SAW_pivot (version 1.0) Documentation

September 4, 2003

1 General considerations

There are two source files: src/public/simul.c and src/public/lib1.c. The latter contains the library routines that actually implement the pivot algorithm. The former uses these routines to run the pivot algorithm, but it only computes the end to end distance for each walk generated. To use this code to do your own simulations you will need to modify simul.c (or write an analogous program) to compute the things you are interested in. Hopefully you won't need to modify the library. The member function

point step_rval(long i)

for the class *walk* returns the ith point along the walk. So it is a good starting point for computing properties of the walk.

The script "run_script" will compile and run the program simul.c with some default values for parameters like number of steps in the walk. The parameters used in simul.c are documented in run_script.

The code is designed so that different lattices (square, cubic, triangular, etc.) are all done by the same code to the extent that this is possible. To avoid having switch statements all over the place for those parts of the code that depend on the particular lattice (which would slow down the code), we use preprocessor directives.

In the file src/public/variable.h, ALGORITHM_NUMBER should be defined to be one of the following numbers:

= 7 for square lattice using all the lattice symmetries
= 11 for triangular lattice
= 5 for hexagonal lattice
= 47 cubic lattice
=383 four dimensional hypercubic lattice
= 2 for square lattice using only two lattice symmetries (reflections in ± 45 deg lines)
= 1 for Manhattan lattice

(The numbers used above are the number of non-trivial lattice symmetries for the lattice.) Note that ALGORITHM_NUMBER determines both the number of dimensions and the particular lattice. It is used in src/public/local.h to determine other variables

(NUM_SYM,SQUARE_LATTICE,TWO_DIMENSIONS...) which are also used by preprocessor directives.

2 Points

There are two classes for points on the lattice: *point* and *rpoint*. The first one is a d-tuple of *long*'s and the second is a d-tuple of *double*. *rpoint* gives the coordinates of the point as a point in R^d. *rpoint* gives the integer coordinates with respect to a basis for the lattice. The bases used are

square : $e1 = (1,0), \quad e2 = (0,1)$
triangular : $e1 = (1,0), \quad e2 = (-1/2, \sqrt{3}/2)$
hexagonal : $e1 = (1,0), \quad e2 = (-1/2, \sqrt{3}/2)$
cubic : $e1 = (1,0,0), \quad e2 = (0,1,0), \quad e3 = (0,0,1)$
hypercubic : $e1 = (1,0,0,0), \quad e2 = (0,1,0,0), \quad e3 = (0,0,1,0), \quad e4 = (0,0,0,1)$

In the pivot algorithm itself we use mainly *point*. *rpoint* will be useful in computing random variables.

3 Walks

Walks are represented by the class *walk*. This data structure is more than just a linear array of points since it is used to avoid carrying out the pivots right after they are accepted. This data structure is explained in "A faster implementation of the pivot algorithm for self-avoiding walks," by Tom Kennedy, J. Statist. Phys. 106, 407-429 (2002). This paper is archived in www.arXiv.org as cond-mat/0109308

long nsteps; The number of steps in the walk.

point steps;* The sites the walk visits. They run from steps[0] to steps[nsteps].

long niter; This is the "age" of the walk, i.e., the number of iterations of the pivot algorithm that have been applied to it. It is in multiples of INNER_LOOP, typically a million.

long npivot; This is the number of pivots that have been accepted but not yet applied to the walk.

long ptime;* $l_1, l_2, \cdots l_n$ in the notation of the paper.

long igroup;* $g_1, g_2, \cdots g_n$ in the notation of the paper.

point shift;* $x_1, x_2, \cdots x_n$ in the notation of the paper.

4 Excluded region

We can simulate the walk in the full space or in several different subsets. The class *excluded_class* specifies the region the walk is excluded from.

int region; region specifies the region the walk is excluded from:
=0 : no excluded region, full plane
=1 : half plane or space, walk must satisfy $y > 0$
=2 : cut: walk not allowed to hit set of points with $x \geq 0$ and $y = 0$.
=3 : quarter: walk not allowed to hit pos x-axis or pos y-axis

In three dimensions region can also be $-1, -2$, but this is to implement some unusual simulations that are not documented here.