3 lectures on Gunter Lunter's *fastLS*

Nick Patterson
Reich Lab (virtually)
October/November 2020

In a remarkable paper *Haplotype matching in large cohorts ...* (Bioinformatics, 2019)
Lunter gives an algorithm *fastLS*
that finds the best scoring path (we will define this precisely) through a set of reference haplotypes in time independent of the size of the reference set.
We will describe our version in detail

The setup

We have a reference array H of haplotypes.
m rows (haplotypes)
n columns (biallelic SNPs)
Pick an allele for each column (reference allele).
Code $H(i, j) = 0$ if allele for haplotype i, SNP j matches reference
else $H(i, j) = 1$.
So H is an $m \times n$ binary array.

We define a path

$$\mathbf{p} = (p_0, p_1 \ldots p_{n-1}) \;\; 0 \;\; \leq p_i < m$$

and this induces a map $h(\mathbf{p})$ to a binary string in the obvious way.

$$h(\mathbf{p}) = (h_0, h_1, \ldots) \; where \; h_j = H(p_i, j)$$

Given a path $\mathbf{p}$ and a binary string $\mathbf{w} = (w_0, w_1, \ldots w_{n-1})$ we define the number of *jumps* J and *garbles* G. J is number of columns j with $p_j \neq p_{j-1}$ and G number of columns j with $h(\mathbf{p})_j \neq w_j$. The *score* $S(\mathbf{p}, \mathbf{w})$ is

$$S = J\rho + G\mu$$

The problem we aim to solve is:
Preprocess H in work $O(mn)$ and storage $O(mn)$. Then given a $0, 1, 2$ valued string $=(x_0, x_1, \ldots x_{n-1})$ find two binary strings $\mathbf{w_1}, \mathbf{w_2}$ and paths $\mathbf{p_1}, \mathbf{p_2}$ in work $O(n)$, independent of m such that

$$\mathbf{x} = \mathbf{w_1} + \mathbf{w_2}$$

and the score

$$S = \mathbf{S}(\mathbf{p_1}, \mathbf{w_1}) + \mathbf{S}(\mathbf{p_2}, \mathbf{w_2})$$

is minimized.
In fact we don't quite succeed. Our algorithm appears to be $O(n)$ but I have no proof for the worst case. I don't believe Lunter has either.

Preprocessing

We carry out *radix sort*, keeping track of the data flow
We imagine a deck of cards. Initially card i in the deck has haplotype $H(i)$ and index i

We sort lexicographically, with ties broken by the index. We process columns in order $n-1, n-2, \ldots, 0$ from right to left.
For column j we go through the deck in order placing in two piles $A_0(j), A_1(j)$ depending on the value of bit j on the card. Then we simply place $A_0(j)$ on top of $A_1(j)$.
By induction the resulting deck is lexicographically ordered on haplotypes from columns j to $n-1$.
We refer to $deck(j)$ as the ordering after processing SNP j, with $deck(n)$ as the initial ordering. *This sorts without any pairwise comparisons in work* $O(mn)$

As we sort we will compute and store:

1. $A(j,i)$ the index of card i in the deck after processing SNP j.
2. Let z be card i in $deck(j+1)$ ($z=i$, when $j=n-1$)
   $LF(j,i)$ the position of z in $deck(j)$.
   Thus $LF(j,\star)$ records the permutation of cards from $j+1 \rightarrow j$.

It's easy to see that A and LF can be computed in time $O(mn)$.
We need two other arrays U, V
For each card i in deck $j+1$ define $w(i)$ to be the bit for SNP j.

$$U(j,i,x) = \min\{k \geq i | w(k) = x,\ x = 0,1\}$$

We can compute $U(j,\star,\star)$ in time $O(m)$ by scanning backwards
Boundary condition: $U(j,i,x) = m$ if there is no $k \geq i, w(k) = x$ Similarly define

$$V(j,i,x)\ = \max\{k \leq i | w(k) = x,\ x = 0,1\}$$

Boundary condition: $V(j,i,x) = -1$ if there is no $k \leq i, w(k) = x$

The Central Routine: *lfx*

```
#define YES 1
#define  NO 0

int
lfx (int col, int s, int e, int x, int *pnews, int *pnewe)
{
  int ts, te;

  *pnews = m;
  *pnewe = -1;

  ts = U[col][s][x] ;
  te = V[col][e][x] ;
```

```
  if (feasible (ts, te)) { // tests if interval is possible
    *pnews = lf (col, ts);
    *pnewe = lf (col, te);
    return YES;
  }
  return NO;
}
```

Call

```
lfx(col, s,  e, x, &news, &newe) ;
```

Here $I(s, e)$ is an interval on $deck(col+1)$ We claim that $I(news, newe)$ is the interval on $deck(col)$ containing precisely the indices of $I(s, e)$ where bit(col) matches x.

Proof: $I(ts, te)$ is the smallest such interval on $deck(col+1)$ by construction But conditional on bit(col) the map $LF(k, \star)$ maintains the order. the result follows.

First application

Given this machinery we now give a fast algorithm for (partial) haplotype exact matching.

Problem:

Given a haplotype $\mathbf{w} = (w_0, w_1, \ldots w_e)$ find the longest exact match $H(k, s), H(k, s+1), \ldots, H(k, e)$ in time independent of m.

The process is very simple.

Set $a = 0, b = m - 1, s = 0$.

For j = e, e-1, $\ldots 0$ do

Set $x = w(j)$. if $lfx(j, a, b, x, pnewa, pnewb) = NO$ set $s = j + 1$ and break

pnews, pnewb are pointers to newa, newb

Set $a = newa, b = newb$

end do

Set $k = A(s, a)$.

$H(k, s), H(k, s+1), \ldots$ is the required haplotype.

This procedure runs in constant time per SNP processed.

Next lecture: *fastLS* on haploids