Problem Definition

I need to run a euchre tournament. I need a chart that tells which player is partners with who, which pairs of partners will play against eachother, and what table they need to sit at, across many rounds. We refer to this as a *rotation chart.*

Ideally, the rotation chart will be in the following constraints.

- Everyone is partnered with every other person exactly once
- Everyone plays against every other person exactly 2 times (2 opponents)
- Each person is assigned 1 seat every round, including dedicated bye seats who are not in play

Criteria

A rotation chart is defined by the number of *players*, and the number of rounds is determined based on the number of players and the constraints that need to be met.

What is a *valid* rotation chart?

- A rotation chart where each player is present exactly once in each round.

What is an *optimal* rotation chart?

- A valid rotation chart that also meets the criteria described above.

What is a round, seat, player, partner, opponent, and table?

Trivial Charts

For less than 4 players, all players are in bye rounds. There is no gameplay, there is no tournament.

We can develop an optimal rotation chart for 4 people easily.

Round	Pairings
1	{1,2} {3,4}
2	{1,3} {2,4}
3	{1,4} {2,3}

Methods for Producing Optimal Rotation Charts

1. We generate valid *pairings* of players across rounds. Then we join them into valid tables. A valid list of pairing may not necessarily mean that a valid rotation chart could be constructed from it.

Generating Pairings for Powers of 2

I think a inductive argument makes sense here. Here's an intuitive argument for parties of size 2^n.

Suppose we have a valid rotation table for a party of size n. We will construct a valid rotation table R_0 for a party of size $2n$. For players 1 through n, we use the n-player rotation table called R_1. We also use the n-player rotation table for players $n+1$ to $2n$ called R_2. We construct new rounds 1 through $n-1$ such that round i of R_0 of the union of round i in R_1 and R_2. We now construct n new rounds. Let round $k+n-1$ be constructed by pairing player i of R_1 with player $i+kmodn$ of R_2. See that each round pairs each of the $2n$ individuals with another individual. Also see that each round is distinct and unique. We have generated a valid rotation table for $2n$.

what am I smoking here

Using our $n=4$ case we can generate the rotation table for $n=8$.

Round	Pairings
1	{1,2} {3,4} {5,6} {7,8}
2	{1,3} {2,4} {5,7} {6,8}
3	{1,4} {2,3} {5,8} {6,7}
4	{1,5} {2,6} {3,7} {4,8}
5	{1,6} {2,7} {3,8} {4,5}
6	{1,7} {2,8} {3,5} {4,6}
7	{1,8} {2,5} {3,6} {4,7}

Generating Pairings for $4n + 2$ given $4n + 1$ Rotation Table

For $4n + 1$ individuals, a rotation table $4n$ rounds where each individual has exactly one bye round exclusively. To construct the

== Some Goals/Open Questions

- Is there a way to iterate through valid charts? (valid meaning, all players exist exactly once in every round)
- What's the equivalence between two charts? There's lots of symmetry. Knowing this during search would help eliminate duplicate entries.
- Related: what's the proper way to sort a chart? Of all symmetries, there should be exactly one representation that is considered "default" or "first".
- Tackle the 4n case before the others - What rules will always improve the table? Is there an algorithm for always resolving a problem in the table? For instance, if I know players x and y have been partners one too many times, how can I directly solve that problem? I might find one round where the players are partners and switch y with another individual who x hasn't been partners with enough.
- Well defined cost functions. These are done well in the python code. They should be marked up.

Symmetry

Each round shares the same symmetry, so let's first examine a single 12 person round.

Each round has 3 tables that can be any order. So that's 3!=6 permutations. For each of those tables, there are 8 possible configurations: each pair has two orderings, and the two pairs can be in either order. That means 3!*8^3 symmetries. Of the total 12! permutations, only 12!/(3!*8^3) are unique. In decimal form, thats 479 001 600 permutations, with 3072 symmetries, for a total of 155925 unique rounds.

A valid rotation chart for 12 people consists of 11 rounds, so there is 155925^11 permutations, or 1.324e57.

This value is too large to be traversed through. But only a subset of these permutations actually meet our constraints.

We can also use some baselines (there's probably a technical term for this?), for instance, we know at least 1 round will be 1 through 12 no matter what, so we could instead consider the remaining 155925^10 permutations, or 8.495e51.

Another approach

Instead of a cost function on valid tables, I could slowly build a set of rounds and perform depth first search until I find a valid solution. I can perform multiple checks along the way to make sure I'm not

adding certain pairs/opponents that break the conditions. This may be "faster" than evaluating random tables, since the cost function is doing a lot of repeated work (as of now). But I do need some other algorithms that are able to check if the rotation chart currently has no valid outcomes before I get there…that might be hard. I could do an approach that uses "both"…randomly add rounds to a list of rounds and check validity after each round is added. If the next round doesn't work, pick the next one.

The enumeration problem could be isolated down to just valid rounds. How do I enumerate through all 155925 valid permutations?

Symmetry Aware Representation

Each pair can be in either order. To avoid maintain that order, I could use 1 number. For instance, a partnered with b could be p_a*p_b where p_i is the ith prime. This would be a unique value among all a,b combinations but because multiplication is commutative, either permutation gives the same result.

That lowers the memory by 1/2, but MIGHT incur some runtime cost for finding where problems are. This tradeoff is likely worth it.

Can I extend this representation between two pairs of partners? If I do the same trick again, I run into problems: * p_a*p_b=P is a large number, which means p_P*p_P' is even bigger! Too big * Multiplication is commutative between partners, which means we lose information about who are partners and who are opponents.

I could simply add both numbers, where one is scaled by a large value. Suppose the max of p_a*p_b is M. Then I could join both sets of partners together with P + M*P'. The problem is that this value will check if you switch P and P' around. Can I get around that?

Is is possible that p_a*p_b+1 is another product of two primes? If it was not, then it could allow me to compose (p_a*p_b+1)(p_c*p_d) which would uniquely factor…but the order still matters doesn't it? That won't work either.

I may not be able to extend this. I may need to explicitly use tuples that are sorted before use.

I might ask this, can I quickly check if two pairs are equal in values and not by position? This is one of those probabilistic things where it would be easy to check if they were unequal (just multiply and compare) but verifying they are equal is harder. But this is a single conditional we are talking about, this isn't worth thinking about more.

Bounds for Valid and Optimal Charts

We can use some combinatorial reasoning to determine possible upper bounds for how many possible rotation charts exist, as perhaps rotation charts that are *valid* and *optimal*.