A Fast and Optimal Hand Isomorphism Algorithm

Kevin Waugh
waugh@cs.cmu.edu
Department of Computer Science
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213 USA

Abstract

In a section of their 2007 paper, Gilpin, Sandholm, and Sørensen outline a technique for indexing poker hands that accounts for suit isomorphisms. Their implementation is specific to Texas Hold'em as it requires a large case analysis, and is not optimal as many cases are omitted. In this paper, we build on their ideas and provide a fast and optimal technique that generalizes beyond Texas Hold'em as well as provide an inverse mapping from an index to a canonical hand.

Introduction

In Texas Hold'em poker, and many other card games, it is common for a large number of hands to be strategically equivalent, or *isomorphic*. For example, the preflop hands $A\heartsuit K\clubsuit$, $A\diamondsuit K\clubsuit$ and $K\heartsuit A\diamondsuit$ can all be thought of as the same hand, AK offsuit, without sacrificing any value to the opponent. In poker, we are free to permute the suits, and to permute the order of the cards dealt within the same round in any way we would like.

When designing computer-card playing programs it is important to consider hand isomorphisms for both computational and space reasons. For example, computing the expected value of a partial hand is up to 24 times more efficient if only isomorphic hands are considered. As a second example, computer poker programs often store their strategies in large tables. When choosing an action, the agent *indexes* the hand to a number, which is is the same for all isomorphic hands, that offsets into this table. The indexing procedure must be efficient leading to "holes", or offsets that have no pre-image, and unnecessarily wasting space.

In the exposition to come, we will construct an indexing function based on the one described in Gilpin, Sandholm, and Sørensen that is efficient to compute; optimal—it has no holes; has an inverse mapping—we can compute a canonical representative from its index; and general—our implementation needs no substantial change to adapt it to other card games, like Pot-Limit Omaha or Leduc Hold'em.

A C implementation of what is described in this paper can be found at `http://www.github.com/kdub0/hand-isomorphism`.

Rank Sets

Our first building block will be a procedure for indexing M*-rank sets*, which are sets of M cards *of the same suit*. With out loss of generality, let us consider the ranks to be in $\{0, 1, \ldots, N\}$ and consider a set in decreasing order.

The *colex* function (Bollobas 1986), provides precisely this indexing. As much of the indexing scheme is based off the same principles, we describe the construction here.

Without loss of generality, we order the sets lexographically. For example if $M = 2$, $\langle 4, 1\rangle > \langle 3, 0\rangle > \langle 2, 1\rangle$. Given an ordering, we can compute an index for a set A if we can count how many sets come before it in the ordering.

Let us start easy and consider $M = 1$, where we have a single element $a_1 \in \{0, \ldots, N\}$. It is not hard to count the number of sets less than a_1, in particular $\text{indexset}_1(a_1) = a_1$. Note, we use the convention that $\binom{n}{m} = 0$ if $n < m$. Computing a set from an index in this case is equally trivial $\text{unindexset}_1(\text{idx}) = \{\text{idx}\}$. For arbitrary $M > 1$, we can construct the index recursively,

$$\text{indexset}_M(a_1, a_2, \ldots, a_M) = \tag{1}$$

$$\binom{a_1 - 1}{M} + \text{indexset}_{M-1}(a_2, \ldots, a_M). \tag{2}$$

This recurrence works by partitioning the sets less than the one of interest into two groups. The first term counts all the sets of length M whose largest element is smaller than a_1. That is, all ways we can choose M elements from a domain of $a_1 - 1$ elements, which does not depend on the remainder of the set. The second term counts the number of sets of length M whose largest element is a_1 that are less than our set. As these sets are tied at the largest element, the remaining elements are important. Since the first is fixed and larger than the remaining elements, we can recurse and forget it.

We unroll this recurrance into a sum

$$\text{indexset}_M(a_1, a_2, \ldots, a_M) = \sum_{i=1}^{M} \binom{a_i}{M - i + 1}. \tag{3}$$

As the number of ranks is typically small, we can use bit operations to sort the set in linear time and we can precompute the necessary binomial coefficients.

The inverse map for arbitrary M follows a similar recur-

sive pattern. In particular,

$$\text{unindexset}_M(\text{idx}) = \tag{4}$$

$$\{x\} \cup \text{unindexset}_{M-1}\left(\text{idx} - \binom{x-1}{M}\right), \text{ where} \tag{5}$$

$$x = \max\left\{x \mid \binom{x}{M} \leq \text{idx}\right\}. \tag{6}$$

That is, we find the largest element of the set by maximizing the first term in the original recurrence subject to the constraint that it be less than the provided index.

Computationally, unindexing is more expensive than indexing. In particular, we use a binary search to compute x, leading to an $O(M \log N)$ procedure as opposed to a very skinny $O(M)$. Approximating the binomial coefficients can reduce the search to no more than a few indices at the cost of a few floating point operations. For example, using

$$\left(\frac{x}{M}\right)^k \leq \binom{x}{M} \leq \left(\frac{xe}{M}\right)^k \tag{7}$$

we need only search for x in

$$\exp\left(\frac{\text{idx} - k + k\log k}{k}\right) \leq x \leq \exp\left(\frac{\text{idx} + k\log k}{k}\right) \tag{8}$$

Rank Groups

Our second building block will be a procedure for indexing $M_1, M_2, \ldots, M_K$*-rank groups*. This is a sequence of K rank sets *of the same suit* with sizes $M_1, M_2, \ldots, M_K$ that do not share cards. Mathematically these operations are straightforward, but implementing them quickly on a computer is more challenging.

First, we note that an M-rank set index has size $\binom{N}{M}$. This means that we compactly compute an index for the first set using the method in the previous section. If we recursively compute an index for the remaining $M_2, \ldots, M_K$-rank group we can combine them as

$$\text{indexgroup}_{M_1,M_2,\ldots,M_K}(A_1, A_2, \ldots, A_K) = \tag{9}$$

$$\text{indexset}_{M_1}(A_1) + \tag{10}$$

$$\binom{N}{M_1} \text{indexgroup}_{M_2,\ldots,M_K}(A_2, \ldots, A_K | A_1). \tag{11}$$

Second, we note that as A_2 cannot share cards with A_1 we are free to rename the ranks in A_2 to fit in $\{0, 1, \ldots, N - M_1\}$. In particular, we will shift all the ranks above a "used slot" downward by one. Again, we can do this efficiently using bit operations. Pseudocode for indexing is shown in Algorithm 1. Note that we represent the sets as binary numbers where bit i is set if rank i is in our set. As such, $\cup$ is logical or, $\cap$ is logical and, smaller is $b - 1$, $|\cdot|$ counts the number of bits set, and largest is the most significant bit. All of these take a single operation on a modern processor.

To unindex a rank group, we can unindex each rank set separately and shift the ranks in subsequent rank sets upward. Again, we use bit operations and tabulation to perform the shifting in $O(M)$ time. Pseudocode for unindexing is shown in Algorithm 2.

Algorithm 1 $\text{indexgroup}_{M_1,M_2,\ldots,M_K}(A_1, A_2, \ldots, A_K|U)$

```
B ← A_1
next ← indexgroup_{M_2,...,M_K}(A_2, ..., A_K | U ∪ A_1)
idx ← C(N−|U|, M_1) next
for i in 1 to M do
    b ← largest(B)
    rank ← b − |smaller(b) ∩ U|
    idx ← idx + C(rank, M_1+i−1)
    B ← B \ {b}
end for
return idx
```

Algorithm 2 $\text{unindexgroup}_{M_1,M_2,\ldots,M_K}(\text{idx}|U)$

```
this ← remainder(idx, C(N−|U|, M_1))
next ← ⌊idx / C(N−|U|, M_1)⌋
B ← unindexset_{M_1}(idx)
A_1 ← ∅
for b ∈ B do
    a ← unused-rank(U, b)
    A_1 ← A_1 ∪ {a}
end for
A_2, ..., A_K ← unindexgroup_{M_2,...,M_K}(next | U ∪ A_1)
return A_1, A_2, ..., A_K
```

Note that the index size for a group is the multinomial coefficient

$$\text{size}(M_1, M_2, \ldots, M_K) = \prod_{i=1}^{K} \binom{N - \sum_{j=1}^{i-1} M_j}{M_i}. \tag{12}$$

Hands

Finally, we are at the stage where we can describe indexing $M_1, M_2, \ldots, M_K$*-hands*, which cannot contain duplicate cards, but can contain all suits. Our first step will be to compute the group index for each suit using Algorithm 1 and will take a single pass over the cards.

The final concept we will need to create a perfect indexing is a *suit configuration*. In particular, a valid suit configuration is a sequence of K numbers for each suit, $c_1^j, c_2^j, \ldots, c_K^j$ such that (a) all groups are full—the sum of the counts group i is M_i, and (b) suit j's sequence is lexicographically greater than or equal to that of suit $j + 1$.

When we have a hand whose suit configuration has no equal suits, we can deterministically map to a canonical hand. That is, using a fixed *a priori* suit ordering, we relabel the suits by their configurations' lexicographic order and within a group we sort by suit and then card. The only added complexity, from a mathematical perspective, comes when two suits have equal counts in all groups. That is, we need a way to break ties between suits with the same configuration. We choose to break ties by their groupindex. Note that two suits can be completely identical and we will account for this when combining the group indices into a final index.

Algorithm 3 $\text{indexhand}_{C_1,C_2,\ldots,C_S}(\text{idx}_1, \text{idx}_2, \ldots, \text{idx}_S)$

this $\leftarrow \text{multiset-colex}_j(\text{idx}_1, \text{idx}_2, \ldots, \text{idx}_j)$
next $\leftarrow \text{indexhand}_{C_{j+1},\ldots,C_S}(\text{idx}_{j+1}, \ldots, \text{idx}_S)$
return this $+ \binom{\text{size}(C_1)+j-1}{j}$next

For a fixed suit configuration, to combine the group indices into a single index taking special care when two or more suits have the same configuration, we will use the multiset-colex index. The construction of the indexing operator and unindexing operator are exactly the same as for colex, except we use the fact that the number of multisets of M elements drawn from a universe of N elements is $\binom{N+M-1}{M}$. Note here that N may be large, as it will be the size of the group index for the particular suit configuration that is replicated, but as M is no bigger than the number of suits. We can quickly compute this without a table.

The indexing algorithm for a particular suit configuration is shown in Algorithm 3. We must impose an ordering on the suit configurations and offset the index we compute based on this ordering. One can use a hash table, or binary search to compute this offset. We use a brute force approach and tabulate from all suit counts to its suit configuration.

Let us demonstrate on a couple of examples. First, an easier case $2\clubsuit A\clubsuit|6\clubsuit J\heartsuit K\heartsuit$. Our first step is to compute the suit configuration, which is $\clubsuit = \langle 2,1\rangle > \heartsuit = \langle 0,2\rangle > \spadesuit = \diamondsuit = \langle 0,0\rangle$. Our fixed suit ordering is bridge order, or $\spadesuit, \heartsuit, \diamondsuit, \clubsuit$. In this case we can write down the canonical hand without first computing the group indices $A\spadesuit 2\spadesuit|6\spadesuit K\heartsuit J\heartsuit$. Now, let us compute the group indices

$$\clubsuit = \binom{12}{2} + \binom{0}{1} + \binom{13}{2}\left[\binom{3}{1}\right] \tag{13}$$
$$= 72 + 78(3) = 306 \tag{14}$$
$$\heartsuit = 0 + 1\left[\binom{11}{2} + \binom{9}{1}\right] \tag{15}$$
$$= 55 + 9 = 64 \tag{16}$$
$$\spadesuit = \diamondsuit = 0. \tag{17}$$

Note that though $6\clubsuit$ is the 5th rank, we shifted it down by one as the $2\clubsuit$ from the previous round is below it. Finally, we combine the indices together noting that the size of the first index is $\binom{13}{2}\binom{11}{1}$,

$$\text{idx} = 306 + \binom{13}{2}\binom{11}{1}64 \tag{18}$$
$$= 55,218. \tag{19}$$

This suit configuration has a of size $\binom{13}{2}\binom{11}{1}\binom{13}{2}$.

Let us now consider a harder example, where two suits have the same configuration, $6\diamondsuit T\clubsuit|J\clubsuit 7\diamondsuit K\heartsuit$. We see that $\diamondsuit = \clubsuit = \langle 1,1\rangle > \heartsuit = \langle 0,1\rangle > \spadesuit = \langle 0,0\rangle$, so let us start by computing the group indices:

$$\clubsuit = \binom{8}{1} + \binom{13}{1}\left[\binom{8}{1}\right] \tag{20}$$
$$= 8 + 13(8) = 112 \tag{21}$$
$$\diamondsuit = \binom{4}{1} + \binom{13}{1}\left[\binom{4}{1}\right] \tag{22}$$
$$= 4 + 13(4) = 56 \tag{23}$$
$$\heartsuit = 0 + 1\left[\binom{11}{1}\right] = 11. \tag{24}$$

Mapping to a canonical hand, we get $T\spadesuit 6\heartsuit|J\spadesuit 7\heartsuit K\diamondsuit$ and computing the final index

$$\text{idx} = \left[\binom{112+1}{2} + \binom{56}{1}\right] + \binom{156+1}{2}11 \tag{25}$$
$$= [6,216 + 56] + 12,246(11) = 140,078. \tag{26}$$

As a final exercise, let us unindex $6,220$ with suit configuration $\langle 1,2\rangle > \langle 1,1\rangle$. First, we compute the size of the first suit configuration as $\binom{13}{1}\binom{12}{2} = 198$. This means the group indices are $\spadesuit = 182$ and $\heartsuit = 31$, as $182 + 198(31) = 6,220$. Hearts are straightforward to unindex. In particular, the ranks are $5|2$, which map to $7\heartsuit|4\heartsuit$. Spades are slightly more complicated, the two groups have indices $0|14$. We need to find the largest x such that $\binom{x}{2} \leq 14$, which turns out to be 5, meaning the two ranks in the second group are $5,4$. Note $\binom{5}{2} + 4 = 14$. As the rank from the first group, 0, is less than them both, they must be shifted up by one before converting to cards. Our final canonical hand is then $2\spadesuit 7\heartsuit|8\spadesuit 6\spadesuit 4\heartsuit$.

Method	**Flop**		**Turn**	
Hyperborean	1,565,954	1.22x	75,427,404	1.37x
Tartanian	3,032,029	2.36x	160,697,537	2.91x
Perfect	1,286,792	1x	55,190,538	1x
	River			
Hyperborean	3,769,937,756	1.55x		
Tartanian	8,516,969,461	3.50x		
Perfect	2,428,287,420	1x		

Table 1: Indexing sizes for Texas Hold'em

Results

In Table 1, we show the index sizes for the flop, turn and river in Texas Hold'em of Hyperborian and Tartanian, along side our indexing scheme. We see that our scheme provides a 1.5x improvement over Hyperborian on the river, and a 3.5x improvement over Tartanian. On a Core 2 Duo desktop machine, we also observed a 3.5x speed up of our indexing scheme over Hyperborean.

References

Bollobas, B. 1986. *Combinatorics*. Cambridge University Press.

Gilpin, A.; Sandholm, T.; and Sørensen, T. 2007. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis in Texas Hold'em poker. In *Twenty Second National Conference on Artificial Intelligence*.