY [1..tmol_vector_dim] OF REAL;
VAR
       component       :      I32;
       contribution    :      REAL;
       dx              :      vector;
       dy              :      vector;
       dz              :      vector;
       freq_tmol       :      I32;
       i               :      I32;
       last_total      :      I32;
```

```
        num_empty_box   :       I32;
        num_full_box    :       I32;
        num_tmol        :       I32;
        population      :       I32;
        setbit_0        :       INTEGER;
        this_tmol       :       I32;
        tmol_x          :       tmol_vector;
        tmol_y          :       tmol_vector;
        tmol_z          :       tmol_vector;
        total_empty     :       I32;
        word            :       I32;
        xi              :       I32;
        y_factor        :       REAL;
        yi              :       I32;
        z_factor        :       REAL;
        zi              :       I32;
{=============================================================================}
{=============================================================================}
BEGIN   {PROCEDURE calc_fn}
{=============================================================================}
construct_map(x,y,z);                                   {Generate excluded vol map}
num_full_box            :=      0;
FOR yi:=0 TO ybox-1 DO
  FOR zi:=0 TO zbox-1 DO
    num_full_box        :=      num_full_box    +       POP(map[yi,zi]);
num_empty_box           :=      xbox * ybox * zbox      -       num_full_box;
IF (num_empty_box<2*target_numtmol)
  THEN  freq_tmol       :=      1
  ELSE  freq_tmol       :=      num_empty_box DIV target_numtmol;
num_tmol                :=      num_empty_box   DIV     freq_tmol;
setbit_0            :=      1;
word                :=      0;
total_empty         :=      0;
this_tmol           :=      1;
REPEAT
  xi                :=      -1;
  yi                :=      word MOD ybox;
  zi                :=      word DIV ybox;
  population        :=      xbox  -  POP(map[yi,zi]);
  last_total        :=      total_empty;
  total_empty       :=      total_empty     +       population;
  WHILE (total_empty>=this_tmol*freq_tmol) DO       {Take a tmol from this word}
    BEGIN

      WHILE (last_total<this_tmol*freq_tmol) DO     {Update last_total & xi    }
        BEGIN                                       {until next tmol_x is found}
          xi        :=      xi      +       1;
          IF (BAND(map[yi,zi],LSHIFT(setbit_0,xi))=0) {Is bit 0 (is box empty)?}
            THEN  last_total  :=  last_total + 1;
        END;                    {Now we know the proper xi. Set tmol_* to *i values}
      tmol_x[this_tmol]     :=      xi;
      tmol_y[this_tmol]     :=      yi;
      tmol_z[this_tmol]     :=      zi;
      this_tmol             :=      this_tmol       +       1;
    END;
  word      :=      word    +       1;      {Take no more samples from this word}
UNTIL (this_tmol>num_tmol);
FOR this_tmol:=1 TO num_tmol DO      {Now transform the coord indices to coords}
  BEGIN
    tmol_x[this_tmol]:=      twoxmax * ((tmol_x[this_tmol]+0.5)/xbox - 0.5);
    tmol_y[this_tmol]:=      twoymax * ((tmol_y[this_tmol]+0.5)/ybox - 0.5);
    tmol_z[this_tmol]:=      twozmax * ((tmol_z[this_tmol]+0.5)/zbox - 0.5);
  END;
contribution        :=      num_empty_box / (xbox*ybox*zbox*num_tmol);
{=============================================================================}
END;    {PROCEDURE calc_fn}
{=============================================================================}
```

## *References*


[1] G.L. Deitrick, L.E. Scriven and H.T. Davis, "Efficient molecular simulation of chemical potentials," *J. Chem. Phys.*, **90**(4), 2370–85 (1989).

[2] Mihaly Mezei, "A cavity-biased (T, V, $\mu$) Monte Carlo method for the computer simulation of fluids," *Mol. Phys.*, **40**(4), 901–6 (1980).

[3] Mihaly Mezei, "Grand-canonical ensemble Monte Carlo study of dense liquid Lennard–Jones, soft spheres and water," *Mol. Phys.*, **61**(3), 565–82 (1987).

[4] K.E. Gubbins, *Molecular Simulation*, **2**, 223 (1989).

[5] B. Widom, "Some Topics in the Theory of Fluids," *J. Chem. Phys.*, **39**(11), 2808–12 (1963).

[6] B. Widom, "Structure of Interface from Uniformity of the Chemical Potential," *J. Stat. Phys.*, **19**(6), 563–74 (1978).

[7] S. Ramano and K. Singer, "Calculation of the entropy of liquid chlorine and bromine by computer simulation," *Mol. Phys.*, **37**(6), 1765–72 (1979).

[8] K.S. Shing and K.E. Gubbins, "The chemical potential in dense fluids and fluid mixtures via computer simulation," *Mol. Phys.*, **46**(5), 1109–28 (1982).

[9] Y. Guissani, B. Guillot and F. Sokolic, "Chemical potential of triatomic polar liquids: a computer simulation study," *Chem. Phys.*, **96**, 271–276 (1985).

[10] Y. Guissani, B. Guillot and S. Bratos, "The statistical mechanics of the ionic equilibrium of water: A computer simulation study," *J. Chem. Phys.*, **88**(9), 5850–6 (1988).

[11] Charles H. Bennett, "Efficient Estimation of Free Energy Differences from Monte Carlo Data," *J. Comp. Phys.*, **22**, 245–68 (1976).

[12] M. Mezei and D.L. Beveridge, "Free Energy Simulations," *Annals of New York Academy of Sciences*, **482**, 1–23 (1986).

[13] David A. Kofke and Eduardo D. Glandt, "Monte Carlo simulation of multicomponent equilibria in a semigrand canonical ensemble," *Mol. Phys.*, **63**, 1105 (1988).