STATISTICS OF L_1 DISTANCES IN THE FINITE SQUARE LATTICE

RICHARD J. MATHAR

ABSTRACT. The L_1 distance between two points in a square lattice is the sum of horizontal and vertical absolute differences of the Cartesian coordinates and — as in graph theory – also the minimum number of edges to walk to reach one point from the other. The manuscript contains a Java program that computes in a finite square grid of fixed shape the number of point pairs as a function of that distance.

1. NOTATIONS

We consider the statistics of distances in a finite $M \times M$ square lattice, where distances between points at coordinates (x_1, y_1) and (x_2, y_2) are measured in the L_1 norm [3]:

Definition 1. *(Distance)*

$$d \equiv |x_2 - x_1| + |y_2 - y_1|. \tag{1}$$

The Cartesian coordinates are 0-based in this manuscript, as usual in programming:

$$0 \le x, y < M. \tag{2}$$

Example 1. *The example of Figure 1 shows for the off-center reference coordinate $(x, y) = (4, 3)$ one point with distance 0 (the point itself), 4 points with distance 1, 8 points with distance 2,. . . , 3 points with distance 10 and 1 point with distance 11.*

Definition 2. *(Number of points with distance d) $N_M(x, y, d)$ is the number of points in the $M \times M$ square lattice with distance d to the reference point (x, y).*

The points with a common distance d build a square lattice tilted by 45° relative to the host lattice; its lattice constant is larger by a factor $\sqrt{2}$. They fall basically into four sets to the NE, NW, SW and SE directions.

Definition 3. *(Marginal statistics)*

$$N_M^{(u)}(d) \equiv \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} N_M(x, y, d) \tag{3}$$

is the number of unordered pairs of points with distance d in the finite square lattice. If the case $d = 0$ is excluded, it is the statistics of distinct point's distances.

Date: June 13, 2023.

2020 *Mathematics Subject Classification.* Primary 51F10; Secondary 05-04.

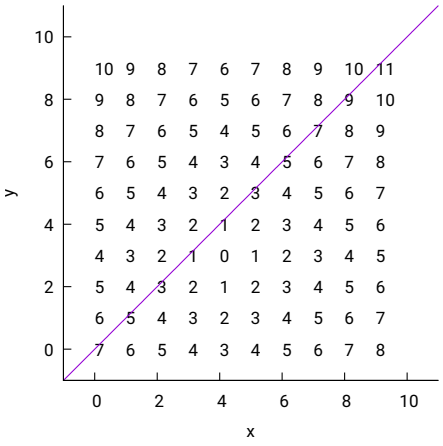

FIGURE 1. Distances in a 10×10 lattice relative to the point $(4,3)$. Magenta diagonal as a guide to the eye.

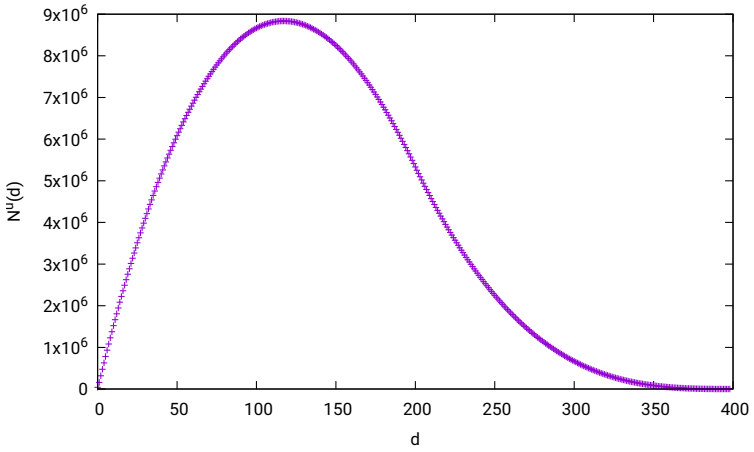

FIGURE 2. Example of distances $N_M^{(u)}(d)$ for a $M=200$ size grid.

Unordered in that definition points out that with this summation all distances between distinct points are counted twice. The distance $d=3$ from $(1,4)$ to $(2,6)$, for example, is counted once in the contribution $N_M(1,4,3)$ and again in the contribution $N_M(2,6,3)$. To count distances between distinct points only once, there is the *ordered* variant,

$$N_M^{(o)}(d) \equiv \begin{cases} N_M^{(u)}(d), & d=0; \\ N_M^{(u)}(d)/2, & d>0. \end{cases} \tag{4}$$

Remark 1. *For infinite lattices in cubic lattices much more is known* [1, A035607].

2. Small neighborhood and Symmetries

The 45° tilted lattice is "complete" as long as the reference point is sufficiently close to the center and d not too large:

(5)
$$N_M(x,y,d) = \begin{cases} 1, & 0 \le x < M \wedge 0 \le y < M \wedge d = 0; \\ 4d, & x + d < M \wedge x - d \ge 0 \wedge y + d < M \wedge y - d \ge 0 \wedge d > 0. \end{cases}$$

If d is larger than the requirements in this formula, N_M is smaller because the edges of the $M \times M$ lattice clip some of these four slanted edges of the sublattices.

As a numerical check, each of the M^2 points of the statistics must have a well-defined distance d:

$$\sum_{d=0}^{2M-2} N_M(x,y,d) = M^2, \tag{6}$$

where the upper limit in the sum refers to the case where the reference point is in one of the 4 corners of the lattice. In consequence

$$\sum_{d=0}^{2M-2} N_M^{(u)}(d) = M^4, \tag{7}$$

so $N_M(d)/M^4$ is the probability of distance d between two independently randomly sampled points in the finite lattice. If we would sample only distances between distinct points, (6) would read $\sum_{d=1}^{2M} N_M(x,y,d) = M^2 - 1$, (7) would read $\sum_{d=1}^{2M} N_M^{(u)}(d) = M^2(M^2-1)$, and $N_M^{(u)}(d)/[M^2(M^2-1)]$ becomes the probability of distance d sample over pairs of distinct points.

The number of points at distance d does not change if the reference point is moved to another point by one of the symmetries (flips along horizontal or vertical axes, along diagonals, rotations by 90°) of the square lattice, for example:

$$N_M(x,y,d) = N_M(M-1-x,y,d) = N_M(x,M-1-y,d) = N_M(y,x,d). \tag{8}$$

To compute the statistics (3) it therefore suffices to sample approximately an octant of the lattice, paying attention to the multiplicity of 4 of points on the diagonals, and (if M is odd) multiplicity of 4 of points on the center lines and multiplicity of 1 of the center point:

$$\begin{aligned} N_M^{(u)}(d) = 8 \sum_{x=1}^{\lfloor M/2 \rfloor - 1} \sum_{y=0}^{x-1} N_M(x,y,d) + 4 \sum_{x=0}^{\lfloor M/2 \rfloor - 1} N_M(x,x,d) \\ + [2 \nmid M] \left\{ 4 \sum_{x=0}^{(M-3)/2} N_M(x, \frac{M-1}{2}, d) + N_M(\frac{M-1}{2}, \frac{M-1}{2}, d) \right\} \end{aligned} \tag{9}$$

where $[\ldots]$ is the Iverson bracket (1 if the statement in the square brackets is true, 0 if false) [2].

Appendix A. Java Implementation

A.1. **Compilation and Use.** The source code in this appendix contains a Java program which is compiled for example with

```
javac -cp . de/mpg/mpia/rjm/SquareGridL1.java
jar cfm SquareGridL1.jar META-INF/mani $de/mpg/mpia/rjm/SquareGridL1.class
```

and then called with the syntax

```
java -jar SquareGridL1.jar [-o] width height
```

or

```
java -jar SquareGridL1.jar [-x xpiv -y ypiv] width height
```

The two last integer parameters `width` and `height` are the nonnegative number of lattice points along the horizontal and vertical direction. (In that respect the program is more versatile than the main section of the manuscript where width and height are the same M.)

- In the first format of the call the counts $N^{(u)}(d)$ are computed if the option `-o` is absent, otherwise $N^{(o)}(d)$.
- In the second format of the call the counts $N^{(u)}(x, y, d)$ are computed with integer coordinates `xpiv` and `ypiv` specifying the pivotal (reference) coordinates for all distances.

The program handles reference coordinates outside the finite grid correctly. It samples all distances exactly by tracking and clipping all four edges of the associated 45° slanted grid. (There is no Monte Carlo-alike approximation in the results.)

Example 2. *The case of Figure 1 is obtained with*

```
java -jar SquareGridL1.jar -x 4 -y 3 10 10
```

Example 3. *The numbers that define Figure 2 are obtained with*

```
java -jar SquareGridL1.jar 200 200
```

For both formats of the call the output is a list of d N pairs starting at $d = 0$, and a last line that starts with a hash (sharp) sign. The last line contains three integers: the sum $\sum_{d\ge 0} N$ in the earlier list, and two median values of d. The first median value considers $d = 0$ to be part of the statistics, the second median value ignores the $d = 0$ contribution of the statistics (and may be larger than the first for that reason).

A.2. **File de/mpg/mpia/rjm/SquareGridL1.java.**

```
/*
* $Header: de/mpg/mpia/rjm/SquareGridL1.java$
*/

/** @file
* Statistics of L1 distances in a finite width x height square grid.
* @author Richard J. Mathar
*/

package de.mpg.mpia.rjm ;

import java.util.* ;
import java.lang.* ;
import java.math.* ;

/**
* @brief compute the number of points in a finite square grid
```

```
* across the range  of L1 distances.
* @since 2023-06-09
* @author Richard J. Mathar
*/
public class SquareGridL1
{
    /** the dimension (width and height, horizontal and vertical) number
    * of points) of the square grid
    */
    int width ;
    int height ;

    /**
    * Constructor with a predefined size of the grid
    * This just stores away the size parameters and does not
    * compute anything.
    * @param sx The number of points along x
    * @param sy The number of points along y
    */
    public SquareGridL1(int sx, int sy)
    {
        width = sx ;
        height = sy ;
    } /* ctor */

    /**
    * L1 (Manhattan) distance between two points
    * @param x1 Horizontal cordinate of first point
    * @param y1 Vertical cordinate of first point
    * @param x2 Horizontal cordinate of second point
    * @param y2 Vertical cordinate of second point
    * @return |x2-x1|+|y2-y1|
    */
    public static int d(int x1, int y1, int x2, int y2)
    {
        return Math.abs(x2-x1)+Math.abs(y2-y1) ;
    }

    /**
    * Number N(xpiv,ypiv,d) of points at distance d to pivotal xpiv,ypiv
    * @param xpiv Horizontal coordinate of pivotal point
    * @param ypiv Vertical coordinate of pivotal point
    * @param d L1 distance of points sampled.
    * @return The number of points in the finite grid with distance d.
    *  The function does not require that xpiv or ypiv are in the range [0,..,width-1] or [0,...,height
    */
    public BigInteger count(int xpiv, int ypiv, int d)
    {
        if ( xpiv+d < width && xpiv-d >= 0 && ypiv+d < height && ypiv-d >= 0)
        {
            /* if point sufficiently close to center and d sufficiently small,
            * employ closed form N=4d.
            */
```

```
    if ( d == 0 )
        return BigInteger.ONE ;
    else
        /* 4d */
        return (new BigInteger(""+d)).shiftLeft(2) ;
}

/* res is the cumulative sum over the 4 slanted edges
*/
BigInteger res = BigInteger.ZERO ;

/* edge to the NE , excluding the point at distance d at the same xpiv (avoid double count).
* x-xpiv + y-ypiv =d
*/
int xmax = Math.min(xpiv+d,width-1) ;
int xmin = Math.max(xpiv+1,0) ; // +1 to exclude the point right to the N
int ymin = Math.max(d+xpiv+ypiv-xmax,0) ;
int ymax = Math.min(d+xpiv+ypiv-xmin,height-1) ;
xmax = Math.min(xpiv+ypiv-ymin+d,width-1) ;
xmin = Math.max(xpiv+ypiv-ymax+d,0) ;
if ( xmax -xmin >=0 )
{
    int xdiff = xmax-xmin+1 ;
    res = res.add(new BigInteger(""+xdiff)) ;
}

/* edge to the NW , excluding the point at distance d at the same y (avoid double count).
* xpiv-x + y-ypiv =d
*/
xmin = Math.max(xpiv-d+1,0) ; // +1 to exclude the point right to the W
xmax = Math.min(xpiv,width-1) ;
ymax = Math.min(-xpiv+xmax+ypiv+d,height-1) ;
ymin = Math.max(-xpiv+xmin+ypiv+d,0) ;
xmax = Math.min(xpiv-ypiv+ymax-d,width-1) ;
xmin = Math.max(xpiv-ypiv+ymin-d,0) ;
if ( xmax -xmin >=0 )
{
    int xdiff = xmax-xmin+1 ;
    res = res.add(new BigInteger(""+xdiff)) ;
}

/* edge to the SW, excluding the point at distance d at the same x (avoid double count).
* xpiv-x + ypiv-y =d
*/
xmin = Math.max(xpiv-d,0) ;
xmax = Math.min(xpiv-1,width-1) ; // -1 to exclude the point right to the S
ymin = Math.max(xpiv+ypiv-xmax-d,0) ;
ymax = Math.min(xpiv+ypiv-xmin-d,height-1) ;
xmin = Math.max(xpiv+ypiv-ymax-d,0) ;
xmax = Math.min(xpiv+ypiv-ymin -d,width-1) ;
if ( xmax -xmin >=0 )
{
    int xdiff = xmax-xmin+1 ;
```

```
        res = res.add(new BigInteger(""+xdiff)) ;
    }

    /* edge to the SE, excluding the point at distance d at the same y (avoid double count).
    * x-xpiv + ypiv-y =d
    */
    xmax = Math.min(xpiv+d-1,width-1) ; // -1 to exclude the point right to the E
    xmin = Math.max(xpiv,0) ;
    ymax = Math.min(xmax-xpiv+ypiv-d,height-1) ;
    ymin = Math.max(xmin-xpiv+ypiv-d,0) ;
    xmax = Math.min(xpiv-ypiv+ymax+d,width-1) ;
    xmin = Math.max(xpiv-ypiv+ymin+d,0) ;
    if ( xmax -xmin >=0 )
    {
        int xdiff = xmax-xmin+1 ;
        res = res.add(new BigInteger(""+xdiff)) ;
    }

    return res ;
} /* count */

/**
* Number N(xpiv,ypiv,d) of points at all distances d.
* @param xpiv Horizontal coordinate of pivotal point.
* @param ypiv Vertical coordinate of the pivotal point.
* @return The number of points in the finite grid with distance d=0,1,2,...
*  The function does not require that xpiv or ypiv are in the range [0,..,width-1]
* or [0,...,height-1].
*/
public BigInteger[] count(int xpiv, int ypiv)
{
    /* maximum distance to the 4 corners of the finite grid
    */
    int dmax = d(xpiv,ypiv,0,0) ;
    dmax = Math.max(dmax,d(xpiv,ypiv,width-1,0)) ;
    dmax = Math.max(dmax,d(xpiv,ypiv,0,height-1)) ;
    dmax = Math.max(dmax,d(xpiv,ypiv,width-1,height-1)) ;

    BigInteger res[] = new BigInteger[1+dmax] ; // +1 because d=0 is included
    /* loop over all reachable distances
    */
    for(int d= 0; d<= dmax; d++)
        res[d] = count(xpiv,ypiv,d) ;
    return res ;
}

/**
* Number N(d) of points at all distances d.
* @param unordered If false, points are considered ordered
*   by some criterion concerning their coordinates.
*   This means that the entries of the vector N[d]
*   that is returned are halved for d>=1.
* @return The number of point pairs in the finite grid with distance d=0,1,2,...
```

```
*  and and point coordinates in the range 0<=x<width and 0<=y<height.
*/
public BigInteger[] count(boolean unordered)
{
    /* maximum distance to the 4 corners of the finite grid
    */
    int dmax = width+height-2 ;

    BigInteger res[] = new BigInteger[1+dmax] ; // +1 because d=0 is included

    /* floor of width/2-1
    */
    int whalf = width/2-1 ;

    /* floor of height/2-1
    */
    int hhalf = height/2-1 ;

    /* loop over all reachable distances
    */
    for(int d= 0; d<= dmax; d++)
    {
        res[d] = BigInteger.ZERO ;
        if ( width == height)
        {
            /* point with 8-fold conjugates in the symmetry
            * This here the branch of the symmetry
            * of the square (dihedral group of order 8)
            */
            for (int xpiv=1 ; xpiv <= whalf ; xpiv++)
            for (int ypiv=0 ; ypiv < xpiv ; ypiv++)
            {
                BigInteger Npiv = count(xpiv, ypiv, d) ;
                res[d] = res[d].add(Npiv.shiftLeft(3)) ;
            }

            /* point with 4-fold conjugates on the diagonals
            */
            for (int xpiv=0 ; xpiv <= whalf ; xpiv++)
            {
                BigInteger Npiv = count(xpiv, xpiv, d) ;
                res[d] = res[d].add(Npiv.shiftLeft(2)) ;
            }

            if ( width %2 !=0 )
            {
                /* point with 4-fold conjugates on center lines
                */
                for (int xpiv=0 ; xpiv <= whalf ; xpiv++)
                {
                    BigInteger Npiv = count(xpiv, (width-1)/2, d) ;
                    res[d] = res[d].add(Npiv.shiftLeft(2)) ;
                }
```

```
        /* point with 1-fold multiplicity in center
        */

        BigInteger Npiv = count((width-1)/2, (width-1)/2, d) ;
        res[d] = res[d].add(Npiv) ;
    }
}
else
{
    /* point with 4-fold conjugates in the symmetry
    * Symmetry of the rectangle, group of order 4
    * by flips along horiz. or vert. axis.
    */
    for (int xpiv=0 ; xpiv <= whalf ; xpiv++)
    for (int ypiv=0 ; ypiv <= hhalf ; ypiv++)
    {
        BigInteger Npiv = count(xpiv, ypiv, d) ;
        res[d] = res[d].add(Npiv.shiftLeft(2)) ;
    }

    if ( width %2 !=0 )
    {
        /* point with 2-fold conjugates on vertical center line
        * if width is an odd number
        */
        for (int ypiv=0 ; ypiv <= hhalf ; ypiv++)
        {
            BigInteger Npiv = count((width-1)/2, ypiv, d) ;
            res[d] = res[d].add(Npiv.shiftLeft(1)) ;
        }
    }

    if ( height %2 !=0 )
    {
        /* point with 2-fold conjugates on horizontal center lines
        * if height is an odd number
        */
        for (int xpiv=0 ; xpiv <= whalf ; xpiv++)
        {
            BigInteger Npiv = count(xpiv, (height-1)/2, d) ;
            res[d] = res[d].add(Npiv.shiftLeft(1)) ;
        }
    }

    if ( height %2 !=0 && width %2 != 0)
    {
        /* point with 1-fold multiplicity in center
        * if width and height are both odd.
        */
        BigInteger Npiv = count((width-1)/2, (height-1)/2, d) ;
        res[d] = res[d].add(Npiv) ;
    }
}
```

```
        }
        /* if ordered, divide all d>0 results by 2
        */
        if ( unordered == false)
            for(int d= 1; d<= dmax; d++)
                res[d] = res[d].shiftRight(1) ;
        return res ;
    }
    /**
    * Print a set of N values to stdout. one distance and count per line
    * followed by two median values (with and without Nstat[0]).
    * @param Nstat The list of integer values
    */
    public static void printN(final BigInteger[] Nstat)
    {
        /* cumulative sum over d=0,1,...
        */
        BigInteger Nsum = BigInteger.ZERO ;

        /* loop over all available distances
        */
        for(int d= 0; d< Nstat.length; d++)
        {
            System.out.println(d+ " " + Nstat[d]) ;
            Nsum = Nsum.add(Nstat[d]) ;
        }
        System.out.print("# " + Nsum) ;

        /* first d (as the median) where Nsum is larger than width*height/2
        * dmed0 includes the d=0 results, dmed ignores them in the median.
        */
        int dmed0 = -1 ;
        int dmed = -1 ;

        /* partial sum (again) to catch the two median values
        */
        BigInteger psum = BigInteger.ZERO ;
        /* loop over all available distances
        */
        for(int d= 0; d< Nstat.length && ( dmed0 <0 || dmed < 0) ; d++)
        {
            psum = psum.add(Nstat[d]) ;
            /* 2*psum > Nsum ?
            */
            if ( dmed0 <0 && psum.shiftLeft(1).compareTo(Nsum) >=0 )
                dmed0 = d ;
            /* 2*(psum-Nstat[0]) > Nsum?
            */
            if ( dmed <0 && d >0 && psum.subtract(Nstat[0]).shiftLeft(1).compareTo(Nsum) >=0 )
                dmed = d ;
        }
        /* print check sum (which should be width*height) and median distance
        */
```

```
    System.out.println(" " + dmed0 + " " + dmed) ;
} /* printN */

/** Main program
* usage: java -jar SquareGridL1.jar [-o] [-x xpivot -y ypivot] width height
*/
public static void main(String[] args) throws InterruptedException
{
    boolean unordered = true ;

    int xpiv = -1;
    int ypiv = -1;

    /* empty list of arguments: usage hint
    */
    if ( args.length == 0 )
    {
        final SquareGridL1 tmp = new SquareGridL1(0,0) ;
        System.out.println("Usage: java -cp . " + tmp.getClass().getName()
            + " [-o] [-x xpivot -y ypivot] width height") ;
        return ;
    }

    for( int optind =0 ; optind < args.length ; optind++)
    {
        if ( args[optind].equals("-o") )
            unordered =false  ;
        if ( args[optind].equals("-x") )
            xpiv = Integer.parseInt(args[++optind]) ;
        if ( args[optind].equals("-y") )
            ypiv = Integer.parseInt(args[++optind]) ;
    }

    /* last two command line arguments are the grid area (width and height)
    */
    int height = Integer.parseInt(args[args.length-1]) ;
    int width = Integer.parseInt(args[args.length-2]) ;

    SquareGridL1 grid = new SquareGridL1(width,height) ;

    if ( xpiv != -1 || ypiv != -1 )
    {
        /* the case where the user provided an explicit pivotal point.
        */
        BigInteger Nstat[] = grid.count(xpiv,ypiv) ;

        grid.printN(Nstat) ;
    }
    else
    {
        /* marginal the case where the user provided no pivotal point
        */
        BigInteger Nstat[] = grid.count(unordered) ;
```

```
            grid.printN(Nstat) ;
        }
    } /* main */
} /* SquareGridL1 */
```

A.3. File META-INF/mani.

```
Manifest-Version: 1.0
Built-By: Richard J. Mathar
Main-Class: de.mpg.mpia.rjm.SquareGridL1
Name: de/mpg/mpia/rjm/SquareGridL1
Specification-Title: vixra:2306.xxxx
Specification-Version: 1.0
```

References

1. O. E. I. S. Foundation Inc., *The On-Line Encyclopedia Of Integer Sequences*, (2023), https://oeis.org/. MR 3822822
2. Donald E. Knuth, *Two notes on notation*, Am. Math. Monthly **99** (1992), no. 5, 403–422. MR 1163629
3. Joan Serra-Sagristà, *Enumeration of lattice points in l_1 norm*, Inf. Proc. Lett. **76** (2000), 39–44.

URL: `https://www.mpia.de/~mathar`

Max-Planck Institute of Astronomy, Königstuhl 17, 69117 Heidelberg, Germany