SAWS AND SAPS ON THE CAYLEY GRAPH OF A GROUP

MIKE ZABROCKI

These are notes from the talk that I gave at the Algebraic combinatorics seminar at Fields on Friday, October 31, 2014.

A walk in a lattices is a sequence of points

$$origin = p_0 \to p_1 \to p_2 \to \ldots \to p_r$$

such that p_i and p_{i+1} are adjacent in the lattice. A walk is self avoiding (SAW) if $p_i \neq p_j$ for any $i \neq j$. The length of the walk is r. A walk is a self avoiding polygon (SAP) if $p_r = origin$ and $p_0 \to p_1 \to p_2 \to \ldots \to p_{r-1}$ is a SAW.

Now the lattice of this walk can be any sort of configuration of regular points. The usual lattice that one considers is the square lattice, but one could also imagine a triangular, hexagonal (as Neal spoke about) or the three dimensional cubic lattice (integer points (a_1, a_2, a_3) for $a_i \in \mathbb{Z}$).

One of the lattices that we can consider is the Cayley graph of a group. Let

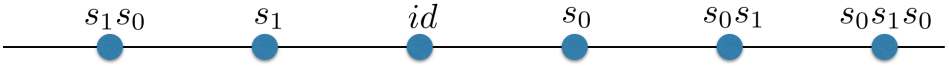
$$identity = g_0 \to g_1 \to g_2 \to \ldots \to g_r$$

where the g_i are the elements of the group G with generators $\{s_1, s_2, \ldots, s_n\}$ and g_i and g_{i+1} are adjacent if $g_{i+1} = g_i s_{d_i}$ with $1 \leq d_i \leq n$.

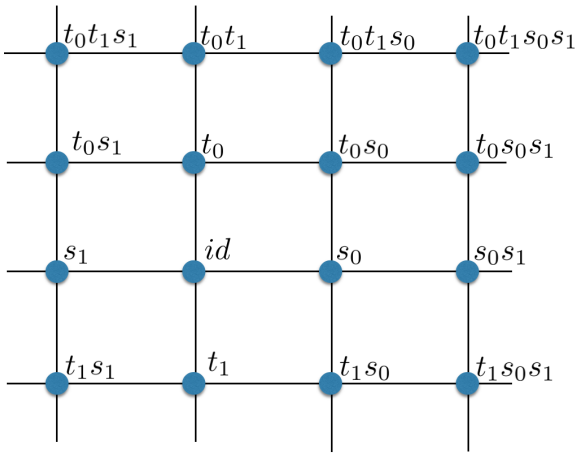
The condition that the path intersects itself is that $g_i = s_{d_1} s_{d_2} \cdots s_{d_i}$ is equal to $g_j = s_{d_1} s_{d_2} \cdots s_{d_j}$ for some $j > i$. This implies that $s_{d_{i+1}} s_{d_{i+2}} \cdots s_{d_j} = id$.

One of the reasons that this could be an interesting way of looking at the problem is that there are classes of groups whose Cayley graphs coincide with lattices that have previously been studied. But is is also interesting because there are other groups in this family whose lattices have not been as well studied. It motivates study of these other lattices and encourages us to look at them from the group theory perspective. We are especially interested in the lattices for groups that arise because they can be classified for geometric reasons.

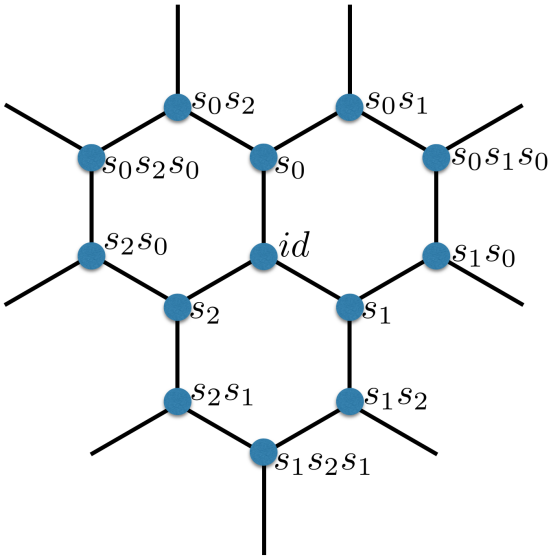
Example 1. Consider the group $\tilde{A}_1$ generated by two generators $\{s_0, s_1\}$ satisfying the relations $s_0^2 = s_1^2 = 1$ and $s_0 s_1 \neq s_1 s_0$. The elements of the group are indexed by $s_0 s_1 s_0 \cdots s_{1/0}$ and $s_1 s_0 \cdots s_{1/0}$. The Cayley graph resembles a single line.



Example 2. Consider the group $\tilde{A}_1 \times \tilde{A}_1$ generated by two generators $\{s_0, s_1, t_0, t_1\}$. The generators satisfy the relations $s_i^2 = t_i^2 = 1$ for $i = 0, 1$, $s_i t_j = t_j s_i$ for $i, j \in \{0, 1\}$ and otherwise we have that $s_0 s_1 \neq s_1 s_0$ and $t_1 t_0 \neq t_0 t_1$. are indexed by $s_0 s_1 s_0 \cdots s_{1/0}$ and $s_1 s_0 \cdots s_{1/0}$. The Cayley graph can be represented as a square lattice.

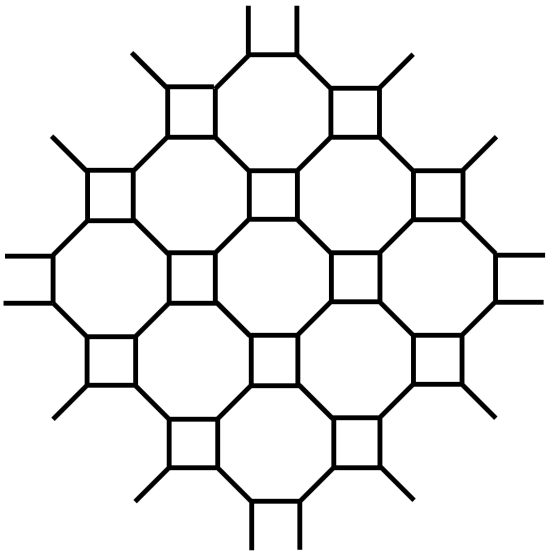


Example 3. Consider the group $\tilde{A}_2$ which has three generators $\{s_0, s_1, s_2\}$. The generators satisfy the relations $s_i^2 = 1$ and $s_i s_{i+1} s_i = s_{i+1} s_i s_{i+1}$ for $i = 0, 1, 2$ where the indices of the generators are all taken to be mod 3. The Cayley graph is the hexagonal lattice.

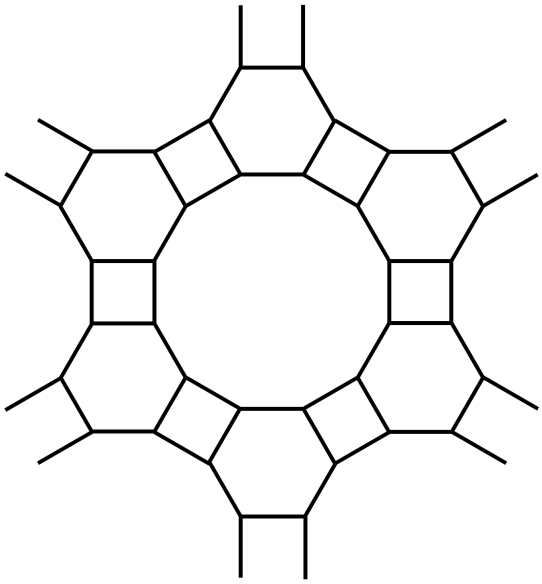


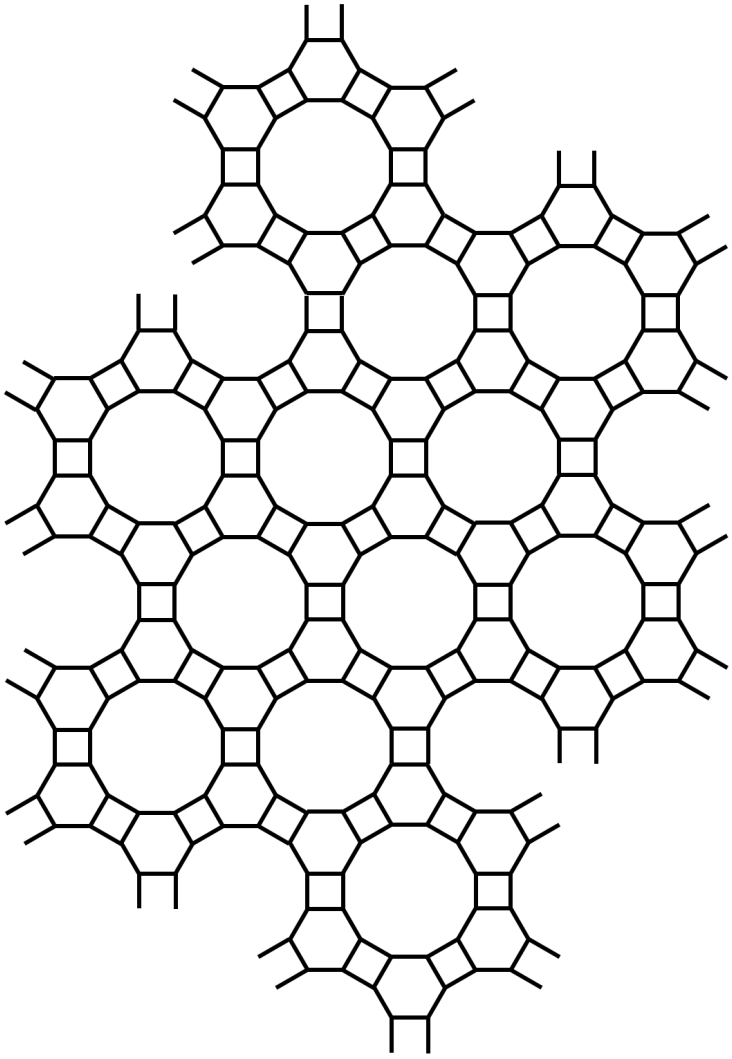
Example 4. The next most general example is the group called $\tilde{C}_2$ (affine C_2) which has three generators again $\{s_0, s_1, s_2\}$ but now this time the relations that the generators

satisfy are $s_i^2 = 1$ for $i = 0, 1, 2$ and $s_i s_{i+1} s_i s_{i+1} = s_{i+1} s_i s_{i+1} s_i$ for $i = 0, 1$ and $s_0 s_2 = s_2 s_0$. This group has the truncated square tiling lattice (also referred to as $(4, 8^2)$) as its Cayley graph. It looks like the following diagram without the labeling of the vertices.



Example 5. The only other two dimensional affine group that can be realized in the plane is $\tilde{G}_2$ (affine G_2) which has three generators $\{s_0, s_1, s_2\}$ that satisfy are $s_i^2 = 1$ for $i = 0, 1, 2$ and $(s_0 s_1)^3 = (s_1 s_2)^6 = (s_0 s_1)^2 = 1$. The Cayley graph has the following form.





It is difficult to find pictures of this lattice on the internet. I have seen in a book that it is sometimes called the cross lattice or a $(4, 6, 12)$ tiling of the plane. I would normally just call it the Cayley graph of affine G_2. It is also called the truncated trihexagonal tiling or (as Conway calls it truncated hexadeltille) and even has a wikipedia page [3].

All of these examples are planar lattices, but we can just as easily define walks on groups that are reflection groups in higher dimensions or in hyberbolic spaces.

Now lets see how we can use algebraic computational tools to count paths in these lattices. These techniques are not supposed to compete with enumeration methods for

computing these paths, because certainly generating them with a low level language is much more efficient. Instead these techniques can be used to look at enumeration for all lattices arising from groups in the same light.

Denote s_i as the generators of my group and let $\tilde{s}_i$ be non-commutative variables in a free algebra. Let $\mathcal{A}_n = \mathbb{Q}\langle \tilde{s}_0, \tilde{s}_1, \cdots, \tilde{s}_n \rangle$. Also let I be the monomial ideal

$$\langle \tilde{s}_{d_1}\tilde{s}_{d_2}\cdots\tilde{s}_{d_\ell} : s_{d_1}s_{d_2}\cdots s_{d_\ell} = id \rangle \ .$$

A basis of $\mathcal{A}$ consists of words in the non-commutative variables $\tilde{s}_i$ which we will consider as a walk on the lattice of the Cayley graph of the group.

The quotient $\mathcal{A}/I$ has as a basis $\tilde{s}_{a_1}\tilde{s}_{a_2}\cdots\tilde{s}_{a_\ell}$ such that no sequence of consecutive letters (factor) in this word are a word in the generators of the group that reduces to the identity.

Moreover we have that the ideal I is minimally generated by

$$\langle \tilde{s}_{d_1}\tilde{s}_{d_2}\cdots\tilde{s}_{d_\ell} \text{ such that } \tilde{s}_{d_1}\tilde{s}_{d_2}\cdots\tilde{s}_{d_\ell} \text{ represents a SAP in the Cayley graph } \rangle \ .$$

That is I is the ideal is generated by the words which do not have a subfactor (a consecutive sequence of letters) which reduces to the identity.

I did a little bit of programming to implement this as generally as possible. The problem is that I was using GAP and Sage in extremely inefficient manner. They constantly had to communicate back and forth between each other while it would have been much easier to just to implement it all in GAP or all in Sage.

I needed to rely on the `GBNP` package in GAP which computes non-commuative Grobner bases. This was not hard to install at all. I just unpacked the file that I downloaded from the internet and placed it in the directory `/Applications/sage/local/gap/latest/pkg`. After that I was able to execute the GAP command `LoadPackage("GBNP");` or the Sage command `gap.eval("LoadPackage(\"GBNP\");")` to load the package so that the commands are available in GAP.

Here are my Sage programs:

```
def is_identity( w, Grp ):
    """
    w is a word in 1...n and Grp is a group with n generators
    then return true if w reduces to the identity
    """
    gens = list(Grp.group_generators())
    return mul([gens[a-1] for a in w], Grp.one())==Grp.one()

def makeGB( ell, Grp ):
    """
    Grp must be a group with generators that can be accessed
    by the method .group_generators()
```

```
    ell is the maximum length of the SAPs that we are looking for
    Note: remove the line mod(d,2)==0 if using a lattice that
    has self avoiding polygons of odd length
    """
    out = [] # list consisting of all self avoiding polygons
    n = len(Grp.group_generators())
    for d in range(1,ell+1):
        if mod(d,2)==0:
            for w in Words(n,d):
                if all(not u.is_factor(w) for u in out)
                  and is_identity(w, Grp):
                    out.append(w)
    return [list(w) for w in out]
```

The function `makeGB` computes a list consisting of all self avoiding polygons in the Cayley graph of `Grp` of length less than or equal to `ell`.

Then if we compute

```
sage: gap.eval("LoadPackage(\"GBNP\");")
'true'
sage: G = WeylGroup(['C',2,1])
sage: gap.set('LM',gap(makeGB(10,G)))
gap.eval("HilbertSeriesQA(LM,3,12);")
'[ 1, 3, 6, 12, 22, 42, 80, 152, 284, 536, 988, 1848, 3448 ]'
```

then the first 11 terms of this sequence are correct, the last one should be off by some amount. Note that in the command `HilbertSeriesQA(LM,3,12);` the 3 indicates that there are three generators of the group. I also computed the number of self avoiding polygons up to length 18 (up to length 20 might be long). I entered this data in the oeis under sequence `A249565`.

Note that the number of unrooted, unoriented SAPs of length $2n$ is equal to the number of oriented SAPs divided by n. These numbers were published in [1] up to length 150. Unfortunately in order to squeeze the number of self avoiding walks out of them you actually need the words themselves and not just the numbers of those words.

length	oriented SAPs	unoriented SAPs	unrooted unoriented SAPs
2	3	3	
4	2	1	1
6	0	0	0
8	4	2	1
10	20	10	4
12	36	18	6
14	84	42	12
16	280	140	35
18	684	342	76

This code really isn't fast, but it can be used to compute the self avoiding walks in the square lattice as well. Assuming that we have already executed the command `gap.eval("LoadPackage(\"GBNP\");")` in the Sage session elsewhere, then

```
sage: G = WeylGroup(['A',1,1])
sage: G2 = G.cartesian_product(G)
sage: gap.set('LM',gap(makeGB(10,G2)))
gap.eval("HilbertSeriesQA(LM,4,12);")
'[ 1, 4, 12, 36, 100, 284, 780, 2172, 5916, 16268, 44100, 120284, 327868]'
```

In this case there are 4 generators of our group and so we compute this with the GAP command `HilbertSeriesQA(LM,4,12);`. The term 327868 is not going to be right because we are using the self avoiding polygons of length 10 to compute the the self avoiding walks of length up to 11. Indeed, if you look at sequence `A001411` in the OEIS you see that there are 324932 self avoiding walks of length 12.

An interesting one is the lattice for $\tilde{G}_2$.

```
sage: gap.set('LM',gap(makeGB(18,WeylGroup(['G',2,1]))))
sage: gap.eval("HilbertSeriesQA(LM,3,19);")
[ 1, 3, 6, 12, 22, 42, 78, 146, 264, 490, 894, 1646, 3012, 5528, 10086,
18472, 33636, 61436, 111612, 203336 ]
```

References

[1] I. Jensen, A. J. Guttmann, Self-avoiding walks, neighbour-avoiding walks and trails on semiregular lattices, J. Phys. A: Math. Gen. 31 (1998) 8137–8145.

[2] A. J. Guttmann, slides for talk "Self-avoiding walks and polygons on non-regular lattices", `http://www.ms.unimelb.edu.au/~tonyg/lectures/hyper1.pdf`.

[3] some relevant Wikipedia entries,
`http://en.wikipedia.org/wiki/Connective_constant`
`http://en.wikipedia.org/wiki/List_of_convex_uniform_tilings`
`http://en.wikipedia.org/wiki/Truncated_square_tiling`
`http://en.wikipedia.org/wiki/Square_tiling`

http://en.wikipedia.org/wiki/Hexagonal_tiling
http://en.wikipedia.org/wiki/Truncated_trihexagonal_tiling