2021-08-17_AppendixC_OptimizingOptimizers

James D. Boyko

17/08/2021

Which optimizer is optimal?

To test which optimizing protocol is best for hOUwie I simulated data under an OUM model which is known to have good results when initializing parameters from corHMM and OUwie. I compared three variables.

The optimization algorithm (algorithm):

nlopt_gn - a global search using nloptr's NLOPT_GN_DIRECT

nlopt_ln - a local search using nloptr's NLOPT_LN_SBPLX

sann - simmulated annealing approach using the R-package GenSA

The way initial parameters were generated (init):

fast - all parameters are based on simple calculations (for example alpha is $ln(2)/TreeHeight$)

good - all parameters are based on an optimization of OUwie and corHMM.

The number of times the search was conducted (n_starts):

1 - one start was conducted with initial parameters being either good or fast

10 - ten starts were conducted with initial parameters being either good or fast and additional starts being sampled from a log uniform distribution centered on the intiial parameters.

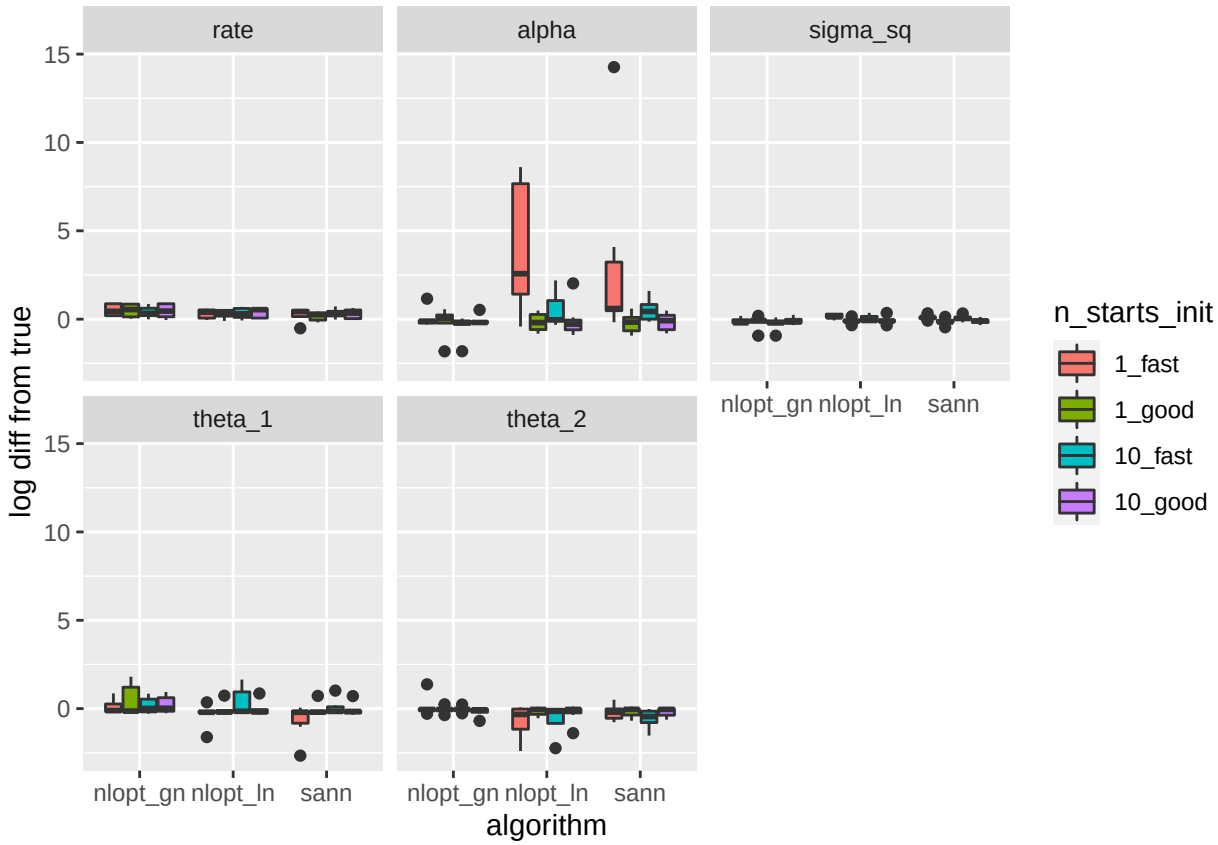
```
files <- dir("~/2020_hOUwie/optim_test/", full.names = TRUE)
results <- do.call(rbind, lapply(files, summarizeFile))
melted_results <- melt(results)
```

```
## Using algorithm, n_starts_init as id variables
```

```
colnames(melted_results)
```

```
## [1] "algorithm"     "n_starts_init" "variable"      "value"
```

```
ggplot(melted_results, aes(x = algorithm, y = value, fill = n_starts_init)) +
  geom_boxplot() +
  ylab("log diff from true") +
  facet_wrap(~variable)
```

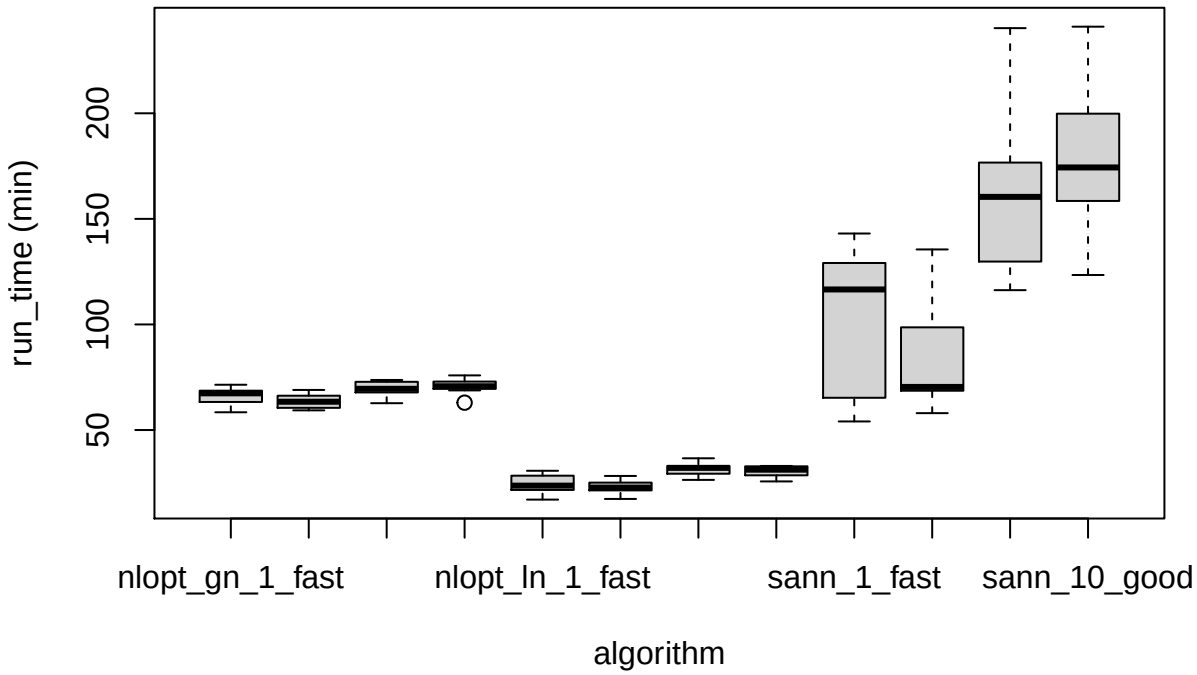


We choose to use the global optimizer because we cannot always rely on good initial parameters to work well. In this case, the corHMM model works well, but in the case of hidden states depending on continuous traits there is no information to fit a good corHMM model. The global search, however, worked about equally well whether we used informed or uninformed starting parameters.

Run time for 100 tip phylogeny

I also looked at run time in minutes.

```
results <- do.call(rbind, lapply(files, summarizeFileB))
boxplot(results$run_time ~ as.factor(paste0(results$algorithm, "_", results$n_starts_init)), xlab = "alg
```



I realize the names are cutoff, but the first four are global optimizers, the second four are local optimizers, and the final four are simulated annealing optimization. The majority of the variation is split amongst these three groups and run time is always secondary to how well the optimization procedure worked. As expected, local searches were the fastest, followed by global searches, and finally simulated annealing.