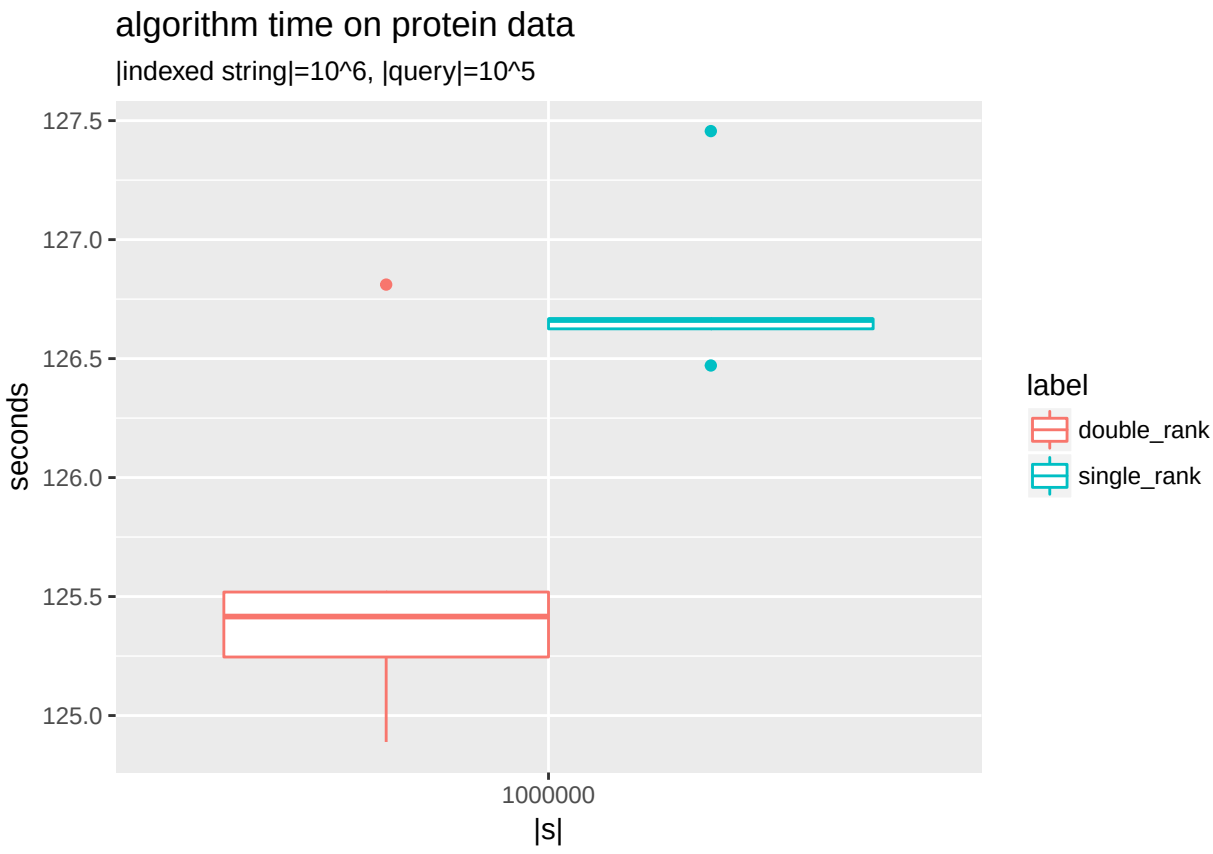
Lazy vs. non lazy

O. Denas

12/28/2016

Contents

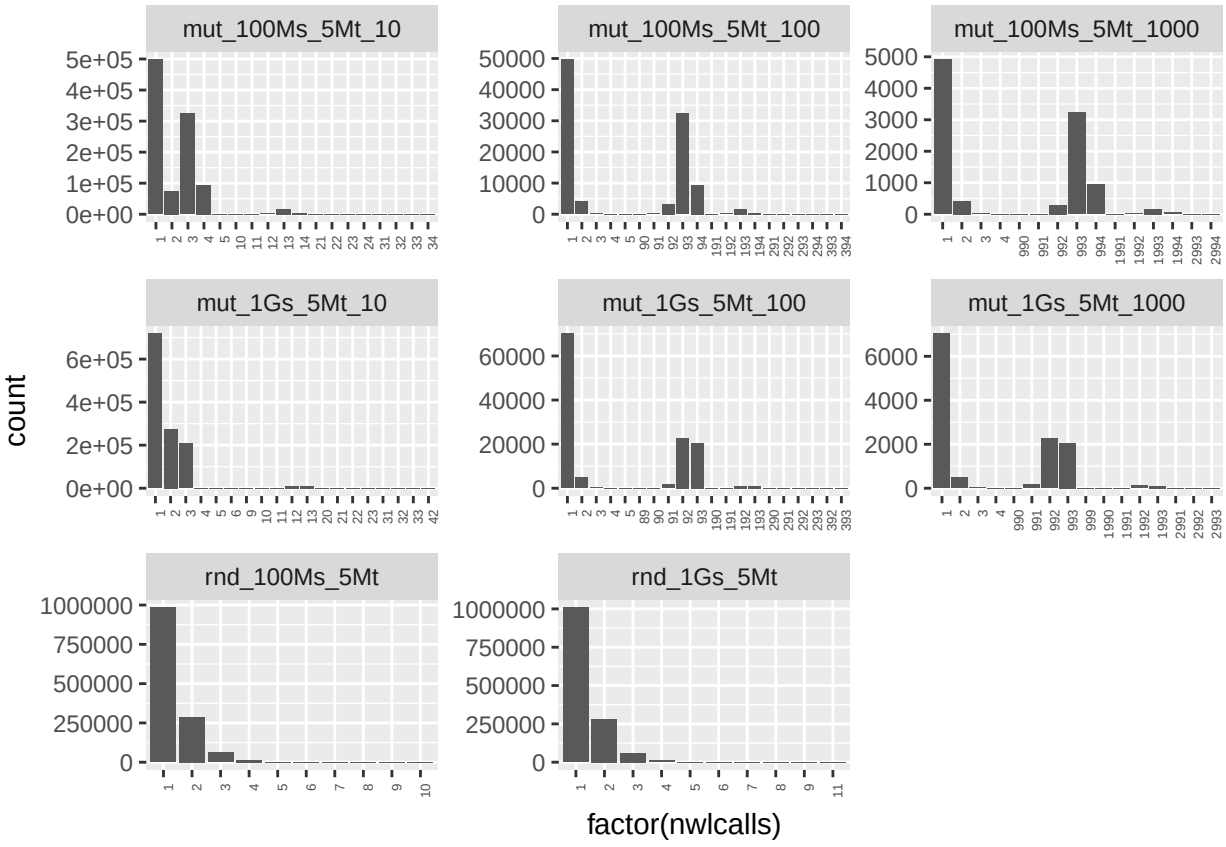
1 Double vs. single rank



2 Lazy vs non-lazy

2.1 Input properties

For various types ("mut_XMs_YMt_Z" means `s` and `t` are random identical strings of length X, and Y million respectively with mutations inserted every Z characters. "rnd_XMs_YMt" means `s` and `t` are random strings of length X, and Y million respectively) of inputs run the MS algorithm and count the number of consecutive `lazy_wl()` calls.

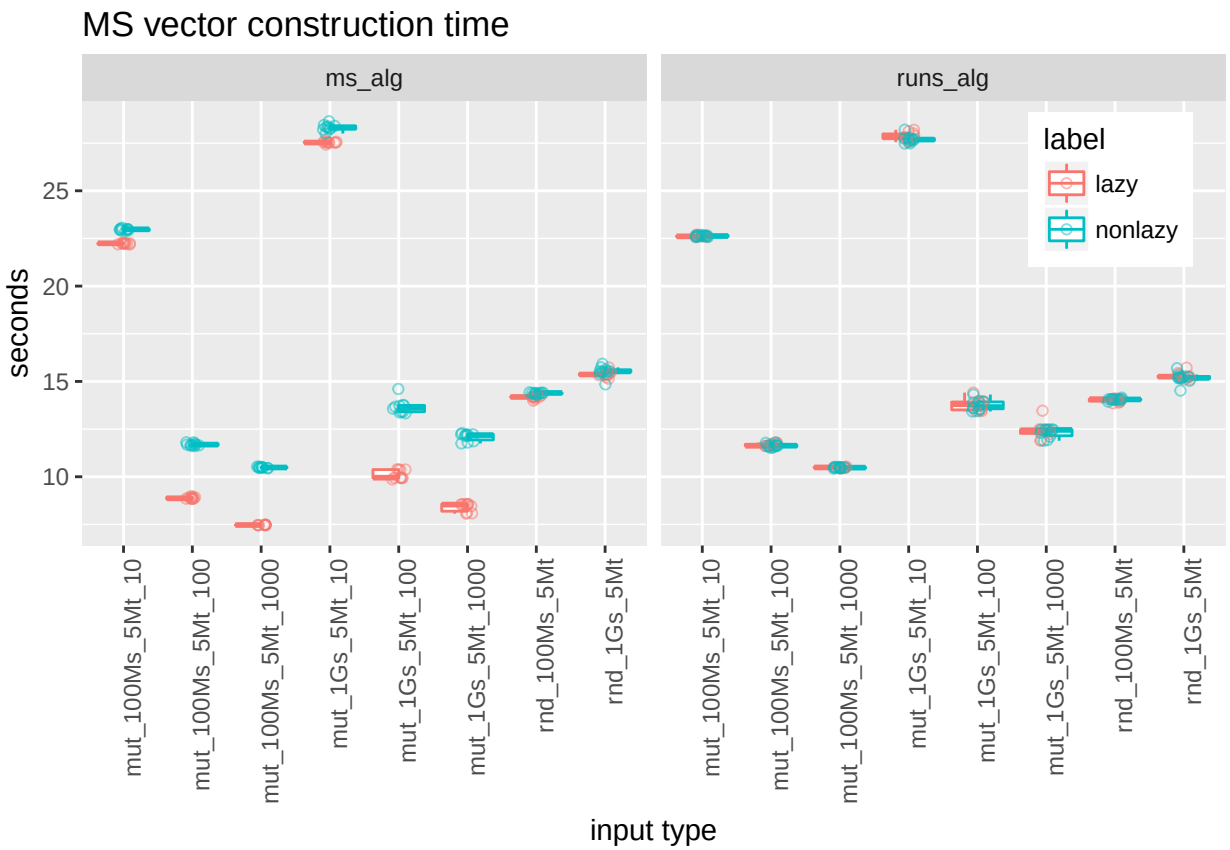


2.2 Code

The lazy and non-lazy versions differ in a couple of lines of code as follows

```
if(flags.lazy){
    for(; I.first <= I.second && h_star < ms_size; ){
        c = t[h_star];
        I = bstep_interval(st, I, c); //I.bstep(c);
        if(I.first <= I.second){
            v = st.lazy_wl(v, c);
            h_star++;
        }
    }
    if(h_star > h_star_prev) // // we must have called lazy_wl(). complete the node
        st.lazy_wl_followup(v);
} else { // non-lazy weiner links
    for(; I.first <= I.second && h_star < ms_size; ){
        c = t[h_star];
        I = bstep_interval(st, I, c); //I.bstep(c);
        if(I.first <= I.second){
            v = st.lazy_wl(v, c);
            h_star++;
        }
    }
}
```

2.3 Run time



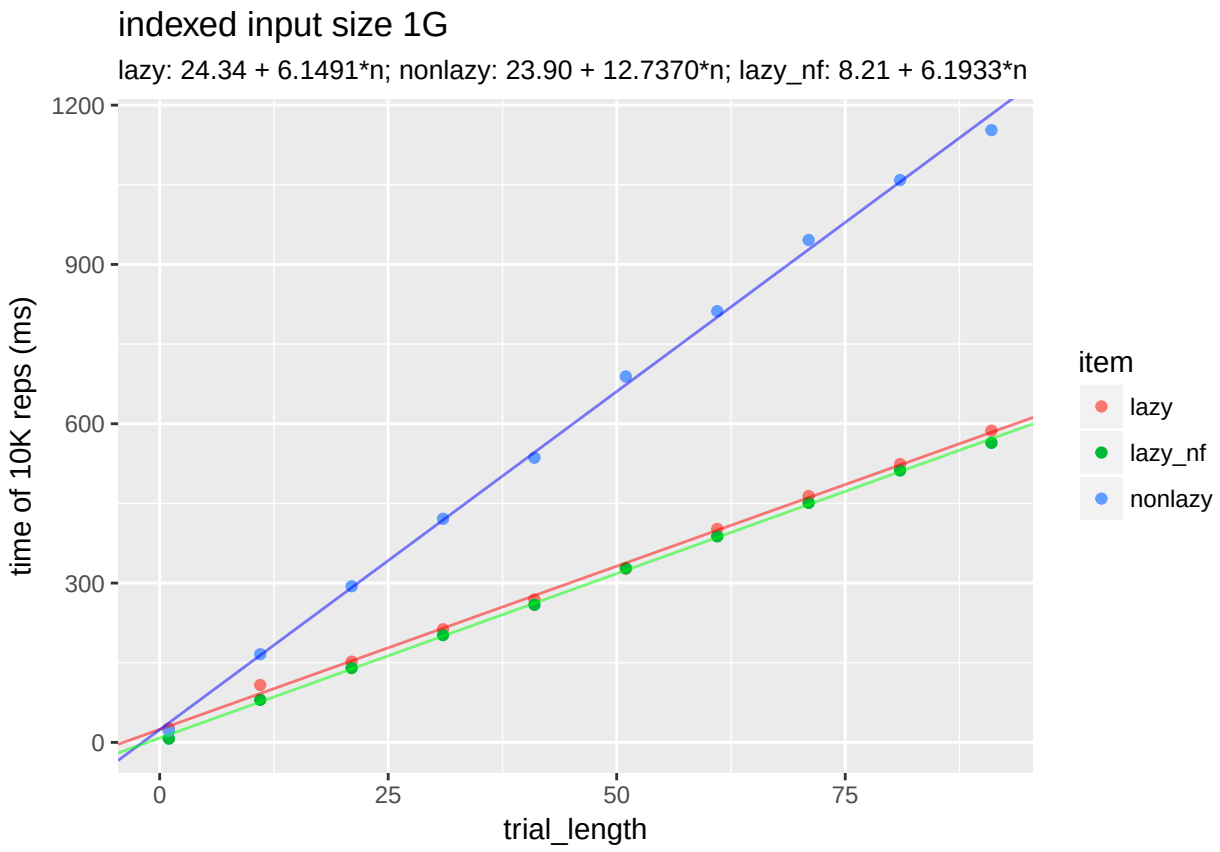
The right panel shows the time to construct the `runs` vector. This stage is the same for both versions and is shown as a control. On the left panel it can be seen that speedup correlates positively with both the size of the indexed string and the mutation period.

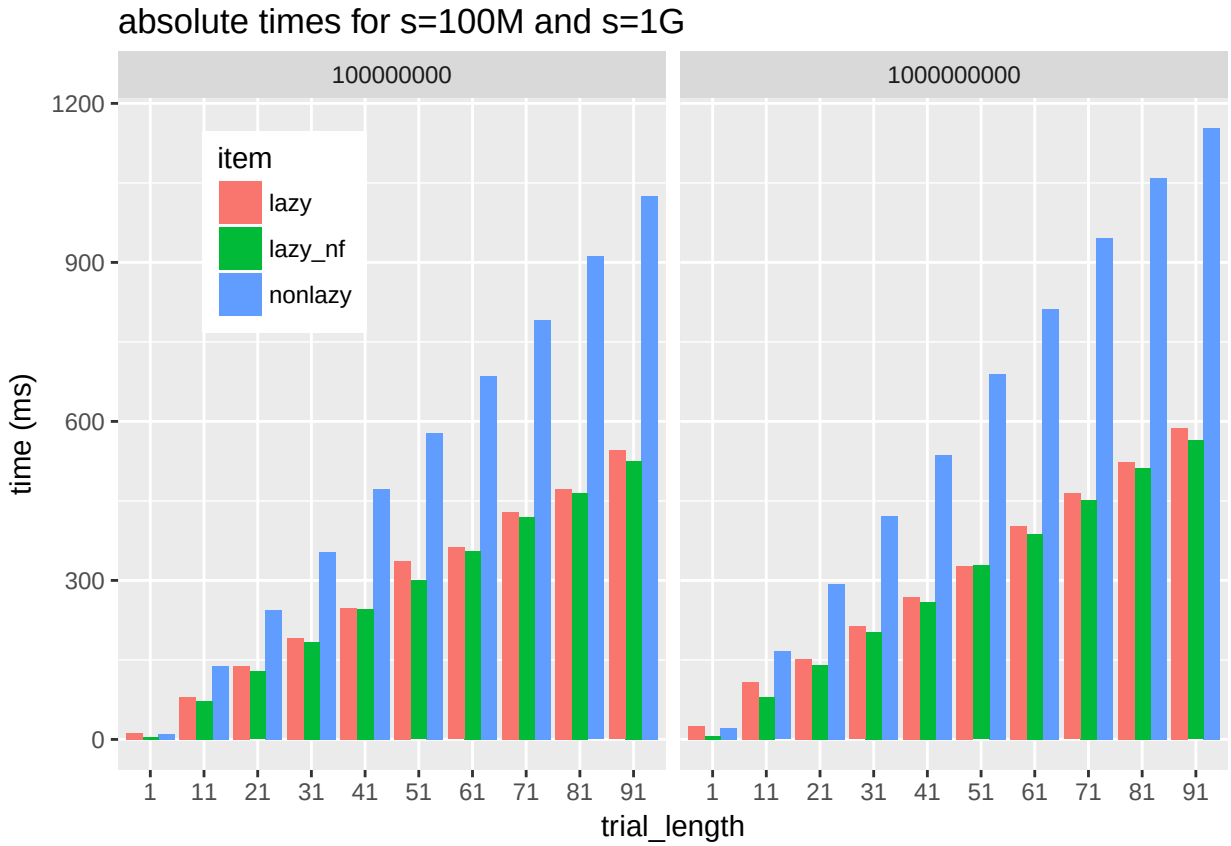
2.4 Sandbox timing

Measure the time of 10k repetitions of

- (lazy) n consecutive `lazy_wl()` calls followed by a `lazy_wl_followup()`
- (nonlazy) n consecutive `wl()` calls
- (lazy_nf) n consecutive `lazy_wl()` calls

```
// lazy
for(size_type i = 0; i < trial_length; i++)
    v = st.lazy_wl(v, s_rev[k--]);
if(h_star > h_star_prev) // // we must have called lazy_wl(). complete the node
    st.lazy_wl_followup(v);
...
// non-lazy
for(size_type i = 0; i < trial_length; i++)
    v = st.wl(v, s_rev[k--]);
...
// lazy_nf
for(size_type i = 0; i < trial_length; i++)
    v = st.lazy_wl(v, s_rev[k--]);
```





2.5 Check

We have measures of both the running time of the `ms` vector construction in the lazy and nonlazy fashion which are $t_l = l_l + a$ and $t_n = l_n + a$ respectively; only the ts being known. Furthermore, we have $\hat{l}_l$ and $\hat{l}_n$ estimations. Hence we should expect $\delta t = t_l - t_n = l_l + a - l_n - a = l_l - l_n$ to be similar to $\delta\hat{l} = \hat{l}_l - \hat{l}_n$.

```
##              b_path   t_l   t_n  l_l  l_n delta_t delta_l_hat
## 1   mut_100Ms_5Mt_10 22.23 22.97 4.05 3.53   -0.74        0.52
## 2  mut_100Ms_5Mt_100  8.89 11.70 2.90 4.07   -2.81       -1.16
## 3 mut_100Ms_5Mt_1000  7.47 10.49 2.79 4.12   -3.02       -1.33
## 4     mut_1Gs_5Mt_10 27.54 28.32 4.08 4.08   -0.78       -0.01
## 5    mut_1Gs_5Mt_100 10.11 13.66 3.15 4.65   -3.55       -1.50
## 6   mut_1Gs_5Mt_1000  8.40 12.09 3.05 4.70   -3.68       -1.65
## 7      rnd_100Ms_5Mt 14.17 14.38 4.68 3.65   -0.20        1.03
## 8        rnd_1Gs_5Mt 15.38 15.51 4.15 3.94   -0.14        0.20
```

The numbers are not identical (process dependent factors might influence the running time of function calls), but they are correlated ($corr(\delta t, \delta\hat{l}) = 0.96$).