To Do/Ideas to think about

- Answer questions about maps - also think about if/how the correspondence between PK_n and PC_{n+1} works when $n \neq m$ (6/12)
- Generalize choosing a nice basis which is linear combinations of characters - use a basis adapted to a chain of subgroups. See Fassler
- Figure out how to fully make the connection to wreath products. Identify which irreducible representations of the wreath product the parking module decomposes into. Should be able to leverage the nice basis?
- Think about indicator functions. One possible candidate would be $1_{I,D}$ which is 1 on preference lists which send the ith car d cars past it's preference. (ie specifying the displacement of the ith car (6/14)
- 3 row partitions, $N_d(\lambda)$?
- Write code to figure out distance from uniform distribution for distribution of desired spots?
- Read (continued) - papers from Prof O
- Read - regarding Mackey theory (see Representation Theory of Finite Groups: Algebra and Arithmetic)
- Read and work through examples of constructing bases from a representation (see Group Theoretical Methods and Their Applications by Albert Fassler)

Prof O - right now I'm a little bit distracted by how many directions there are. Can we narrow things in a little bit for the next direction to focus on?

Reading Notes

Stanley and Yin 2023

- New statistics: Repeats (same pref as previous car), leading elements (want same spot as first car), 1s, left to right maxima (where did last car park), probability that a preference list is a parking function when you do probabilistic parking (probability p to go forward, probability $p-1$ to go backwards
- What is a Prufer code? Referred to Wikipedia. From Prufer code to tree: degrees of node is the number of times it appears in the sequence + 1. Go through the Prufer code in order connecting the first degree 1 node that is left to that node in the Prufer code. Apparently is a bijection with parking functions.
- Seems like the equality of $P_n(x, y, z, w)$ and $Q_n(x, y, z, w)$ implies that a lot of properties of trees are the same. I wonder if you could somehow extend this to the similarities of some pair of actions?

- combinatorics people describe partitions different then people who study the representation theory of S_n

More thoughts because I need to write them down so they aren't just wandering around in my head

On generating functions with multiple variables. If you know that $F(x, y) = G(x, y)$, then you know that $F(x, 1) = G(x, 1)$ and $F(1, y) = G(1, y)$. So generating functions with multiple variables being equal tells you information about all of the individual variables, but this doesn't go the other way. As in $F(x, 1) = G(x, 1)$ and $F(1, y) = G(1, y)$ does NOT imply $F(x, y) = G(x, y)$ (which when you write it down, of course it doesnt).

It seems like being able to prove that the generating function in multiple variables are equal where you are summing over two different types of objects (like between parking functions and trees in this paper) is also a statement about the kind of bijections you should be able to find between the two spaces. In that you should be able to find a bijection f where if an element π had the property of parking functions associated with x before the mapping, then the resulting tree $T = f(\pi)$ has the property of trees associated with x after the mapping.

That makes me wonder what that mapping looks like, and if there is some way to make it more well defined by phrasing it in terms of properties you want as relates to the action of some group. Or I guess you could construct the action of $C_n^m \wr S_m$ on trees if you had a map that you liked the properties of. I don't know how actually meaningful that is though.

See annotations on page 29 and 30 for working out what this bijection would look like for the elements for which there is no choice. I didn't see any obvious patterns. Also I don't know how they are getting the (1s) column.

- You can generate a prime parking function at random using $\mathbb{Z}/(n-1)\mathbb{Z}^n$. There will be exactly 1 prime parking function in each coset of the diagonal subgroup $\mathbb{Z}/(n-1)\mathbb{Z}^n$
- For information on generalized displacement in u-parking functions, points to Yan section 1.4.3.

Beck et al 2015 - Parking functions, shi arrangements, and mixed graphs

Note - one bijection from parking functions to trees goes as follows. Send parking function π to the Prufer code $(\pi_2 - \pi_1, \cdots, \pi_n - \pi_{n-1})$, mod $n+1$. Then make the tree for the Prufer code.

Gessel and Seo 2005

I skipped to page 17

- When I was reading page 18, and the generating function that they give for parking functions which take an additional c spots, it got me thinking about defect even though

it's just describing $PK_{n+c,n}$. That made me wonder if there's a way in which you could use the circular construction of a parking function for n spots and m cars to construct a defect d parking function for m spots and m cars.

This is a thought we have talked about before. The difficulty comes from the the fact that you would have to filter out any preference lists which include cars wanting to park after m and any preference lists where the last $n + c$ spots are unoccupied

- Gives an explicit expression for the number of lucky cars for $n + c$ parking spots and n cars

Decomposing some functions from Stanley and Yin
ie I have a hammer and someone handed me more nails

I looked at what happened when I decomposed defect using the two different bases that we have been working with. This adapted basis does particularly poorly for desents in the parking outcome for a circular parking lot and skyscrapers (left to right max in the paper), which are both associated with the resulting parking outcome of a preference list.

```
In [271]: by_value(decompose(des, 3,3, False), 3,3)
8
-7.5-2.598j          : 002,
-7.5+2.598j          : 001,
-3.-5.196j           : 021,
-3.+0.j              : 010, 020,
-3.+5.196j           : 012,
1.5-2.598j           : 022,
1.5+2.598j           : 011,
24.+0.j              : 000,

In [272]: by_value(character_to_new_basis_3.dot(decompose(des, 3,3, False)), 3,3, labels = basis_labels_3)
11
-7.5-2.598j          : S_3*200,
-7.5+2.598j          : S_3*100,
-3.-5.196j           : S_3*012,
0.+10.392j           : 012,
1.5-2.598j           : 220,
1.5+2.598j           : 110,
3.+0.j               : 001, 002,
3.+5.196j            : S_2*012, S_2*102,
4.5-2.598j           : S_2*001,
4.5+2.598j           : S_2*002,
24.+0.j              : 000,
```

```
In [276]: by_value(decompose(skyscrapers, 3,3, False), 3,3)
9
-6.-5.196j           : 020,
-6.+0.j              : 021, 012,
-6.+5.196j           : 010,
-1.5-12.99j          : 002,
-1.5+12.99j          : 001,
7.5-2.598j           : 022,
7.5+2.598j           : 011,
39.+0.j              : 000,

In [277]: by_value(character_to_new_basis_3.dot(decompose(skyscrapers, 3,3, False)), 3,3, labels = basis_labels_3)
11
-6.+0.j              : S_3*012,
-4.5-7.794j          : S_2*001,
-4.5+7.794j          : S_2*002,
-1.5-12.99j          : S_3*200,
-1.5+12.99j          : S_3*100,
6.-5.196j            : 001,
6.+0.j               : S_2*012, S_2*102,
6.+5.196j            : 002,
7.5-2.598j           : 220,
7.5+2.598j           : 110,
39.+0.j              : 000,
```

```
In [282]: by_value(character_to_new_basis_3.dot(decompose(expected(.5), 3,3, False)), 3,3, labels = basis_labels_3)
13
-2.+0.j              : S_3*012,
-0.5-0.866j          : S_3*100,
-0.5+0.866j          : S_3*200,
-0.375-0.65j         : 220, 112,
-0.375+0.65j         : 110, 221,
-0.125-0.217j        : S_3*220,
-0.125+0.217j        : S_3*011,
0.25-0.433j          : S_3*221,
0.25+0.433j          : S_3*112,
0.75-1.299j          : 002,
0.75+1.299j          : 001,
2.5-0.j              : 111, 222,
16.+0.j              : 000,

In [283]: by_value(decompose(expected(.5), 3,3, False), 3,3)
8
-2.-0.j              : 210, 120, 201, 021, 102, 012,
-0.5-0.866j          : 010, 001, 022,
-0.5+0.866j          : 020, 011, 002,
-0.125-0.217j        : 220, 211, 202,
-0.125+0.217j        : 110, 101, 122,
0.25-0.433j          : 200, 221, 212,
0.25+0.433j          : 100, 121, 112,
2.5+0.j              : 111, 222,
16.+0.j              : 000,
```

```
In [266]: by_value(decompose(lel, 3,3, False), 3,3)
5
-10.-17.321j         : 001, 101, 011, 111, 211, 121, 221,
-10.+17.321j         : 002, 202, 112, 212, 022, 122, 222,
-1.-17.321j          : 201, 021,
-1.+17.321j          : 102, 012,
20.+0.j              : 100, 200, 010, 110, 210, 020, 120, 220,
65.+0.j              : 000,

In [267]: by_value(character_to_new_basis_3.dot(decompose(lel, 3,3, False)), 3,3, labels = basis_labels_3)
14
-30.-17.321j         : S_2*110,
-30.+17.321j         : S_2*220,
-21.-17.321j         : 201,
-10.-17.321j         : S_3*100, 111, S_3*221,
-10.+17.321j         : S_3*200, S_3*112, 222,
-1.-17.321j          : S_3*012,
0.-34.641j           : S_2*112,
0.+34.641j           : S_2*102, 012, S_2*221,
20.-0.j              : S_3*011, S_3*220,
21.-17.321j          : 120,
21.+17.321j          : S_2*012,
30.-17.321j          : S_2*002,
30.+17.321j          : S_2*001,
65.+0.j              : 000,
```

```
In [268]: by_value(decompose(ones, 3,3, False), 3,3)
2
9.-0.j               : 100, 200, 010, 020, 001, 002,
27.+0.j              : 000,

In [269]: by_value(character_to_new_basis_3.dot(decompose(ones, 3,3, False)), 3,3, labels = basis_labels_3)
2
9.+0.j               : S_3*100, S_3*200,
27.+0.j              : 000,
```

```
In [280]: by_value(decompose(repeats, 3,3, False), 3,3)
2
9.+0.j               : 210, 120, 021, 012,
18.+0.j              : 000,

In [281]: by_value(character_to_new_basis_3.dot(decompose(repeats, 3,3, False)), 3,3, labels = basis_labels_3)
3
-9.+0.j              : S_2*102, 201,
9.+0.j               : S_3*012, 120,
18.+0.j              : 000,
```