Card Partitioning

Martin Dupont

October 4, 2018

When solving schafkopf games computationally, we often have the problem in that we need to guess the other players hands. We can often exclude some cards from some players hands, such as if they didn't follow the suit, then they don't have that suit. But in general, we can't really conclude what cards they DO have. Given a set of constraints, we need to divide the remaining unplayed cards into peoples hands, without violating any constraints. This is a constraint satisfaction problem (CSP). CSP's in general may require complicated optimization algorithms, however, I have found an algorithm that can easily find a parsimonious solution in linear time.

This problem has certainly been solved before, but I couldn't find the name of this problem. I'm not claiming that this proof is unique. If anyone finds out the name of the problem, feel free to contact me.

In Schafkopf, we are always trying to guess 3 peoples hands, and we must distribute at most 32 cards. However, the problem and its solution can be easily generalized. So, without further ado:

Theorem 1. *Suppose one has a finite set S that must be divided into n partitions, each of size N_i for $i \in \{1 \cdots n\}$. Call these partitions $\{S_i\}$. In addition, for each S_i, it may only contain a certain subset of elements, which we call A_i. Then this problem has a solution if:*

$$S = \bigcup_i A_i \qquad |A_i| \geq N_i \quad \forall i \qquad \sum_i N_i = |S| \tag{1}$$

and $\forall p$, where p is a choice of an arbitrary number of elements from $\{1 \cdots n\}$,

$$|\bigcup_{i_p} A_{i_p}| \geq \sum_{i_p} N_{i_p}. \tag{2}$$

Which is to say that, for any arbitrary union of A_i's, the size of the union must be greater than or equal to the sum of their number constraints N_i. Call the A_i's constraint sets, the N_i's number constraints, and the S_i's partition sets.

Proof. We prove by propagating constraints:

Firstly, for each i, find the elements in S which are only found in one A_i. We can assign those elements to S_i, and now consider the reduced problem in which those elements are not present. If the original problem satisfied the starting criteria, then the reduced problem will too. This allows us to reduce the problem to the case in which all $A_j \cap A_j \neq \emptyset$.

Going further, suppose one had the case where $|A_i| = N_i$, then we already have partially solved the problem. S_i must equal A_i, and all elements in A_i may be subtracted from the other A_j's. Now we have a new problem with $n \Rightarrow n-1$ and $S \Rightarrow S - A_i$. This operation can also be performed on unions of constraint sets. Suppose $|A_i \cup A_j| = N_i + N_j$ for some $i \neq j$, then the elements in A_i and A_j cannot logically be assigned to any other partition sets, so they may be removed from the other A_k's, and the problem has decomposed into two separate sub-problems, one involving the two constraint sets in question, and one involving all the other constraint sets. It is easy to see that this principle extends to unions of up to $n-1$ constraint sets. We call the process of breaking up the problem into subproblems *propagating constraints.*

So, with this in hand, it is easy to see that we can always assume that we are dealing with a problem for which all $A_i \cup A_i \neq \emptyset$ and all $|\bigcup_p A_{i_p}| > \sum_p N_{i_p}$, as opposed to $\geq$ in the problem definition. If that were not true, we could reduce the problem so that it was true.

Given that all elements $s \in S$ belong to at least two A_i's, we can pick any arbitrary s, and assign it to any arbitrary S_i for which $s \in A_i$. This s can now be removed from all the constraint sets, and we now have a new problem with one less element. Because in the previous problem, $\forall i$, $|A_i| > N_i$, and $|\bigcup_p A_{i_p}| > \sum_p N_{i_p}$, in the new problem, $\forall i$, $|A_i| \geq N_i$, and $|\bigcup_p A_{i_p}| \geq \sum_p N_{i_p}$. We then apply constraint propagation, and assign one more s to an S_i. We repeat this process of assigning and propagating constraints until S is empty.

At every stage of the process, if the initial conditions in eqns. 1 and 2 are satisfied, then they will be satisfied at every later point in the process. Once we have reached the base case, in which there is only one remaining element, the problem is solved. $\square$

Because the above proof is constructive, I implement the constraint optimization problem in my code in the exact same way as the problem is solved, under *distribute_cards.py.* The problem is much simpler in the $n = 3$ case, as there are only two constraints which need to be propagated, checking that $A_i = N_i$ and that $A_i \cup A_j = N_i + N_j$, which only requires checking 6 conditions. For problems with larger n, the combinatorial explosion of constraints to check would become difficult. My current solution is reasonably performant; on my laptop it can solve a typical schafkopf problem in 27 μs.